Syllabus:1. To Understand Software Implementation and Testing

- 1.Explain programming principles and coding guidelines2.Describe the method of incrementally developing code3.Explain how to manage the evolving code4.Define software testing5.Explain unit testing and code inspection6.Explain the testing concepts and testing process7.Design test case and test plan8.Describe Black box testing9.Describe White box testing

---

1. Explain programming principles and coding guidelines

Coding and Testing

- Coding is started when the design phase is complete.
- Every module specified in the design document is coded and unit tested. During unit test each module is tested in isolation from other modules.
- After all the modules have been unit tested and coded, the integration and system testing phase is undertaken.
- The full product takes shape only after all the modules have been integrated together.
- System testing is conducted on the full product.

Coding

The objective of the coding phase is to transform the design of a system in to code in a high-level language, and then to unit test this code.

- The main advantages of using standard style of coding are the following:
  - A coding standard gives a uniform appearance to the codes written by different engineers.
  - It facilitates code understanding.
  - It promotes good programming practices.
- Good organizations have its own coding standards and guidelines.

Coding Standards

- Mandatorily followed by programmers and compliance of coding standards verified before testing phase starts

Coding guidelines

- Give some general suggestions regarding coding, but leave actual implementation depends on individual developers.

1. Give programming principles, guidelines and programming practices.

- The principle and concepts that guide that guide coding task are closely aligned programming style, programming language andmethods.
- Before starting the code, follow someprinciples

1. Select data structures that will meet the need of thedesign.
2. Keep conditional logic as simple aspossible.
3. Understand the software architecture and create interfaces that are consistent withit.
4. Select meaningful variable names and follow other local codingstandards.
5. Write code that isself-documenting.
6. Create a visuallayout.
7. Constrain your algorithm by structured programmingpractice.

- Programming practices help to avoid commonerrors.

1. Controlconstruct

The single entry and exit constructs need to be used.

2. Use ofgoto

The goto statements make the program unstructured. So avoid use of goto statements as possible.

3. Informationhiding

4. Nesting

Structure inside another structure is called as nesting. If there is too deep nesting then it becomes hard to understand the code as well as complex.

5. User defined datatypes

User can define data type to enhance the readability of the code.

6. Modularsize

The size of program may be large or small. There is no rule for size of the program. So as possible generate different module but not of large size.

7. Sideeffects

Sometimes if some part of code may change then it may generate some kind of problems called as side effects.

8. Robustness

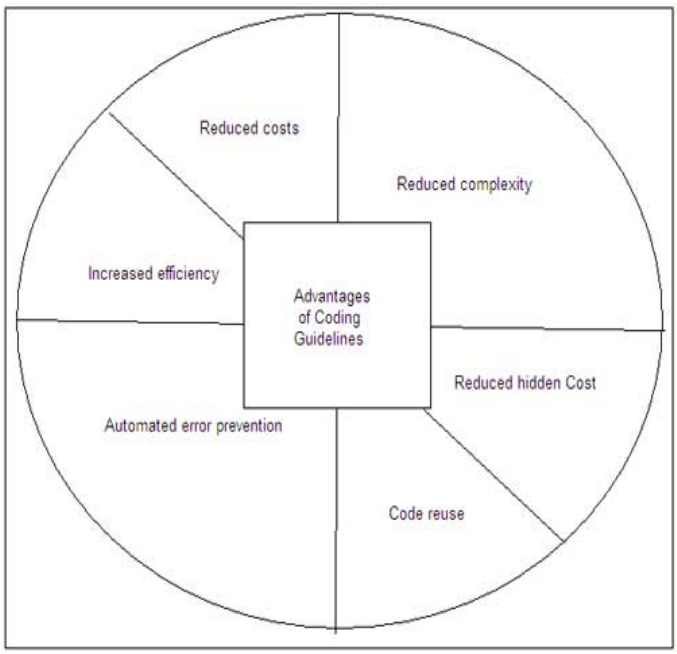
If any kind of exception is generated, the program should generate some kind of output. Then it is called as robustness. In this situation the programs do not crash.

