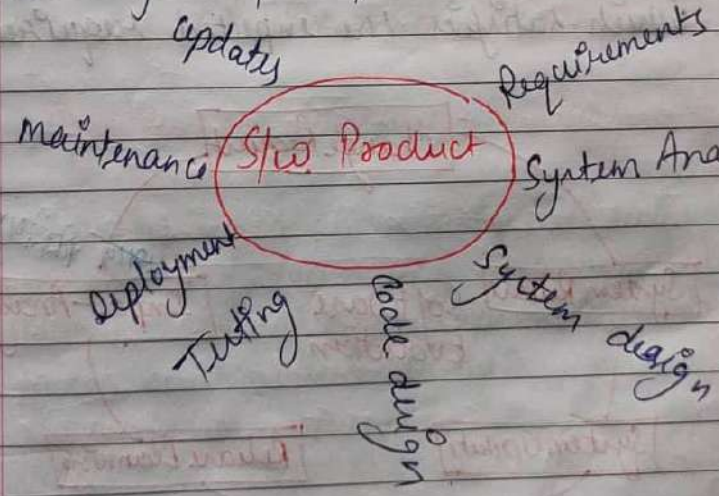
SOFTWARE TESTING & AUDITING

9/Aug/19

Overview of Software Evolution →

S/w is more than just a program code. A program is an executable code which serves some computational purpose. S/w is considered to be collection of executable programming code, associated libraries & documentations. Software when made for a specific requirement is called Software product.

Engineering on the other hand, is all about developing products, using well-defined scientific principles & methods.



IEEE defines S/w Engg. as:

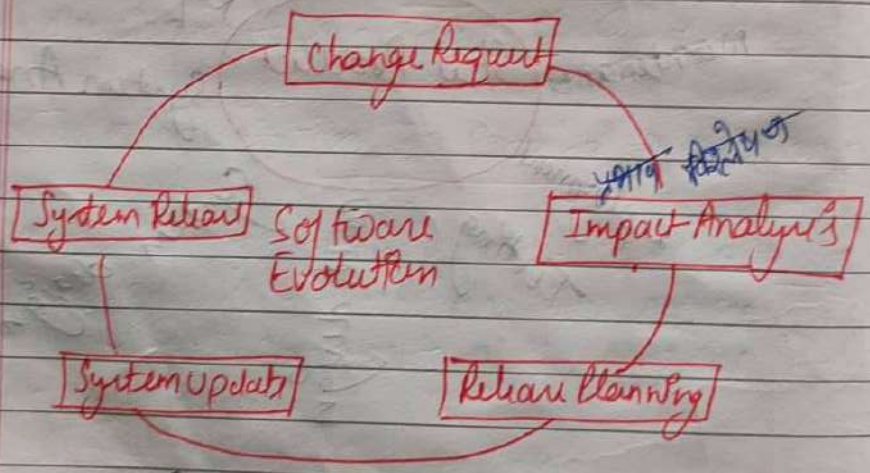
1) The appln of a systematic, disciplined quantifiable approach to the development,

operation and maintenance of S/W: that is the appl^n of engg to software.

2) The study of approaches as in the above statement.

Software Evolution-

The process of developing a software product using S/W engineering principles and methods is referred to as Software evolution. This includes the initial development of S/W and its maintenance and updates, till desired S/W product is developed which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended S/W and show it to the users to get their

Waterfall SDLC → A life cycle model prescribes the different activities that need to be carried out to develop a s/w product and the sequencing of these activities.

feedback at the early stage of s/w product development. The user suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original s/w, till the desired s/w is accomplished.

10/Aug/19

Software Development Life Cycle (SDLC) A s/w life cycle model is a descriptive and diagrammatic representation of the s/w life cycle.

Requirement gathering & Analysis
Design
Implementation or coding
Testing
Deployment
Maintenance

→ Requirement Gathering & Analysis →

During this phase, all the relevant info^n is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Eg. A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood SRS document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

2) Design → In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding → It starts once the developer gets the design document. The software design is translated into source code. All the components of the s/w are implemented in this phase.

4) Testing → Testing starts once the coding is complete and the modules are released for testing. In this phase, the

developed s/w is tested thoroughly and any defects found are assigned to developers to get them fixed.

5) Deployment → Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the appl^n as expected then sign off is provided by the customer to go live.

6) Maintenance → After the deployment of a product on the production environment maintenance of the product i.e if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

Software Evolution →

Exploratory Programming Style → Early commercial computers were very slow and too elementary as compared to today's standards. Even simple processing task took considerable computation time on those computers. That programs at that time were very small in size and lack sophistication. These programs were usually written in assembly language. Programmers wrote programs without formulating any proper solution strategy, plan or design and jump to the terminal and start coding immediately on having out the problem. This is called exploratory Programming style.

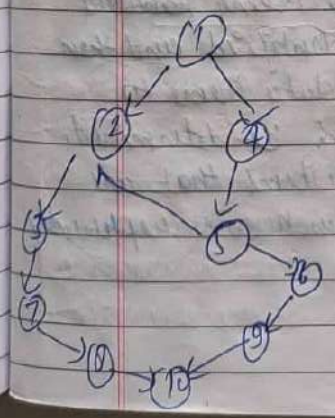
High Level Languge Programming → Introduce of the semiconductor technology. This removes the vacuum tube-based circuits in a computer. It become possible to solve larger and more complex problems. High languages such as FORTRAN, ALGOL and COBOL. Programmers were still using the exploratory style of software development.

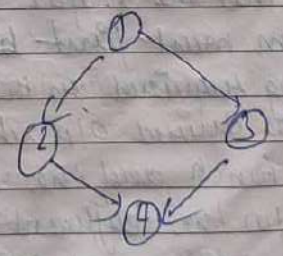
Control Flow Based design → A program's control flow structure indicates the sequence in which the programs instructions are executed. In this flow charting technique was developed

```
1. if(customer_saving_balance > withdrawal_req) {
2.     100: issue_money = TRUE;
3.         GOTO 110;
   }
4. else if (privileged_customer == TRUE)
5.         GOTO 100;
6. else GOTO 120.
7. 110: activate_cash_dispenser(withdrawal_request);
8.         GOTO 130;
9. 120: print(error);
10. 130; End_transaction();
```

or

```
1. if(privileged_customer || (customer_saving_balance >
        withdrawal_req)) {
2.     activate_cash_dispenser(withdrawal_req);
   }
3. else print(error)
4. end_transaction
```





A program is called structured when it uses only the sequence, selection and iteration types of constructs and is modular.

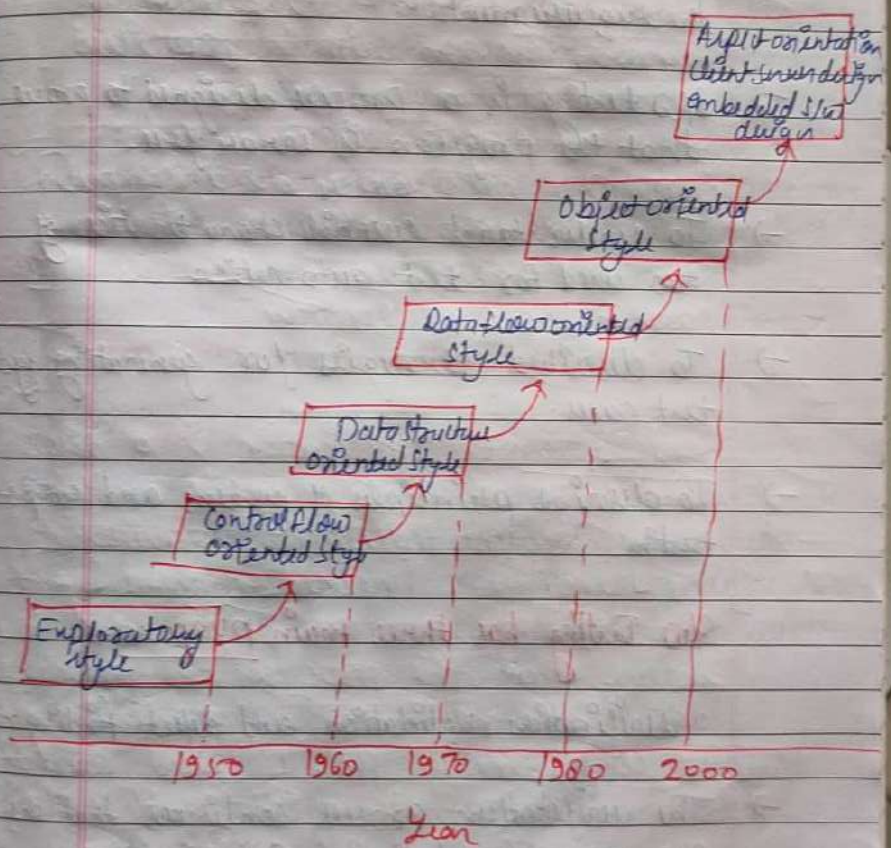
Data structure-oriented Design → Semiconductors replaced by the integrated circuits. S/w engineers were tasked to develop larger and more complicated s/w products which often required writing in excess of several tens of thousands of lines of source code. Design of the important data structure of the program than to the design of its control structure. This technique is called data structure oriented design technique.

Data-flow oriented Design → Introduce to the Very large Scale Integrated circuits and some new arch concepts, more complex and sophisticated s/w products were needed to solve further challenging problems.

The data flow oriented technique advocate that the major data items handled by a system must first be identified and then the processing required on these data items to produce the desired o/p should be determined.

The flow and the data items that are exchanged b/w the different flow are represented in a diagram known as DFD.

Object oriented Design → object oriented design technique is an intuitively appealing approach where the natural objects relevant to a problem are first identified and then the relationship among the objects such as composition, reference and inheritance are determined.



Testing Process → Software testing is a process of verifying and validating that a S/W application or program.

→ Software testing is designed to establish that the S/W is working satisfactorily as per the requirements.

→ S/W testing is a process designed to prove that the program is error free.

→ To understand essential characteristics of tools used for test automation.

→ To described strategies for generating system test cases.

→ To describe principles of system and component testing.

S/W Testing has three main purposes →

Verification, Validation and defect finding.

→ The verification process confirms that the S/W meets its technical specifications. A specification is a description of a function in terms of a measurable o/p

value given a specific input value under specific pre conditions.

→ Validation process confirms that the S/w meets the business requirements. Eg of this is "After choosing a branch office name, info^n about the branch's customer account managers will appear in a new window.

→ A defect is a variance b/w the expected and actual result. The defects ultimate source may be ~~traced~~ traced to a fault introduced in the specification, design or development phases.

Terminologies in Testing →

Faults, Errors and Failure →

→ Fault → An incorrect step, process or data definition in a computer program which causes the program to perform in an unintended manner.

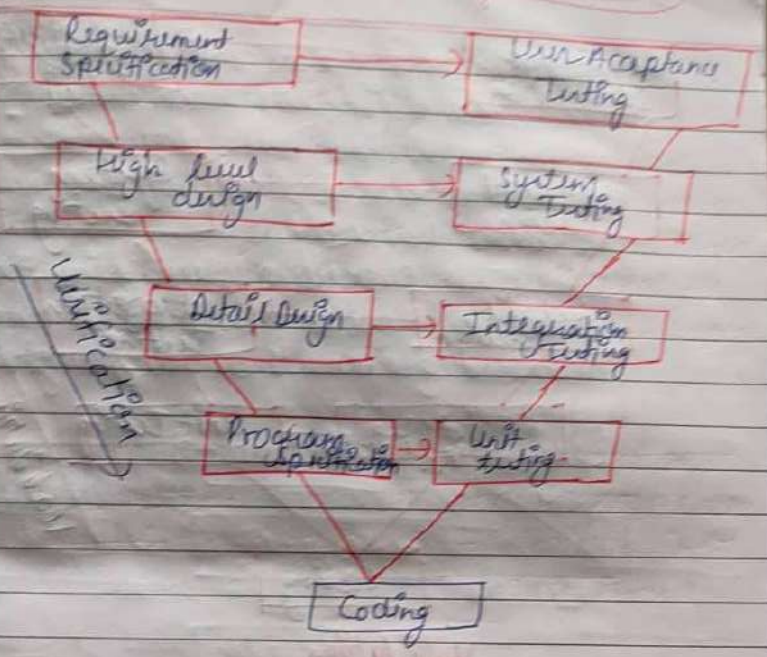
Fault Avoidance → Using techniques and procedures which aim to avoid the introduction of faults during any phase of the safety lifecycle of the safety-related system.

