SOFTWARE TESTING

1. INTRODUCTION To Software Testing :

✓ What is software testing ?

→ Software testing is a process of executing a programme or application with the intent of finding the software bugs or errors.

Or : Software testing is a process used to help identify the correctness, completeness & quality of developed computer system software.

✓ * Purpose of Software Testing :-

Software testing has different goals & objectives. The major objectives of software testing are follows.

→ Finding defects which may get created by the programmer while developing the software.

→ Gaining confidence in & providing information about the level of quality.

→ To prevent defects.

→ To make sure that the end result meets the business & user requirements.

→ To ensure that it satisfy the business requirement specifications & system requirement specifications.

→ To gain the confidence of the customers by providing them a quality product.

* Who Should Software Testing ?

1. Test Managers :

→ Manage & control a software test project

→ Supervise test engineer.

→ Define & Specify a test plan

2. Software Test Engineers & Testers :-

→ Define test cases, write test Specifications, run test.

3. Independent Test group.

4. Development Engineers :-
→ Only perform unit test & integration testing.
5. Quality Assurance Group & Engineers :-
→ Perform System testing
→ Define software testing standards & quality control process.
* What to Software Test ?
→ 5 key software testing steps every engineer should perform.
1. Basic Functionality Testing
2. Code Review Testing
3. Static Code Analysis
4. Unit Testing
5. Single User Performance Testing
1. Basic Functionality Testing :-
→ Begin by making sure that every button on every screen works.
→ You also need to ensure that you can entered simple test into each field without crashing the software.
2. Code Review Testing :-
→ Before basic functionality testing, code review testing will be occure.
→ Main purpose of the code review testing is the to findout missing codes in the programme.
3. Static Code Analysis :-
→ The static code analysis can looks for many weakness in the source code, such as security vulnerabilities & potential concurrency issueses.
→ Use static code analysees tools to enforce coding standards & configure those tools to run automatically

as part of the build.

4. Unit Testing :-

→ Developers will write unit test to make sure that the unit (be it a method, class or component) is working as expected & test accross a range of valid & invalid inputs.

5. Single User Performance Testing :-

→ Sometimes some teams have load & performance testing backed into their continuous process & run load test as soon as code is checked in.

* Testing Process :- Imp DT-12.01.18

→ Testing is different from debuging.

→ Removing errors from your programmes is known as debuging but testing aims to locate an as yet undiscovered errors.

→ Software testing process is of 2 types

(a) Static

(b) Dynamic

(a) Static Testing Process :-

→ Static testing where in the software requirement specification is tested to check wheather it is as per user requirements or not.

→ We use techniques of code reviews, code inspections, work throughs & software technical review to do static testing.

(b) Dynamic Testing Process :-

→ It is dynamic testing as now the code is tested.

→ We use varrious techniques for dynamic testing like black box, white box, gray box testing.

* Selection of Good Test Cases :-

→ Designing good test case is a complex art. It is complex because

(a) Different types of test cases are needed for different classes of information.

(b) All test cases within a test suite will not be good. Test cases may be good in variety of ways.

→ People create test cases according to certain testing styles like domain testing or risk best testing.

* Measurement of Progress / Testing :-

→ There is no single scale i.e available to measure the testing progress.

→ A good project manager wants that worse conditions should occure in the very begining of the project only than in the later phases.

→ If errors are large in numbers, we can say either testing was not done throughly, so, their is no standard way to measure our testing process.

→ But matrix can be computed are the organizational process, project & product level. Each set of these measurements has its value in monitoring, planning & control.

/*// Matrics is assisted by 4 core components.

1. Schedule
2. Quality
3. Resources
4. Size

* Incremental Testing Approach :- IMP

→ To be effective, a software testers should be knowledge-able in 2 key areas

(a) & Software Testing Techniques

(b) The application under test

→ Our goal is to define a suitable list of test to perform within a tight dead line. There are 8 stages for this approaches

1. Stage - 1 → Exploration :

purpose - To gain familarity with the application.

2. stage - 2 - Base line Test :

purpose - To devise & execute a simple test case

3. Stage - 3 - Trains Analysis :

purpose - To evaluate wheather the application perform as expected when actual output can not be predetermine.

4. Stage - 4 :- Inventory

purpose - To identify the different catagories of data & create a test for each catagory item

5. Stage - 5 - Inventory Combinations -

purpose - To combine different input data.

6. Stage - 6 - Push the Boundries

purpose - To evaluate application behaviour at date boundries

7. Stage - 7 - Devious Data .

purpose - To evaluate system response when speciting bad data

8. Stage - 8 - & Stress the enveronment -

purpose - To attempts to break the system.

* Basic Terminology Related To software Testing :
We must define the following terminologies one by one
1. Error (or Mistake or Bugs) :
→ People make errors. A good synonym is mistake. When people make mistakes while coding, we call these mistakes bugs. Errors tend to propagate
2. Fault (or Defect) :
→ A missing or incorrect statement(s) in a program resulting from an error is a fault. So, fault is the representation of an error.
3. Failure :
→ A failure occurs when a fault executes. The manifested inability of a system or component to perform a required function within specified limits is known as a failure
4. Incident :
→ When a failure occurs, it may or may not be readily apparent to the user. An incident is the symptom associated with a failure that alerts the user to the occurrence occurrence of a failure
5. Test :
Testing is concerned with errors, faults, failures & incidents. A test is the act of exercising software with test cases.
6. Test Case :
→ A test case has an identity & is associated with program behaviour. A test case also has a set of inputs & a list of expected outputs. The essence of software testing is to determine a set of test cases for the item to be tested.

The test case template is discussed next.

Test Case ID
Purpose
Preconditions
Inputs
Expected outputs
Postconditions
Execution History
Data Result Version Run By

→ Inputs are of two types

(a) Preconditions : Circumstances that hold prior to test case execution.

(b) Actual Inputs : That were identified by some testing method.

7. Test Suite :

→ A collection of test scripts or test cases that is used for validating bug fixes (or finding new bug) within a logical or physical area of a product.

8. Test Script :

The step-by-step instructions that describe how a test case is to be executed. It may contain one or more test cases.

9. Test Ware :

→ It includes all testing documentation created during testing process. For e.g. test specification, test scripts, test cases, test data, environment specification

10. Test Oracle :

Any means used to predict the outcome of a test.

11. Test Log :

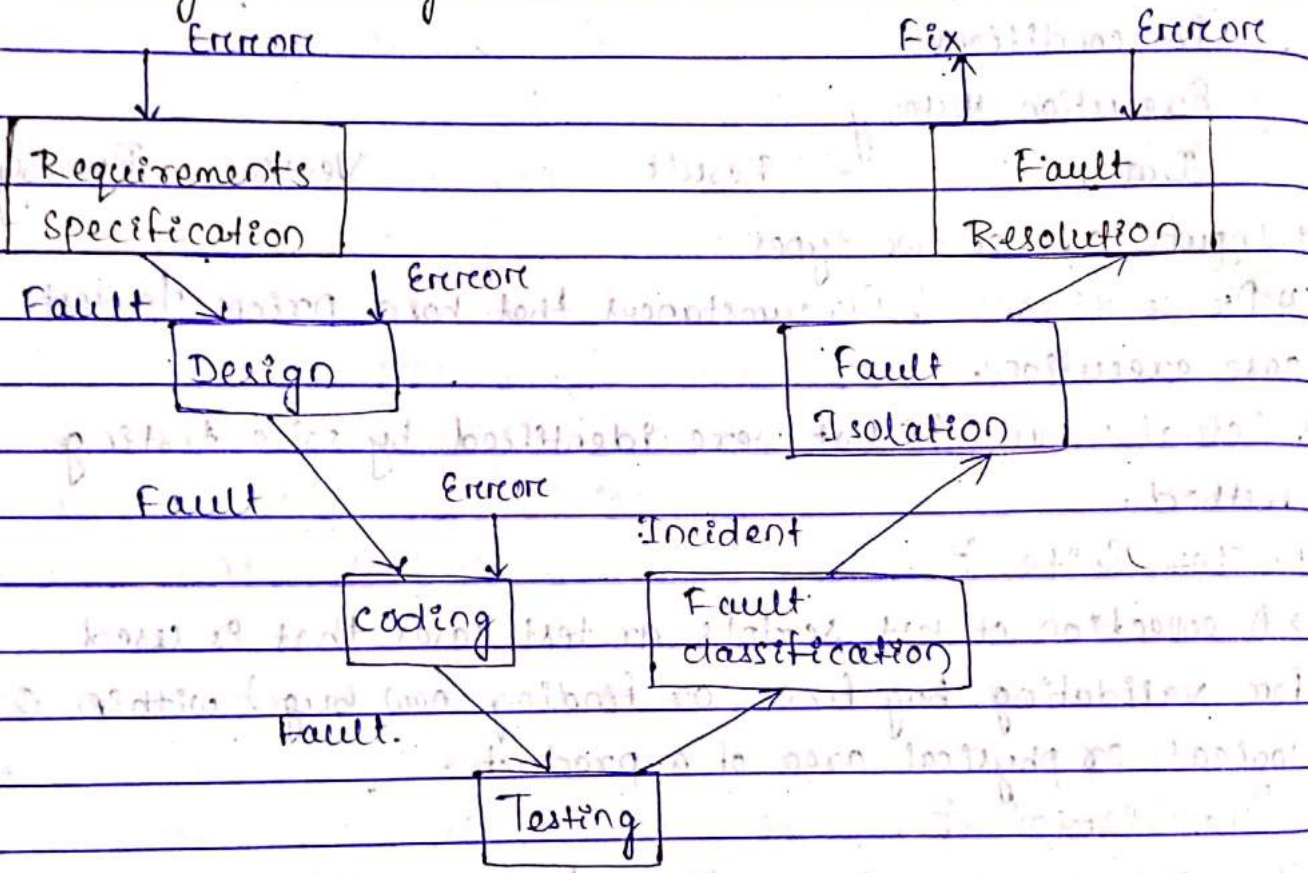
A chronological record of all relevant details about

the execution of a test.

12. Test Report :-

→ A document describing the conduct & results of testing carried out for a system.

* Testing Life Cycle :-



(A Testing Life Cycle)

→ In the development phase, three opportunities arise for errors to be made, resulting in faults that propagates through the remainder of the development process.

→ The first three phases are putting bugs IN, the testing phase is finding bugs & the last 3 phases are Getting Bugs OUT.

→ The fault resolution step is another opportunity for errors & new faults. When a fix causes formerly

correct software to misbehave, the fix is deficient.

* When to stop Testing ?

→ Testing is potentially endless. We cannot test till all defects are unearthed & removed. It is simply impossible.

→ At some point, we have to stop testing & ship the software.

→ Testing is a trade-off between budget, time & quality.

→ It is driven by profit models.

→ The pessimistic approach to stop testing is whenever some or any of the allocated resources time, budget or test cases are exhausted.

→ The optimistic stopping rule is to stop testing when either reliability meets the requirement, or the benefit from continuing testing cannot justify the testing cost.

* Principles of Testing :-

To make software testing effective & efficient we follow certain principles. These principles are stated below

1. Testing should be Based on User Requirements :-

This is in order to uncover any defects that might cause the program or system to fail to meet the client's requirements

2. Testing Time & Resources are Limited :-

Avoid redundant tests.

3. Exhaustive Testing is Impossible :-

As stated by Myers, it is impossible to test everything due to huge data space & large number of paths that a program flow might take.

4. Use Effective Resources to Test :-

This represents use of the most suitable tools, procedures & individuals to conduct the tests.

5. Test Planning Should be Done Early :-
→ This is because test planning can begin independently of coding & as soon as the client requirements are set.
6. Testing Should Begin "in small" & progress Toward Testing "in large" :-
The smallest programming units (or modules) should be tested first & then expand to other parts of the system.
7. Testing should be conducted by an independent third party.
8. All tests should be traceable to customer requirements.
9. Assign best persons for testing. Avoid programmers.
10. Test should be planned to show software defects & not their absence.
11. Prepare test reports including test cases & test results to summarize the results of testing.
12. Advance test planning is must & should be updated timely.

* Limitation of Testing :-
→ Testing can show presence of error's - not their absence
→ No matter how hard you try, you would never find the last bug in an application.
→ The domain of possible inputs is too large to test.
→ There are too many possible paths through the program to test.
→ In short, maximum coverage through minimum test-cases. That is the challenge of testing.

→ Various testing techniques are complementary in nature & it is only through their combined use that one can hope to detect most errors.

* Availability of Testing Tool, Techniques, Metrics :
There are an abundance of software testing tools that exist. Some of them are listed below.

(a) Mothora :
It is an automated mutation testing tool-set developed at purdue university. Using mothora, the tester can create & execute test cases, measure test case adequacy, determine input-output correctness, locate & remove faults or bugs & control & document the test.

(b) NuMega's Bounds checker, Rational's Purify :
They are run times checking & debugging aids. They can both check & protect against memory leaks & pointer problems.

(c) Ballista COTS Software Robustness Testing Harness [Ballista] :
- It is a full-scale automated robustness testing tool. It gives quantitative measures of robustness comparisons across operating systems. The goal is to automatically test & harden commercial off-the-shelf (COTS) software against robustness failures.

Software Verification & Validation :-

2.1 INTRODUCTION :-

→ The evolution of software that satisfies its user expectations is a necessary goal of a successful software developement organization.

→ To achieve this goal, software engineering practices must be applied throughout the evolution of the software product.

→ Most of these practices attempt to create & modify *software in a manner that maximizes the probability of satisfying its users expectations.

* Verification & validation :-

→ Software verification & validation is a "a systems engineering process employing a rigorous methodology for evaluating the correctness & quality of software product through the software life cycle.