9. Switch case withdefaults

Inside the switch case statement if any value which is unpredictable is given as argument then there should be default case to execute it.

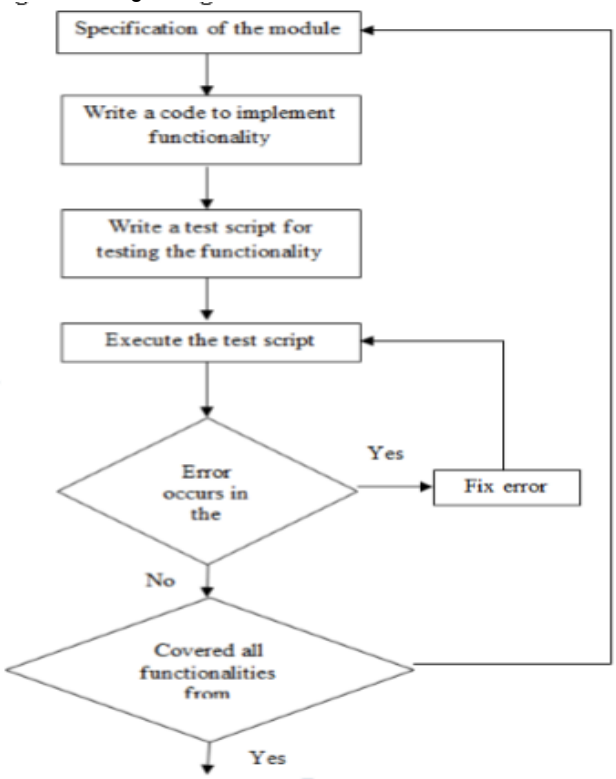
2. Representative coding guidelines

1. **Do not use a coding style that is too clever or too difficult to understand .**
2. **A void obscure side effects :**Modifications to parameters passed by reference, can't identify while casual examining.
3. **Do not use an identifier for multiple purpose :** Problems caused by this are:
4. Variable for multiple purpose can lead to confusion.
5. Future enhancements more difficult.
6. **The code should be well- documented .**
7. **The length of any function should not exceed 10 source lines .**
8. **Do not use GO TO statements .**



2.Describe the method of incrementally developing code

- To create any kind of good quality software, there is needed to follow some kind ofstandards.
- These standards should follow at each and every stage. Like for coding there are also some standards called as coding standard which are givenbelow:



e **Incremental Development of Code**

Naming Convention:

- To provide name there are some rules to be followedlike,
- Variable name must not begin with numbers.
- Package name and variable name should begin with lowercase.
- Constants must be in uppercase.
- Name of variable should bemeaningful.
- The variable with large scope must have long name. For example count_total, sumetc.
- The prefix must be used for Boolean type ofvariables.

Files

- Reader must get an idea about the purpose of the file by its name. in some programming langue there is some extension given tofile.
- From the name of file, reader will have idea what is the content offile.
- Name of class and file name must be same.

Comment

- Comments are non-executable part of the code. But it is very important because it enhances the readability of the code.
- Single line comment is given by//.
- Multi line comment is given /* and*/.
- For the name of the variables comments must be given.

Statements

Guidelines about declaration and executable statements:

- Class variable should never be declared.
- Avoid use of break and continue statements in the loop.
- Avoid complex conditional expressions, do...while statements.

- Declare some related variables on some line and unrelated variable in another line.

Advantages:

- The code become readable and can be understood easily.
- It helps in good programming practises.
- It brings uniform appearance in system implementation.

