UNIT - III

INTRODUCTION OF CODING

The goal of the coding or programming phase is to translate the design of the system produce during the design phase into code in a given programming language, which can be executed by a computer and that performs the computation specified by the design.
For Example:-a given design the aim is to implement the design in the best possible manner.
During implementation, it should be kept in mind that the program should not be constructed so that they are easy to write, but so that they are easy to read and understands.
A Program is read a lot more often and by a lot more people during the later phases. Often, making a program more readable will require extra work by the programmers.
They are many different criteria for judging a program, including readability, size of the program, execution time and required memory.
Having readability and understandability as a clear objective of the coding activity can itself help in producing software that is more maintainable.
If readability is an objective of the coding activity.

QUE : EXPLAIN TYPES OF PROGRAMMING PRACTICE

The primary goal of coding phase is to translate the given design into source code in a given programming language, so that the code is simple, easy to test and easy understand and modify Simplicity and clarity.
Good programming (Solid Code) is a skill that can only be acquired by practice.
However, much can be learned from the experience of others, and some general rules and guide-lines can be laid for the programmer.
Good programming (producing correct and simple programs) is a practice independent of the larger programming language, although some well-structured languages like Pascal, ADA, and module make the programmer's job simpler.

There are four types of programming practices:-

1) Top Down Approach & Bottom up Approach.
2) Structured Programming.
3) Information Hiding.
4) Programming style Verification Metrics.

Top Down Approach & Bottom Up Approach

The question at coding time is given the hierarchy of modules produced by design, in what order should the modules be built-starting from the top level or starting from the bottom level?

In top-down implementation, the implementation starts from the top of the hierarchy and proceeds to the lower levels. First the main module is implemented. Then it's subordinates(or lower level modules) are implemented and their subordinates and so on.

In a bottom-up implementation, the process is reverse. The developments starts with implementing the modules at the bottom of the hierarchy and proceeds through the higher levels until it reaches the top.

All the modules are to be developed and then put together to from the system for testing purposes as is done for small System; it is immaterial which module is coded first.
When modules have to be tested separately, top-down and bottom-up lead to top-down and bottom-up approaches to testing.

QUE : EXPLAIN STRUCTURED PROGRAMMING

In unstructured programming language we can use GO TO statement in programs.
For example : assembly language programming

There are three programming constructs: sequence, selection, and iteration-were sufficient to express any programming logic.

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

Structured programs avoid unstructured control flows by restricting the use of GO TO statements.
Structured programming is facilitated, if the programming language being used supports single-entry, single-exit program constructs such as if-then-else, do=while, etc.
Thus, an important feature of structured programs is the design of good control structures.
When we use GO TO statements in a program then it is much more complex.

A structured program should be modular. A modular program is one which is decomposed into a set of modules1 such that the modules should have low interdependency among each other.

advantages of writing structured programs compared to the unstructured ones

1) less error-prone, structured programs are normally more readable, easier to maintain.
2) require less effort to develop structured programs.
3) structured programming feature are usually permitted in certain specific programming situations, such as exception handling, etc.
4) These programming languages facilitated writing modular programs.
5) Programs having good control structures.

Example of structure programming languages such as PASCAL, MODULA, C, etc.

INFORMATION HIDING

Write short note on Information Hiding and programming style.

The private information of module, which is not access or use in the rest of the modules it is known as information hiding.

The principles of the information hiding. The information capture in the data structures to be hidden from the rest of the system, and only the access functions on the data structures that represent the operations performed on the information should be visible.
When the information is captured in the data structures and then on the data structures that represents some information hiding can reduce the coupling between modules and make the system more maintainable.
If data structures are directly used in modules, then all modules that use some data structures are coupled with each other and if change is made in one of them, the effect on all other modules needs to be evaluated.
Information hiding is also an effective tool for managing the complexity of developing software. As we have seen whenever possible, problem partitioning must be used so that concern can be separated and different part solved separately.
By using information hiding, we have the separated the concern of managing the data from the concern of using the data to produce some desired result.
Another from of information hiding is to let a module see only data items needed by it. The other data items should be "hidden" from such modules and the modules should not be allowed to access these data items. Thus, each module is given access to data items on a "need-to-know" basis.

PROGRAMING style and VERIFICATION METRIES

Names : selecting proper module and variable name.

Control constructs : it is desirable that as much possible single-entry, single – exit Control constructs be used.

Gotos : when alternative to using gotos is more complex should the gotos be used.

Information Hiding : if it is possible we can use data hiding facility for hide information.
User defined types : Modern langu.. ges allow users to define types like the enumerated type.
Type days (mon, tue, wed, thu fri, sat, sun).

Nesting : we use nesting of control structure and conditional control

Module size : A programmer should carefully examine any routine with very few statements (say fewer than 5) or with too many statements (say more than 50).

Module interface : A module with a complex interface should be carefully examined. Such modules might not be functionality cohesive and might be implementing multiple functions.

Program layout : how the program is organized and proper indentation and blank spaces and parentheses should be used to enhance the readability of programs.

