www.jntukmaterials.com

UNIT – IV

Coding: Coding, Code Review, Software Documentation
Testing: Testing, Unit Testing,Black-Box Testing, White-Box Testing, Debugging, Program Analysis Tool, Integration Testing,Testing Object-Oriented Programs, System Testing, Some General Issues Associated with Testing

Coding:

Introduction:

Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards. Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards rigorously. The purpose of requiring all engineers of an organization to
adhere to a standard style of coding is the following:

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It enhances code understanding.
- It encourages good programming practices.

Coding standards and guidelines :

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop. The following are some representative coding standards.

Rules for limiting the use of global: These rules list what types of data can be declared global and what cannot.

Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Error return conventions and exception handling mechanisms: The way error conditions are reported by different functions in a program are handled should be standard within an organization.

Do not use a coding style that is too clever or too difficult to understand: Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.

Avoid obscure side effects: The side effects of a function call include modification of parameters passed by reference, modification of global variables, and I/O operations.

The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

Do not use goto statements: Use of goto statements makes a program unstructured and makes it very difficult to understand.

Code review :

Code review for a model is carried out after the module is successfully compiled and the

www.jntukmaterials.com

all the syntax errors have been eliminated. Code reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code. Normally, two types of reviews are carried out on the code of a module. These two types code review techniques are code inspection and code walk through.

Code Walk Throughs :

Code walk through is an informal code analysis technique. In this technique, after a module has been coded, successfully compiled and all syntax errors eliminated. A few members of the development team are given the code few days before the walk through meeting to read and understand code. The main objectives of the walk through are to discover the algorithmic and logical errors in the code. The members note down their findings to discuss these in a walk through meeting where the coder of the module is present

Some of these guidelines are the following.

- The team performing code walk through should not be either too big or too small. Ideally, it should consist of between three to seven members.
- Discussion should focus on discovery of errors and not on how to fix the discovered errors.
- In order to foster cooperation and to avoid the feeling among engineers that they are being evaluated in the code walk through meeting, managers should not attend the walk through meetings.

Code Inspection

In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In other words, during code inspection the code is examined for the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk throughs. Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed. Such a list of commonly committed errors can be used during code inspection to look out for possible errors.

Following is a list of some classical programming errors which can be checked during code inspection:

- Use of uninitialized variables.
- Jumps into loops.
- Nonterminating loops.
- Incompatible assignments.
- Array indices out of bounds.
- Improper storage allocation and deallocation etc..

Software documentation :

Introduction
Internal Documentation
External Documentation
Gunning's Fog Index

Introduction:

When various kinds of software products are developed then not only the

www.jntukmaterials.com

executable files and the source code are developed but also various kinds of documents such as users' manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process. All these documents are a vital part of good software development practice. Good documents are very useful and server the following purposes:

• Good documents enhance understandability and maintainability of a software product.
• Use documents help the users in effectively using the system.
• Good documents help in effectively handling the manpower turnover problem.
• Production of good documents helps the manager in effectively tracking the progress of the project.

Internal documentation :

Internal documentation is the code comprehension features provided as part of the source code itself. Internal documentation is provided through appropriate module headers and comments embedded in the source code. Internal documentation is also provided through the useful variable names, module and function headers, code indentation, code structuring, use of enumerated types
and constant identifiers, use of user-defined data types, etc. Careful experiments suggest that out of all types of internal documentation meaningful variable names is most useful in understanding the code.

External documentation:

External documentation is provided through various types of supporting documents such as users' manual, software requirements specification document, design document, test documents, etc. A systematic software development style ensures that all these documents are produced in an orderly
fashion.