→ Fault Tolerance → The ability of a functional unit to continue to perform a required func^n in the presence of faults or errors

Failure → The inability of the system or component to perform its required function with in specified performance requirements.

Error → A discrepancy b/w a computed, observed or measured value or cond^n and the true, specified or theoretically correct value or condition.

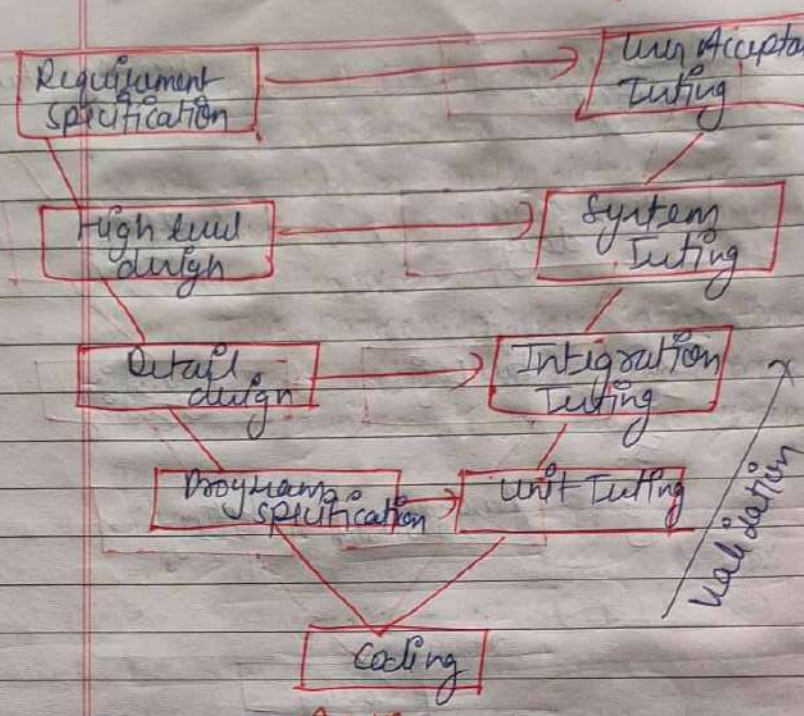
Verification → Verification is a disciplined approach to evaluate whether a s/w product fulfills the requirements or condition imposed on them by the standard or process. It is done to ensure that the process and procedures defined by the customer for development and testing are followed correctly. It is also called static technique as it does not involve execution of any code, program or work product.



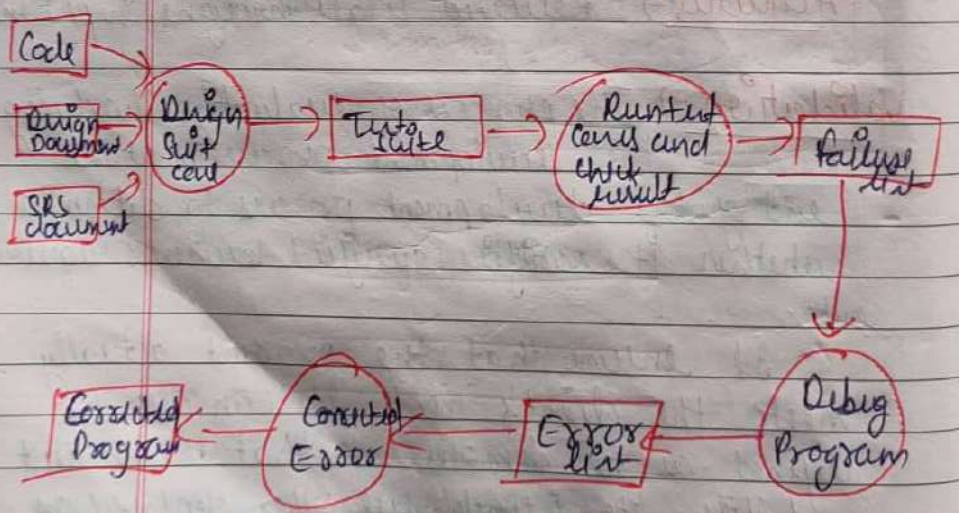
Activities → Reviews, Walkthroughs, Inspection

Validation → The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

It ensure that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.



Validation



Testing Process

Difference b/w Verification & Validation

Verification	Validation
1- Verification is a static practice of verifying documents, design, code and program.	1- Validation is a dynamic mechanism of validating & testing the actual product.
2- It does not involve executing the code.	2- It always involves executing the code.
3- It is human based checking of documents & files	3- It is computer based execution of program
4. Verification uses methods like Inspection, walkthroughs and Desk-checking etc.	4. Validation uses methods like black box testing, gray box testing and white box testing.
5- Verification is to check whether the S/w conforms to specifications.	5- Validation is to check whether S/w meets the customer expectations and requirements.
6- It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is high level exercise.

Test Case → A Test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

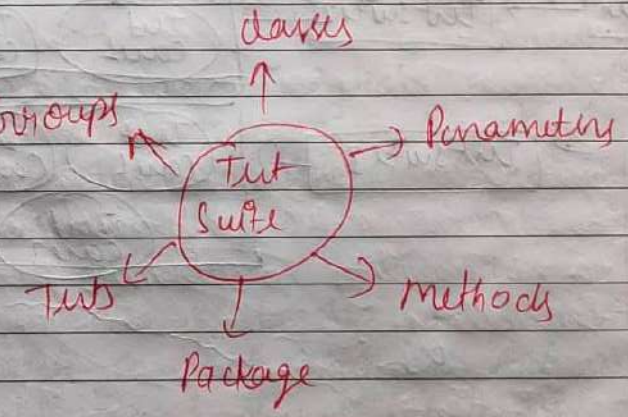
The process of developing test cases can also help find problems in the requirements or design of an appln.

Designing Test Cases →

Create a test plan
↓
Plan the test lab
↓
Design the test lab
↓
Develop the test lab
↓
Design test cases
↓
Conduct test
↓
Use the lab after deployment.

A test case includes-
→ The purpose of the test
→ Special hardware requirements, such as a modem.
→ Special S/w requirement, such as a tool.
→ Specific setup or configuration requirement.
→ A description of how to perform the test.
→ The expected results or success criteria for the test.

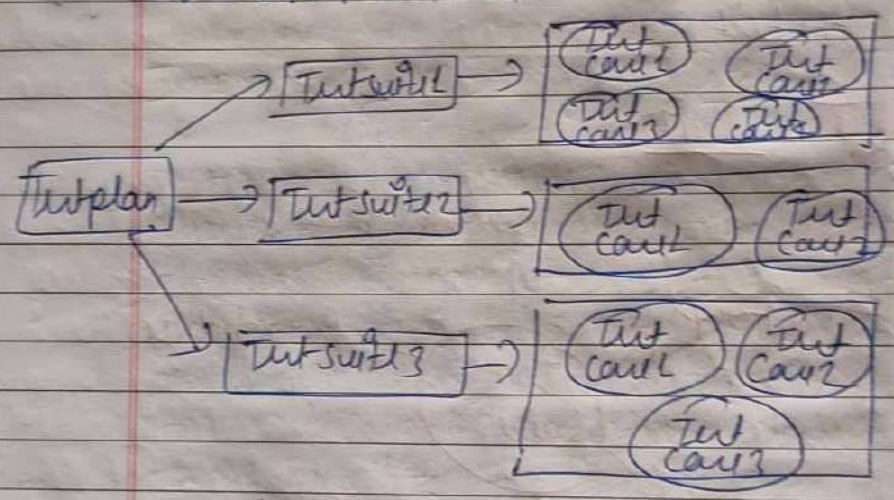
Test Suit →



Test Suite is a container that has a set of tests which helps testers in executing and reporting the test execution status. It can take any of three states namely Active, In Progress and Completed.

A test case can be added to multiple test suites and test plans. After creating a test plan, test suites are created which in turn can have any number of tests.

Test suites are created based on the cycle or based on the scope. It can contain any type of tests, functional or non-functional.



Test oracles → A test oracle is a mechanism, different from the program itself that can be used to check the correctness of the output of the program for the test cases.

An oracle is a mechanism for determining whether the program has passed or failed a test.

A complete oracle would have three capabilities and would carry them out perfectly.

→ A generator → to provide predicted or expected results for each test.

→ A comparator, to compare predicted and obtained results.

→ An evaluator, to determine whether the comparison results are sufficiently close to be a pass.

One of the key problems with oracle is that they can only address a small subset of the inputs and outputs actually associated with any test. The tester might intentionally set the value of some variables, but all of the programs other variables have values too. Configuration settings, amount of ~~variable~~ available memory, and program options can also affect the test results.

Impracticality of Testing All Data →

For most programs, it is impractical to attempt to test the program with all possible inputs, due to a combinatorial explosion. For those inputs selected, a testing oracle is needed to determine the correctness of the O/P for a particular test input.

Impracticality of Testing All Paths →

For most programs, it is impractical to attempt to test all execution path through the project due to a combinatorial explosion. It is also not possible to develop an algo for generating test data for paths in an arbitrary product, due to the inability to determine path feasibility.

Verification →

Verification is the process of evaluating work products of a development phase to determine whether they meet the specified requirements.

Verification ensures that the product is built acc to the requirements

and design specification.

Methods of verification →

Self Review → Self review may not be referred as an official way of review in most of the S/W verification descriptions, as it is assumed that everybody does a self check before giving work product for further verification.

Peer review → Peer review is the most informal type of review where an author and a peer are involved. It is a review done by a peer and review records are maintained. A peer may be a fellow developer or tester as the case may be.

Walk through → Walk through is a semi-formal type of review as it involves larger teams alongwith the author reviewing a work product. It may involve a project team or part of a project team doing a review jointly as the case may be.

SRS verification →

SRS is a detailed description of a S/w system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement b/w customer and contractors. It may include the use cases of how user is going to interact with S/w system. SRS document contains all necessary requirements required for project development.

A good SRS defines the how S/w system will interact with all internal modules, hardware, commⁿ with other programs and human user interactions with wide range of real life scenarios.

1- **Correctness of SRS should be checked** → Since the whole testing phase is dependent on SRS, it is very important to check its correctness. There are some standards with which we can compare & verify.

2- **Ambiguity should be Avoided** → Sometimes in SRS, some words have more than one meaning and this might confused testers making it difficult to get the exact requirement.

3- Requirements should be complete → When tester writes test cases, what exactly is required from the appln is the first thing which needs to be clear. For eg → If appln needs to send the specific data of some specific size then it should be clearly mentioned in SRS that how much data and what is the size limit to send.

4- Consistent requirement → SRS should be consistent within itself and consistent to its reference documents.

5- Verification of expected result → SRS shouldn't have statement like "Work as expected". It should be clearly stated that what is expected since different testers would have diff thinking aspects and may draw diff results from this statements.

User Documentation Verification →

It is a type of non-functional testing. Any written or pictorial form describing, defining, specifying, reporting or certifying activities, requirements, procedures, or results. Documentation is as important to a product success as the product itself.

If the documentation is poor, non-existent or wrong, it reflects on the quality of the product and the vendor.

→ As per the IEEE documentation describing plans for, or results of, the testing of a system or components, types include test case specification, test incident reports, test log, test plan, test procedure, test report. Hence the testing of all the above mentioned documents is known as documentation testing.

→ This is one of the most cost effective approaches to testing. If the documentation is not right; there will be major & costly problems. The documentation can be tested in a no of different ways to many different degrees of complexity.