Verification	Validation
→ It is a static process of verifying documents, design & code.	→ It is a dynamic process of validating / testing the actual product.
→ It does not involve executing the code.	→ It involves executing the code
→ It is human based checking of documents / files.	→ It is computer based execution of program
→ Target is requirements specification, application architecture, high level & detailed design, database	→ Target is actual product a unit, a module, a set of intergrated modules, final product

design.	
→ It uses methods like inspections, walk throughs, Desk-checking etc.	→ It uses methods like black box, gray box, white box testing etc.
→ It generally comes first - done before validation.	→ It generally follows verifications.
→ It answers to the question - Are we building the product right ?	→ It answers to the question - Are we building the right product ?
→ It can catch errors that validation cannot catch.	→ It can catch errors that verifications cannot catch.

* Quality Assurance (QA) & Quality Control (QC) :-

Quality Assurance (QA) - The planned & systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled, is known as Quality Assurance. (QA)

Quality Control (QC) : The observation techniques & activities used to fulfill requirements for quality is known as Quality Control (QC)

* Difference Between QA & QC :-

QA	QC
→ It is process related	→ It is product related.
→ It focuses on the process used to develop a product.	→ It focuses on testing of a product developed for a product under development.
→ It involves the quality of the processes.	→ It involves the quality of the products.
→ It is a preventive control	→ It is a detective control.
→ Allegiance is to development.	→ Allegiance is not to development
→ QA will be come before the QC	→ QC will be come after QA

* Verification & Validation (V&V) Limitations :-

→ The overall objective of software V & V approaches is to ensure that the product is free from failures & meets its user's expectations.

→ There are several theoretical & practical limitations that make this objective impossible to obtain for many products.

1. Theoretical Foundations :-

Howden claims the most important theoretical result in program testing & analysis is that no general purpose ~~purpose~~ testing or analysis procedure can be used to prove program correctness.

2. Impractically Impracticality of Testing all Data :-

→ For most programs, it is impractical to attempt to test the program will all possible inputs, due to a combinational explosion.

→ For those inputs selected, a testing oracle is needed to determine the correctness of the output for a perticular test input.

3. Impracticality of Testing All Paths :-

→ For most programs, it is impractical to attempt to test all execution paths through the product, due to a combinational explosion.

→ It is also not possible to develop an algorithm for generating test data for paths in an arbitrary product, due to the mobility to determine path feasibility.

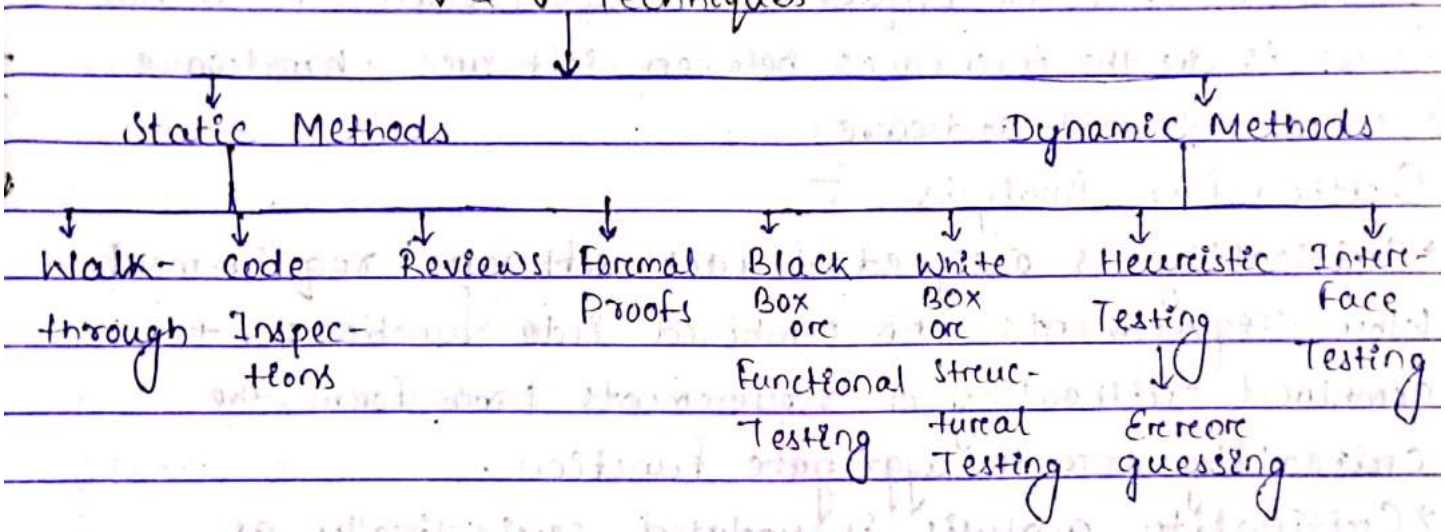
4. No Absolute Proof of Correctness :-

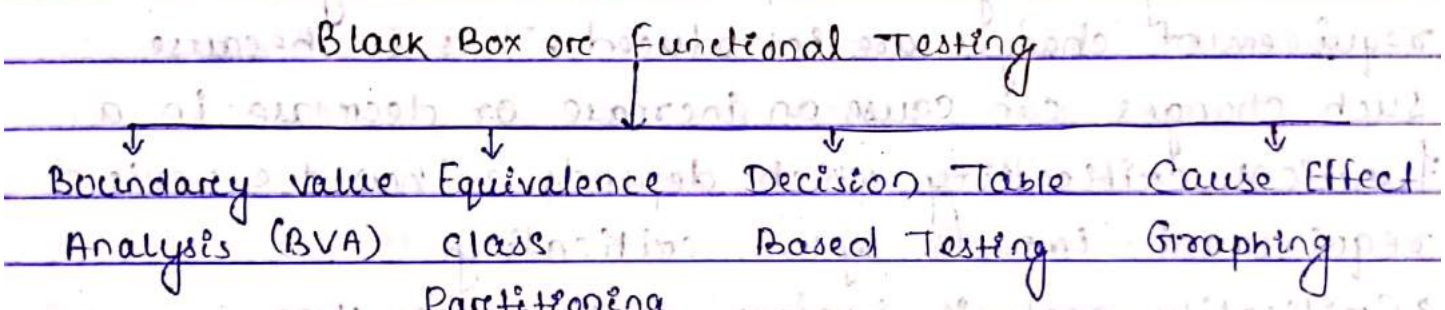
→ Howden claims that there is no such thing as an absolute proof of correctness.

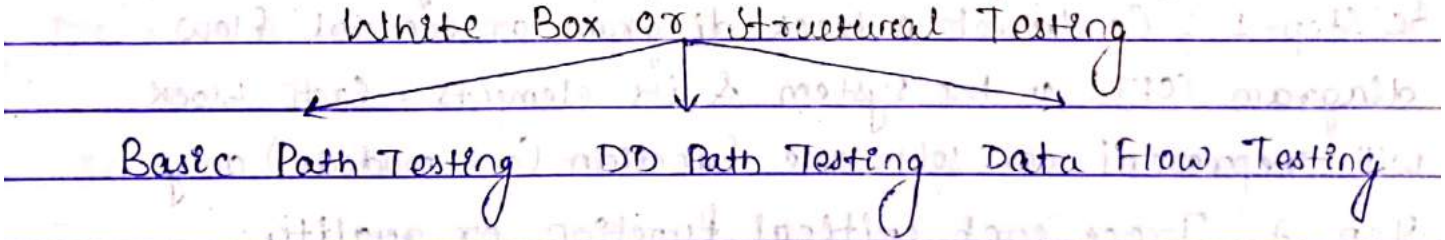
→ Instead, he suggests that there are proofs of

equivalency, i.e proofs that one description of a product is equivalent to another description.

* Categorizing V & V Techniques :







* Role of V & V in SDLC

Traceability Analysis :

→ It traces each software requirement back to the system requirements established in the concept activity.

→ This is to ensure that each requirement correctly satisfies the system requirements & that no extraneous software requirements are added.

Interface Analysis :-

→ It is the detailed examination of the interface requirements specifications. The evaluation criteria is same as that for requirements specification. The main focus is on the interfaces between software, hardware, user & external software

Criticality Analysis :-

→ Criticality is assigned to each software requirement. When requirements are combined into functions, the combined criticality of requirements form form the critrality for the aggregate function.

→ Criticality analysis is updated periodically as requirement changes are introduced. This is because such changes can cause an increase or decrease in a functions criticaltity which depends on how the revised requirement impacts system criticality.

→ Criticality analysis involves the following steps.

Step-1 : Construct a block diagram or control flow diagram (CFD) of the system & its elements. Each block will represent one software function (or module) only

Step-2 : Trace each critical function or quality requirement through ~~CFD~~ CFD CFD

Step-3 : Classify all traced software functions as critical to

(a) Proper execution of critical software functions

(b) Proper execution of critical quality requirements.

Step-4 : Focus additional analysis on these traced critical software functions.

Step-5 : Repeat critality analysis for each life cycle process to determine whether the implementation

details shift the emphasis of the criticality.

Hazard & Risk Analysis :-

→ It is done during the requirements definition activity. Now hazards or risks are identified by further refining of the system requirements into detailed software requirements. These risks are assessed for their impact on the system.

* *

* Proof of Correctness (Formal Verification) :-

→ A proof of correctness is a mathematical proof that a computer program or a part there of will, when executed, yield correct results i.e results fulfilling specific requirements. Before proving a program correct, the theorem to be proved must, of course, be formulated

Hypothesis :- The hypothesis of such a correctness theorem is typically a condition that the relevant program variables must satisfy immediately before the program is executed. This condition is called the 'precondition'.

Thesis :- The thesis of the correctness theorem is typically a condition that the relevant program variables must satisfy immediately 'after' execution of the program. This latter condition is called the 'postcondition'.

"If the condition, V, is true before execution of the program, S, then the condition, P, will be true after execution of S"

Where V is precondition & P is postcondition

* Simulation & Prototyping :-

→ Simulation & prototyping are techniques for analyzing the expected behaviour of a product. There are

many approaches to constructing simulations & prototypes that are well documented in the literature.

→ For V&V purposes, simulations & prototypes are normally used to analyze requirements & specifications to ensure that they reflect the user's needs. Since they are executable, they offer additional insight into the completeness & correctness of these documents.

→ Simulations & prototypes can also be used to analyze predicted product performance, especially for candidate product designs, to insure that they conform to the requirements.

→ It is important to note that the utilization of simulation & prototyping as V & V technique requires that the simulations & prototypes themselves be correct.

* Requirements Tracing :-

"It is a technique for insuring that the product, as well as the testing of the product, addresses each of its requirements." The usual approach to performing requirements tracing uses matrices.

(a) One type of Matrix maps Requirements to Software modules. Construction & analysis of this matrix can help insure that all requirements are properly addressed by the product & that the product does not have any superfluous capabilities.

(b) Another type of Matrix maps requirements to test cases. Construction & analysis of this matrix can help insure that all requirements are properly tested.

(c) A third type of Matrix maps requirements to their evalution approach. The evalution approaches may

consist of various levels of testing, reviews, simulation etc.

* Software V & V Planning (SVVP) :-

→ The development of a comprehensive V & V plan is essential to the success of a project. This plan must be developed early in the project

→ Depending on the development approach followed, multiple levels of test plans may be developed, corresponding to various levels of V & V activities.

→ IEEE 836 has documented the guidelines for the contents of system, software, build & module test plants.

Step-1 : Identification of V & V Goals

V & V goals must be identified from the requirements & specifications. These goals must be address those attributes of the product that correspond to its user expectations. These goals must be achievable, taking into account both theoretical & practical limitations.

Step-2 : Selection of V & V Techniques

Once step-1 (above) is finished, we must select specific techniques for each of the product that evolves during ~~SDLC~~ SDLC.

(a) During Requirements Phase - The application applicable techniques for accomplishing the V & V objectives for requirements are technical reviews, prototyping & simulations. The review process is often called as a System Requirement Review (SRR).

(b) During Specifications Phase - The applicable techniques The for this phase are technical reviews, requirements tracing, prototyping & simulations. The requirements must be traced to the specifications.

(c) During Design Phase - The techniques for accomplishing the V&V objectives for designs are technical reviews, requirements tracing, prototyping, simulation & proof of correctness.

(d) During Implementation Phase :- The applicable techniques for accomplishing V&V objectives for implementation are technical reviews, requirements tracing, testing & proof of correctness. Various code review techniques such as walk throughs & inspections exist.

(e) During Maintenance Phase :-
Since changes describe modifications to products, the same techniques used for V&V during development may be applied during modification. Changes to implementation require regression testing.

Step-3 : Organizational Responsibilities
The organizational structure of a project is a key planning consideration for project managers. An important aspect of this structure is delegation of V&V activities to various organizations.

(a) Development Organization -
(b) Independent Test Organization (ITO) -
(c) Software Quality Assurance (SQA) Organizations -
(d) Independent V&V Contractor -

Step-4 : Integrating V&V Approaches :
Once a set of V&V objectives has been identified, an overall integrated V&V approach must be determined. This approach involves integration of techniques appl to various life cycle phases as well as delegation of these tasks among the projects organization.

Step - 5 : Problem Tracking
→ It involves documenting the problems encountered during V & V effort.
→ Routing these problems to appropriate persons for correctness.
→ Insuring that correctness corrections have been done.
Step - 6 : Tracking Test Activities
Step - 7 : Assessment.

~~Technical Testing~~

* Software Testing Reviews (STRs) :-
→ A review process can be defined as a critical evaluation of an object. It includes techniques such as walk-throughs, inspections & audits. Most of these approaches involve a group meeting to assess a work product.
→ Software technical reviews can be used to examine all the products of the software evolution process.

Rationale for STRs :-
(a) Error Prone Software Development & Maintenance Process :- The complexity & error-prone nature of developing & maintaining software should be demonstrated with statistics depicting error frequencies for intermediate software products.
(b) Inability to Test all Software :- It is not possible to test all software. Clearly exhaustive testing of code is impractical.
(c) Reviews are a form of Testing :- The degree of formalism, scheduling & generally positive attitude afforded to testing must exist for software technical reviews if quality products are to be produced.