Gunning's Fog Index:

In linguistics, the Gunning fog index is a readability test for English writing. The index estimates the years of formal education a person needs to understand the text on the first reading. A fog index of 12 requires the reading level of a U.S. high school senior (around 18 years old). The test was developed by Robert Gunning, an American businessman, in 1952. The fog index is commonly used to confirm that text can be read easily by the intended audience. Texts for a wide audience generally need a fog index less than 12. Texts requiring near-universal understanding generally need an index less than 8

The complete formula is:

$$0.4\left[\left(\frac{\text{words}}{\text{sentences}}\right)+100\left(\frac{\text{complex words}}{\text{words}}\right)\right]$$

Fog Index	Reading level by grade
17	College graduate
16	College senior
15	College junior
14	College sophomore
13	College freshman
12	High school senior
11	High school junior
10	High school sophomore
9	High school freshman
8	Eighth grade
7	Seventh grade
6	Sixth grade

www.jntukmaterials.com

Testing

Testing:

Basic Concepts and Terminologies
Testing Activities
Why design test cases

Basic Concepts and Terminologies:

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

Failure: It is the inability of a system or component to perform required function according to its specification. Faliure indicates that External behavior is incorrect.
Error: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
Test case: This 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.
Test suite: This is the set of all test cases with which a given software product is to be tested.

Testing Activities:

Test Suite Design: The set of testcases using which a program is to be tested is designed using different techniques (blackbox, whitebox etc..).
Running Test cases and checking the result to detect failures: Each test case is run and the results are compared with the expected results. A mismatch between tha actual result and expected results indicates failure.
Debugging: For each failure observed during the previous activity, debugging is carried out to identify the statements that are in error.
Error correction: After the error is located in the previous activity, the code is appropriately changed to correct the error.

Why design test cases:

Exhaustive testing of almost any non-trivial system is impractical due to the fact that the domain of input data values to most practical software systems is either extremely large or infinite. Therefore, we must design an optional test suite that is of reasonable size and can uncover as many errors existing in the system as possible. Actually, if test cases are selected randomly, many of these randomly selected test cases do not contribute to the significance of the test suite, i.e. they do not detect any additional defects not already being detected by other test cases in the suite. Thus, the number of random test cases in a test suite is, in general, not an indication of the effectiveness of the testing. In other words, testing a system using a large collection of test cases that are selected at random does not guarantee that all (or even most) of the errors in the system will be uncovered.

www.jntukmaterials.com

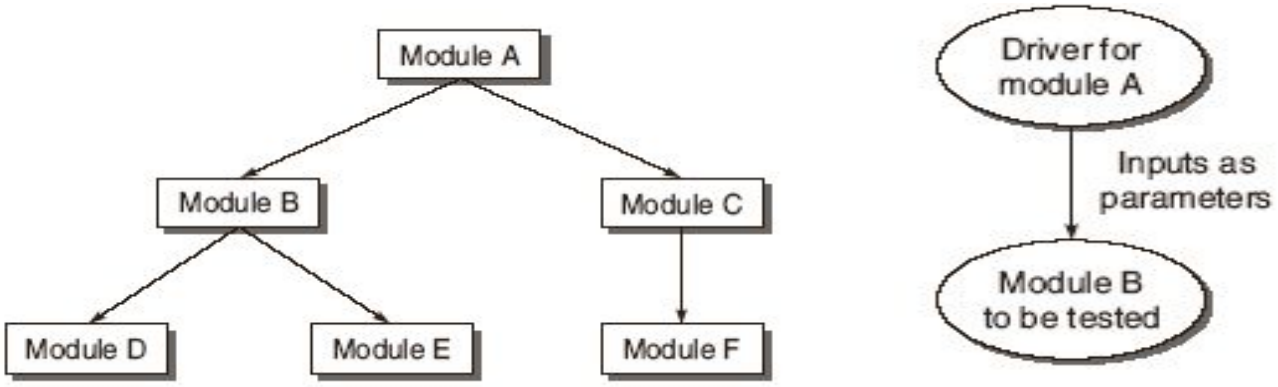
Unit Testing:

A unit is the smallest testable part of an application like functions, classes, procedures, interfaces. Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. Unit tests are basically written and executed by software developers to make sure that code meets its design and requirements and behaves as expected. The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly. This means that for any function or procedure when a set of inputs are given then it should return the proper values.

Drivers:

Drivers are also kind of dummy modules which are known as "calling programs", which is used when main programs are under construction. Suppose a module is to be tested, where in some inputs are to be received from another module. However this module which passes inputs to the module to be tested is not ready and under development. In such a situation, we need to simulate the inputs required in the module to be tested. This module where the required inputs for the module under test are simulated for the purpose of module or unit testing is known as **driver module**.