QUE : EXPLAIN VERIFICATION:-

Verification of the output of the coding phase is primarily intended for detecting errors introduced during this phase.

That is, the goal of verification of the code produced is to show that the code is consistent with the design it is supposed to implement.

It should be pointed out that by verification we do not mean proving correctness of programs, which for our purpose is only one method for program verification.

For our purpose is only one method for program verification.

Program verification methods fall into two categories.

1. Dynamic verification method
2. Static verification method.

Dynamic verification Method

In dynamic verification method the program is executed on some test data and the outputs of the program are examined to determine if there are any errors present. Hence, Dynamic techniques follow the traditional pattern of testing. And common notion of testing refers to these techniques.

Static verification Method

Static techniques, on the other hand, do not involve actual program execution on actual numeric data. Though it may involve some form of conceptual execution.

In static techniques, the program is not compiled and then executed, as in testing. Common forms of static techniques are program verification code, reading, code reviews and walkthroughs and symbolic execution.

In static techniques often the errors are detected directly, unlike dynamic techniques where only the presence of an error is detected. This aspect of static testing makes it quite attractive and economical.

It has been found that the types of errors detected by two categories of verification techniques are different. The types of errors detected by static techniques are often not found by testing or it may be more cost effective to detect these errors by static methods.

Consequently, Testing and static methods are complimentary in nature, and both should be used for reliable software.

QUE : EXPLAIN CODE READING:-

Code reading involves careful reading of the code by the programmer detect any discrepancies between the design specification and the actual implementation.

It involves determining the abstraction of the module and then comparing it with its specifications.

The process is the reverse of design. In design, we start from an abstraction and move towards more details of the program and move towards an abstract description.

The process of code reading is best done by reading the code inside out, starting with the innermost structure of the module.

First determine its abstract behavior and specify the abstraction. Then the higher level structure is considered, with the inner structure replaced by its abstraction.

QUE : EXPLAIN STATIC ANALYSIS.

Analysis of programs by methodically analyzing the program text is called static analysis.

Static analysis is usually performed mechanically by the aid of software tools. During static analysis the program itself is not executed but the program text is the input to the tools. The aim of the static analysis tools is detect errors or potential errors or to generate information about the structure of the program that can be useful for documentation or understanding of the program. Different kind of static analysis tools can be designed to perform different types of analysis.

Many compilers perform some limited static analysis. More often, tools explicitly for static analysis are used.

Static analysis can be very useful for exposing errors that may escape other techniques. As the analysis is performed with the help of software tools, static analysis is the very cost-effective way of discovering errors.

An advantage is that static analysis sometimes detects the errors themselves, not just the presence of errors. As in testing. This saves the effort of tracing the errors from the data that reveals the presence of errors.

QUE : WHAT IS TESTING ?

The aim of program testing is to identify all defects in a program. and reducing defects in a system.

OR

Testing perform very critical role for quality assurance and for ensuring the reliability of software. During the testing the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as expected.

BASIC CONCEPT AND TERMINOLOGIES OR TESTING FUNDAMENTAL:-

Testing program involves providing the program with a set of test inputs (or test cases).

The following are some commonly used terms associated with testing.

- **An error** is a mistake committed by the development team during any of the development phases. The mistake might have been committed in the requirements, design, or code. An error is also sometimes referred to as a fault, a bug, or a defect.

OR

Error refers to the difference between the actual output of software and correct output. In this interpretation, error is essentially a measure of the difference between the actual and the ideal. Error is also used to refer to human action those results in software containing a detect or fault. This definition is quite general and encompasses all the phases.

-**Fault** is a condition that causes a system to fail in performing its required function.
A fault is the basic reason for software malfunction and is synonymous with the commonly used term bag.

-**A failure** is a manifestation of an error (or defect or bug). In other words, a failure is the symptom of an error. But, mere presence of an error may not necessarily lead to a failure.

OR

A software failure occurs if the behavior of the software is different from the specified behavior. Failure may be caused due to functional or performance reasons.
A failure is produced only when there is a fault in the system.

-**A test suite** is the set of all test cases with which a given software product is tested.

VERIFICATION AND VALIDATION

Verification is the process of determining whether the output of one phase of software development conforms to that of its previous phase.

Validation is the process of determining whether a fully developed system conforms to its requirements specification.

Alternatively, we can state the difference between verification and validation in the following words

While verification is concerned with phase containment of errors, the aim of validation is to make the final product error free.

TESTING ACTIVITIES

Testing perform the following main activities:

1. Test suite design: The set of test cases using which a program is to be tested is designed using different test case design techniques.

2. Running test cases and checking the results to detect failures: Each test case is run and the results are compared with the expected results. A mismatch between the actual result and expected results indicates a failure. The test cases fails for which the system fails are noted down for later debugging.

3. Debugging: For each failure observed during the previous activity, debugging is carried out to identify the statements that are in error. In this, the failure symptoms are analyzed to locate the errors.

4. Error correction: After the error is located in the previous activity, the code is appropriately changed to correct the error.
Of all the above mentioned testing activities, debugging is possibly the most time consuming activity.