Documentation testing is performed at three level

a) Read test → Documentation is reviewed for clarity, organization, flow & accuracy w/o executing the documented instrn

b) Hands-on test → The online help is exercised and the error messages are verified to evaluate their accuracy and usefulness.

c) Functional test → The instruction embedded in the documentation are followed to verify:

Software Project Audit →

A software audit review or software audit, is a type of software review in which one or more auditors who are not members of the s/w development organization conduct "An independent examination of a s/w product software process, or set of s/w processes to assess compliance with specifications, standards, contractual agreements or other criteria.

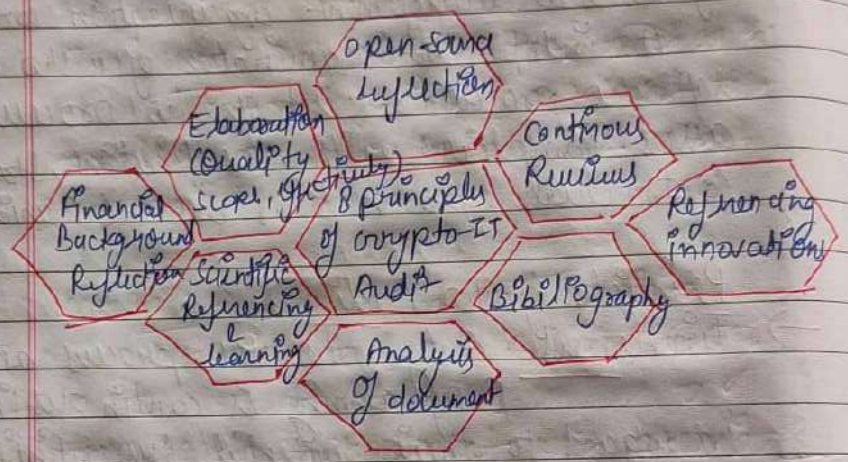
Principle of S/w Audit →

→ **Timeliness** → Only when the process and programming is continuous inspected in regard to their potential susceptibility to faults and weakness, but as well with regard to the continuation of the analysis of the found strengths.

→ **Source openness** → It requires an explicit reference in the audit of encrypted programs, how the handling of open source has to be understood. Eg- Program

→ **Elaborateness** → Audit process should be oriented to certain minimum standard. The recent audit process of encrypting s/w often vary greatly in quality, in the scope and effectiveness and also experience in the media reception

Often differing perceptions -



→ Financial content → further transparency is needed to clearify whether the S/W has been developed commercially and whether the audit was funded commercially.

⇒ Scientific referencing of learning perspectives

Each audit should describe the finding in detail within the content and also highlight progress and development needs constructively.

Software Tailing →

a) Tailoring → Tailoring is the responsibility of the development team in consultation with the process champion or S/W quality assurance who actually runs the development process.

→ Purpose of tailoring is to eliminate tasks that add unnecessary costs, cycle time and data that donot add value to the process or the definition of the product.

→ tailoring of a standard takes the form of deletion of wasteful steps. Alteration of activities to more explicitly reflect the appl^n to a particular context or addition of activities to satisfy program requirements.

b) Tailoring S/W Quality assurance program by criteria →

(i) Continuous Improvement → All the standard process in SQA must be improved frequently and made official so that the other can follow.

(ii) Documentation → All the QA policies and methods, which are defined by QA team, should be documented for training and use for future projects.

(iii) Experience → Choosing the members who are seasoned SQA auditors is a good way to ensure the quality of management review.

(iv) Tool usage → utilizing tool such as the tracking tool, management tool for SQA process reduces SQA effort and project cost.

(v) metrics → Developing and creating metrics to track the S/W quality in its current state, as well as to compare the improvement with previous version.

Inspection →

→ Inspection improves the reliability, availability and maintainability of a S/W product.
→ Anything readable that is produced during S/W development can be inspected.
→ Inspections can be combined with structured, systematic testing to provide a powerful tool for creating defect free programs.

→ The inspection activity follows a specified process and the participants play well defined roles.

→ An inspection team consists of three to eight members who play the roles of moderator, author, reader, recorder and inspector.

Steps in the Inspection process →

→ Planning
→ Overview meeting
→ Preparation
→ Inspection meeting
→ Rework
→ Follow up

Configuration Audit →

→ Configuration audit provides a mechanism for determining the degree to which the current state of the system is consistent with the latest baseline and documentation.
→ They provide greater visibility into the status of a project by evaluating the status of the items.

→ Configuration audits provide such a review.

→ They are not the same as quality audits or product test.

→ Configuration audits are typically performed at the time of delivery and major upgrades to the S/w.

Types of configuration Audit →

→ FCA (functional configuration Audit)
→ Physical configuration Audit.

Assignment - 1

Q Differentiate b/w verification and validation.

Q Define the term test, test case and test suite.

Q Explain the different types of verification methods.

Unit-2

(Functional Testing & Structural Testing)

Functional Testing →

Functional testing is a Quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the S/W component under test.

Functions are tested by feeding them input and examining the O/P and internal program structure is rarely considered.

Function testing usually describes what the system does.

Functional testing is a testing technique that is used to test the features of the system or S/W, should cover all the scenarios including failure paths and boundary cases.

Functional testing verifies that each function of the S/W appln operates in conformance with the requirement specification.

Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual result with the expected result.

Boundary Value Analysis →

* Boundary value analysis is a method useful for arriving at test that are effective in catching defects that happen at boundaries.

* Boundary value analysis believes and extends the concept that the density of defects is more towards the boundaries.

* It has been found that most defects in situations such as this happen around the boundaries.

* Boundary value analysis is extremely useful in uncovering defects when there are internal limits placed on certain resources, variables or data structures.

Advantages →

1. Small set of test case is prepared.
2. It reduces the no of test cases for a tester.
3. Very good at exposing potential user interface input problems.
4. Easy to test the boundary values.
5. Provide clear guidelines to prepare the test cases.

Disadvantages →

1. Difficult to prepare the test cases if the boundary values are not known correctly.
2. Does not test dependencies b/w combination of inputs.
3. Not useful for large amount of values.
4. Not all possible inputs are tested.

Equivalence class Testing →

→ Equivalence partitioning is a s/w testing technique that involves identifying a small set of representative input values that produce as many different o/p conditions as possible.

→ This reduces the no of permutations and combinations of input, output values used for testing, thereby increasing the coverage and reducing the effort involved in testing.

→ The set of input values that generate single expected o/p is called a partition.

→ When the behaviour of the s/w is the same for a set of values, then the set is termed as an equivalence class or a partition.

Steps to prepare an equivalence partition table -

1- Choose criteria for doing the equivalence partitioning, list of values.
2- Identify the valid equivalence classes based on the above criteria.
3- Select a sample data from that partition.
4- Identify special values, if any, and include them in a table.
5- Check to have expected results for all the cases prepared.
6- If the expected result is not clear for any particular test case, mark appropriately and escalate for corrective actions.

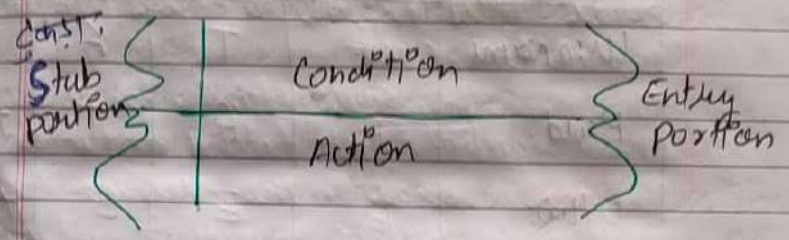
Decision Table based testing →

A decision table is a good way to deal with combinations of things. This technique is sometimes also referred to as 'cause-effect' table.
The reason for this is that there is an associated logic diagramming technique called 'cause-effect graphing' which was sometimes used to help in deriving the decision table.
It helps in reducing test effort in verifying

Each and every combination of test data, at the same time ensuring complete coverage.

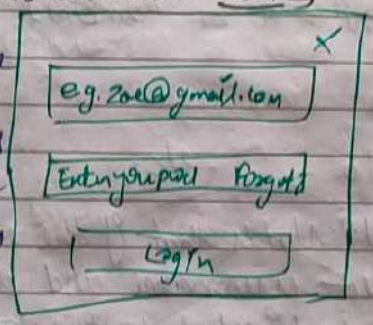
→ Decision Table provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.

→ Decision tables can be used in test design inorder to determine whether or not they are used in specification or not.



- Why to use Decision table →

DT is a tabular representation of inputs, various rules, cases or test condition



Eg. Let's take an example and see how to create a decision table for a login screen.

The condition states that if the user provides the correct username and pass the user will be redirected to the homepage. If any of the input is wrong, an error msg will be displayed.

The decision table is used for testing the system behaviour for diff input combinations. This is a systematic approach where the different input combinations and their corresponding system behaviour are captured behaviour in tabular form.

Conditions	Rule1	Rule2	Rule3	Rule4
Username	F	T	F	T
Pwd	F	F	T	T
O/ip	E	E	E	H

2no of things to be combined

$2^3 = 8$

$2^4 = 16$ ---etc.

The condn states that if the user provides the correct username and pwd the user will be redirected to the homepage. If any of the input is wrong, an error msg will be displayed.

T → Correct username/pwd
F → wrong username/pwd
E → error msg is displayed
H → Home screen is displayed.

$2^6 = 64$

Case1 Username and Pwd both were wrong. The user is shown an error message.

Case2 - Username was correct, but the Pwd was wrong. The user is shown an error message.

Case3 - Username was wrong, but the Pwd was correct. The user is shown an error message.

Case4 Username and Pwd both were correct, and the user is navigated to the homepage.

Why is Decision table Important?

A decision table is an outstanding technique used for testing and requirements management.

→ DT are very much helpful in test design technique.

→ It helps testers to search the effects of combinations of diff inputs and other sys states that implement business rules.

→ It provides a regular way of stating complex business rules which benefits the developers as well as testers.

→ It is the most preferable choice for testing and requirements management.

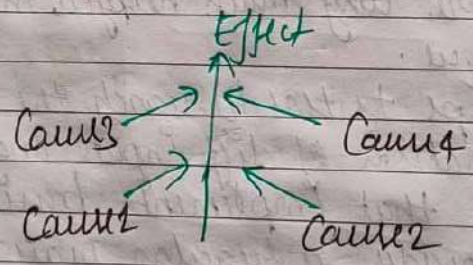
→ It is a structured exercise to prepare requirements when dealing with complex business rules.

Cause-Effect Graphing Technique →

Cause-Effect Graph graphically shows the connection b/w a given outcome and all causes that manipulate the outcome. Cause effect graph is a black box testing technique.

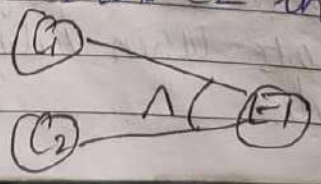
It is a testing technique that aids in choosing test cases that logically relate causes (inputs) to Effects (outputs) to produce test cases.

A "Cause" stands for a separate input condition that fetches 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.

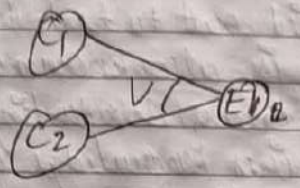


Notation we are going to use →

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

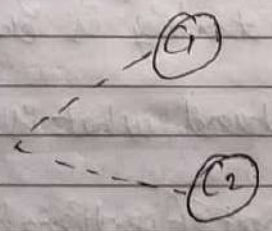


OR → For effect E1 to be true, either of causes C1 OR C2 should be true.



NOT → For effect E1 to be true, cause C1 should be false.

MUTUALLY EXCLUSIVE → When only one of the causes will hold true.



Cause → Input
Effect - output

Structural Testing →

It is the type of testing carried out to test the structure of code. It is also known as White Box testing or Glass Box testing. This type of testing requires knowledge of the code, so it is mostly done by the developers. It is more concerned with how system does it rather than the functionality of the system.