(d) Reviews are a way

(d) Reviews are a way of Tracking a Project :-
Through identification of deliverables with well defined entry & exist criteria & successful review of these deliverables, progress on a project can be followed & managed more easily.

(e) Reviews Provide Feedback :-
The instructor should discuss & provide examples about the value of review processes for providing feedback about software & its development process.

(f) Educational Aspects of Reviews :-
It includes benefits like a better understanding of the software by the review participants that can be obtained by reading the documentation as well as the opportunity of acquiring additional technical skills by observing the work of others.

* Types of STRs :-
→ A variety of Software Technical Reviews (STRs) are possible on a project depending upon the development model followed, the type of software product being produced & the standards which must be adhered. These developmental models may be
(i) Waterfall Model
(ii) Rapid Prototyping
(iii) Iterative enhancement
(iv) Maintenance activity Modeling
And the current standards may be
(i) Military Standards
(ii) IEEE standards (iii) NBS standards

Informal Review	Formal Reviews
→ It is a type of review that typically occurs spontaneously among peers.	→ It is a planned meeting
→ Reviewers have no responsibility.	→ Reviewers are held accountable for their participation in the review
→ No review reports are generated.	→ Review reports containing action items is generated & acted upon.

* Review Methodologies :

There are 3 approaches to reviews

(a) Walk through (or presentation reviews)

(b) Inspection (or user product reviews)

(c) Audits

Inspection	Walk throughs
→ It is a five-step process i.e well-formalized.	→ It has fewer steps than inspection & is a less formal process.
→ It uses checklists for locating errors	→ It does not use a checklist.
→ It is used to analyze the quality of the process.	→ It is used to improve the quality of the product
→ This process takes longer time.	→ It does not take longer time.
→ It focuses on training of junior staff.	→ It focuses on finding defects.

* Independent V & V Contractor (IV&V) :

→ An independent V&V contractor may sometimes be used to insure independent objectivity & evaluation for the

customers.

→ The use of a different organization, other than the software development group, for software V&V is called independent verification & validation (IV&V).

→ 3 types of independence are usually required.

1. Technical Independence :

→ It requires that members of the IV&V team (organization or group) may not be personnel involved in the development of the software.

→ This team must have some knowledge about the system design or some engineering background enabling them to understand the system.

→ Technical independence is crucial in the team's ability to detect the subtle software requirements, software design & coding errors that escape detection by development testing & SQA reviews.

→ The IV&V team uses or develops its own set of test & analysis tools separate from the developers tools whenever possible.

2. Managerial Independence :

→ It means that the responsibility for IV&V belongs to an organization outside the contractor & program organizations that develop the software.

→ The IV&V team provides its finding in a timely fashion simultaneously to both the development team & the systems management who acts upon.

3. Financial Independence :

→ It mean that control of the IV&V budget is retained in an organization outside the contractor & program organization that develop the software.

→ This independence protects against diversion of funds or adverse financial pressures or influences that may cause delay or stopping of IV&V analysis & test tasks & timely reporting of results.

* Positive & Negative Effect of Software V&V on Project

→ Software V&V has some positive effects on a software project. These are -

1. Better quality of software. This includes factores like completeness, consistency, readability & testability of the software.
2. More stable requirements.
3. More rigorous development planning, at least to interface with the software V&V organization.
4. Better adherence by the development organization to programming language & development standards & configuration management practices.
5. Early error detection & reduced false starts.
6. Better schedule compliance & progress monitoring.
7. Greater project management visibility into interim technical quality & progress.
8. Better criteria & results for decision-making at formal reviews & audits.

Some Negative Effects of software V&V on a software development project include -

1. Additional project cost of software V&V (10-30% extra)
2. Additional interface involving the development team, user & software V&V organization. For example attendance at software V&V status meetings, anomaly resolution meetings.

3. Additional documentation requirements, beyond the deliverable products. If software V&V is receiving incremental program & documentation releases.
4. Need to share computing facilities with & to provide access to, classified data for the software V&V organization.
5. Lower development staff productivity if programmers & engineers spend time explaining the system to software V&V analysis, especially if explanations are not documented.
6. Increased paper work to provide written responses to software V&V error reports & other V&V data requirements. For example, notices of formal review & audit meetings, updates to software release schedule & response to anomaly reports.
7. Productivity of development staff affected adversely in resolving invalid anomaly reports.

→ Some steps can be taken to minimize the negative effects & to maximize the positive effects of software V&V. To recover much of the software V&V costs, software V&V is started early in the software requirements phase.

✓

* Standard For Software Test Documentation (IEEE 829)

→ The IEEE 829 standard for software test documentation describes a set of basic software test documents. It defines the content & form of each test document.

→ In this addendum we give a summary of the structure of the most important IEEE 829 defined

Ieee test documents.

→ This addendum is based on the course materials by Jukka Paakki (& the IEEE 829 standard)

Test Plan :

→ Test-plan Identifier
→ Introduction
→ Test Items
→ Features tof be Tested
→ Features not to be tested.
→ Approach
→ Item pass/fail Criteria
→ Suspension Criteria & resumption
→ Test Deliverables
→ Testing Tasks
→ Environmental needs
→ Responsibilities
→ Staffing & Training Needs
→ Schedule.
→ Risks & contingencies.
→ Approvals

Test-case Specification :

→ Test-case specification identifier
→ Test Items
→ Input Specifications
→ Output Specifications
→ Environmental Needs
→ Special procedural Requirements
→ Intercase dependecies

Test - Incident Report (Bug Report)

→ Bug - Report Identifiers.

→ & Bug Description

→ Impact

Chap - 3

Black Box (or Functional Testing Techniques)

* Introduction To Black Box (or Functional Testing) :-

→ The term 'Black Box' refers to the software which is treated as a black box. By treating it as a black box, we mean that the system or source code is not checked at all.

* Boundary Value Analysis (BVA) :-

→ It is a black box testing technique that believes & extends the concept that the density of defect is more towards the boundaries. This is done the following reasons :

(i) Programmers usually are not able to decide wheather they have to use <= operator or < operator when trying to make comparisons.

(ii) Different terminating conditions of for-loops, while loops & repeat loops may cause defects to move around the boundary conditions.

(iii) The requirements themselves may not be clearly understood, especially around the boundaries, thus causing even the correctly coded program to not perform to the correct way.

* What is BVA (Boundary Value Analysis) ?

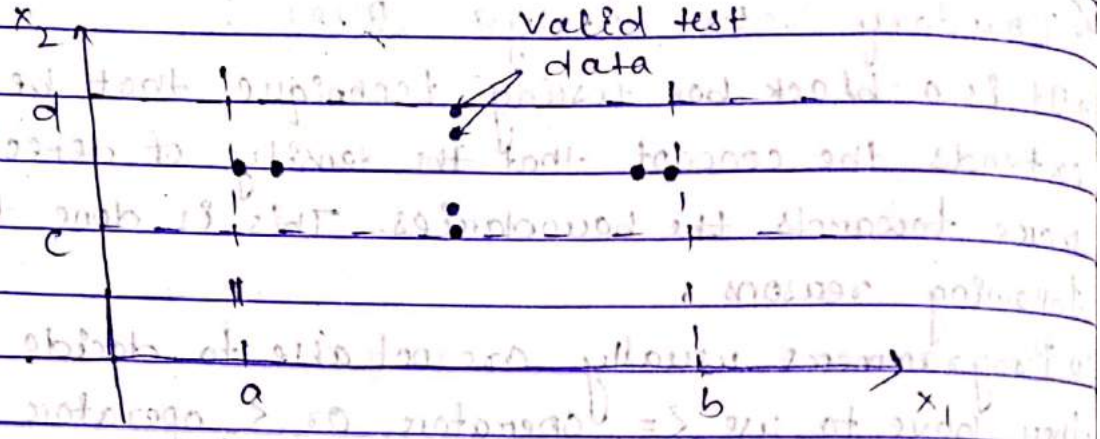
→ The basic idea of BVA is to use input variable values at their minimum, just above the minimum, a nomial value, just below their maximum & at their maximum. i.e {min, min+, nom, max-, max}

→ BVA is based upon a critical assumption that is known as 'Single fault assumption theory'.

→ According to this assumption, we derive the test

cases on the basis of the fact that failures are not due to simultaneous occurrence of two (or more) faults.

→ So, we derive test cases by holding the values of all but one variable at their nominal values & letting that variable assume its extreme values.



(BVA Test cases)

For a function of n variables, BVA yields (4n + 1) test cases.

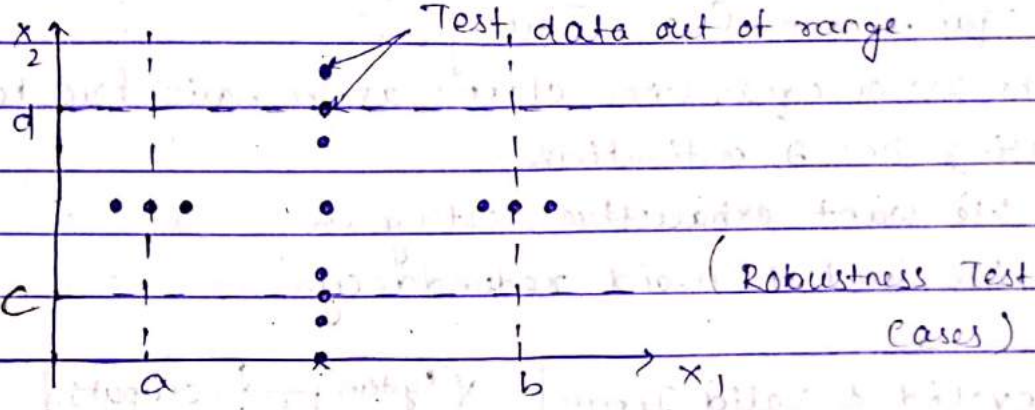
* Limitation of BVA :-

1. Boolean & logical variables present a problem for Boundary value Analysis.
2. BVA assumes the variables to be truly independent which is not always possible.
3. BVA test cases have been found to be rudimentary because they are obtained with very little insight & imagination.

* Robustness Testing :-

→ Another variant to BVA is Robustness testing. In BVA, we are within the legitimate boundary of the range i.e we consider the following

values for testing.
{min, min+, nom, max-, max, max+} whereas in Robustness testing, we try to cross these legitimate boundaries also. So, now we consider these for testing. {min-, min, min+, nom, max-, max, max+}

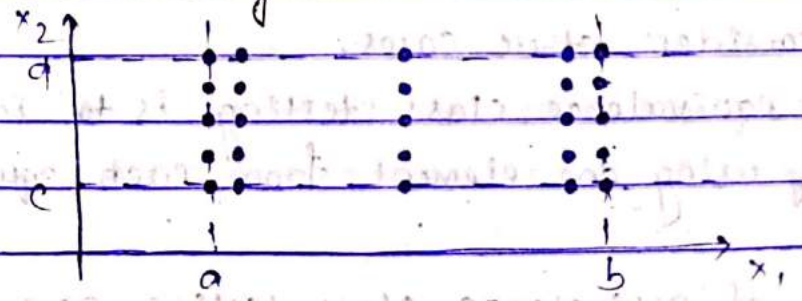


Again, with robustness testing we can focus on exception exception handling. With strongly typed languages, robustness testing may be very awkward.

→ For a program with n-variables, robustness testing will yield (6n + 1) test-cases.

→ Each dot represent a test value at which the program is to be tested. In robustness testing, we cross the legimate boundaries of input domain.

* Worst-Case Testing :-



→ If we reject our basic assumption of single fault assumption theory & focus on what happens when we reject this theory, it simply means that we want to

see that what happens when more than one variable has an extreme value.

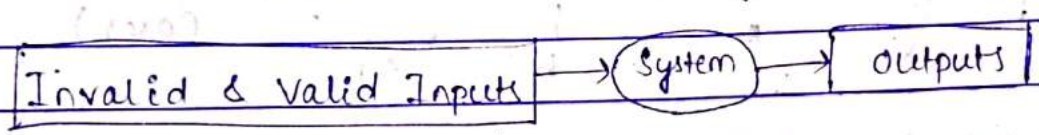
→ This is multiple path assumption theory. In electronic circuit analysis, this is called as "Worst - case analysis"

* Equivalence Class Testing :-

→ The use of equivalence classes as the basic for functional testing has 2 motivations.

(a) We want exhaustive testing &

(b) We want to avoid redundancy



(Equivalence class Partitioning)

→ This is not handled by BVA technique as we can see massive redundancy in the tables of test cases.

→ In this technique, the input & the output dommain is divided into a finite number of equivalence classes. Then, we select one representative of each class & test our program against it.

→ It is assumed by the testers that if one representative from a class is able to detect error then why should he considers other cases.

→ The idea of equivalence class testing is to identify test cases by using one element from each equivalence class.