QUE : EXPLAIN ABOUT TEST CASES AND TEST CRITERIA

-A TEST CASES : is the triplet [I.S.O.], where I is the data input to the system, S is the state of the system at which the data is input, and O is the expected output of the system.

1. That are good at revealing the presence of fault is central to successful testing. The reason for this is that if there is a fault in a program, the program can still provide the expected behavior for many inputs. Only for the set inputs that exercises the fault in the program will the output of the program deviate from excepted behavior.

2.We would like to determine a set of test cases such that successful execution of all of them implies that are no errors in the program.

3.While selecting test cases the primary objective is to ensure that if there is an error or fault in the program, it is exercised by one of the test cases. An ideal test case set is one that succeeds (meaning that its execution reveals no errors) only if there are no errors in the program. One possible ideal set of test causes is one that includes all the possible inputs to the program. This is often called **exhaustive testing.** However, in feasible, as even for small programs. The number of elements input domain can be extremely large.

4. Realistic goal for testing is to se' ct a set of test cases that is close to ideal.

5. How should we select our test cases?

TEST CRITERIA:-

1) Test selection criteria can be used. For a given program P and its specification S. A test selection creation specifies the condition that must be satisfied by a set of test cases T. the creation become a basis for case selection.

2) Once during testing set of test cases T satisfies this criterion for a program P if the execution of P with T ensures that each statement in P is executed at least once.

3) Specifying a criterion for evaluating a set of test cases and generating a set of test cases that satisfy a given criterion.

4) There are two fundamental properties for testing criterion. 1) Reliability 2)Validity.

QUE : EXPLAIN TEST ORACLE:-

To test any program, we need to have a description of its expected behavior and the method of determining whether the observed behavior conforms to the expected behavior for this we need a test oracle.

1. 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. Conceptually we can consider testing a process in which in test cases are given to the test oracle and the program under testing. The output of the two is then compared to determine if the program behaved correctly for the test cases.

2. Test oracles are necessary for testing. ideally, we would like an automated oracle, which gives a correct answer. Often the oracle is human being, who mostly computed by hand what the output of the program should be. As it is often extremely difficult to determine whether the behavior conforms to the excepted behavior, out "human oracle" may make mistakes. As a result when there is a discrepancy between the results of the program and the oracle. we have to verify the result produced by the oracle, before declaring that there is a fault in the program this is one of the reasons testing is so cumbersome and expensive.

3. The human oracles generally use the specification of the program to decide what the 'correct' behavior of the program should be. To help the oracle determine the correct behavior, it is important that the behavior of the system or component be us ambiguously specified and that the specification itself is error free.

QUE : EXPLAIN PSYCHOLOGY OF TESTING :

Selecting test cases is creative activity that relies on the agility of the tester. Due to this reason psychology of the person performing he testing becomes important. The aim of testing is often to demonstrate that a program works by showing that it has no errors.

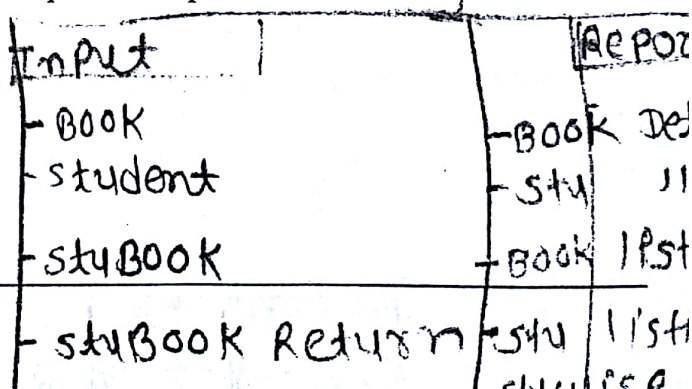
The basic purpose of testing is to detect errors that may be present in the p ogram.

QUE : EXPLAIN INTEGRATION TESTING WITH TOP DOWN AND BOTTOM UP APPROACHES.

The objective of integration testing is to check whether the different modules of a program interface with each other properly.
Any one of the following approach can be used to develop the test plan.

L.S

> Big bang approach
> Top down approach
> Bottom up approach
> Mixed Approch(sandwiched)



In the following, we provide an overview of these approaches to integration testing

BOTTOM-UP INTEGRATION TESTING:

Large software products are often made up of several sub systems.

A sub-system might consist of many modules which communicate among each other through well-defined interfaces. In bottom-up integration testing, thus the modules for the each sub-system are integrated. Thus, the subsystems can be integrated separately and independently.

The bottom up approaches starts from the bottom of the hierarchy. First the modules at the very bottom, which have no subordinates are tested the modules are combined with higher level modules for testing. A driver is needed to setup the appropriate environment and invokes the module under testing with the different set of test cases.

Advantages : 1) The principal advantage of bottom-up integration testing is that several disjoint sub systems can be tested simultaneously.

2) bottom up testing is that the low level modules get tested thoroughly, since they are exercised in each integration step.

Disadvantage of bottom-up testing is the complexity that occur when the system is made up of large number of small subsystems that are at the same level.