→ The objective of structural testing is not to check diff input or o/p cond^n but to check diff data and programming structure used in the program.

→ Structural testing is done -

a- To understand what is missing in our test suit.

b. To complement functional testing.

c. It helps to identify obvious ~~and~~ independent

Categories of Structural testing →

1- Statement Coverage → It is the weakest form of testing, as it requires that every statement in the code has to be executed at least once

2) Branch coverage → Each branch condition for the program is tested for its true or false values.

3) Path coverage → For path coverage, the path of the program is executed at least once, it test individual path for the program.

4) Condition coverage → In this type of testing, it checks all possible combination of condition.

Diff between structural and functional Testing

Structural Testing	Functional Testing
1. This testing will uncover error occured during the coding of the program	This testing will uncover error occured during the implementation of requirements and design specification.
2. It is concerned about both the result and the process.	It is concerned only about the result but not the processing.
3. Structural Testing is often referred as white box testing. The knowledge of code is very much essential	It is often referred as black box testing, no need to know about the coding of the program

Control Flow Testing → Control Flow testing is a testing technique that comes under white box testing. Aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path.

Control Flow Graph is formed with the node, edge, decision node, junction node to specify all possible execution path.

Notation used for Control Flow Graph

1) Node → Nodes in the control flow graph are used to create a path of procedures. It represents the seq of procedures which procedure is next to come so, the tester can determine the seq of occurrence of procedure.

2) Edge → Edge in control flow graph is used to link the direction of nodes.

3) Decision node → It is used to decide next node of procedure as per the value.

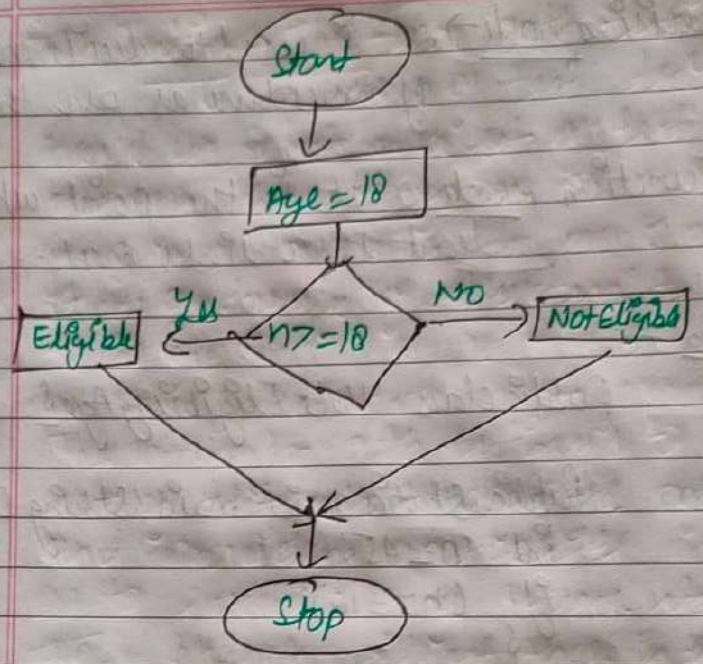
4) Junction Node → It is the point where at least three links meets.

Eg -

```
Public class Vote Eligibility Age{

    Public Static void main (String [] args) {
        int n = 45;
        if (n >= 18)
        {
            System.out.println("You are eligible for voting");
        } else
        {
            System.out.println("You are not eligible for voting");
        }
    }
}
```

Control Flow Graph ⇒ The above example shows eligiblity criteria of age for voting where if age is 18 or more than 18 so print message "You are eligible" if it is less than print "You are not eligible".



Program for this scenario is written above and the control flow graph is designed for the testing process.

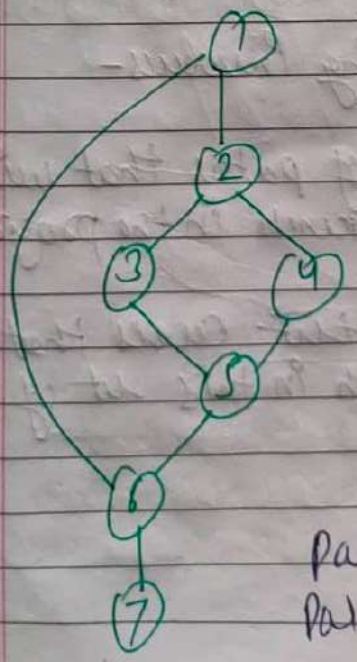
In the control flow graph Start, age, eligible, not eligible and Stop are the nodes, n>=18 is a decision node to decide which part will execute as per the given value.

Connectivity of the eligible node and not eligible node is that on the Stop node.

Path Testing → It is a structural testing method that involves using the source code of a program in order to find every possible executable paths. It helps to determine all faults lying within a piece of code. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

Basis Path Testing → It is based on white box testing method, that defines test cases based on the flows or logical path that can be taken through the program.

The objective behind basis path in S/W testing is that it defines the no of independent paths, thus the no of test cases needed can be defined explicitly.



1- if A=50
2- Then if B>C
3. THEN A=B
4. Else A=C
5. End if
6. End if
7. Print A

Path1 → 1,2,3,4,5,6,7
Path2 → 1,2,4,5,6,7
Path3 → 1,6,7

Path Testing → The term path testing represents a group of test techniques based on selecting a set of test paths through the program.

→ It is a family of structural testing technique.

→ The basis for path testing is to select a set of test path through the program and providing appropriate inputs so as to execute every source statement and branch at least once in that program.

→ Path testing is applicable to unit testing. It is effective for small and medium program.

→ This type of testing involves-

a) Generating a set of paths that will cover every branch in the program.

b) Finding a set of test cases that will execute every path in this set of program path.

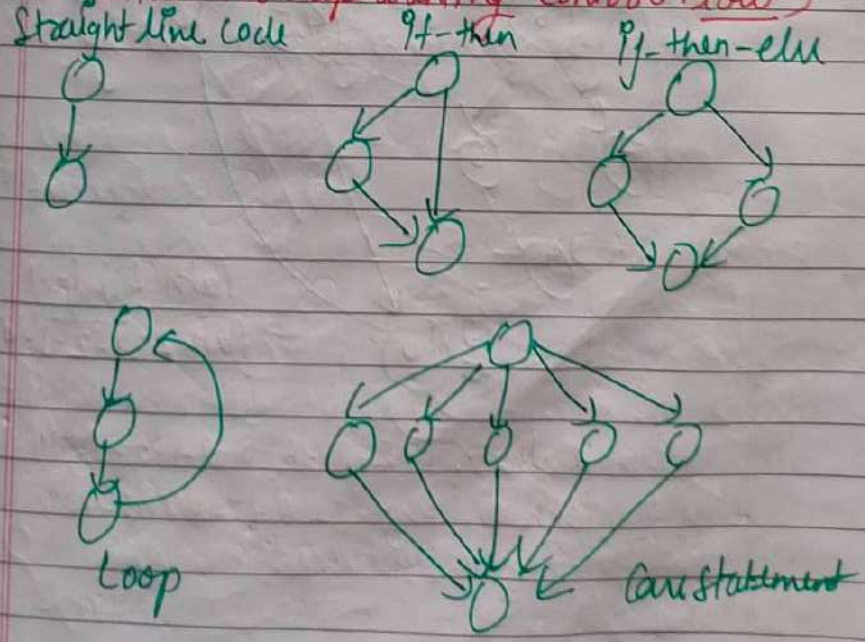
Path Testing Technique →

1) Control flow Graph (CFG) :- The program is converted into flow graph by representing the code into nodes, regions and edges.

2) Decision to Decision path (D-D) → CFG can be broken into various decision to decision paths and then collapsed into individual nodes.

3) Independent Paths → It is a path through a DD path graph which cannot be reproduced from other paths by other methods.

Notation for representing control flow →

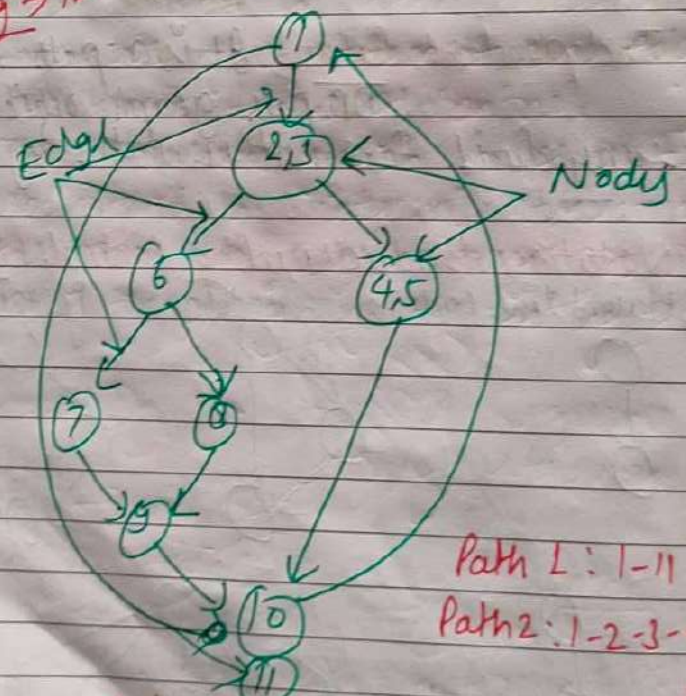


Independent Path →

→ An independent path is any path through the program that introduces atleast one new set of processing statements or a new condition.

→ When stated in terms of a flow graph, an independent path must move along atleast one edge that has not been traversed before the path is defined.

Eg →



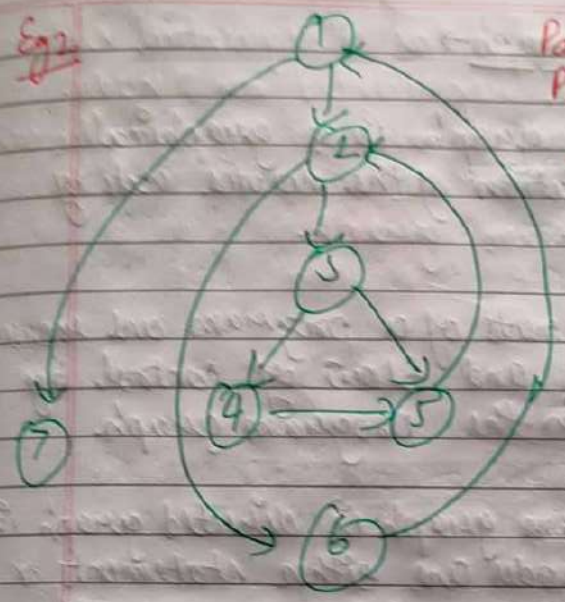
Path 1 : 1-11

Path 2 : 1-2-3-4-5-10-1-11

Path 3 : 1-2-3-6-8-9-10-1-11

Path 4 : 1-2-3-6-7-9-10-1-11

Eg 2



Path 1: 1-7
Path 2: 1-2-6-1-7
Path 3: 1-2-3-4-5-2-6-1-7
Path 4: 1-2-3-5-2-6-1-7

Generation of a graph from Program-

1- Graph are extensively used in testing of computer programs. We may represent the flow of control of any program in terms of directed graph.

2. A graph representing the flow of control of a program is called a program graph.

3. The program graph may further be transformed into the DD path graph.

4. Both of these graphs may provide foundation to many testing techniques.

Program Graph →

1- Program graph is a graphical representation of the source code of program.

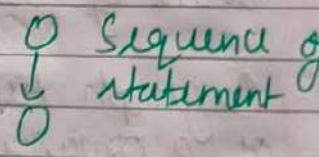
2. The statements of a program are represented by nodes and flow of control by edges in the program graph.

3. A program graph is directed graph in which nodes are either statements or fragments of a statement and edges represent flow of control.