Incremental Development of Code

- Whenever the design has been completed, the coding technique will be started. Before write a code, some specification for the functionalities must beavailable.
- For that purpose there will be some incremental approach for development.
- Here first of all specify the module and then write code to implement functionality.
- Then test that code based on some cases. Then execute the test script.
- It should be checked that any kind of errors are generated then fix the errors.
- If any kind of errors are not generated then covered all the functionalities mentioned in the specification, the process is terminated.
- Each and every functionality is written and immediately tested is one of its advantages

3.Explain how to manage the evolving code

- Inside any organization each software developer will create a program module. After completion of every module they are integrated with each other. At that time there is needed to manage such code.
- The code management system contains a central repository in which there exists a control directory structure which keeps the full revision history of all the files.
- File history should be stored and using the older version new versions can be created. The repository is official source of the files.
- Various commands are performed on the repository are

1. Get a local copy

- The local copy of the repository is obtained by the programmer and programmer works on it. Making local copy is called checkout.

2. Make changes in the file

- The changes are made in local copy of the file. These changes are committed back in the repository. The operation is called as checkin.

3. Update local copy

- Changes made in the local copy are not reflected in the local copy of the repository. Hence using the update command these changes are updated.

4. Get report

- A detailed report on the evolution made can be obtained. For example the report containing the old data and corresponding changes made, responses for the changes, corresponding files in which these changes must be reflected and soon.
- After making these changes, these changes must be available to all the team members of theproject.
- If two processes try to access the same file at a time then it creates the conflicts. These conflicts must be solved manually.

CODE REVIEW

- Code review for a module is undertaken after the module successfully complies.
- More cost effective than testing is that reviews directly detect errors.
- Eliminating an error from code involves three main activities- testing(certain inputs), debugging(locating error) and correcting errors.
- Normally the following two types of reviews are carried out on the code of a module

1. Code inspection
2. Code walkthrough

Code walkthrough

- A few members of the development team are given the code couple of days before walkthrough meeting.
- Each member select some test cases and simulate the execution code.
- Members note down their findings and discuss in meeting.
- The main objective of code walkthrough is to discover the algorithmic and logical errors in the code.
- Some of these guidelines are the following
    - The team performing code walkthrough should not be either too big or too small. Ideally, it should consist of between three to seven members.
    - Focus on discovery of errors.
    - In order foster cooperation, avoid managers.

Code inspection

- During the code inspection the code is examined for the presence of some common programming errors.
- Following is a list of some classical programming errors which can be checked during code inspection:
    - Use of uninitialized variables
    - jumps into loops
    - Non-terminating loops
    - Array indices out of bounds

Clean Room Testing : Was pioneered at IBM.Testing relies heavily on walkthroughs, inspection and formal verification.

4.Define software testing

TESTING

- The aim program testing is to identify all defects in a program. It is not possible to guarantee that a program is error free.
- A software product is normally tested in three level or stages :
    - unit testing
    - Integration testing
    - System testing
- During unit testing , the individual components (or units) of a program are tested.
- After testing all the units individually, the units are slowly integrated and tested after each step of integration(integration testing).
- Finally , the fully integrated system is tested(system testing).
- Unit testing is referred to as **testing in the small**, where as integration and system testing are referred to as **testing in the large**.

6.Explain the testing concepts and testing process

Basic concepts and terminologies

- Testing a program involves providing the program with a set of test inputs(or test cases) and checking if the programs behaves as expected.
- The following are some commonly used terms associated with testing:

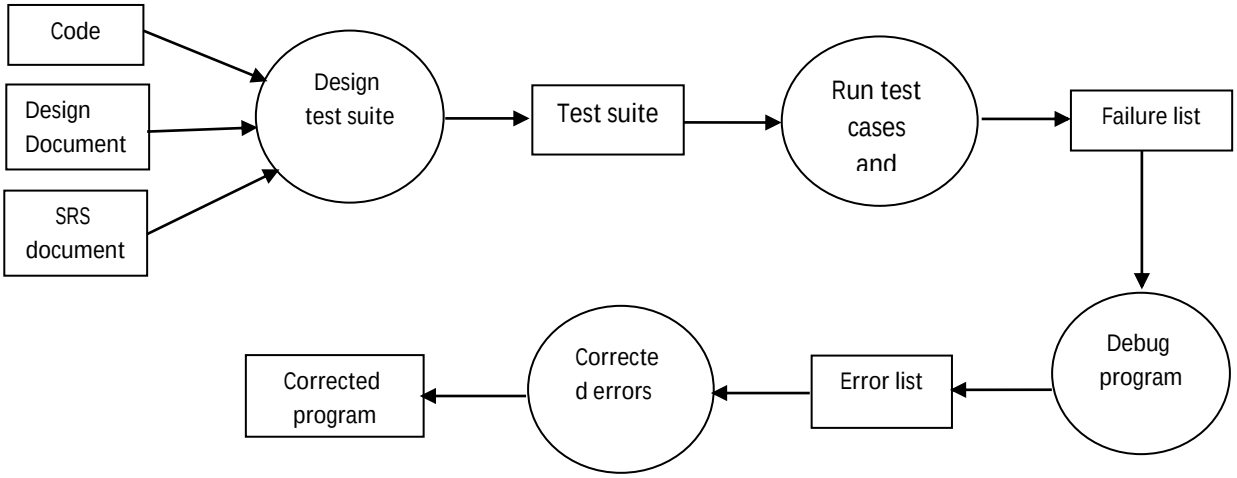
- An **error** is a mistake committed by the development team during any of the development phases.
- A **failure** is a manifestation(action) of an error (or *defect* or *bug).* In other words , a failure is the symptom of an error.
- A **test case** is the triplet [I,S,O] , Where I is the data input to the system , S is the state of the system at which the data is input , O is the excepted output of the system.
- A **test suite** is the set of all test cases with which a given software product is tested.

Testing activities

Testing involves performing the main activities:

1. **Test suite design** : The set of test cases using which a program is to be tasted is designed.
2. **Running test cases and checking the result to detect failure**: Each test cases in run and the results are compared with the expected results.
3. **Debugging**: for each failure observed during the previous activity , debugging is carried out to identify the statements that have error.
4. **Error correction:** after the error is located in the previous activity, the code is appropriately changed to correct the error.

Diagrammatic representation of testing Process



5.Explain unit testing and code inspection

UNIT TESTING

- Unit testing is undertaken after a module has been coded and reviewed.
- Before carrying out unit testing, the unit test cases have to be designed and the test environment for the unit under test has to be developed.

Driver and stub modules

- Stubs and drivers are designed to provide the complete environment for a module so that testing can be carried out .
- Unit testing focuses verification effort on the smallest unit of software design the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of themodule.
- The module interface is tested to ensure that information properly flows into and out of the program unit undertest.
- Tests of data flow across a module interface are required before any other test isinitiated.
- Selective testing of execution paths is an essential task during the unit test. Test cases should be

designed to uncover errors due to erroneous computations, incorrect comparisons, or improper controlflow.

- Among the more common errors in computation are (1) misunderstood or incorrect arithmetic precedence, (2) mixed mode operations, (3) incorrect initialization, (4) precision inaccuracy, (5) incorrect symbolic representation of anexpression.
- The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm'sexecution.
- Boundary testing is the last task of the unit test step. Software often fails at its boundaries. That is, errors often occur when the nth element of an n-dimensional array is processed, when the ith repetition of a loop with i passes is invoked, when the maximum or minimum allowable value isencountered.
- Test cases should uncover errors such as comparison of different data types, incorrect logical operators or precedence etc.
- All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths aretested.
- A driver is a "main program" that accepts test case data, passes such data to the component, and prints relevant results. Stubs serve to replace modules that are subordinate the component to betested.
  - A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoingtesting.

Stub

- A stub procedure is a dummy procedure that has the same I/O parameters as the given procedure.

Driver

- A driver module should contain the non-local data structures accessed by the module under test.