→ Four types of equivalence class testing are -

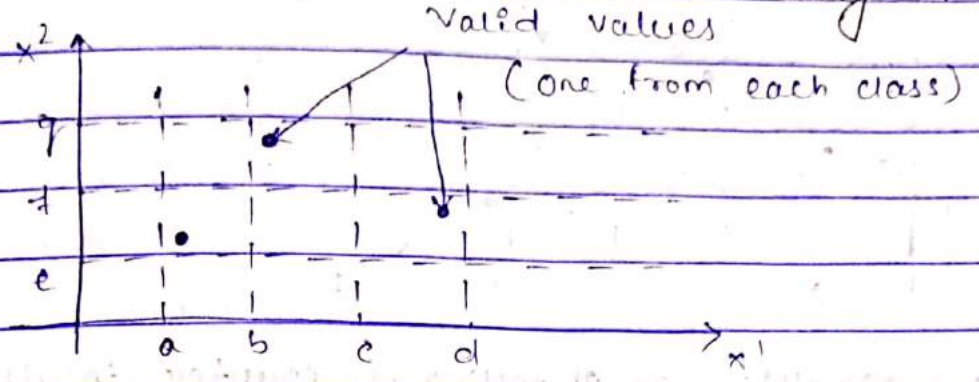
(a) Weak Normal Equivalence Class Testing

(b) Strong Normal Equivalence Class Testing

(c) Weak Robust Equivalence Class Testing

(d) Strong Robust Equivalence Class Testing.

a. Weak Normal Equivalence Class Testing :-

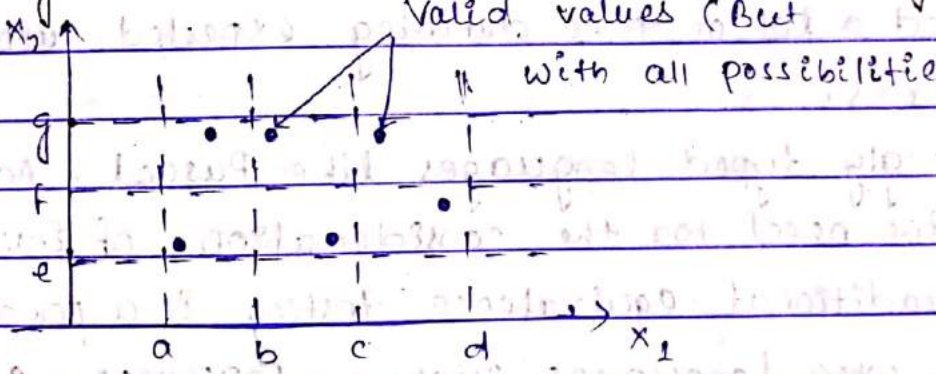


(Weak Normal Equivalence class Testing)

→ The word 'weak' means 'single fault assumption'. This type of testing is accomplished by using one variable from each equivalence class in a test case.

→ From each class we have one dot meaning that there is one representative element of each test case.

(b) Strong Normal Equivalence class Testing :-

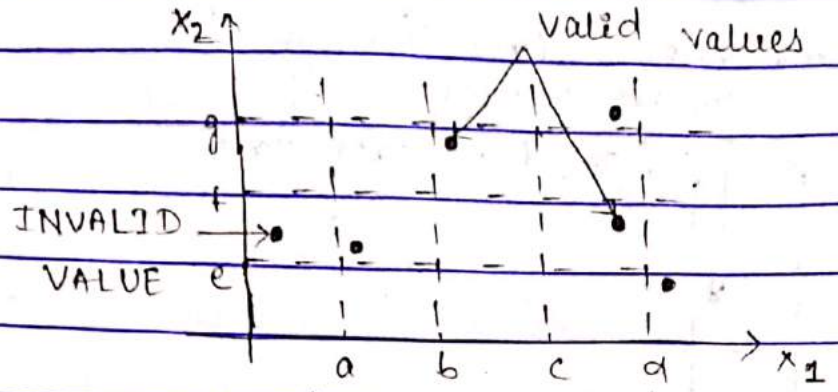


→ This type of testing is based on the multiple fault assumption theory. So, now we need test cases from each element of the cartesian product of the equivalence classes.

→ Just like we have truth tables in digital logic, we have similarities between these truth tables & our pattern of test cases. The cartesian product guarantees that we have a notion of 'completeness' in 2 ways - (a) we cover all equivalence classes.

(b) we have one of each possible combination of inputs

(c) Weak Robust Equivalence class Testing :-



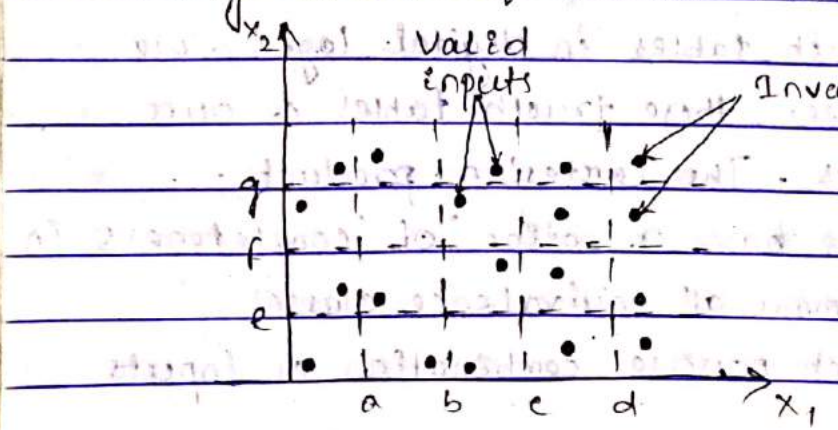
→ The name for this form of testing is counter intuitive & oxymoronic. The word 'weak' means 'single fault assumption' theory & the word Robust refers to invalid values.

→ 2 problems occure with robust equivalence testing. They are listed below :

(i) Very often the specification does not define what the expected output for an invalid test case should be. Thus, testers spend a lot of time defining expected outputs for these cases.

(ii) Also, strongly typed languages like Pascal, Ada, eliminate the need for the consideration of invalid inputs. Traditional equivalence testing is a product of the time when languages such as FORTRAN, C & COBOL were dominate. Thus this type of error was common.

(d) Strong Robust Equivalence Class Testing :-



→ This form of equivalence class testing is neither counter intuitive nor oxymoronic, but is just redundant.
→ As explained earlier also, 'robust' means consideration of invalid values & the 'strong' means multiple fault assumption.
→ We obtain the test cases from each element of the cartesian product of all the equivalence classes.
→ We find here that we have 8 robust (invalid) test cases & 12 strong or valid inputs. Each is represented with a dot. So, totally we have 20 test cases (represented as 20 dots) using this technique.

* Solved Examples -

Equivalence class Test Cases for the Triangle Problem

As stated in the problem definition earlier, we note that in our triangle problem four possible outputs can occur : (a) NOT - A - TRIANGLE (b) Scalene (c) Isosceles (d) Equilateral
We can use these to identify output (range) equivalence classes as follows :

$O_1 = \{\langle a, b, c\rangle$: the triangle is equilateral$\}$
$O_2 = \{\langle a, b, c\rangle$: the triangle is isosceles$\}$
$O_3 = \{\langle a, b, c\rangle$: the triangle is scalene$\}$
$O_4 = \{\langle a, b, c\rangle$: sides a, b & c do not form a triangle$\}$

Now, we apply these four techniques of equivalence class partitioning one by one to this problem.

(a) The four weak normal equivalence class test cases are

Case ID	a	b	c	Expected Output
WN_1	5	5	5	Equilateral
WN_2	2	2	3	Isosceles
WN_3	3	4	5	Scalene
WN_4	4	1	2	Not a triangle

* Decision Table Based Testing :-

→ Decision tables are of all the functional testing methods, those based on decision tables are the most rigorous because decision tables enforce logical rigour.

* What are Decision Tables ?

→ Decision tables are a precise & compact way to model complicated logic. They are ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions

→ It is another popular black-box testing technique.

A decision table has 4 portions.

(a) Stub portion

(b) Entry portion

(c) Condition portion

(d) Action portion

Rules →

Condition Stub	Condition Entry
Action Stub	Action Entries

Structure of Decision Table

→ A column in the entry portion is a rule. Rules indicate which actions are taken for the conditional circumstances indicated in the condition portion of the rule. Decision tables in which all conditions are binary are called as limited entry decision tables.

→ If conditions are allowed to have several values, the resulting tables are called extended entry decision tables.

→ To identify test cases with decision tables, we follow certain step.

Step-1 : For a module identify input condition (causes) & action (effect).

Step-2 : Develop a cause - effect graph

Step-3 : Transform this cause - effect graph, so obtained in step 2 to a decision table.

Step-4 : Convert decision table rules to test cases.

Each column of the decision table represents a test case. i.e | Number of Test Cases = Number of Rules |

For a limited entry decision tables, if it n conditions exist, there must be 2^n rules.

* Advantages, Disadvantages & Applications of Decision Tables :-

* Advantages of Decision Tables -

1. This type of testing also works ~~store~~ iteratively. The table that is drawn in the first iteration, acts as a stepping stone to derive new decision table (s), if the initial table is unsatisfactory.

2. These tables guranatee that we consider every possible combination of condition values. This is known as its "Completeness property". This property promises a form of complete testing as compared to other techniques.

3. Decision tables are declarative. There is no particular order for conditions & actions to occur.

Disadvantages of Decision Tables :-

Decision tables do not scale up well. We need to "factor" large tables into smaller ones to remove redundancy.

Applications of Decision Tables :-

→ The technique is useful for applications characterized by any of the following.

(a) Prominent if-then-else logic

(b) Logical relationships among input variables.

(c) Calculations involving subsets of the input variables.

(d) Cause-&-effect relationships between inputs & outputs.

(e) High cyclomatic complexity.

Examples -

Test cases for the triangle problem using decision Table Based Testing Technique

Case ID	a	b	c	Expected Output
D_1	4	1	2	Not a triangle
D_2	1	4	2	Not a triangle
D_3	1	2	4	Not a triangle
D_4	5	5	5	Equilateral
D_5	?	?	?	Impossible
D_6	?	?	?	Impossible
D_7	2	2	3	Isosceles
D_8	?	?	?	Impossible
D_9	2	3	2	Isosceles
D_{10}	3	2	2	Isosceles
D_{11}	3	4	5	Scalene

* Cause - Effect Graphing Technique :-

→ Cause - Effect graphing is basically a hardware testing technique adapted to software testing. It is black-box method. It considers only the desired external behaviour of a system. This is

a testing technique that aids in selecting test cases that logically relate causes (inputs) to effects (outputs) to produce test cases.

Causes & Effects :

A 'cause' represents a distinct input condition that brings about an internal change in the system. An 'effect' represents an output condition, a system transformation or a state resulting from a combination of causes.

Myer suggests the following steps to derive test cases :

Step-1 : For a module, identify the input conditions (causes) & actions (effect).

Step-2 : Develop a cause-effect graph

Step-3 : Transform cause-effect graph into a decision table.

Step-4 : Convert decision table rules to test cases. Each column of the decision table represents a test case.

Test Cases for the triangle Problem :

Step 1 - Firstly, we must identify the causes & its effects. The causes are -

C_1 : Side x is less than sum of y & z

C_2 : Side y is less than sum of x & z

C_3 : Side z is less than sum of x & y

C_4 : Side x is equal to side y

C_5 : Side x is equal to side z

C_6 : Side y is equal to side z

The effects are -

e_1 : Not a triangle

e_2 : Scalene triangle

e_3 : Isosceles triangle

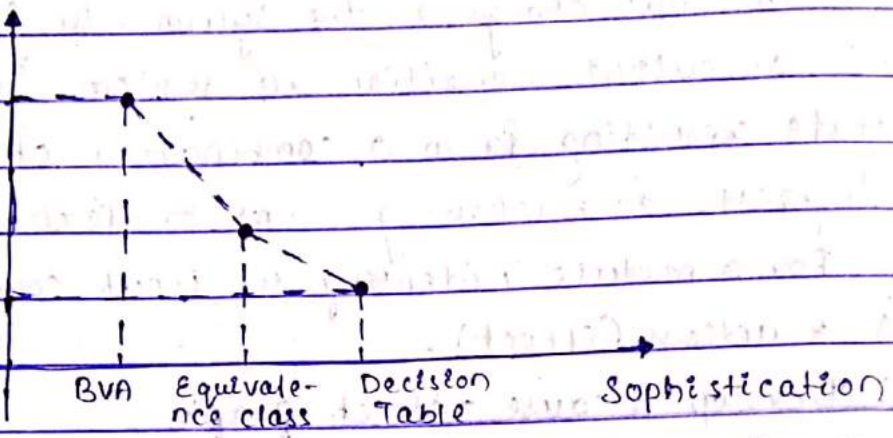
e_4 : Equilateral triangle.

e_5 : Impossible.

→ Step 2 : It's cause-effect graph.

* Comparison on Black Box (or Functional) Testing Techniques :-

1. Testing Efforts →



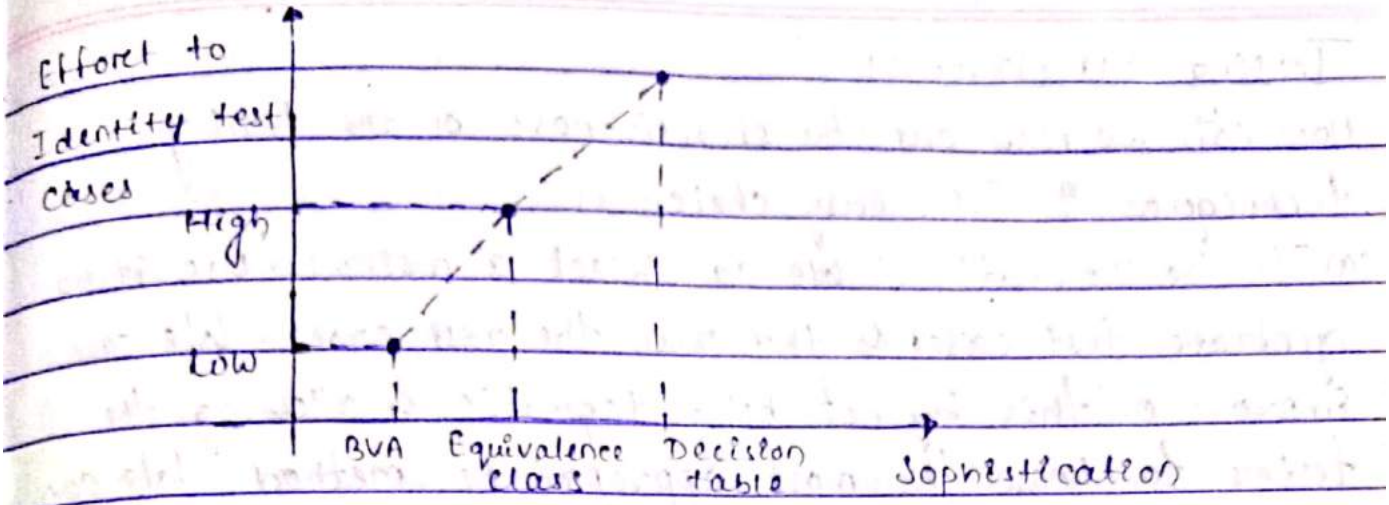
(Test cases as per the Testing Method)

→ The functional methods that we have studied so far, vary both in terms of the number of test cases generated & the effort to develop these test cases.