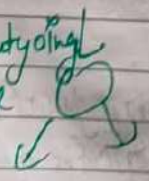
4. The program graph helps us to understand the internal structure of the program which may provide the basis for designing the testing techniques

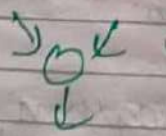
Node O

Edge →

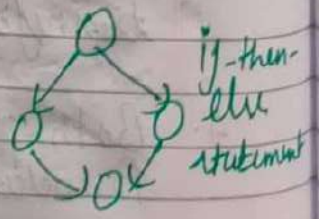


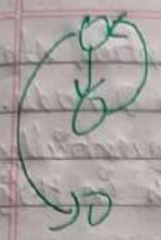
Two outgoing edge





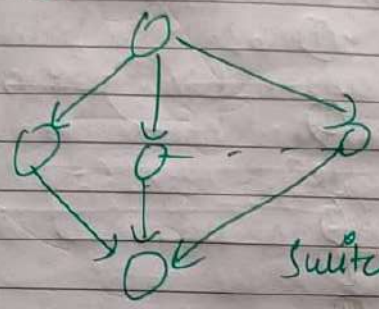
(more than two outgoing edge)





(while loop)

Statement representation



Switch statement

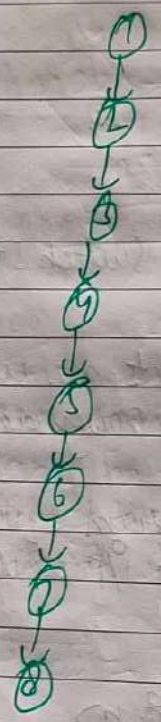
→ The basic constructs are used to construct a program is its program graph.

→ Consider the program square which takes a no as an i/p and generates the square of the no.

```
# Include< stdio.h>
  Void main()
  {
    int number, result
     Pf("Enter the value")
     Scanf("%d", & number)
     result = number * number
     Pf(" The result is %d", result);
    }
```

→ This program has 8 sequential statements which are represented by 8 nodes.

→ All nodes are arranged sequentially which may lead to only one path in this program graph.



Identification of Independent Paths →

→ There are three equations from graphing theory that will we to calculate the no of linearly independent paths through any structured system.

2- These three equations and the theory of linear independence were the work of a Dutch scholar named C.

3. The cyclomatic No $V(G)$ of a strongly connected graph G with N nodes, E edges and one connected component.

4- This cyclomatic number is the no of linearly independent paths through the sys.

→ $G = IP = Edges - Nodes + 2$ $(IP = E - N + 2)$

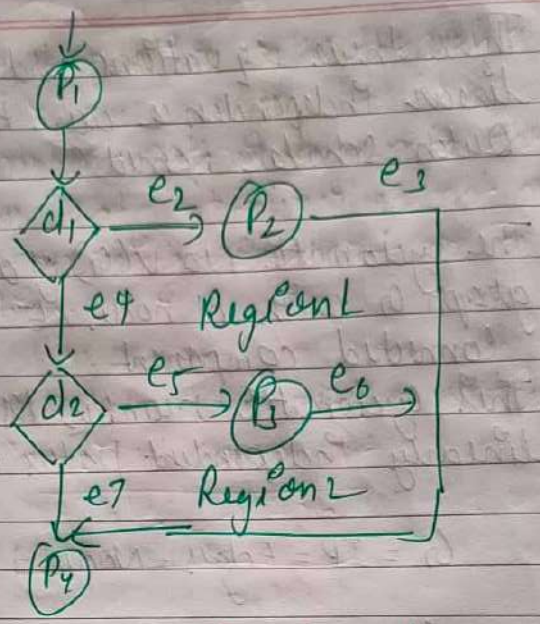
$G = IP = Regions + 1$ $(IP = R + 1)$

$G = IP = Decisions + 1$ $(IP = D + 1)$

5- Even though the case statement and the series of required decisions do not have the same no of total paths, they do have the same no of linearly independent paths.

6- The no of linearly independent paths through a system is usually the minimum no of end-to-end paths required to touch every path segment at least once.

7- If a looping structure is traversed one time.



Looping structure

→ All three equations must be equal to the same no for the logic circuit to be valid. If the system is not a valid logic circuit, it cannot work.

→ ~~Yo~~ Nodes = Decision + Process

→ (a) Independent paths = Edges − Nodes + 2

(b) Independent paths = Regions + 1

(c) Independent paths = Decisions + 1

Edges = 7, Decision = 2, Process = 4, Region = 2

Nodes = Decisions + Processes = 2+4 = 6

Independent paths = Edges - Nodes + 2
= 7 - 6 + 2 = 3

Independent paths = Regions + 1 = 2 + 1 = 3

Independent paths = Decisions + 1 = 2 + 1 = 3

Cyclomatic Complexity →

→ The cyclomatic complexity is also known as structural complexity because it gives internal view of code.

→ This approach is used to find the no of independent paths through a program. This provides us to upper bound for the no of tests that must be conducted to ensure that all statements have been executed atleast once and every condn has been executed on its true and false side.

→ If a program has backward branch then it may have infinite no of paths. Although it is possible to define a set of algebric expression that gives the

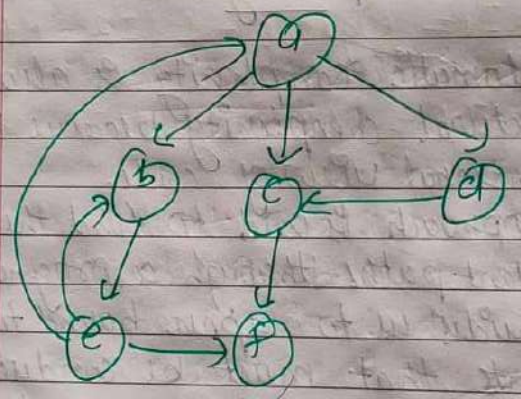
total no of possible paths through a program

→ An independent path is any path through the program that introduces atleast one new set of processing statement or a new condition.

→ V(G) of a graph G with n vertices, e edges and P connected component

$$\boxed{V(G) = e - n + 2p}$$

→



$V(G) = 9 - 6 + 2 = 5$

$e = 9,\ n = 6\ \ P = 1$

~~Independent Path~~ →

Path 1 : a c f

Path 2 : a b e f

Path 3 : a d c f

Path 4 : a b e a c f or a b e a b e f

Path 5 : a b e b e f

Properties →

(a) $V(G) >= 1$

(b) $V(G)$ is the max^m No of independent paths in Graph G.

(c) Inserting and deleting functional statements to G does not affect $V(G)$.

(d) G has only one path if and only if $V(G)=1$

(e) Inserting a new row in G increases $V(G)$ by unity.

(f) $V(G)$ depends only on the decision structure of G.

Eg →

```
if (Condition 1)
  Statement 1
else
  Statement 2
  if (Condition 2)
    Statement 3
  else
    Statement 4
```

e = 9
n = 7
p = 1

$C.C = 9 - 7 + 2 = 4$

Data flow Testing →

→ It is a technique used to detect improper use of data in a program.

→ By looking data usage, risky areas of code can be found and more test cases can be applied.

→ A data flow graph is a graph which represents data dependencies b/w a no of operations.

→ In control flow testing, we find various path of a program and design test cases to evaluate those paths.

→ Eg →

```
# include <stdio.h>
void main()
{
int a, b, c;
  a = b+c;
Pf ("%d", a);
}
```

Value of 'a' may be previously stored in the memory location assigned to variable 'a' or garbage value.

→ If we execute the program, we may get an unexpected value.

→ main Point of concern are -
a- Statements where variable receive value (definitions)
b- Statements where these values are used (referenced)

Data Flow Strategies →

1. All definitions → Test cases for each definition of each variable.

2. All predicate uses → Test cases are generated so that there is at least one path of each variable definition to each P-use of variable.

3. All ~~P uses some~~ Computational uses → Test Cases are generated so that there is atleast one path of each variable definition to each C-use of variable.

4. All P uses some C uses → Test cases for every variable, there is a path from every definition to every p-use of that definition.

5. All C uses some P uses → Test cases for every variable, there is a path from

every definition to every c-use of that definition.

6- All uses → Test cases for every use of the variable, there is a path from the definition of that variable to the use.

7- All du paths → This is the strongest data flow testing strategy. Every du path from every definition of every variable to every use of that definition.

Mutation Testing →

Mutation Testing is a method of s/w testing in which program or source code is deliberately manipulated followed by suite of testing against the mutated code.

→ The mutations introduced to source code are designed to imitate common programming errors.

→ Mutation testing is a kind of testing in which the appln is tested for the code that was modified after fixing a particular bug.

→ It also helps in finding out which code and which strategy of coding can help in developing the functionality

effectively.

→ Mutation testing ensures the accuracy of our unit test.

Types of mutation Testing →

1) Value mutations → An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.

2) Decision mutations → The condⁿ are changed to check for the design errors. One changes the arithmetic operators to locate the defects.

3) Statement mutations → Changes done to the statements by deleting the line which might arise when a developer is copy pasting the code from somewhere else.

Eg → Original if (x>y) Print "Hello" Else Print "Hi"

Mutant if (x < y < Strong = " ">) <|y < > Print "Hello" Else Print "Hi".

Unit-3 (Regression Testing)

Regression Testing →

1. S/w undergoes constant changes.

2. Such changes are necessitated because of defects to be fixed, enhancements to be made to existing functionality or new functionality to be added.

3. Anytime such changes are made, it is important to ensure that -

a) The changes or additions work as designed.

b) The changes or additions do not break something that is already working & should continue to work.

Eg → (a) Assume that in given release of a product, there are three defects i.e, D^1, D^2 and D^3, the development team will fix these defects and the testing team will perform test to ensure that these defects are indeed fixed.

b) When the customers start using the product, they may encounter new defects D^4 and D^5.

c) Again the development and testing team will fix and test these new defect fixes.

d) But in this process defect b' is again occur.

Types of Regression Testing →

1) Regular Regression Testing →

* A regular regression testing is done b/w test cycles to ensure that the defect fixes that are done and the functionality that were working with the earlier test cycles continue to work.

* A regular regression testing can use more than one product build for the test cases to be executed.
* A build is an aggregation of all the defect fixes and features that are present in the product.

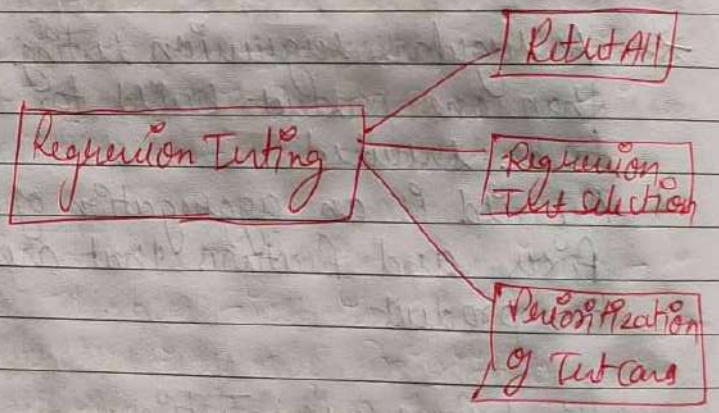
2) Final Regression Testing →

* It is done to validate the final build before release.
* The final regression test cycle is conducted for a specific period of duration, which is mutually agreed upon b/w the development and testing teams. This is called "cook time" for regression testing.

Regression Testing Process →

1- **Performing an Initial "Smoke" or "Sanity" test →**

Identifying the basic functionality that a product must satisfy.
Eg → If one builds a database, then any build of the database should be able to start it up and perform basic oprn such as queries, data manipulation and shutdown of the database.



1- **Retest All →** This is one of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

2- Regression Test Selection →

→ Instead of re-executing the entire test suite, it is better to select part of the test suite to be run.

→ Test cases selected can be categorized as Reusable Test cases, Obsolete Test cases.

→ Reusable Test cases can be used in subsequent regression cycles.

→ Obsolete Test cases can't be used in succeeding cycles.

3- Prioritization of Test Cases →

Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Regression Test Cases Selection →

(i) Once the test cases are prioritized, test cases can be selected. There could be several approaches to regression testing which need to be decided on a case by case basis.