TOP-DOWN INTEGRATIONTESTING:

Top-down integration testing starts with the root module ands one or two sub ordinate modules in the system. After the top level skeleton has been tested, the modules that are at the immediately lower layer of the skeleton are combined with it and tested.

Advantage of top-down integration testing is that it require writing only stubs, and stubs are simpler to write compared to drivers.

Disadvantage of the top down integration testing approach is that in the absence of lower level routines, many it may become difficult to exercise the top level routines in the desired manner since the lower level routines perform several low level functions such as the I/O operations.

QUE : EXPLAIN DRIVER AND STUB MODULES:-

In order to taste a single module, we need a complete environment to provide all relevant code that is necessary for execution of the module. That is besides the module under the test. The following are the needed to the test module.

1. The processor belonging to the other module that the module under the taste calls.
2. Non local data structure that the module access
3. A processor to the call the function of the module under taste with appropriate parameter.

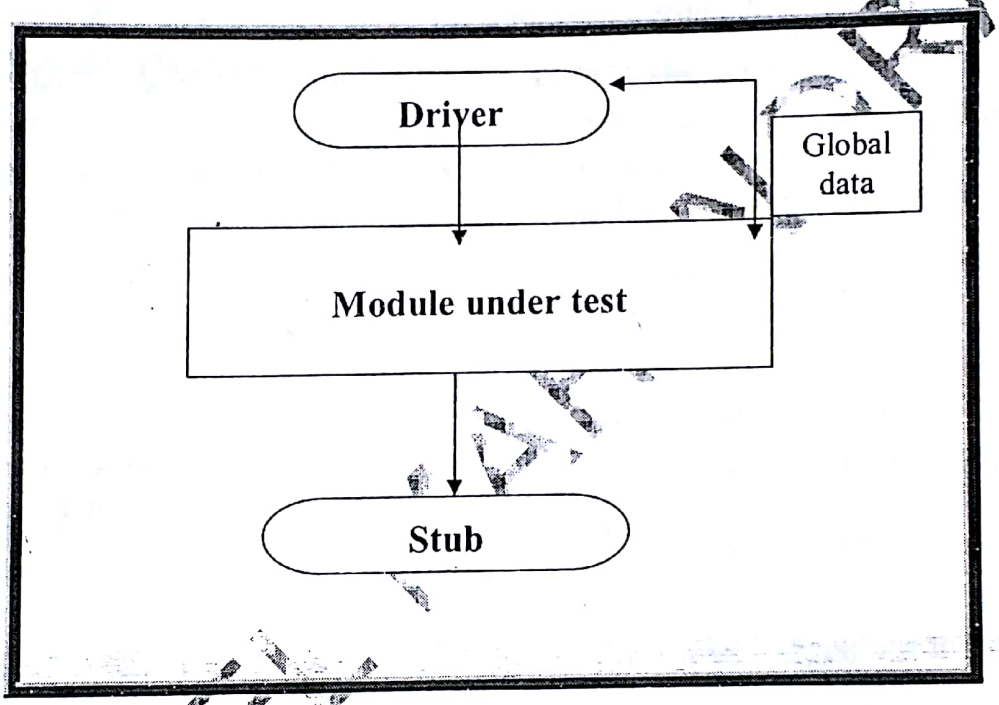
Stub

The role of stub and driver module is pictorially show in following figure. A stub processor is a dummy processor that has the same I/O parameter as the given processor, but has a highly simplified behavior for ex. A stub processor may produce the expected behavior using simple table look up mechanism.

Driver

A driver module should contain the non-local data structure access by the module under the taste. Additionally, it should also have a code to the call the different function of the module under the taste with appropriate parameter values for tasting.

Figure



QUE : THERE ARE TWO BASIC APPROACHES OF TESTING.

1. Functional testing 2. Structure Testing Explain black box testing

1. Functional Testing (BLACK BOX testing). Explain functional testing.

In functional testing the structure of the program is not considered. Test cases are decided solely on the basis of the requirements or specifications of the program or module. And the internals of the module or the program are not considered for selection of test cases.

Functional is often called "Black Box Testing"

Software testing techniques where by the tester does not know the internal working of the item begins tested. For example, in a black box test on software design the tester only know the

inputs and what the expected outcomes should be and not how the program arrives at those outputs.

The tester does not ever examine the programming code and does not any further knowledge of the program other that its specifications.

The Advantages of this type of testing includes:

- The test is unbiased the because the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

The disadvantages of this type of testing include:

- The test can be redundant if the software designer has already run a test case.
- The test cases are difficult to design.
- Testing every possible input stream is unreliable because it would take a inordinate amount of time therefore many program paths will go untested.

In black box testing, test cases are designed on the basis of input/output and values and no knowledge of design or code is required.

There are two main approaches to design black-box test cases are given below.

1. Equivalence class partitioning
2. Boundary value analysis
3. Cause effect graphics

1) Equivalence Class Partitioning

Because we cannot do exhaustive testing. The next natural approach is to divide the domain of all the inputs into a set of equivalence classes, so that if any test in an equivalence class succeeds, then every test in that ass will succeed.