→ The domain-based techniques have no recognition of data or logical dependencies. They are very mechanical in the way they generate test cases. Because of this they are also easy to automate.

→ The techniques like equivalence class testing focus on data dependecies & thus we need to show our craft. The thinking goes into the identification of the equivalence classes & after that, the process is mechanical.

→ Also note that from the graph, that the decision table based technique is most sophisticated because it requires the testere to considere both data & logical dependencies.



(Test case Identification Effort as per Testing method)

→ Effort required to identify test cases is lowest in BVA & highest in decision tables.

→ The end result is a trade-off between the test case effort identification & test-case execution effort.

→ Also note that, judging testing quality in terms of the sheere number of test cases has drawbacks similare to judging programming productivity in terms of lines of code.

Testing Efficiency :-

→ What we found in all these functional testing strategies is that either the functionality is untested or the test cases are redundant. So, gaps do occure in functional test cases & these gaps are reduced by using more sophisticated techniques.

→ When we see several test cases with the same purposes, sense redundancy, detecting the gaps is quite difficult. If we use only functional testing, the best we can do is to compare the test cases that result from two methods.

→ In general, the more sophisticated method will help us recognize gapi but nothing is guaranted.

Testing Effectiveness :-

How can we find out the effectiveness of the testing techniques ? The easy choice is

(a) To be Dogmatic : We can select a method, use it to generate test cases & then run the test cases. We can improve on this by not being dogmatic & allowing the tester to choose the most appropriate method. We can again another incremental improvement by devising appropriate hybrid methods.

(b) The second choice can be the structural testing techniques for the test effectiveness. This will be discussed in subsequent chapters.

CHAPTER - 4

White Box or Structural Testing Techniques

4.1 INTRODUCTION :

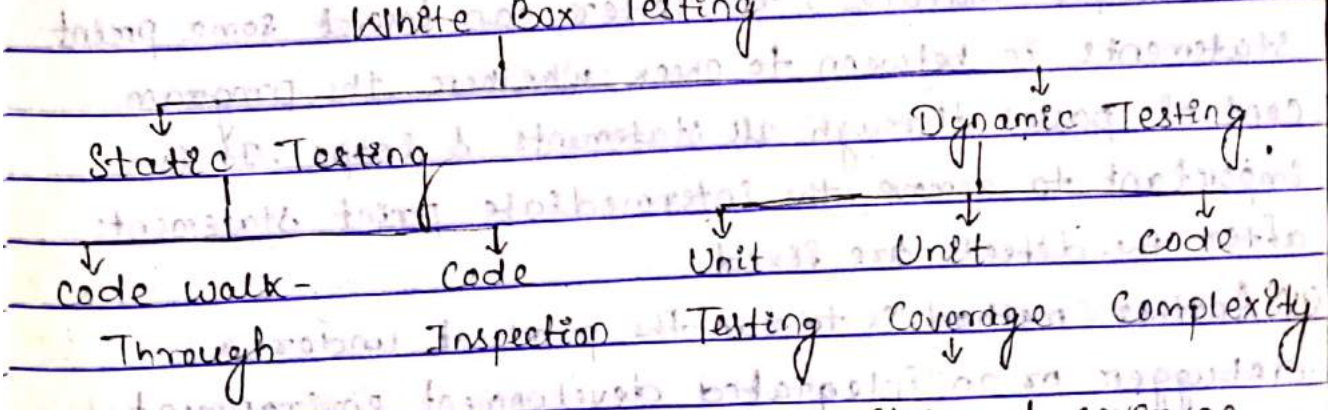
→ White box testing is a way of testing the external functionality of the code by examining & testing the program code that realizes the external functionality.

→ It is a methodology to design the test cases that uses the control structure of the application to design test cases. White box testing is used to test the program code, code structure & the internal design flow.

4.2 Static Versus Dynamic White Box Testing :

→ A number of defects get amplified because of incorrect translation of requirements & design into program code.

- The code works according to the functional requirements.
- The code has been written in accordance with the design developed earliere in the project life cycle.
- The code for any functionality has been missed out.
- The code for any functionality has been missed out.
- The code handles errors properly



→ Static testing can be done by humans or with the help of specialized tools. So, static white box testing is the process of carefully & methodically reviewing the software design, architecture or code for bugs without executing it.

* Dynamic White Box Testing Techniques :-

→ In dynamic testing, we test a running program. So, now binaries & executables are desired.

• Unit / code Functional Testing →

→ It is the process of testing in which the developer performs some quick checks prior to subjecting the code to more extensive code coverage testing or code complexity testing. It can be performed in many ways.

(i) At the initial stages, the developers or testers can perform certain tests based on the input variables & the corresponding expected output variables. This can be a quick test. If we repeat these tests for multiple values of input variables also then the confidence level of the developer to go to the next level increases.

(ii) For complex modules, the tester can insert some print statements in between to check whether the program control passes through all statements & loops. It is important to remove the intermediate print statements after the defects are fixed.

(iii) Another method is to run the product under a debugger or an integrated development environment (IDE). These tools involve single stepping of instructions, setting break points at any function or instruction.

• Code Coverage Testing :-

→ Code coverage testing involves designing & executing test cases & finding out the percentage of code that is covered by testing.

→ The percentage of code covered by a test is found by adopting a technique called as the instrumentation of code.

• Statement Coverage :-

→ Statement coverage refers to writing test cases that execute each of the program statements. We assume that "more the code covered, the better is the testing of the functionality."

→ The statement coverage for a program, which is an indication of the percentage of statements actually executed in a set of tests, can be calculated as follows

$$\text{Statement Coverage} = \left[\frac{\text{Total Statements Exercised}}{\text{Total Number of Executable Statements in Program}} \right] \times 100$$

For ex, if total number of statements exercised = 08

Total no. of executable statements in program = 10

$$\therefore \text{Statement coverage} = \frac{08}{10} \times 100 = 80\%$$

• Path Coverage :-

In path coverage technique, we split a program into into a number of distinct paths. A program or a part of a program can start from the beginning & take any of the paths to its completion. The path coverage of a program may be calculated based on

on the following formula.

$$\text{Path Coverage} = \left[\frac{\text{Total Path Exercised}}{\text{Total Number of paths in program}}\right] \times 100$$

- Condition Coverage :

→ Path testing is not sufficient as it does not exercise each part of the Boolean expressions, Relational expressions & so on. This technique of condition coverage or predicate monitors whether every operand in a complex logical expression has taken on every TRUE / FALSE value.

→ For e.g, in if-then-else, there are 2^2 or 4 possible True / False conditions. The condition coverage which indicates the percentage of conditions which indicates the percentag covered by a set of setest cases, is defined by the formula

$$\text{Condition Coverage} = \left[\frac{\text{Total Decisions Exercised}}{\text{Total Number of Decisions in Program}}\right] \times 100$$

→ This technique of condition coverage is much stronger criteria than path coverage, which in turn is a much stronger criteria than statement coverage.

- Function Coverage :

→ In this white box testing technique we try to identify how many program functions are covered by test cases.

→ The following are the advantages of this technique.

1. Functions (like functions in C) are easier to

identify in a program & hence it is easier to write test cases to provide function coverage.
2. Since functions are at higher level of abstraction than code, it is easier to achieve 100% function coverage.
3. It is easier to prioritize the functions for testing.
4. Function coverage provides a way of testing traceability, i.e tracing requirements through design, coding & testing phases.
5. Function coverage provides a natural transition to black box testing.
→ Function coverage can help in improving the performance as well as the quality of the product.
→ For e.g. if in a networking software, we find that the function that assembles & disassembles the data packets is being used most often, it is appropriate to spend extra effort in improving the quality & performance of that function. Thus, function coverage can help in improving the performance as well as the quality of the product.

* Code Complexity Testing :-
• Cyclomatic complexity, its properties & its meaning in Tabular form :-
(i) Mccabe Metrics based on Cyclomatic Complexity, V(G)
(ii) Execution Coverage Metrics based on any of Branch, path or Boolean coverage.
(iii) Code Grammar Metrics based around line counts & code structure counts such as Nesting.
(iv) OO metrics based on the work of chidamber & Kemerer

(v) Derived Metrics based on abstract concepts such as understability, maintainability, comprehension & testability.

(vi) Custom Metrics imported from third party software / system, eg. defect count.

→ Catagories of Metrics :- There are 3 catagories of metrics

(a) McCabe Metrics

(b) OO Metrics

(c) Grammer Metrics

• Cyclomatic Complexity, its properties & its meaning in tabular form -

$$V(G) = e - n + 2$$

* Mutation Testing Versus Error Seeding - Differences in Tabular form :-

Error Seeding	Mutation Testing
→ No mutants are present here	→ Mutants are developed for testing
→ Here source code is tested within itself	→ Here mutants are combined, compared for testing to find error introduced.
→ Errors are introduced directly	→ Special techniques are used to introduce errors
→ Test cases which detect errors are used for testing	→ Here, case test cases which kill mutants are used for testing
→ It is less efficient error testing technique	→ It is more efficient than error seeding.
→ It requires less time.	→ It is more time consuming

→ It is economical to perform → It is expensive to perform.
→ It is better method for bigger problems. → It is a better method for small size programs

→ This mutation testing scheme was proposed by De Millo in 1978.
→ In this technique, we mutate (change) certain statements in the source code & check if the test code is able to find the errors.
→ It is a technique i.e used to access the quality of the test cases i.e whether they can reveal certain types of faults.
→ A mutant remains live because it is equivalent to the original program i.e, it is functionally identical to the original program or the test data is inadequate to kill the mutant.

$$\text{Adequacy of Test set} = \frac{\text{No. of killed mutants}}{\text{No. of non-equivalent mutants}}$$

* Advantages of Mutation Testing :
1. It can show the ambiguities in code.
2. It leads to more reliable product
3. A comprehensive testing can be done.

* Disadvantages of Mutation Testing :
1. It is difficult to identify & kill equivalent mutants
2. Stubborn mutants are difficult to kill
3. It is time consuming technique, so automated tools are required.
4. Each mutation will have the same size as that of the

of the original program. So a large number number of mutant programs way may need to be tested against the candidate test suite.

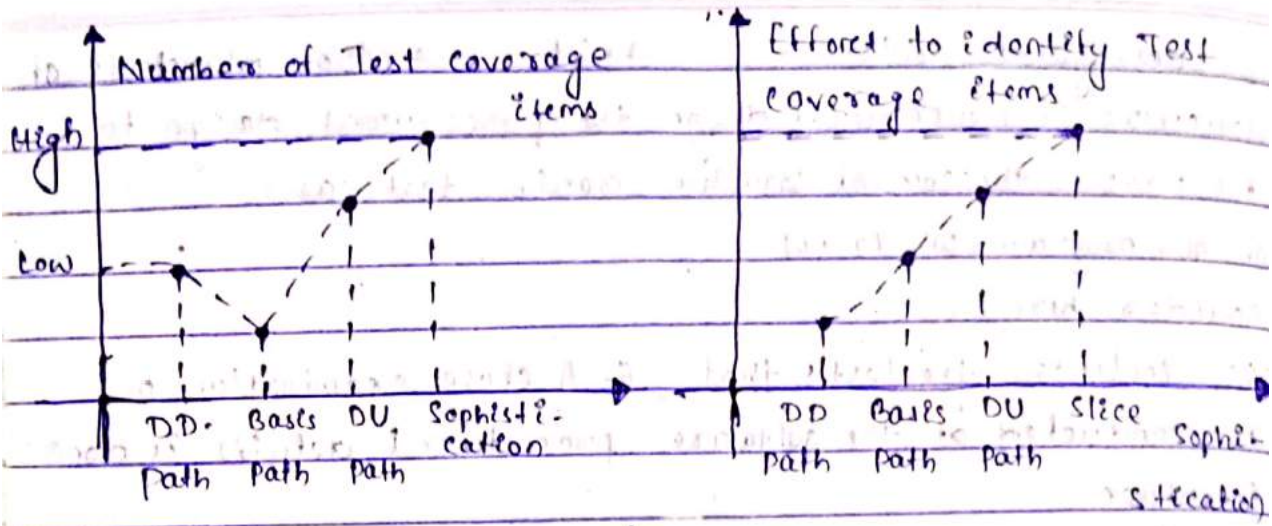
* Comparision of Black Box (BB) & White Box (WB) Testing Techniques :-

Functional or Black Box Testing	Structural or White Box Testing
1. This method focus on functional requirements of the software i.e, it enable the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.	1. This method focusses on procedural details i.e, internal logic of a program
2. It is NOT an alternative approach to white box technique rather is complementary approach i.e likely to uncovere a different class of errors.	2. It concentrates on internal logic, mainly.
3. Black box testing is applied during later stages of testing.	3. Where as, white box testing is performed early in the testing process.
4. It attemps to find errors in following catagories - (a) Incorrect or missing functions (b) Interface errors (c) Errors in data structures or external database access (d) Performance errors.	4. Where as, white box testing performed in following cases - (a) Internal logic of your program. (b) Status of program.

5. It desregards control structure of procedural design (i.e what is the control structure of our program, we do not consider here).	5. It uses control structure of the procedural design to derive test cases.
6. It includes the tests that are conducted at the software interface.	6. A close examination of procedural details is done.
7. Are used to uncovere errors.	7. Logical paths through the software are tested by providing test cases, that exercise specific sets of conditions or loops.
8. To demonstrate that software functions are operational i.e input is properly accepted & output is correctly produced. Also, the integrity of external information (e.g a data base) is maintained.	8. A limited set of logical paths be however examined.