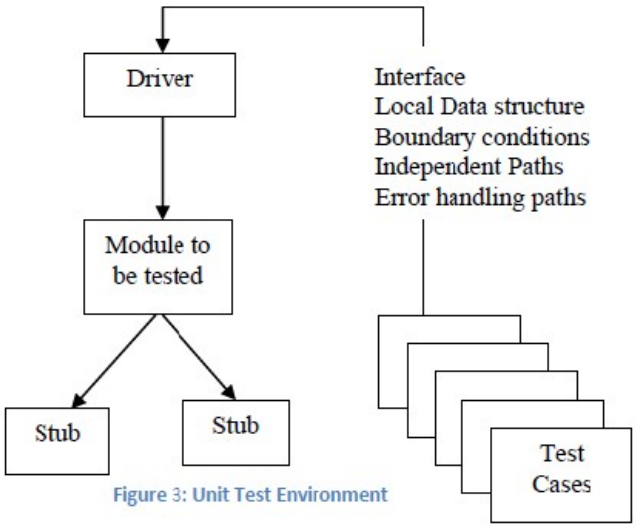


Figure 3: Unit Test Environment

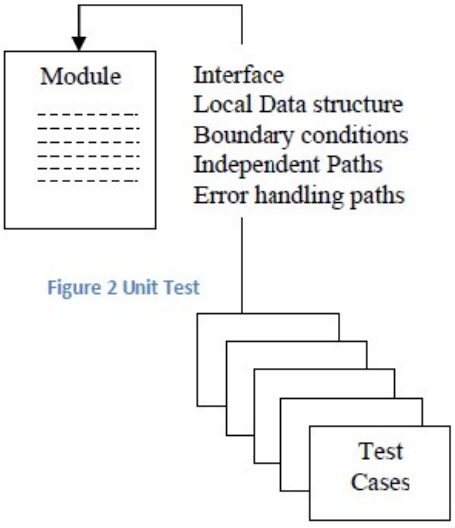


Figure 2 Unit Test

Code Inspection

- The goal of code inspection technique is to find errors and defects in the code. It will compile your code successfully.
- For that purpose there will be a team of member like,
- Moderator: - Manager or the leader of the code in spectionteam.
- Designer: - The team responsible for the current phase.
- Implementer: - The team responsible for the next phase.
- Tester: - The person who tests the code. This person must be preferably from SQA team.
- The design document and the document containing the code for the review are distributed to the team members during the code inspection.

- It also looks for the quality issue of the code.

A checklist must be prepared while reviewing the code. A checklist is given below:

- Are the definitions exhibitsthe typing with capacities ofthe language?
- Are there anydanglingpointers?
- Is there any use ofNULLpointer?
- Are all indexes withinabound?
- Are all indexesproperlyinitialized?
- Is there anyinfinite loop?

Is there any condition such as divideby zero?

Are all the output variables got assigned somevalue?
Does the code satisfy all the localcoding standards?
Do actual and formal parameters ofthe functionmatch?
Is there any undeclaredvariable?
Are all labels referredcorrectly?

Advantages

- It is very effective for finding the defects from the code.

Disadvantages

- The code inspection process is time consuming.
- A large number of people are involved in the code inspection process, it can be turned out to be costly process.

Due to these drawbacks of the code inspection, a single person may carry out the code inspection and code reviews.

8.Describe Black box testing

BLACK-BOX TESTING

- In black –box testing , test cases are designed from an examination of the input/output values only and no knowledge of design or code is required .
- The following are the two main approaches available to design to design black-box test cases:

1. Equivalence e class partitioning
2. Boundary value analysis

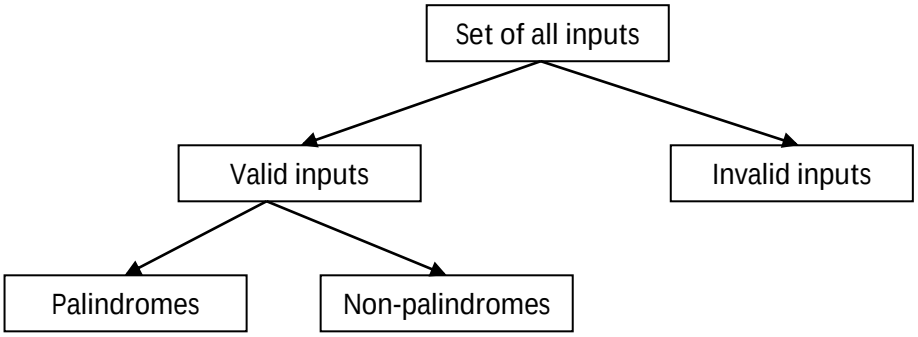
Equivalence Class Partitioning

- The domain of input value to the program under test is partitioned in to a set of equivalence classes .
- The partitioning is done such that for every data input data belonging to the same equivalence class, the program behaves similarly.
- The following are two general guidelines for designing the equivalent classes:

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

2.If the input data assumes values from a set of discrete members of some domain, then one equivalent classes for the valid input values and another equivalence class for the input values should be defined .

- Eg: Design equivalence class partitioning test suite for function that reads a character string of size less than five characters and displays whether it is palindrome.



Boundary Value Analysis

- Test suite design involves designing test cases using the values at the boundaries of different equivalence classes.
- For those equivalence classes that are not a range of values no boundary value test cases can be defined.
- For an equivalence class that is a range of values, the boundary values need to be included in the test suite.

The important steps in the black –box test suite design approach:

- Examine the input and output values of the program.
- Identity the equivalence classes.
- Design equivalence class test cases by picking one representative value from each equivalence class.
- Design boundary value test cases

9.Describe White box testing

WHITE BOX TESTING

- White-box testing is an important type of unit testing .
- A large number of white-box testing strategies exist.

Basic Concepts

- A White-box testing strategy can either be *coverage based or fault based*.

Fault based testing

- A fault – based testing strategy targets to detect certain types of faults.
- An example of fault-based strategy is mutation testing.

Coverage based testing

- A coverage –based testing strategy attempts to execute (or cover) certain elements of a program.
- Popular examples of coverage based strategy are statement coverage, branch coverage, and path coverage- based testing.

1. **Statement coverage**
   - The statement coverage-based strategy aims to design test cases so as to execute every statement in a program at least once.
   - The principal idea governing the statement coverage strategy is that unless a statement is executed, there is no way to determine whether an error exists in the statement .
   - The weakness of the statement coverage strategy is that executing a statement once and observing that it behaves properly for one input value is no guarantee that it will behave correctly for all input values.
2. **Branch coverage**
   - Branch coverage-based testing requires test cases to be designed so as to make each branch condition in the program to assume true and false values in turn.
   - Is also known as edge testing, since in this testing scheme, each edge of program's control flow graph is traversed at least once.
   - Branch coverage-based testing is a stronger testing than statement coverage-based testing.

3. **Condition coverage**
   - In the condition coverage-based testing, test cases are designed to make each component of a component conditional expression to assume both true and false values.
   - Condition coverage-based testing is a stronger testing than branch coverage-based testing.
4. **Path coverage**
   - Path coverage-based testing strategy requires designing test cases such that all linearly independent paths (or basis paths) in the program are executed at least once.
   - A linearly independent path can be defined in terms of the control flow graph (CFG) of a program.

Control flow graph - A control flow graph describes how the control flows through the program.Describes the sequence in which the different instructions of a program get executed.First number all the statements of program.

Path-A**e**dge sequence from the start node to a terminal node of the control flow graph of a program.

Linearly independence set of paths(or basis path set) - If each path in the set introduces at least one new edge of basis paths or simply the basis set.