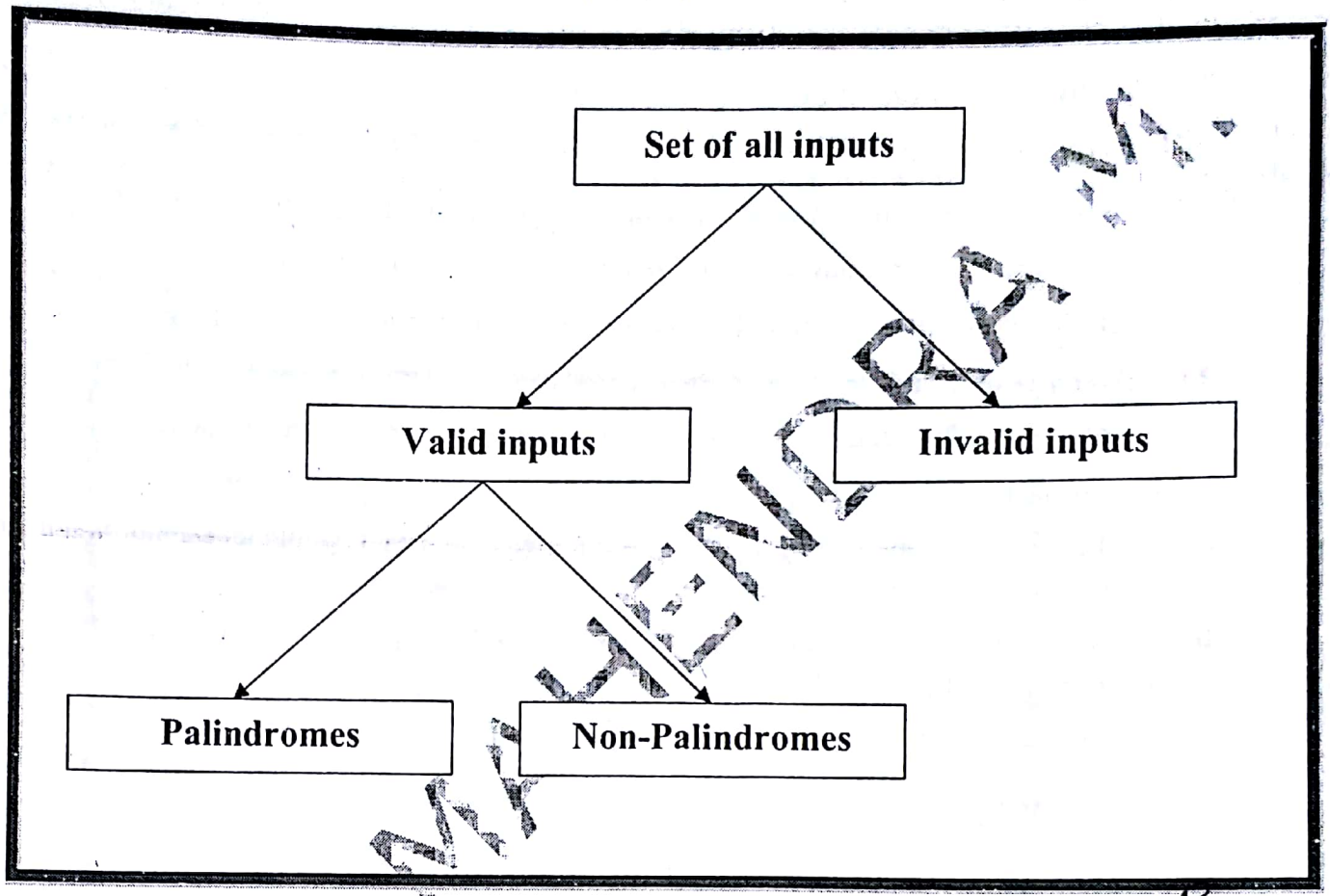
In this approach, the domain of i nput values to a program partitioned into asset of equivalence classes. The partitioning is done such that the behavior of the program is similar to every input data belonging to the same equivalence class.

The main idea behind defining the equivalence classes is that testing the code with one value belonging to an equivalence class is as good as testing the software with any other values belonging to that equivalence class. Equivalence classes for a software can be designed by examining both the input and output data. The following are some guidelines for designing the equivalence classes:

- If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes need to be defined.

- some domain the one Equivalence Class for the valid input value and another Equivalence Class for the invalid input value should be define

for example, if the valid Equivalence Class are {A,B,C}
then the invalid Equivalence Class is U - {A,B,C}, where U is the universal class.



BOUNDARY VALUE ANALYSIS :

It has been observed that programs that work correctly for a set of values in an equivalence class fails on some special values. These values lie often on the boundary of the equivalence class.

In boundary value analysis, we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes. Boundary values for each equivalence classes of the output should be covered.

For example, Programmers may improperly use <, instead of <= ,or conversely <= instead of <. Boundary value analysis leads to selection of test cases at the boundaries of different equivalence classes.