* Comparision on Various White Box Testing Techniques :-

→ Functional testing techniques always result in a set of test cases & structural metrics is always expressed in terms of something countable like the number of program paths, the number of decision-to-Decision ~~DD~~ (DD-paths) & so on.



(Shows Trends of Test Coverage Item(s)) (Shows Trend of Test Method Effort)

→ Figure show the trends for the number of test coverage items & the effort to identify them as functions of structural testing methods, respectively.

* Advantages of White Box Testing :-

→ White box testing helps us to identify memory leaks. When we allocate memory using malloc() in C, we should explicitly release that memory also. If this is not done then over time, there would be no memory available for allocating memory on requests. This can be done using debuggers also that can tally allocated & freed memory.

→ ~~Performance~~ Performance :-

→ Performance Analysis :-

Code coverage tools can identify the areas of a code that are executed most frequently. Extra efforts can be then made to check these sections of code. To do further performance improvement techniques like caching, coprocessing or even parallel processing

can be considered.

→ Coverage tests with instrumented code is one of the best means of identifying any violations of such concurrency constraints through critical sections.

→ White box testing is useful in identifying bottlenecks in resource usage.

→ White box testing can help identify security holes in dynamically generated code.

CHAPTER - 5

GRAY BOX TESTING

5.0 Introduction To Gray Box Testing :-

→ Code coverage testing involves "dynamic testing" methods of executing the product with pre-written test cases & finding out how much of code has been covered.

→ Understanding of code & logic means white box or structural testing where as writing effective test cases means black box testing. So, we need a combination of white box & black box techniques for test effectiveness. This type of testing is known as "gray box testing".

WHITE + BLACK = GRAY

5.1 What is Gray Box Testing :-

→ Black box testing focuses on software's external attributes & behaviour. Such testing looks at an application's expected behaviour from the user's point of view.

→ White box testing / glass-box testing, however, tests software with knowledge of internal data structure, physical logic flow, & architecture at the source code level.

→ White box testing looks at testing from the developer's point of view.

→ Both black-box & white-box testing are critically important complements of a complete testing effort.

* Definitions of Gray Box Testing :-

→ Gray box testing incorporates the elements of both black box & white box testing.

➤ "Gray box testing consists of methods & tools derived from the knowledge of the application internals & the environment with which it interacts, that can be applied in black box testing to enhance testing productivity, bug finding & bug analyzing efficiency"

Or

➤ "Gray box testing is using inferred or incomplete structural or design information to expand or focus black box testing."

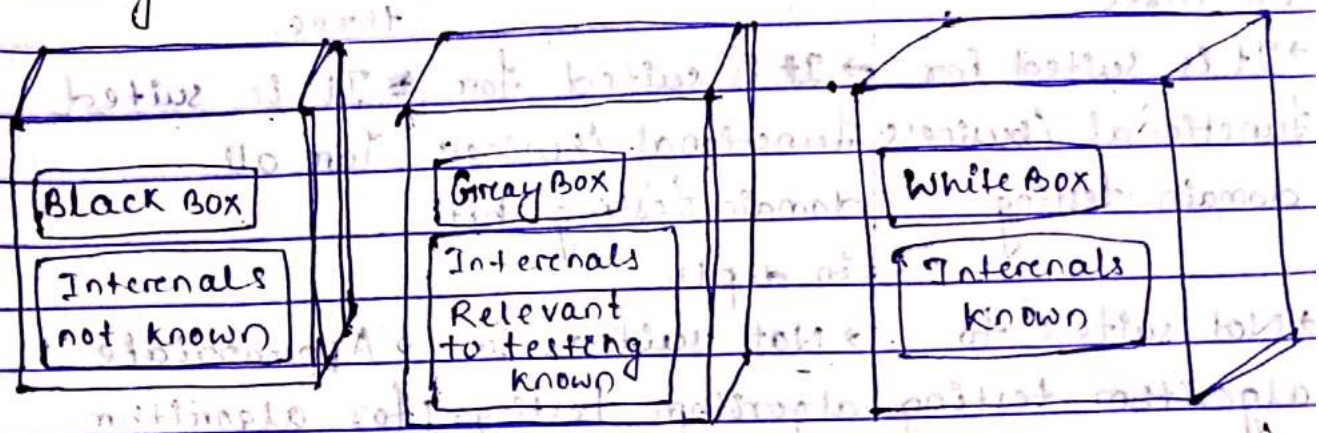
Or

➤ "Gray box testing is designing of the test cases based on the knowledge of algorithms, internal states, architectures or other high level descriptions of program behaviour."

Or

➤ "Gray box testing involves inputs & outputs, but test design is educated by information about the code or the program operation of a kind that would normally be out of scope of view of the tester."

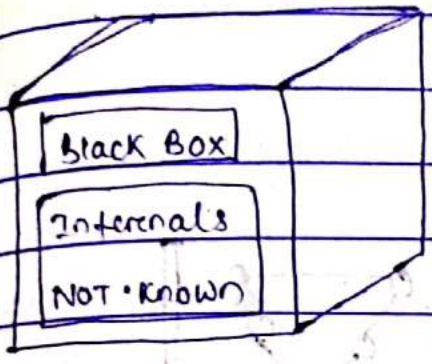
* Comparison of White Box, Black Box & Gray Box Testing :



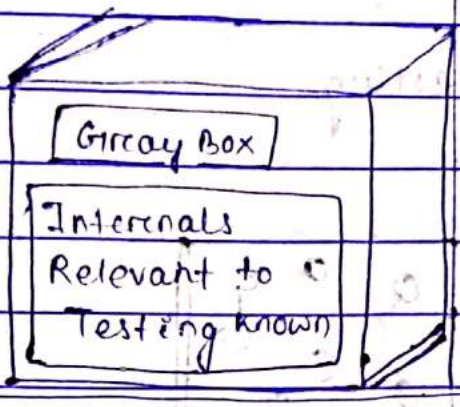
(Comparison of ~~BB~~ BB, GB & WB Techniques)

Black Box Testing	Gray Box Testing	White Box Testing
→ Low granularity.	→ Medium granularity.	→ High granularity.
→ Internals NOT known.	→ Internals partially known	→ Internals fully known
→ Internals not required to be known	→ Internals relevant to testing are known.	→ Internal code of the application & database known
→ Also known as • Opaque box testing • closed box testing • Input output testing • Data driven testing • Functional testing • Behavioural.	→ Also known as translucent box testing.	→ Also known as • Glass box Testing • Clear box testing • Design based testing • Logic based testing • Structural Testing • Code based Testing.
→ It is done by end-users (User acceptance testing). Also done by testers, developers.	→ It is done by end-users (User acceptance testing. Also done by testers, developers.	→ Normally done by testers & developers.
→ It is likely to be least exhaustive of the three	→ It is somewhere in between.	→ Potentially most exhaustive of the three.
→ It is suited for functional /business domain testing	→ It is suited for functional /business domain testing bit in depth	→ It is suited for all.
→ Not suited to algorithm testing	→ Not suited to algorithm testing.	→ Appropriate for algorithm testing

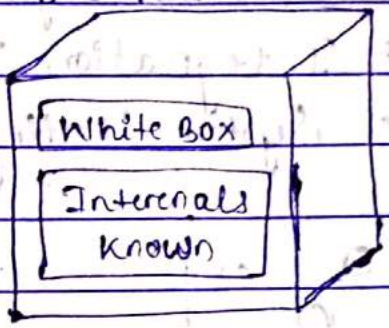
→ It can test only by trial & error data domains, internal boundaries & overflow



→ It can test data domains, internal boundaries & overflow, if known.



→ It can determine & therefore test better; data domains, internal boundaries & overflow.



CHAPTER - 7

LEVELS OF TESTING

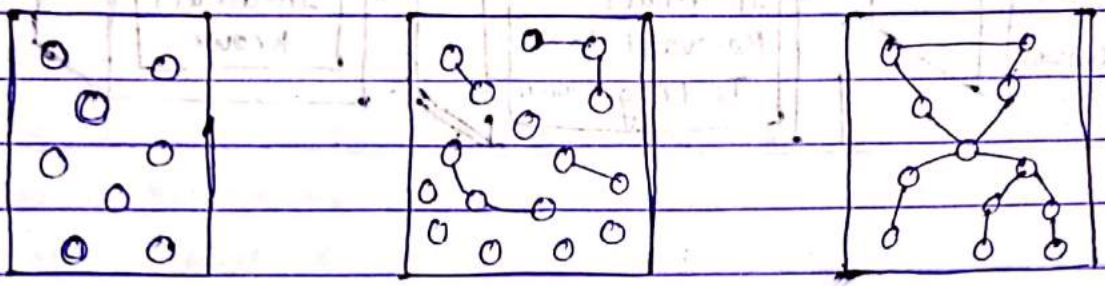
7.01 Introduction -

When we talk of levels of testing then we are actually talking of 3 levels of testing.

(a) Unit Testing

(b) Integration Testing

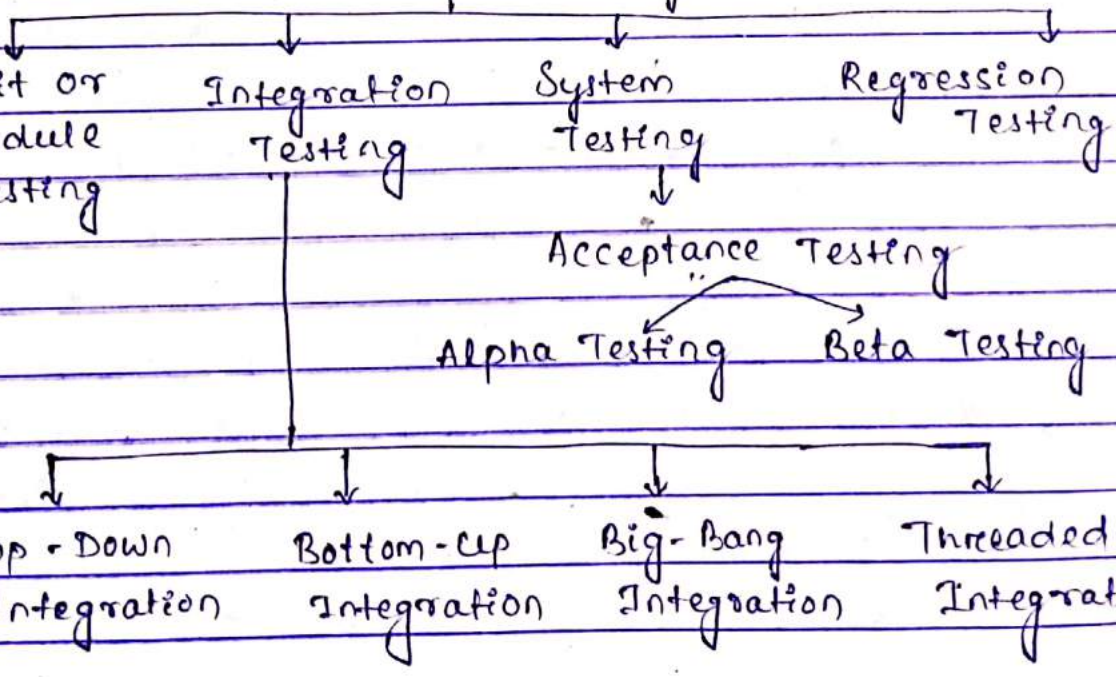
(c) System Testing.



Unit Testing Integration Testing System Testing

→ Generally, system testing is functional rather than structural testing

* Unit, Integration, System & Acceptance Testing :-



→ Unit testing "is a the process of taking a module (an atomic unit) & running it in isolation from the rest of the software product by using prepared test cases & comparing the actual results with the results predicted by the specification & design module." It is a white box testing technique.

Importance of Unit testing :-

1. Since modules are being testing tested individually so testing becomes easier.
2. It is more exhaustive.
3. Interface errors are eliminated.

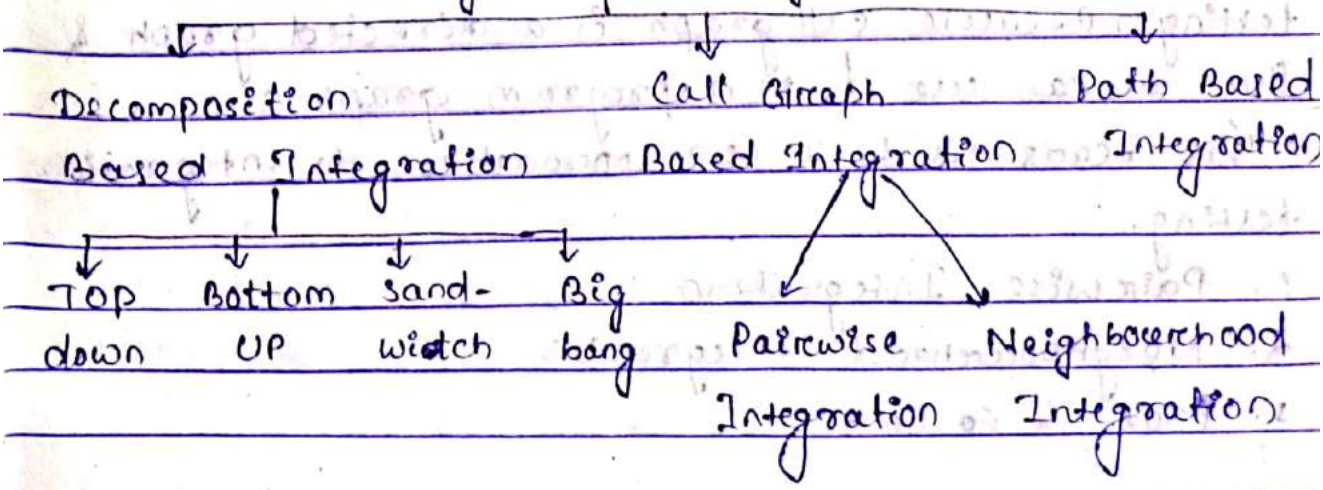
7.3 Integration Testing :

→ Integration is defined as the set of interactions among components.