(ii) For eg -

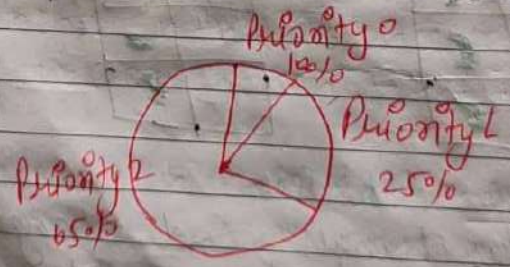
Prioritization of test cases in Regression Testing →

Prioritizing the test cases depends on the business impact, critical and frequently used functionality. Selection of test cases based on priority will reduce the test suite.

Test cases can be classified -

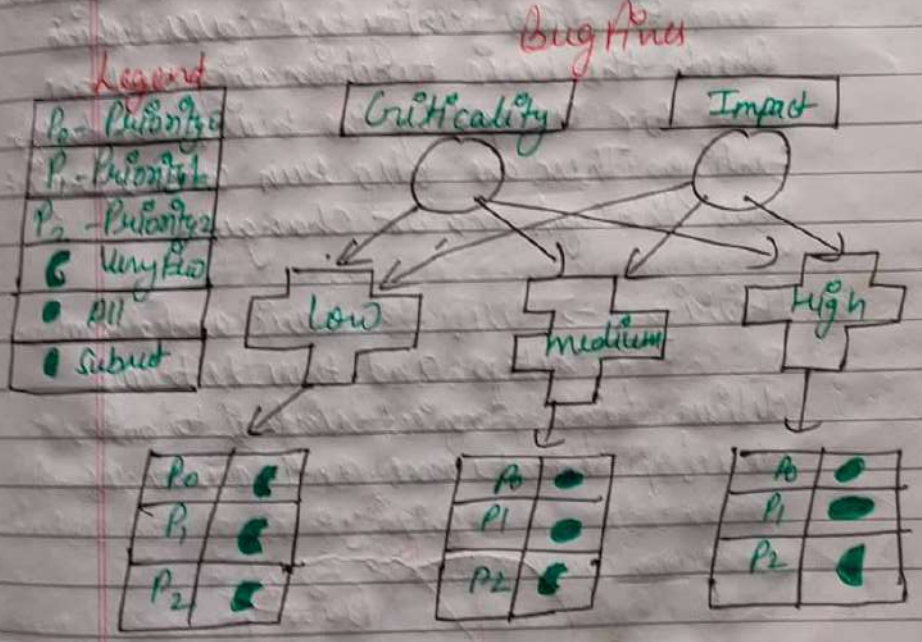
a) Priority 0 :- These test cases can be called as Sanity test cases which check the basic functionality and are run for accepting the build for further testing. These are also run when a project goes through major changes.

b) Priority-1 → Uses the basic and normal setup and these test cases deliver high project value to both development teams and customers.



c) Priority-3 → These test cases deliver moderate project value and are executed as a part of S/w testing life cycle and selected for regression on need basis.

1. Regression all → For regression testing, all priority 0, 1, and 2 test cases are selected.
2. Priority based regression → For regression testing based on the priority, all priority 0, 1 & 2 test cases are run in order based on the availability of time.
3. Random Regression → Random test cases are selected and executed.



4. Regress changes → Code changes are compared to the last cycle of testing and test cases are selected based on their impact on the code.

1- Case1 -> If criticality and Impact of the defect fixes are low, then it is enough to select few test cases from test case Database and execute them.

2- Case2 ->
a- If the criticality and the Impact of the bug fixes are medium, then we need to execute all Priority-0 and priority-1 test cases.
b- If bug fixes need additional test cases from priority-2 then those test cases can also be selected and used for regression testing.
c- Selecting Priority-2 test cases in this case is desirable but not a must.

3- Case 3-> If the criticality and impact of the bug fixes are high, then we need to execute all Priority-0, Priority-1 and carefully selected Priority-2 test cases.

iii) The above methodology requires Impact analysis of bug fixes for all defects. It can be a time consuming process.

iv) If there is not enough time and the risk of not doing Impact analysis is low, then the following methodologies are used.

→ Test case Prioritization orders test cases with the goal of finding faults in regression testing as quickly as possible.

→ For seed-based symbolic execution, we can also order the seeds, with the goal of gaining incremental coverage as quickly as possible.

→ A large body of work exists on prioritization methods for regression testing, and it is not our purpose to explore all possible strategies.

Five Ranking Strategies →

a) Random ordering → Our first prioritization is not in fact a proposed solution but a baseline to measure the importance of ordering.

b) Branch Total → Prioritize test cases by their branch coverage such that the test with the highest branch coverage is used as a seed first.

c) Branch Additional → Prioritize by incremental branch coverage over all tests previously ranked. The first test chosen is therefore the same as with the BT method, but further tests are selected by the additional coverage provided, not absolute coverage.

d) Farthest - Point - First (FPF) :-

→ The first test ranked is chosen randomly. Future test cases are selected by computing the minimum distance to all already ranked tests and adding the test case with the largest minimum distance to the ranking.

→ The distance func^n in our case is the Hamming distance b/w the two test cases branch coverage vectors; even if a test does not produce incremental branch coverage

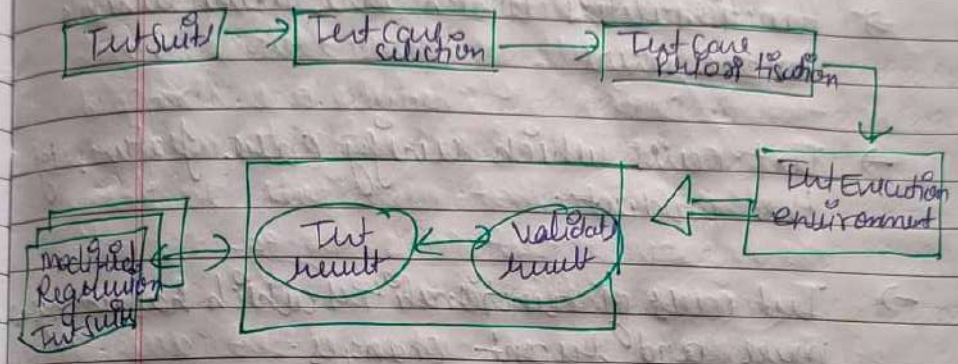
e) Shortest Path →

Tests are ranked in order of increasing path length. The no of branches taken during is counted and paths that execute fewer total branches in the source of the tested program are chosen first.

Reducing the no of test cases →

→ For most projects it is nearly impossible

Regression Test case Selection →



Regression Test Selection is a way of selecting the most appropriate test case to ensure that it delivers the correct result. It enables a tester to select a valid set of test cases to make sure the unmodified part of the program is error free.

→ Identify the Affected part of the code →

The unmodified part of the program that gets affected due to modification is identified here.

→ Test case Selection → Here the test cases that involve the selection of the subset of test cases, from the initial test suit, are selected by the team.

→ Test cases which have frequent defects.

→ Functionalities which are more visible to the users.

→ Test cases which verify core features of the product.

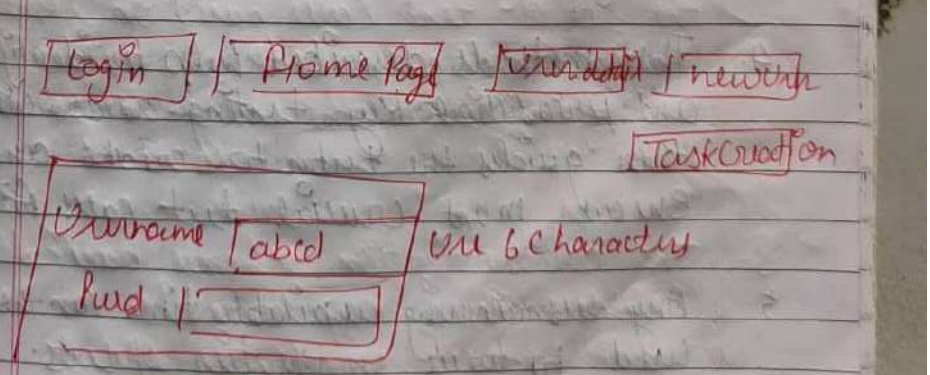
→ Test cases of functionalities which has undergone more and recent changes.

→ All integration Test cases.
→ All complex Test cases.
→ Boundary value Test cases.
→ A sample of successful test cases.
→ A sample of failure test cases.

Smoke Testing → It is a kind of S/w testing performed after S/w build to ascertain that the critical functionalities of the program are working fine. It is executed "before" any detailed functional or regression tests are executed on the S/w build. The purpose is to reject a badly broken appln so that the QA team does not waste time installing and testing the S/w appln.

The test cases chosen to cover the most imp functionality or component of the system. The objective is not to perform exhaustive testing but to verify that the critical functionalities of the system are working fine.

Sanity testing → Sanity testing is a kind of S/w testing performed after receiving a software build, with minor changes in code or functionality to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.



Reducing the number of test cases →

1- For most projects it is nearly impossible to execute all of these tests. Although many project schedules are strapped due to tight time constraints, many other reasons exist that necessitate reducing the number of test cases, for eg-

a- Imminent ship date.
b- Impossibly large number of test cases.
c- Limited staffing resources.
d- Limited access to test equipment.

2. There are several techniques that reduce the number of test cases.

3. Although no one method will identify the best test cases, the intent is to categorize the tests on a scale from most important down to least important.

4. The methods described help to identify and prioritize key testing areas, which then guides the tester to create and execute most crucial test cases first.

5. Any remainning available time can then be used to focus on the next tier of tests.

6. Failure to assess the appl^n potential problem often results in testing the least critical features.

7. Four schemes that focus on prioritizing the existing set of test cases.
a. Priority category scheme.
b. Risk analysis.
c. Interviewing to identify problem areas.
d. Combination schemes.

Code coverage prioritization technique →

1. We consider a program P with its modified program P_1 and its test suit T created to test P.
2. When we modify P to P_1 we would like to execute modified portion of the source code and portion affected by the modification to see the correctness of modification.
3. We neither have time nor resources to execute all test cases of T.
4. Our objective is to reduce the size of T to T_1 using some selection criteria, which may help us to execute the modified portion of the source code and the portion affected by modification.

5. A code coverage based technique has been developed which is based on version specific test case prioritization and select T_1 from T which is a subset of T.
6. The technique also prioritizes test cases of T_1 and recommends case of high priority test cases first then low priority test cases in descending order till time and resources are available or a reasonable level of confidence is achieved.

Prioritization Guidelines →

1. Test case prioritization is a compromise, the tester selects the situations to execute at the expense of those to omit. The risk is that some application features will not be tested.

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

3. The prioritization schemes basically address three key concepts:
(a) What features must be tested?
(b) What are the consequences if some features are not tested?

4. The main guide for selecting tests is to assess risk: what price are the project authority and the company willing to pay if a test is not executed?

5. The main guide for selecting tests is to assess risk: what price are the project authority and the company willing to pay if a test is not executed?

6. If the excluded test had been executed it might have found a problem prior to product delivery.

Priority Category Scheme →

1. The easiest scheme for categorizing tests is to assign a priority code directly to each test description.
2. Although this approach may appear to be arbitrary, it is based on the comparison "Is test x more important than test y?"
3. The test descriptions can vary in form, such as a test outline, a spreadsheet, a list of tests, a test document's table of contents, or actual test case descriptions.

4. The tester can conduct this exercise himself or as a group exercise with input from the developers, managers and customer representatives.

5. Following three level priority categorization scheme.

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

6. Assigning priority codes is as simple as writing number adjacent to each test description.

Risk Analysis and Risk matrix:

Risk Analysis →

1. ~~Where the~~ It's a well defined process that prioritizes modules for testing. Risk contains two components

(a) Probability of occurrence
(b) Severity of impact of the problem.