CAUSE-EFFECT GRAPHICS:

One weakness with the equivalence class partitioning and boundary value methods is that they consider each input separately. Cause effect graphing is a technique that aids in selecting combination of input conditions in a systematic ways, Such that the number of test cases does

not become unmanageably large. The technique starts with identifying causes and effects of the system under testing. For example, an input condition can be "file is empty" which can be set to true by having empty input like and false by a non-empty file.

STRUCTURAL TESTING WHITE BOX TESTING OR Glass Box Testing

In structure approaches, test cases are generated based on the actual code of the program or module to be tested.

This structure approach is some time called "Glass Box Testing".

The basis for deciding test cases in functional testing is the requirements or specifications of the system or module. For the entire system, the test cases are designed from the requirements specification document for the system, for modules created during design, test cases for functional testing are decide from the module specifications produced during the design.

The most obvious functional testing procedure is exhaustive testing, which as we have started is impractical.

Structural testing on the other hand is concerned with testing the implementation of the program. The intent of the structural testing is not to exercises all the different input or output conditions but to exercise the different programming structure and data structure used in the program.

To test the structure of the program, structure testing aims to achieve test cases that will force the designed coverage of different structure. Various criteria have been proposed for this. The criteria for functional testing.

A white box testing strategy can either we coverage based or fault based

1. Fault based : A fault based testing strategy targets to detect certain types of faults.

2. Coverage based : A popular example of coverage based testing strategy are statement coverage, branch coverage and the path coverage based testing.

1) STATEMENT COVERAGE

The simplest coverage criteria is statement coverage, which requires that each statement of the program to be executed at least once during testing. This coverage criteria is not very strong and can leave errors undetected.

It is difficult to determine whether it leads to a failure due to some illegal memory access. Wrong result computation due to improper arithmetic operation, etc.

2) BRACH COVERAGE:

Requires that each edge in the control flow graph be traversed at least once during testing.

In other words, Brach coverage requires that each decision in the program be evaluated to true and false values at least once during testing. Testing based on branch coverage is often called branch testing.

3) PATH COVERAGE :

Path coverage requires that all possible paths in the control how graph be executed during testing. This is also known as all paths criterion and the testing. The difficulty with this criterion is that programs that contain loops can have an infinite number of possible paths.

Advantages of white Box testing:

1) As the knowledge of internal coding structure is prerequisite, it becomes very to find out which type of input data can help in testing the application effectively.

2) The other advantage of white box testing is that is helps in optimizing the code.

3) It helps in removing the extra lines of code, which can bring in hidden defects.

Disadvantages of white box testing:

1) As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type testing, which increases the cost.

2) And it is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.

CONTROL FLOW BASED TESTING:

The control flow graph of a program is considered and coverage of various aspect of the graph are specified as criteria, let us precisely define a control flow graph.

Let the control flow graph of the program P be G. A node in this graph, represents a block of statements that is always executed together,

i.e. The first statement is executed, all other statements are also executed. And edge represent a possible transfer of control after executing a last statement of block represented by node to the first statement of the block represented by node.

A path is finite sequence of nodes (n1, n2……nk) k>1. A complete path is a path whose first node is the start node and the last node us and exit node. The path executed during testing includes all the nodes in the graph.

DATA FLOW BASED TESTING:

In data flow based testing, besides the control flows, about where the variables are defined and where the definitions are used is also used to specify the test cases. The basic idea behind data flow based testing is to make sure that during testing. The definitions of variables and their subsequent use are tested.

For data flow based criteria, a definition use graph for the program is first constructed from the control flow graph representing a block of code has variables occurrences in it. Variable occurrences can be one of the following three types:

1 def represents the definition of a variable.

2 C-use represents computational use of a variable.

3 P-use represents predicate use.

Because we are interested in the flow of data between nodes, a C-use of a variable x is consider red global. C-use if there is no def of x within the block preceding the C-use. With each node i, we associated all the global C-use variables in that node the P-use is associated with edges.

QUE : EXPLAIN TESTING PROCESS:

The basic goal of the software development process is to produce software that has no errors.

In an effort to detect errors soon after they are introduced. Each phase end with a Verification activity such as a review. However, most of these verification activities in the early phases of software development are based on human evaluation and cannot detect all errors.

This unreliability of the quality assurance activities in the early part of the development cycle places a very high responsibility on testing.