→ "Testing is the interaction between the modules & interaction with other systems externally is called as integration testing."

→ It is both a type of testing & a phase of Testing.

* Classification of Integration Testing :



* Decomposition - Based Integration :-

→ When we talk of decomposition based integration testing techniques, we usually talk of the functional decomposition of the system to be tested which is represented as a tree or in textual form.

→ The goal of decomposition-based integration is to test the units that have been separately tested interfaces among separately tested units.

→ Types of Decomposition Based Techniques are :-

- Top - down Integration Approach
- Bottom - Up Integration Approach
- Sandwitch Integration Approach
- Big Bang strategy
- Pros & cons of Decomposition - Based Integration

* Call Graph Based Integration :-

→ One of the drawbacks of decomposition-based integration is that the basis is the functional decomposition tree.

→ But if we use call graph based technique instead, we can remove this problem.

→ Also, we will move in the direction of structural testing, Because call graph is a directed graph & thus we can use it as a program graph also.

→ This leads us to 2 new approaches to integration testing.

1. Pairwise Integration
2. Neighborhood Integration
3. Pros & Cons

* Path - Based Integration with its Pros & cons :-

→ Our goal for integration testing is - "Instead of testing interfaces among separately developed & tested units, we focus on interactions among these units." Here, Cofunctioning might be a good term. Interfaces are structural whereas interaction is behavioural.

→ Some basic terminologies that are used in this technique :-

1. Statement Fragment :- It is a complete statement.
2. Source Node :- A source node is a program a statement fragment at which program execution beings or resumes.
3. Sink Node :- It is a statement fragment in a unit at which program execution terminates.
4. Module Execution Path (MEP) :- It is a sequence of statements that begins with a source node & ends with a sink node, with no intervening sink nodes.
5. Message :- We follow the convention that the unit that receives a message, always eventually returns control to the message source.
6. Module - to - Module Path (MM - Path) :- An MM - path is an interleaved sequence of module execution path (MEP) & message
7. Module - to - Module Path Graph (MM - path Graph).

* System Testing :

→ System testing focuses on a complete, integrated system to evaluate compliance with specified requirements.

→ Tests are made on characteristics that are only present when the entire system is run.

✓ What is System Testing ?

→ The testing i.e conducted on the complete integrated products & solutions to evaluate system compliance with specified requirements on functional & non-functional aspects is called as System testing.

→ A System is defined as a set of hardware, software & other parts that together provide product features & solutions.

System Testing = Functional Testing + Non-Functional Testing

Functional Testing	Non-functional Testing
→ It involves the product's functionality	→ It involves the product's quality factors
→ Failure here occur due to code.	→ Failures occur due to either architecture, design or due to code.
→ It is done during unit, component integration & system testing phase.	→ It is done in our system testing phase
→ Configuration remains same for a test suite.	→ Test configuration is different for each test suite.

Integration Testing	System Testing
→ The test cases are created by looking at interfaces.	→ The test cases are created first & verified with design & architecture.
→ The integration test cases focus on interactions between modules or components.	→ The functional system test focuses on the behaviour of the complete product.
→ It starts from the interface specification.	→ It starts from the requirements specification.
→ It requireds some scaffolding	→ It does not require any scaffolding.
→ It checks the integration among modules.	→ It checks the system functionality
→ It tests the visibility of the integration structure.	→ It does not test the visibility of code

Unit Testing	Integration Testing.
→ It starts from the module specification.	→ It starts from the interface specification.
→ It requires complex scaffolding.	→ It required some scaffolding.
→ It checks the behaviour of single module.	→ It checks about the integration among module.

* Acceptance Testing :-

→ It is a phase after system testing i.e done by the customers.

→ These test cases are executed by the customers are normally small in number.

* Acceptance test cases are developed by both customers & product organization. Acceptance test cases are black-box type of tests cases.

→ A user acceptance test is -

- A chance to complete test software
- A chance to completely test business processes.
- A condensed version of a system
- A comparision of actual test results against expected results.
- A discussion forum to evaluate the process

The main objectives are as follows -

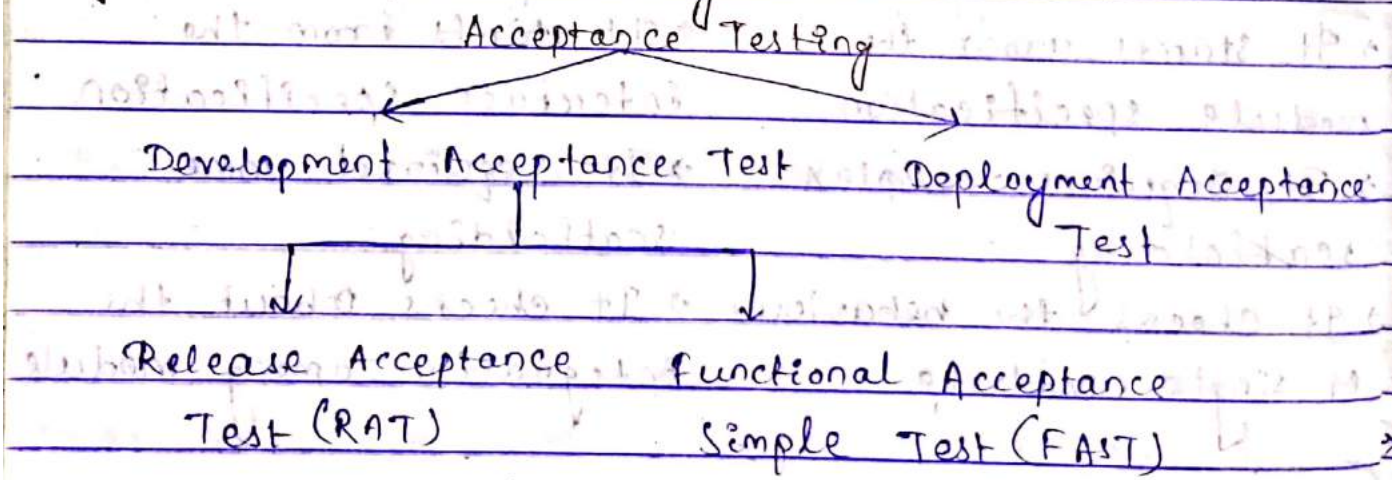
→ Validate system set-up for transactions & user access.

→ Confirm use of system in performing business process.

→ Verify performance on business critical functions.

→ Confirm integrity of converted & additional data

Types of Acceptance Testing :



CHAPTER - 6

REDUCING THE NUMBER OF TEST CASES

* Prioritization Guidlines :-

→ The goal of prioritization of test cases is to reduce the set of test cases based on some rational, non-arbitary, criteria, while aiming to select the most appropriate tests.

→ If we prioritize the test cases, we run with a risk that some of the application features will not be tested at all.

→ The prioritization schemes basically address these key concept.

(a) What features must be tested ?

(b) What are the consequences if some features are not tested ?

6.1 Priority Category Scheme :-

The most simplest scheme for categorizing tests is to assign a priority code directly to each test description. One possible scheme is a three-level priority categorization scheme. It is given below.

Priority 1 - This test must be executed

Priority 2 - If time permits, execute this test.

Priority 3 - If this test is not executed, the team won't be upset.

→ So, assigning priority code is as simple as written writing a number adjacent to each test description

→ Once priority codes have been assigned, the tester estimates the amount of time required to execute the tests selected in each category.

6.3 Risk Analysis :-

→ All software projects benefits from risk analysis. Even non-critical software, using risk analysis at the beginning of a project highlights the potential problem areas.

→ This helps developers & managers to mitigate the risks. The tester uses the results of risk analysis to select the most crucial tests.

How Risk Analysis is Done ?

Risk analysis is a well-defined process that prioritizes module for testing. A risk contains 3 components -

(a) The risk, r_i, associated with a project ($i \leftarrow 1$ to)

(b) The probability of occurrence of a risk, (I_i)

(c) The impact of the risk, (x_i)

Problem ID	Potential Problem (r_i)	Probability of occurrence (I_i)	Impact of risk (x_i)	Risk Exposure = $I_i * x$
A	Loss of Power	1	10	10
B	Corrupt file header	2	1	2
C	Unauthorized access	6	8	48
D	Databases not synchonized	3	5	15
E	Unclear user documentation	9	1	9
F	Lost sales	1	8	8
G	Slow throughput	5	3	15
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮

Risk Analysis Table (RAT)

6.4 Regression Testing - Overview :-

→ Regression testing is done to ensure that enhancements or defect fixes made to the software works properly & does not affect the existing functionality.
→ It is usually done during maintenance phase.
→ Regression testing is a testing process i.e used to determine if a modified program still meets its specifications or if new errors have been introduced.

Normal Testing	Regression Testing
→ It is usally done during fourth phase of SDLC.	→ It is done during maintainance phase
→ It is basically software's verification & validation.	→ It is also called as program revalidation
→ New test suites are used to test our code.	→ Both old & new test cases can be used for testing.
→ It is done on the original software.	→ It is done on modified software.
→ It is cheapere.	→ It is a costlier testing.

Types of Regression Testing :-

→ 4 types of regression testing techniques are discussed one-by-one. They are
(a) Corrective Regraession Testing
(b) Progressive Regression Testing
(c) Retest all Regression Testing
(d) Selective Regression Testing

a. Corrective Regression Testing - It applies when specification are unmodified & test cases can be reused

b. Progressive Regression Testing - It applies when specifications are modified & new test cases must be designed.

c. The Retest - All strategy - The retest - all strategy reuses all tests, but this strategy may waste time & resources due to execution of unnecessary tests.

6.5 Prioritization of Test Cases for Regression Testing :

→ Prioritization of tests case requires a suitable a cost criterion. Please understand that tests with with lower costs are placed at the top while those with higher cost at the bottom.

What cost criterion to use ?

→ We could use multiple criteria to prioritize tests.

→ "The tests being prioritized are the ones selected using some test selection technique."

→ Prioritization of regression tests offers a tester an opportunity to decide how many & which tests to run under time constraints. When all tests cannot be run, one needs to find some criteria to decide when to stop testing. This decision could depend on variety of factors like -

(a) Time Constraint

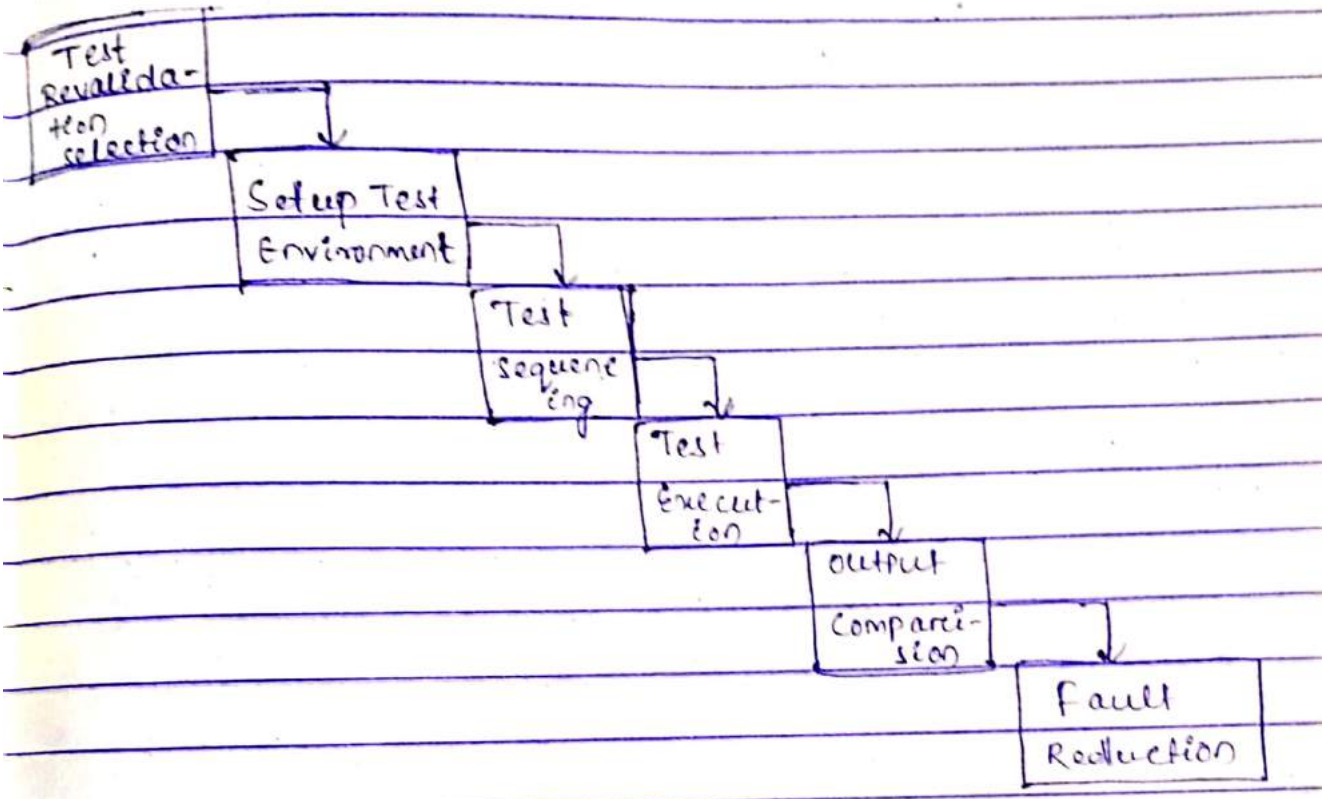
(b) Test Criticality

(c) Customer requirements.

* Regression Testing Technique - :

→ Regression testing is used to confirm that fixed bugs have been fixed & that new bugs have not been introduced.

→ How many cycles of regression testing are required will depend upon the project size.
→ Cycles of regression testing may be performed once per build. Regression tests can be automated.