Probability of occurance

→ Security of impact.
→ Risk exposure.

Problem Id	Problem description	Probability	Security	Risk exposure
A	Lossy Power	1	10	10
B	Corrupted file	2	1	2
C	Unauthorized user access	6	8	48
D	Database not synchronized	3	5	15
E	Unclear User documentation	9	1	9
F	Slow throughput	5	3	15

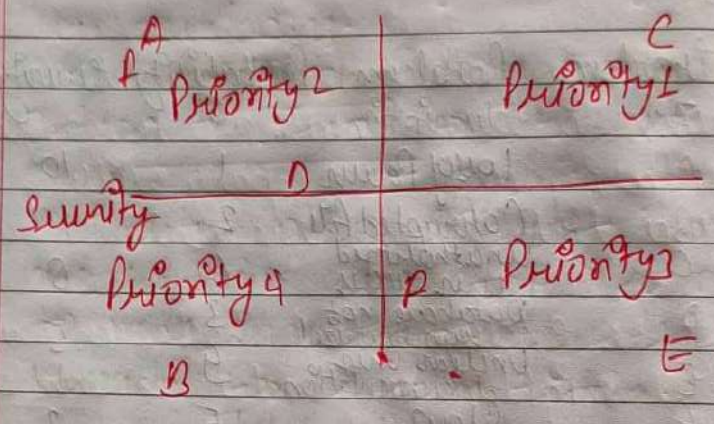
Risk matrix →

A risk matrix allows the tester to evaluate and rank problems by giving more weight to probability or security as necessary.
Risk matrix contains four quadrants with each quadrant representing the priority class as defined.
Priority 1 - high security & high Probability

Priority 2 - High security low Probability

Priority 3 - low urgency high probability

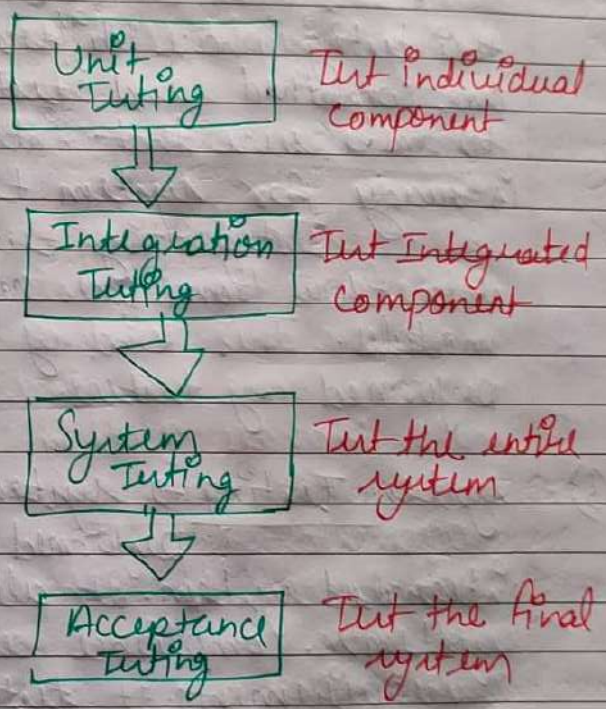
Priority 4 - low urgency low probability



Unit-4

Levels of Testing :-

A level of ST is a process where every unit or component of a software system is tested. The main goal of system testing is to evaluate the system's compliance with the specified needs.



1) **Unit Testing** → A unit is a smallest testable portion of system or application which can be compiled, linked, loaded and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

2) Integration Testing → Integration means combining. For eg - In this testing phase, different S/w modules are combined and tested as a group to make sure that integrated system is ready for system testing.

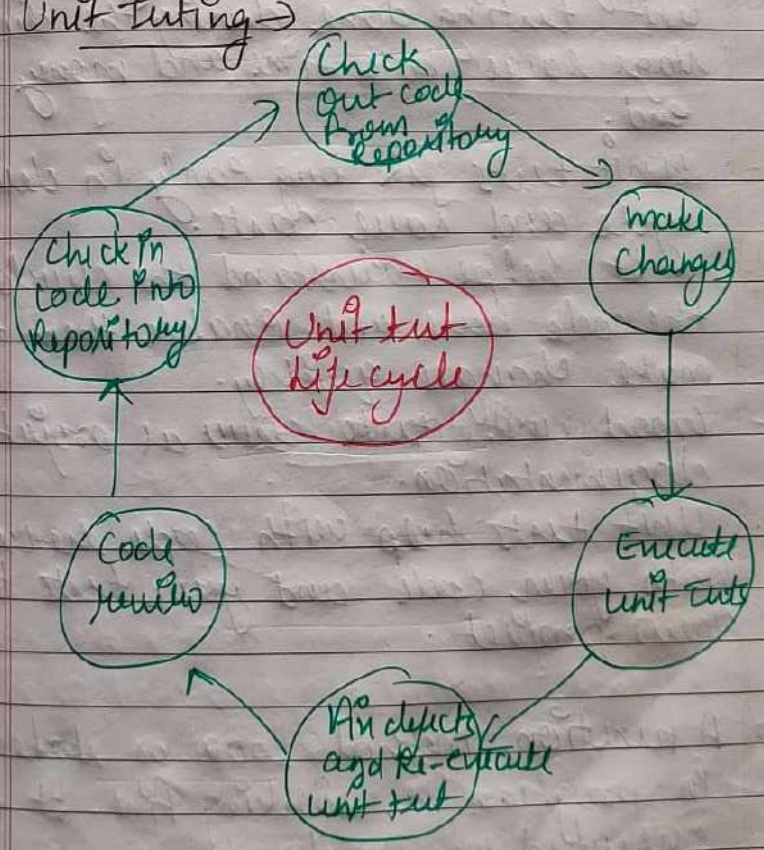
Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) System Testing → system testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluated both functional and non-functional need for the testing.

4) Acceptance Testing → Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However other stakeholders can be involved in this process.

Unit Testing →



Unit testing is defined as a type of s/w testing where individual units of a software are tested.

unit testing of a s/w appl^n is done during the development of an application. The objective of unit testing is to isolate a section of code and verify its correctness.

Why unit Testing?

Sometimes software developers attempt to save time by doing minimal unit testing. Proper unit testing done during the development stage saves both time and money in the end.

1- unit tests fix bug early in development cycle and save costs.
2- It helps understand the developers the code base and enable them to make changes quickly.
3- Good unit tests serve as project documentation.
4. Unit tests help with code re-use. Migrate both your code and your tests to your new project.

Eg-> A function, method, loop or statement in program is working fine. It is executed by Developer.

Integration Testing →

It is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction b/w integrated units.

A module in general is designed by an individual s/w developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the s/w modules work in unity.

Eg → Integration Test case differs from other test cases in the sense it focuses mainly on the interfaces & flow of data/info b/w the modules. Here priority is to be given for the integrating links rather than the unit functions which are already tested.

Appln has 3 modules say 'LoginPage' 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here donot concentrate much on the Login Page testing as its already been done in unit testing.

Test case ID	1
Test case objective	Check the interface link b/w the Login and mailbox module.
Test case Description	Enter login credentials and click on the login button.
Expected Result	To be directed to the mail Box.

Approaches of Integration Testing →

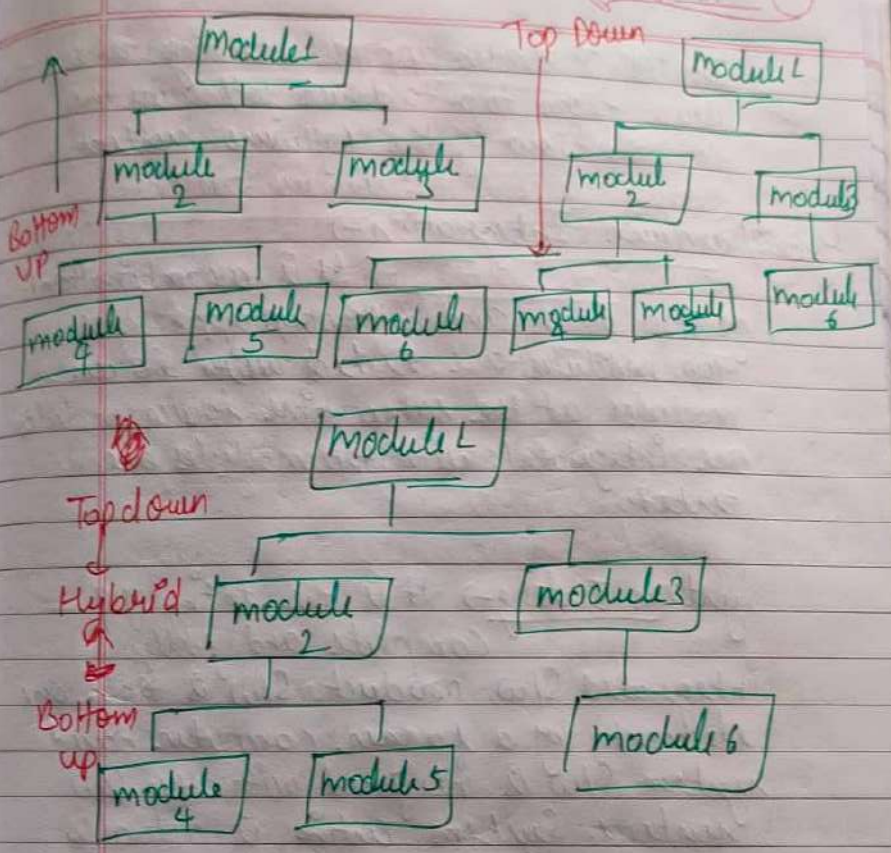
Big Bang Approach →

Here all Component are integrated together at once and then tested.

Incremental Approach →

In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning.

→ Bottom up
→ Top Down



Bottom up Integration → In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing.

Top-down Integration →

In top to down approach, testing takes place from top to down following the control flow of the software system.

Hybrid/Sandwich Integration → It is a combination of Top down and bottom up approaches. Top modules are tested with lower modules at the same time lower modules are integrated with top modules are tested.

System Testing → It is the testing of a complete and fully integrated S/W product. S/W is only one element of a larger computer based system. S/W is interfaced with other S/W systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

→ Black BOX Testing
→ White Box Testing

Types of System Testing →

1) Usability Testing → It mainly focuses on the user's ease to use the

appln, flexibility in handling controls and ability of the system to meet its objectives.

2) Load Testing → Load Testing is necessary to know that a S/W solution will perform under real-life loads.

3) Regression Testing → It involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.

4) Recovery Testing → It is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.

5) Migration Testing → Migration testing is done to ensure that the S/W can be moved from older S/m infrastructures to current system infrastructures w/o any issues.

6) Functional Testing → Also known as Functional completeness testing
Functional Testing involves trying to think of any possible missing functions.

7) Hardware Testing → IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions b/w the h/w and s/w during system testing.

Assignment - 3

Q What is regression testing? Differentiate b/w regression and development testing?

Q What are the categories to evaluate regression test selection techniques? Why do we use such categorization?

Q What is the difference b/w equivalence partitioning and boundary value analysis methods?

Q Why do bugs occur in software?

Q Explain the various steps of the regression testing process? Which step is the most important and why?

Acceptance criteria used in Acceptance Testing →

1) Acceptance criteria - product acceptance →

(a) During the requirement phase, each requirement is associated with acceptance criteria.
(b) It is possible that one or more requirements ~~the acceptance~~ may be mapped to form acceptance criteria.
(c) whenever there are changes to requirements the acceptance criteria are accordingly modified & maintained.
(d) Acceptance testing is not meant for executing test cases that have not been executed before.

2) Acceptance criteria - procedure acceptance →

(a) Acceptance criteria can be defined based on the procedures followed for delivery. An example of procedure acceptance could be documentation and release media.

(b) The nature of acceptance criteria are as follows -

i) User, administration and troubleshooting documentation should be part of the release.
ii) Along with binary code, the source code of the product with build scripts to be delivered in a CD.
iii) A minimum of 20 employees are trained on the product usage prior to deployment.