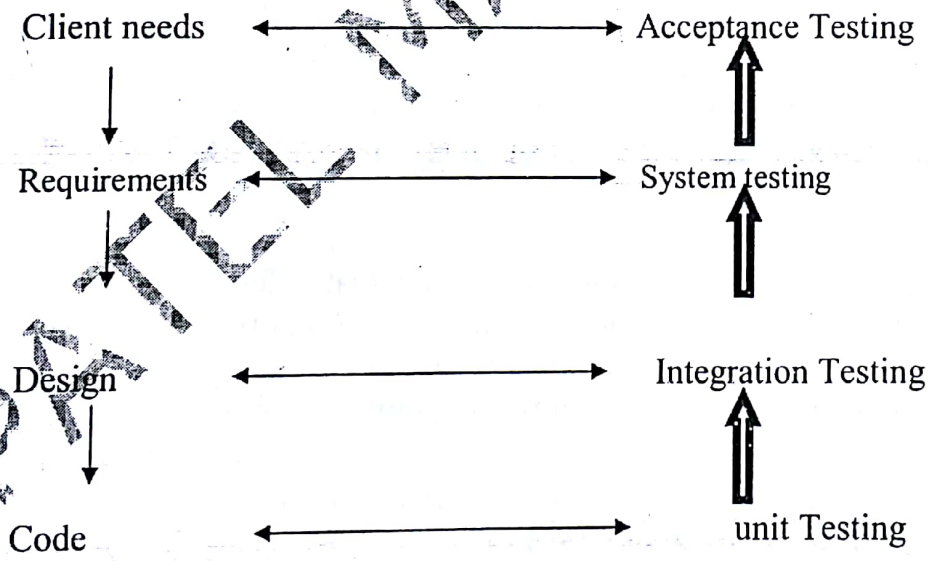
In other words, as testing is the last phase before the final software is delivered. It has the enormous responsibility of detecting any type of errors that may be in the software.

QUE : COMPARISON OF DIFFERENT TESTING TECHNIQUES:

It is not easy to compare the effectiveness of the different techniques. By effectiveness, we mean the fault detecting capability. The effectiveness of a technique for testing particular software will In general, depends on the type of errors that exist in the software.

Various studies have been done to evaluate or compare some techniques. Techniques to partition the input domain for testing purpose as is generally done by any testing criterion were evaluated against random testing. In which test cases are selected randomly, through simulation in. The simulation results seem to indicate that random testing may be quite cost effective.

QUE : EXPLAIN LEVEL OF TESTING:



The basic level are unit testing, integration testing, and system and acceptance testing. These different levels of testing attempts to detect different types of faults. The relation of the faults introduced in different phases, and the different levels of testing are shown in figure.

1) Unit testing : In this, different modules are tested against the specification produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test internal logic of the modules. It is typically done by the programmer of the module.

In unit testing, perform the testing with individual module or program.

2) Integration Testing: In this, Different modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly. Hence the emphases are on testing interfaces between modules. This testing activates can be considered testing the design.

3) System testing and Acceptance Testing:

Here the entire software system is tested. The reference document for this process is the requirements documents, and the goal is to see if the software meets its requirements. This is essentially a validation exercise, and in many situations it is the only validation activity for regression testing.

Some test cases that have been executed on the old system are maintained, along with the output produced by the old system. These test cases re executed again on the modified system and its output compared with the earlier output to make sure that the system is working as before on this test case. A consequence of this is that the test cases for system should be properly documented for future use.

QUE : DEFINE α , β ,ACCEPTANCE AND GAMMA TESTING.

System testing consists of three different kinds of testing activities as follows:

1) **α-testing** :a testing is the system testing performed by the development team.

2) **β-testing** : This is the system testing performed by a customers.

3) **Acceptance testing**:-This is the system testing performed by the customer himself after the product deliver to determine whether to accept the delivered product or to reject it.

4) **Gamma testing** is a little-known informal phrase that refers de isively to the release of "buggy" (defect-ridden) products. It is not a term of art among tester, but rather an example of referential humor. Cynics have referred to all software release as "gamma testing" since defects are found in almost all commercial, commodity and publicly available software eventually.

QUE : EXPLAIN MUTATION TESTING :

Mutation testing is a fault based testing techniques in the sense that mutation taste case are designed to help detect specific type of faults in a program.

The idea behind the mutation testing of few arbitrary changes to a program at a time.

Each time a program is changed, it is called **mutated program** and the change effected is called **mutant.**

Depending on the change made to the code, a mutated program may or may not introduce some error.

An important advantage of mutation testing is that it can be automated to a grate extent. The process of generation and a killing of mutant can be automated by predefining a set of primitive changes that can be applied to the program.

These primitive changes can be simple program alterations such as deleting a statement, deleting variable a variable definition , changing the type of an operators, changing data type of a variable, etc.

A major disadvantage of mutation based testing approach is that it is very expensive, since a large numbers of possible mutations can be generated.

QUE : WRITE SHORT NOTE ON TEST PLAN:

In general, Testing commences with a test plan and terminates with acceptance testing. A test plan is general document for entire project that defines the scope, approach to be taken and the schedule of testing as well as identifies the test items for the entire testing process and the personnel responsible for the different activity of testing.

There are three types of inputs for forming the test plan are:-
1) Project Plan
2) Requirement Documents
3) System Design Documents

A test plan should contain the following

- Test unit specification
- Features to be tested
- Approach for testing
- Test deliverables
- Schedule
- Personnel allocation

Test unit specification:-

A test unit is a set of one or more modules, together with associated data, that are from a single computer program and that are the object of testing. A test unit can occur at any level and can contain from a single module to the entire system. Thus a test unit may be a module, or a complete system.

Features to be tested:-

All software features and combinations of features that should be tested. A Software features is a software characteristic specified or implied by the requirements or design documents. These may module functionality performance, design constraints and attributes.