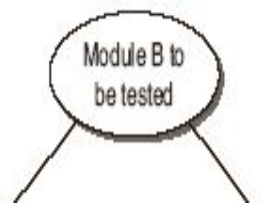
For example: When we have modules B and C ready but module A which calls functions from module B and C is not ready so developer will write a dummy piece of code for module A which will return values to module B and C. This dummy piece of code is known as driver.



Stubs:

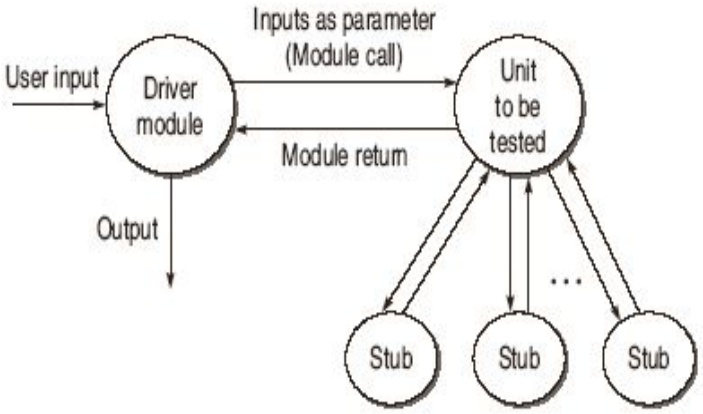
Stubs are dummy modules which are known as "called programs" which is used when sub programs are under construction. The module under testing may also call some other module which is not ready at the time of testing. Therefore, these modules need to be simulated for testing. In most cases, dummy modules instead of actual modules, which are not ready, are prepared for these subordinate modules. These dummy modules are called **Stubs.**

Assume you have 3 modules, Module A, Module B and module C. Module A is ready and we need to test it, but module A calls functions from Module B and C which are not ready, so developer will write a dummy module which simulates B and C and returns values to module A. This dummy module code is known as stub.



www.jntukmaterials.com

Benifits of using Stubs and Drivers:



--Stubs allow the programmer to call a method in the code being developed, even if the method does not have the desired behaviour yet.
--By using stubs and drivers effectively, we can cut down our total debugging and testing small parts of a program individually, helping us to narrow down problems before they expand.
--Stubs and Drivers can also be an effective tool for demonstrating progress in a business environment.

Example:

```
main()
{
        int a,b,c,sum;
        scanf("%d%d",&a,&b);
        sum=calsum(a,b);
        diff=caldiff(a,b);
        mul=calmul(a,b);
        printf("The sum is %d:",sum);
}
calsum(int x, int y)
{
        int d;
        d=x+y;
        return d;
}
```

*Suppose if the main() module ot caldiff() & calmul() moduel is not ready, then the driver and stub module can be designed as:

Solution:

www.jntukmaterials.com

-->Driver for main() Module:

```
driver_main()
{
        int a,b,c,sum;
        scanf("%d%d",&a,&b);
        sum=calsum(a,b);
        printf("The sum is %d:",sum);
}
```

-->Stub for call_sum() module:

```
call_sum(int x, int y)
{
        printf("Difference calculating module")
        return 0;
}
```

Black-Box Testing:

Introduction
Equivalance class partitioning
Boundary Value Analysis

Introduction:

Black Box Testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional. This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:
--- Incorrect or missing functions
-- Interface errors
-- Errors in data structures or external database access
-- Behavior or performance errors
-- Initialization and termination errors

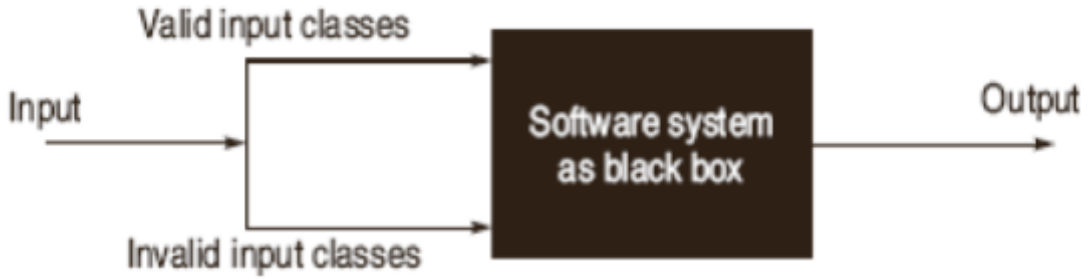
Equivalance class partitioning:

Equivalence class partitioning is a specification-based or black-box technique. It can be

www.jntukmaterials.com

applied at any level of testing and is often a good technique to use first. The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence class partitioning'.