3) Acceptance criteria - Service level agreement

a) Service level agreement can become part of acceptance criteria.
b) They are generally part of a contract signed by the customer and product-organization.
c) For example, time limits to resolve those defects can be mentioned part of SLA.

Selecting Test cases for Acceptance Testing →

1- End-to-end functionality verification →
Test cases that include the end-to-end functionality of the product are taken up for acceptance testing.

2) Domain Test → Load testing is necessary to know that a software solution ~~Domain~~ Since Acceptance tests focus on business scenario, the product

domain tests are included. Test cases that reflect business domain knowledge are included.

3) User Scenario tests → Acceptance tests reflect the real-life user scenario verification. So test cases that portray them are included.

4) Basic Sanity tests → Tests that verify the basic existing behaviour of the product are included.

5) New functionality → When the product undergoes modifications or changes, the acceptance test cases focus on verifying the new features.

6) A few non-functional tests → Some non-functional tests are included and executed as part of acceptance testing to double-check that the non-functional aspects of the product meet the expectation.

7) Acceptance test data → Test cases that make use of customer real-life data are included for acceptance testing.

Executing Acceptance tests →

1. Sometimes the customers themselves do the acceptance tests. In such cases, the job of the product organization is to assist the customers in acceptance testing and resolve the issues that come out of it.

2. If the acceptance testing is done by the product organization, forming the acceptance test team becomes an important activity.

3. An acceptance test team usually comprises members who are involved in the day to day activities of the product usage or are familiar with such scenarios.

4. The product management, support and consulting team, who have good knowledge

5. The no of test team members needed to perform acceptance testing is very less, as the scope and effort involved in acceptance testing is not much when compared to other phases of testing.

Acceptance Testing →

Acceptance testing is the level in the software testing process which decides whether a product is given the green light or not. The aim of this type of testing is to evaluate whether the system complies with the end-user requirements and if it is ready for deployment.
The testing team will utilize a variety of methods, such as pre-written scenarios and test cases to test the s/w and use the results obtained from these tools to find ways in which the system can be improved.
The scope of acceptance testing ranges from simply finding spelling mistakes and cosmetic errors to uncover bugs that could cause a major error in the appl^n.

Debugging Process →

Debugging is the process of finding a bug in the software. It refers to identifying, analyzing and removing errors. This activity begins after the s/w fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task becoz errors

needs to be resolved at all stages of debugging.

Debugging Process →

→ Problem identification and report preparation
→ Assigning the report to software engineer to the defect to verify that it is genuine.
→ Defect analysis using modeling documentations finding and testing candidates etc.
→ Defect Resolution by making required changes to the system.
→ Validation of corrections.

Debugging Strategy →

→ Study the system for the larger duration in order to understand the system. It helps debugger to construct different representations of system to be debugging depends on the need.

→ Backwards analysis of the problem which involves tracing the program backward from the location of failure message in order to identify the region of faulty code. A detailed study of the region is conducting to find the cause of defects.

→ forward analysis of the program involves tracing the program forwards using breakpoints or point statements at different points in the program and studying the results.

→ Using the past experience of the s/w debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

Difference b/w Testing and Debugging →

Testing	Debugging
1. Testing always starts with known condn, uses predefined methods and has predictable outcomes too.	Debugging starts from possibly unknown initial condn and its end cannot be predicted, apart from statistically
2. Testing can and should definitely be planned designed and scheduled.	The procedures for, and period of debugging cannot be so constrained.
3. It proves a programmers failure.	It is the programmer's vindication.
4. It is demonstration of error correctness.	It is always treated as a deductive process.

5- Testing as executed should strive to be predictable, dull, constrained, rigid, and inhuman.	Debugging demands intuition leaps, conjecture, experimentation and some freedom also.
6- Much of the testing can be done w/o design knowledge.	Debugging is impossible w/o detailed design knowledge.

Testing Techniques →

1) People-based techniques →

a) User testing
b) Alpha testing
c) Beta Testing
d) Bug Bashes
e) Subject-matter expert testing
f) Paired testing

2) Coverage-Based techniques →

a) function testing
b) Feature or function integration testing
c) menu tour
d) Domain testing
e) equivalence class Analysis
f) Boundary testing
g) Best representative testing
h) map and test all the ways to edit a field.
i) Logic testing
j) State based testing
k) Path testing
l) Specification based testing
m) Requirements-based testing
n) combination Testing.

appln handles such exceptions and throws proper error.

2) Valid set of test data → refers to the valid or supported files by the appln. There should give the expected o/p when given as i/p.

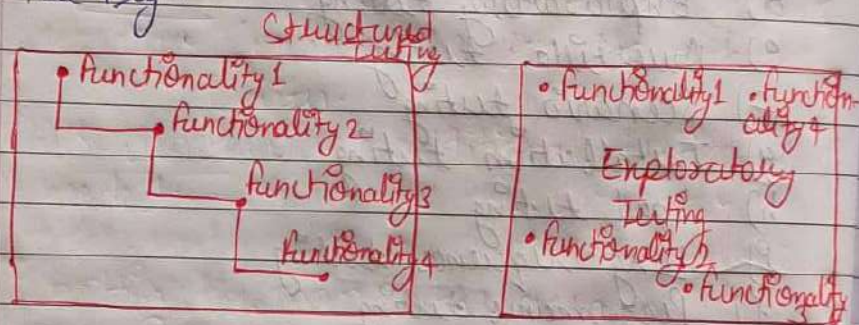
3) Invalid set of Test data → refers to all the unsupported file formats in order to see that appln handles all of them properly w/o breaking and throws user with proper error message.

4) Huge test data → Cannot be made at the time of execution and should be prepared while making your test environment ready.

5) Test data to check all the boundary condns → Includes data which has all possible combinations of boundary values. For eg- If a text box can have no 2-20 then i/p 2 and 20 values. It include all those values which are just enough to handle by the appln.

Exploratory Testing →

Exploratory Testing is all about discovery, investigation and learning. It emphasizes personal freedom and responsibility of the individual tester. It is defined as a type of testing where test cases are not created in advance but testers check system on the fly.



functionality are checked in a structured manner

functionality are checked in a ad-hoc manner.

→ It is not random testing but it is ad-hoc testing with a purpose of find bugs.

→ It is structured and rigorous.

→ It is thinking structured as compared to the procedural structure of scripted testing. This structure comes from charter, time boxing etc.

→ It is highly teachable and manageable.

→ It is not a technique but is an approach.

How to do Exploratory Testing →

1. Create a Bug Taxonomy →

→ Categorize common types of faults found in the past projects.

→ Analyze the root cause analysis of the problems or faults.

→ Find the risk and develop ideas to test the appl^n.

2. Test charter →

→ Test charter should suggests-
→ what to test?
→ how it can be tested?
→ what needs to be looked?

→ Test ideas are the starting point of exploration testing.

→ Test charter helps determine how the end user could use the system.

3- Time Box → This method includes a pair of testers working together not less than 90 minutes.

4- Review Results → Evaluation of the defects. Learning from the testing. Analysis of coverage areas.

5- Debriefing → Compilation of the o/p results. Compare the results with the charter.
Check whether any additional testing is needed.

Eg Getting the map of the area.
Travelling in some random direction to figure out the place.
Calling up and asking a friend for the route.
Asking for directions by going to a nearby gas station.

Benefits →

1- Exploratory testing takes less preparation.

2. Critical defects are found very quickly.

3. The testers can use reasoning based approach on the results of previous results to guide their future testing on the fly.

Demerits →

1. Testing depends on testers experience, skill and knowledge.

2. Require time to learn the appl^n. ~~Testers~~

3. ~~Not~~ few issues cannot be catch in this type of testing.

Exploratory testing technique :-

1) Guesses → Guesses are used to find the part of the program that is likely to have more errors.

2) Architecture diagrams, use cases →

Architecture diagram depict the interactions and relationships b/w different components and modules.

3) Study of Past defects → Defects reports of the previous releases act as a pointer to explore an area of the product further.

4) Error Handling → Error handling is a portion of the code which prints appropriate messages or actions in case of failures. Error handling provides a message or corrective action in failure situation.

5) Discussions → Exploration may be planned based on the understanding of the system during project discussions or meeting

6) Questionnaires → Question like "what, when, How and why" can provide leads to explore areas in the product.

Test Data → Test data is a commonly used term in a tester's day to day life. While executing test cases, he needs some data to input in order to get the expected output. Sometimes to load the appln with data or to check the break point of the appln, huge data is required. This data can be valid or invalid.

Test data is data which is required in executing the test cases properly and to verify the expected o/p in any s/w appln under test.

Eg- You want to test mobile software appln. mobile has many diff appln so to test them you need diff input data such as photos of different formats, music files supported and unsupported formats, video files, contacts files, different emails etc. Without this test data tester cannot proceed with testing and also won't get desired output.

Types of Test data →

1) Blank Files → or no data refers to those files which do not have any data i.e no input is given to the appln and this verifies that

3) Problems - based techniques →

a) Risk - based testing

4) Activity based techniques →

a) Regression testing
b) Scripted testing
c) Smoke testing
d) Exploratory testing
e) Guerrilla testing
f) Scenario testing
g) Installation testing
h) Load testing
i) Long sequence testing
j) Performance testing

5) Evaluation based techniques →

a) Self - verifying data
b) Comparison with saved results
c) Comparison with the specification or other authoritative document.
d) Heuristic consistency
e) Oracle - based testing

Create Test Data →

1. Always make sure that Test data files are not corrupted.

2. Test data should be updated on a regular basis. This will give a clear picture of expected o/p.

3. Test data should be created before Test Cases execution to save time and meet deadline.

4. Test data should have invalid inputs to test negative scenarios.

5. Testers can take developer's help to create test data.

6. It is always a better practice to include all possible combinations of supported and unsupported formats in test data to ensure that test coverage is max^m.

A	B	C	D	E	F	G
First Name	Last Name	Gender	Birthday	e-mail id	Add	Phone
Anil	Jain	~~24/3/90~~	24/3/90	xyz@gmail.com		234
XYZ	moon	K	40/13/00		USA	

Approaches of test data Generation →

1) Random Test data generators →

Random test data generation is probably the simplest method for generation of test data. It can be used to generate input for any type of program. Thus, to generate test data we can randomly generate a bit stream and let it represent the data type needed. However random test data generation does not generate quality test data as it does not perform well in terms of coverage.

2) Pathwise test data generators :-

Pathwise test data generation is considered to be one of the best approaches to test data generation. This approach does not give the generator the choice of selecting

G	H	I
Phone no	Pwd	Confirm pwd
1234567890	xxxx	xxxxx

b/w multiple paths but just gives it one specific path for it to work on. Hence it is named as pathwise test data generator. Thus, except for the fact that this method uses specific paths, it is quite similar to goal-oriented test data generation.

3) Goal-oriented test data generators →
The goal oriented approach provides guidance towards a certain set of paths. The test data generators in this approach generate an input for any path instead of the usual approach of generating I/P from the entry to the exit of a block of code.

Two methods follow this technique-
i) The chaining Approach
ii) Assertion oriented Approach.

4) Intelligent test data generators →
Intelligent test data generators depends on sophisticated analysis of the code to guide the search of the test data. Its are essentially utilizing one of the test data generation method coupled with the detailed analysis of the code.

Benefits of Automated test data Generation:

1) Faster feedback :- If the number of test data items is large, automated test data generation may be substantially faster than creating the same test data by hand.

2) Accurate :- Since test data generation is a tedious task, you are likely to make mistakes. Test data generated by a tool can be much more accurate.

3) Can use built-in algorithms →
The test data generator may implement algorithm to generate correctly formatted special data like postal codes and e-mail addresses.
Therefore you need not look up the algorithms for generating the special test data yourself.

4) Create both valid and invalid test data:

Just as we can create valid test data, we can also create invalid test data by specifying incorrect constraints.

Further by specifying the quantity of the valid and invalid data, we can control the percentage distribution b/w valid & invalid data.