→ Test minimization ignores redundant tests. For eg if both t_1 & t_2, test function, F in P_t, then one might decide to reject t_2 in favour of t_1. The purpose of minimization is to reduce the number of tests to execute for regression testing.
→ Test prioritization means prioritizing tests based on some criteria.
→ Test setup means the process by which AUT (Application Under Test) is placed in its intended or simulated environment & is ready to receive data & output the required information
→ The sequence in which tests are input to an application is an important issue.

CHAPTER - 8

AUTOMATED TESTING

8.1 Automated Testing :

→ Automated testing is automating the manual testing process. It is used to replace (or supplement manual testing with a suite of testing tools.

→ Manual testing is used to document tests, produce test guides based on data queries, provide temporary structures to help run tests & measure the result of the test.

8.2 Consideration During Automated Testing :

While performing testing with automated tools, the following points should be noted :

1. Clear & reasonable expectations should be established in order to know what can & what can not be accomplished with automated testing in the organization.
2. There should be clear understanding of the requirements that should be met in order to achieve successful automated testing. This requires that the technical personnel should use the tools effectively.
3. The organization should have detailed, reusable test cases which contain exact expected results & a stand alone test environment with a restorable database.
4. Testing tool should be cost - effective. The tool must ensure that test cases developed for manual testing are also useful for automated testing.

5. Select a tool that allows the implementation of automated testing in a way that conforms to the specified long-term testing strategy.

8.3 Types of Testing Tools - Static V/S Dynamic Testing is of 2 types - (a) Static Testing
(b) Dynamic Testing
And also that the tools used during these testing are accordingly named as (a) Static testing tools
(b) Dynamic testing tools.

→ Static testing tools seek to support the static testing process whereas dynamic testing tools support dynamic testing process.

→ Static testing is different from dynamic testing.

Static Testing	Dynamic Testing
→ Static testing does not require the actual execution of software.	→ Dynamic testing involves testing the software by actually executing it
→ It is more cost effective.	→ It is less cost effective
→ It may achieve 100% statement coverage in relatively short time.	→ It achieves less than 50% statements coverage because it finds bugs only in part of codes that are actually executed.
→ It usually takes shorter time.	→ It may involve running several test cases, each of which may take longer than compilation

compilation

→ It may uncover variety of bugs.

→ It uncovers limited types of bugs that are explorable through execution.

→ It can be done before compilation.

→ It can take place only after executables are ready.

A. Static Test Tools :-

(a) Flow analyzers → They ensure consistency in data flow from input to output.

(b) Path tests → They find unused code & code with contradictions.

(c) Coverage analyzers → It ensures that all logic paths are tested

(d) Interface analyzers → It examines the effects of passing variables & data bet^n modules.

B. Dynamic Test Tools :-

(a) Test drivers → It inputs data into a module - under - test (MUT)

(b) Test beds → It inputs data into & It simultaneously displays source code along with the program under execution.

(c) Emulators → The response facilities are used to emulate parts of the system not yet developed.

(d) Mutation analyzers → The errors are deliberately 'fed' into the code in order to test fault tolerance of the system

8.4 Problems with Manual Testing :-

The main problems with manual testing are listed be

a. Not Reliable -

→ Manual testing is not reliable as there is no yardstick available to find out whether the actual & expected results have been compared.

b. High Risk -

→ A manual testing process is subject to high risks of oversights & mistakes. People get tired, they may be temporarily in attentive, they may have too much tasks on hand, they may be insufficiently trained & so on.

c. Incomplete Coverage -

Testing is quite complex when we have mix of multiple platforms, O.S. Servers, clients, channels, business processess etc. Testing is non-exhaustive. Testing is non Full manual regression testing is impractical.

4. Time Consuming -

Limited test resources makes manual testing simply too time consuming. As per a study done, 90% of all IT projects are delivered late due to manual testin

5. Fact & Fiction -

The fiction is that manual testing is done while the fact is only some manual testing is done depending upon the feasibility.

* "Manual testing is used to document tests", produce test guides based on data queries, provide temporary structures to help run tests & measure

the results of the test".

8.5 Benefits of Automated Testing :-

→ Automated testing is the process of automating the manual testing process.

→ We shall now list some more benefits of test automation. They are given below.

1. Automated execution of test cases is faster than manual execution. This saves time. This time can also be utilized to develop additional test cases, thereby improving the coverage of testing.

2. Test automation can free test engineers from mundance tasks & make them focus on more creative tasks.

3. Automated tests can be more reliable. This is because manually running the tests may result in boredom & fatigueness, more chances of human error.

4. Automation helps in immediate testing as it need not wait for the availability of test engineers.
In fact, Automation = Lesser Person Dependence.

5. Test cases for certain types of testing such as reliability testing, stress testing, load & performance testing can not be executed without automation.

6. Manual testing requires the presence of test engineers but automated tests can be run round the clock (24 x 7) environment. So, automated testing provides round the clock coverage.

7. Tests, once automated, take comparatively less resources to execute.

8. Automation produces a repository of different tests which helps us to train test engineers to increase their knowledge.
9. Automation does not end with developing programs for the test cases.

8.6 Disadvantages of Automated Testing :
→ An average automated test suite development is normally 3-5 times the cost of a complete manual test cycle.
→ Automation is too cumbersome. Who would automate? Who would train? Who would maintain? This complicates the matter.
→ In many organizations, test automation is not even a discussion issue.
→ There are some organizations where there is practically no awareness or only some awareness on test automation.
→ Automation is not an item of highere priority for managements. It does not make much difference to many organisations.
→ Automation would require additional trained staff. There is no staff for the purpose.

8.7 Skills Needed for Using Automated Tools :
1. Capture / playback & test harness tools (first generation).
2. Data driven tools (Second generation)
3. Action driven (third generation)

Generation)

1. Capture / Playback & Test Harness Tools (1st Generation)

→ One of the most boring & time-consuming activity during testing life cycle is to retrun manual tests number of times.

→ Here, capture / playback tools are of great help to the testers.

→ A capture / playback tool can be either intrusive or non-intrusive.

→ Test harness tools are the category of capture / playback tools used for capturing & replaying sequence of tests.

2. Data-driven Tools :

→ This method helps in developing test scripts that generates the set of input conditions & corresponding expected output.

→ The approach takes as much time & effort as the product

3. Action-driven Tools :

→ This technique enables a layman to create automated testes.

8.8 Test Automation :- "No Silver Bullet"

→ Test automation is a partial solution & not a complete solution.

→ One does not go in for automation because it is easy. It is painful & resource-consuming exercise but once it is done, it has numerous benefits.

→ Test automation is "no silver bullet". It is only one of the many factors that determine S/W quality.

8.9 Debugging :-

→ "Debugging is an activity of locating & correcting errors".

→ Debugging is not testing but it always occurs as a consequence of testing.

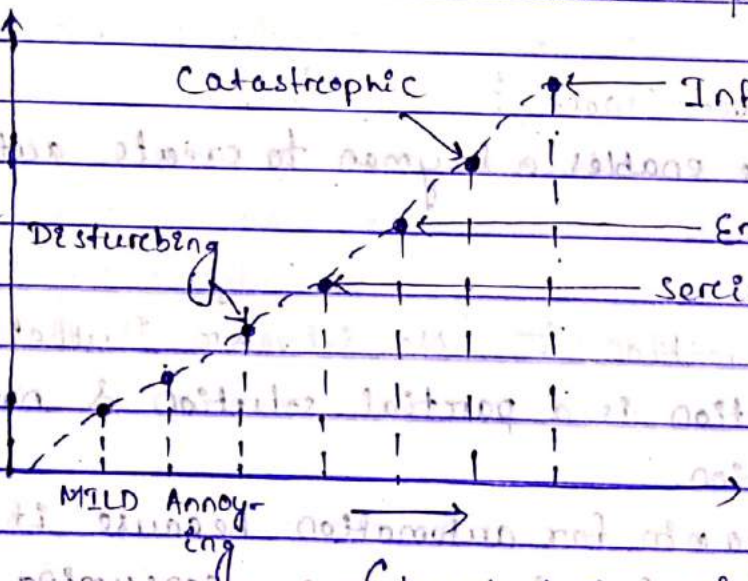
→ The debugging process begins with the execution of a test case. The debugging process will always have one of the 2 outcomes.

(1) The cause will be found, corrected & removed

(2) The cause will not be found.

* The debugging Process :-

→ During debugging, errors are encountered that range from less damaging (like input of an incorrect function) to catastrophic (like system failure, which leads to economic or physical damage)



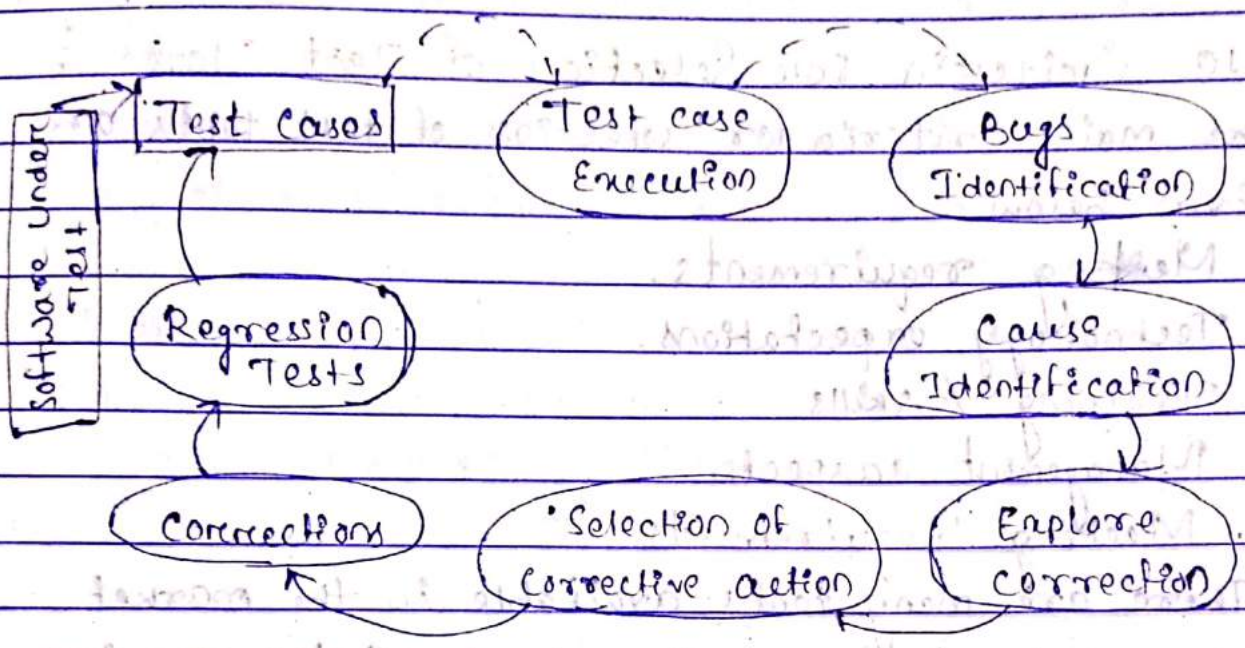
(Level of Error & its Effect)

Once errors are identified in a software system, to debug the problem, a number of steps are followed

Step-1 - Identify the errors

Step-2 - Design error report.

Step 3 - Analyze the errors
Step 4 - Debugging tools are used
Step 5 - Fix the errors
Step 6 - Reset the Retest the software.



(Overall Debugging Life cycle)

Software Testing	Debugging
→ It is the process of executing the program with the intent of finding faults/errors.	→ It is an activity of locating & correcting errors.
→ It is a phase of software development life cycle (SDLC).	→ It occurs as a consequence of testing.
→ Once code is written, testing commences.	→ The debugging process begins with the execution of a test case.
→ It is further comprised of validation & verification of software.	→ It attempts to match symptom with cause, thereby leading to error correction.
→ It uses unit, integration & System level testing.	→ It checks the correctness & performance of the software.

* Debugging Approaches :-

1. Brute Force Method
2. Back Tracking Method
3. Cause Elimination
4.

8.10 Criteria For Selection of Test Tools :-

The main criteria for selection of test tools are given below -

1. Meeting requirements.
2. Technology expectations.
3. Training / skills
4. Management aspects.

a. Meeting Requirements :-

→ There are many tools available in the market today but rarely do they meet all the requirements of given product or a given organization.

→ Test tools may not provide backward or forward compatibility with the product-under-test (PUT).

→ A number of test tools cannot distinguish bet' a product failure & a test failure. This increases analysis time & manual testing.

→ The test tools may not provide the required amount of trouble-shooting / debug / error messages to help in analysis.

b. Technology Expectations :-

→ In general, test tools may not allow test developers to extend / modify the functionality of the framework. So, it involves going back to the tool vendor with additional cost & effort.

→ A good number of test tools require their libraries to be linked with product binaries. When these libraries are linked with the source code of the product, it is called as the "instrumented code"
→ Finally, test tools are not 100% cross-platform. They are supported only on some O.S. platforms. & the scripts generated from these tools may not be compatible on other platform.

C. Training Skills :-
→ Test tools require plenty of training, but very few vendors provide the training to the required level.

D. Management Aspects :-
→ A test tool increases the system requirement & requires the hardware & software to be upgraded.
→ This increases the cost of the already-expensive test tools.
→ Deploying a test tool requires as much efforts as deploying a product in a company.

8.11 Steps For Tool Selection :-
There are 7 steps to select & deploy a test tool in an organization. These steps are:
Step-1 : Identify your test suite requirements & list among the generic requirements discussed. Add other requirements, if any.
Step-2 : Make sure experiences discussed in previous sections are taken care of.

Step - 3 : Collect the experiences of other organizations which used similar test tools.
Step - 4 : Keep a checklist of questions to be asked to the vendors on cost / effort / support.
Step - 5 : Identify list of tools that meet the above requirements & give priority for the tool which is available with the source code.
Step - 6 : Evaluate & shortlist one / set of tools & train all test developers on the tool.
Step - 7 : Deploy the tool across test teams after training all potential users of the tool.