Approach for testing:-

The approach for testing specifies the overall approach to be followed in the current project. The technique that will be used to judge the testing effort should also be specified. This is sometimes called the testing criterion for evaluating the set of test cases used in the testing. In the previous sections we discussed many criteria for evaluating and selecting test cases.

Test deliverables:-

Testing deliverables should be specified in the test plan before the actual begins. Deliverables could be a list of test cases that were used detailed results of testing, test summary report, test log, and data about the code coverage.

In general a test case specification report, test summary and a test log should always be specified as deliverables.

Schedule:-

The schedule specifies the amount of time and effort to be spent on different activities of testing and testing of different units that have been identified.

Personnel Allocation:-

Personnel allocation identifies the person responsible for performing the different activities.

QUE : EXPLAIN TEST CASES, SUITES, SCRIPTS AND SCENARIOS:

A test case is usually a single step, and its expected result, along with various additional pieces of information. It can occasionally be a series of steps but with one expected outcome. The optional fields are a test case Id, test step or order of execution number, related requirement(s), depth, test category, author and also check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps and descriptions.

A test can should also contain a phase for the actual result. These steps can be stored in a word processor document, spreadsheet, database or other common repository. In a database system, you may also be able to see past test results and who generated the results and the system configuration used to generate those results. These past results would usually be stored in a separate table.

The most common term for a collection of test cases is test state. **The suite** often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps and descriptions of the following tests.

Collection of test cases is sometimes incorrectly termed a test plan. They may also be called a test script, or even a test scenario.

Most white box testers write and use **test scripts** in unit, system, and regression testing. Test scripts should be written for modules with the highest risk of failure and the highest impact if the risk becomes an issue companies that use automated testing will call the code that is used in their test scripts.

A scenario test is test based on a hypothetical story used to help a person think through a complex problem or system. They can be as a diagram for testing environment or they could be a description written in prose.

The ideal scenario test has five key characteristic. It is (a) a story that is (b) motivating, (c) credible, (d) complex and (e) easy to evaluate. They are usually different from test cases in that test cases are single steps and scenarios cover a number of steps. Test suites and scenario can be used in concert for complete system tests. See an introduction to scenario testing.

Scenario testing is similar to but not the same as session-based testing, which is more closely relates to exploratory testing but the two concepts can be used in conjunction. See adventures in Session-Based Testing and Session-Based Test Management.

QUE : EXPLAIN TEST CASE SPECIFICATION :

The test plan focuses on how the testing for the project will proceed, which units will be tested and what approaches are to be used during the various stages of testing. However, it does not deal with the details of testing a unit nor does it specify which test cases are to be used.

Test case specification to be done separately for each unit based on the approach specified in test plan, first the features to be tested for this unit must be determined.

Test case specification gives for each unit to be tested, all test cases, inputs to be used in test cases, conditions being tested by the test case and outputs expected for those test cases.

Test case specification is a major activity in the testing process. Careful selection of test cases that satisfy the criterion and approach specified is essential for proper testing.

There are two basic reasons test cases are specified before they are used for testing.

1) The effectiveness of testing depends very heavily on the exact natures of the test cases.

2) Constructing good test cases will reveal errors in programs are still a very creative activity that depends a great deal on the ingenuity of the tester.

The test case specification document is reviewed, using a formal review process, to make sure that the test case are consistent with the policy specified in the plan, satisfy the chosen criterion; and in general cover the various aspects of the unit to be tested. For this purpose, the reason for selecting the test case and expected output are also given in the test case specification document.

Another reason for formal test case specification is that specifications can be used as scripts during regression testing, particularly if regression testing is to be preformed manually. Generally the test case specification document itself if is used to record the result of testing.

QUE : EXPLAIN A SAMPLE TESING CYCLE :

There is a cycle to testing

1) Requirement Analysis: - Testing should begin in the requirement phase of the software development life cycle.

2) Design Analysis: - During the design phase, tester work with developer in determining what aspects of a design is testable and under what parameter those tests work.

3) Test Planning: - Test Strategy, Test Plan(s), Test Bes creation.

4) Test Development: - Test Procedures, Test Scenarios, Test cases, Test Scripts to use in testing software.

5) Test Execution: - Tester executes the software based on the plans and tests and report any errors found to the development team.

6) Test Reporting:- Once testing is completed, tester generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.

7) Retesting the Defect

Not all errors or defects reported must be fixed by a software development team. Some may be caused by error in configuring the test software to match the development or production environment. Some defects can be handled by a workaround in the production environment. Others might be deferred to future release of the software, or the deficiency might be accepted by the business user. There are yet other defects that may be rejected by the development team (of course, with due reason) if they deem it inappropriate to be called a defect.

QUE : EXPLAIN REGRESSION TESTING:

This type of testing is require when the system being tested is an up gradation of an already existing system to fix some bugs or enhance functionality, performance, etc.

Regression testing is the practice of running an old test suite after each change to the system or after each bug fix to ensure that no new bug has been introduced due to the change or the bug fix

If only a few statements are changed, then the entire test suite need not be run – only those test cases that test the functions that are likely to be affected by the change need to be run.

A software is changed to fix a detected bug , regression testing is carried out.