CFG

CFG for *sequence:*

1. a= 5;
2. b=a*2-1

CFG for selection:

1. If(a>b)
2. c=3;
3. else c=5;
4. C=c*c;

CFG for Iteration:

1. while(a>b){
2. b=b-1;
3. b=b*a;}
4. c=a+b;

5. **Mutation testing**
   - The idea behind mutation testing is to make a few arbitrary changes to a program at a time.
   - Each time the program is changed, it is called a mutated program and the changes effect is called mutant.
   - Depending on the change made to the code, a mutated program may or may not introduce errors.
   - After initial testing is complete mutation testing can be taken up.
   - If there exist at least one test case in the test suite for which a mutated program yields an incorrect result, then the mutant is said to be dead, since the error introduced by the mutation operator has successfully been detected by the test suite.

- If a mutant remains alive even after all the test cases have been exhausted, the test suite is enhanced to kill the mutant.

7.Design test case and test plan

- A test plan describe how testing would be accomplished.
- It is a document that specifies the purpose ,scope, and method of software testing.
- It determines the testing tasks and the persons involved in executing those tasks, test items and the features to be tested.
- It also describes the environment for testing and the test design and measurement techniques to be used.
- A properly defined test plan is an agreement between testers and users describing the role of testing in software.
- A complete test plan helps the people who are not involved in test group to understand why product validation is needed and how it is to be performed.

COMPONENTS	PURPOSE
Responsibilities	Assigns responsibilities to different people and keeps them focused
Assumptions	Avoids any misinterpretation of schedules
Test	Provides an abstract of entire process and outlines specific tests. Testing scope ,schedule and duration are also outlined.
Communication	Communication plan is developed
Risk analysis	Identifies areas that are critical for success.
Defect reporting	Specifies the way in which a defect should be documented so that it may reoccur and be retested and fixed.
environment	Describes the data, interfaces ,work area, and the technical environment used in testing . All this is specified to reduce or eliminate the misunderstanding and sources of potential delay.

Test plan steps

1. Set objectives of test plan:-

- Before developing a test plan, it is necessary to understand its purpose.
- Before determining the objectives of a test plan , it is necessary to determine the objectives of the software.
- The objectives of a test plan are highly dependent on that software.

2. Develop a test matrix.:-

- A test matrix indicates the components of the software that are to be tested.
- It also specifies the tests required to check these components.
- Test matrix is also used as a test proof to show that a test exists for all components of the software that require testing.

- Test matrix is used o indicates the testing method,which is used to test the entire software.

3. Develop test administrative components:-

- The purpose of administrative component of a test plan is to specify the time schedule and resources required to execute the test plan.
- If the implementation plan of software changes, the test plan also changes.

4. Write the test plan :-

- The components of a test plan such as its objectives, test matrix, and administrative component are documented.
- All these documents re then collected together to form a complete test plan.
- These documents are organized either in an informal or formal manner.

TEST CASE

- A test case provides the description of inputs and their expected outputs to observe whether the software or a part of the software is working correctly.
- A test case is associated with details like identifier, name, purpose, required inputs,test conditions and expected outputs.
- The purpose of generating test cases helps to identify the problem that exist in the software requirements and design.
- For generating test cases,firstly the criterion to evaluate a set of test cases is specified and then the set of test cases satisfying that criterion is generated.

- There are two methods used to generate test cases:-
  - Code- based test cases generation.
  - Specification based test cases generation.

- Test case specification
  - ✓ A test plan is neither nor related to the details of testing units nor it specifies the test cases to be used for testing units.
  - ✓ Thus , test case specification is done in order to test each unit separately.
  - ✓ Depending on the testing method specified in a test plan, the features of the unit to be tested are determined.
  - ✓ The overall approach stated in the test plan is refined into two parts:-
    - Specific test method.
    - Evaluation criteria.

  - ✓ Based on these test methods and the criteria, the test cases to test the unit are specified.
  - ✓ For each unit being test, these test case specifications describe the test cases, required inputs for test cases,test conditions, and the expected outputs from the test cases.
  - ✓ Test cases specifications are written in the form of a document.
  - ✓ The quality of test cases is evaluated by performing a test case review, which requires a formal document.

- ✓ The benefit of specifying test cases in a formal document is helps testers to select an effective set of test cases.