In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.



"Refer the PPT for Example"

Boundary Value Analysis:

Boundary value analysis(BVA) is based on testing at the boundaries between partitions.Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).

"Refer the PPT for Example"

White-Box Testing:

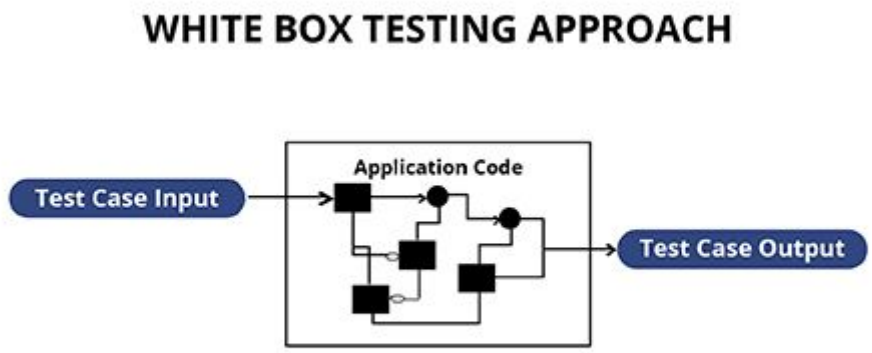
Basic concepts
Statement Coverage
Branch Coverage
Condition Coverage
Path Coverage
Cyclomatic complexity
Mutation Testing

Basic concepts:

--White box testing (WBT) is also called Structural or Glass box testing.
--White box testing involves looking at the structure of the code.
--We need WBT to ensure:
-->That all independent paths within a module have been exercised at least once.
-->All logical decisions verified on their true and false values.
-->All loops executed at their boundaries and within their operational bounds

www.jntukmaterials.com

internal data structures validity.



Example:

```
scanf("%d%d",&x,&y);
while(x!=y)
{
        if(x>y)
                x=x-y;
        else
                y=y-x;
}
printf("x=%d  y=%d",x,y);
```

Statement Coverage:

It is assumed that if all the statements of the module are executed once, every bug will be notified. If we want to cover every statement in the above code, then the following test cases must be designed:

Test case 1: x = y = n, where n is any number
Test case 2: x = n, y = n', where n and n' are different numbers.
Test case 3: x>y
Test case 4: x<y

Decision or Branch Coverage:

Branch Coverage states that each decision takes on all possible outcomes. In other words each branch direction must be traversed atleast once. The different test cases that can be designed are:

Test case 1: x = y
Test case 2: x != y
Test case 3: x < y
Test case 4: x > y

Condition Coverage:

It states that each condition in a decision takes on all possible outcomes at least once. For example consider the following statement: while((I<5) && (J<COUNT))
The different test cases are:

Test case 1: I <= 5, J < COUNT
Test case 2: I > 5, J > COUNT

www.jntukmaterials.com

Basis Path Coverage:

Basis Path Coverage / Basis Path Testing(BPT) is the oldest structural testing technique. The technique is based on the control structure of the program. Based on the control structure, a flow graph is prepared and all the possible paths can be covered and executed during testing. The guidelines for effective path testing are:

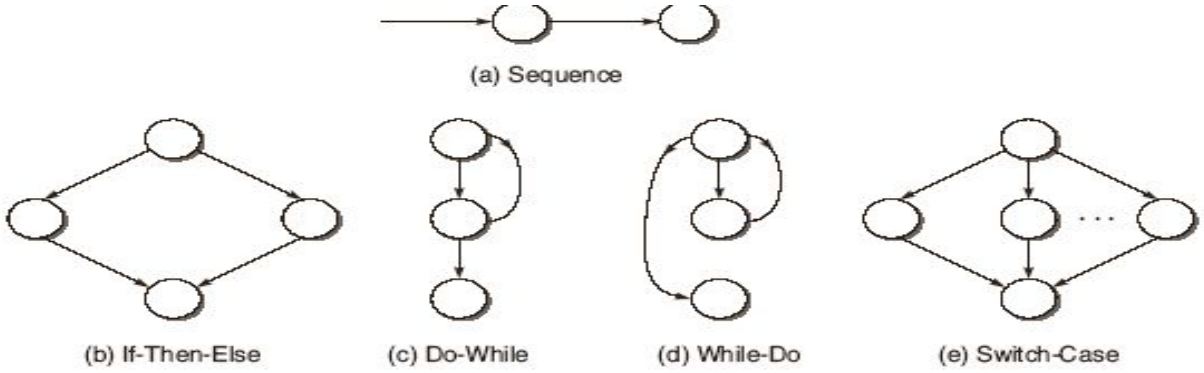
--Path Testing is based on control structure of the program for which flow graph is prepared.
--It requires complete knowledge of the program's structure.
--It is closer to the developer and used by him to test his module.
--The effectiveness of path testing is reduced with the increase in size of software under test.
--Choose enough paths in a program such that maximum logic coverage is achieved.

Control Flow Graph:

The control flow graph is a graphical representation of control structure of a program. Flow graph can be prepared as a directed graph. A Directed graph (V,E) consists of set of vertices V and edges E. Flow graph notations are:

--Node, Edges or links, Decision Node,
--Junction node :A node with more than one arrow entering it is called junction node
--Regions: Areas bounded by edges and nodes

Flow Graph Notations fro Different Programming Constructs: Flow graph is also known as Decision to Decision (D&D) graph.



Path Testing Terminology:

Path: A path through a program is a sequence of instructions or statements that starts at an entry, junction, or decision and ends at another or possibly the same junction, decision or exit.

Segment: Path consists of segments. The smallest segment is a link, that is, a single process that lies between two nodes.

Path Segment: A path segment is a succession of consecutive links that belongs to some path.

Length of a Path: The length of a path is measured by the number of links in it and not by the number of instructions or statements executed along the path.

Independent Path: An independent path is any path through the graph that introduces at least one new set of processing statements or new conditions.

Cyclomatic Complexity:

--It provides a quantitative measure of the logical complexity of a program

www.jntukmaterials.com

--Defines the number of independent paths in the basis set
--Provides an upper bound for the number of tests that must be conducted to ensure all statements have been executed at least once
--Can be computed three ways
-->The number of regions
-->V(G) = E – N + 2, E is the number of edges and N is the number of nodes in graph G
-->V(G) = P + 1, where P is the number of predicate nodes in the flow graph G

Guidelines for Basis Path Testing:
-- Draw the flow graph using the code provided for which we have to write test cases.
--Determine the cyclomatic complexity of the flow graph.
--Cyclomatic complexity provides the number of independent paths.
--Based on every independent path, choose the data such that this path is executed.

"Refer PPT for examples"

Mutation Testing:
Mutation testing is the process of mutating some segment of code(putting some error in the code) and then testing this mutated code with some test data. If the test data is able to detect the mutations in the code. Mutation testing helps a user create test data by interacting with the user to iteratively strengthen the quality of test data. During mutation testing, faults are introduced into a program by creating many versions of the program, each of which contains one fault. Test data are used to execute these faulty programs with the goal of causing each faulty program to fail.

Faulty programs are called *mutants* of the original program and a mutant is said to be *killed* when a test case causes it to fail. When this happens, the mutant is considered *dead*

Primary Mutants:
When the mutants are single they are called as primary mutants. Mutation operators are dependent on programming languages.

Example: if(a>b)
x = x+y;
else
x=x-y;
printf("%d",x);

Example mutants are:

M1: x = x – y; M2: x = x / y; M3: x = x+1;M4: printf("%d",y);

Secondary Mutants:

Example: if(a<b)
c=a;
M1: if(a<=b-1)
c=a;

M2: if(a+1 <=b)

www.jntukmaterials.com

```
    c=a;

M3: if(a==b)
    c=a+1;
```

Debugging:

Need for debugging :

Once errors are identified in a program code, it is necessary to first identify the precise program statements responsible for the errors and then to fix them. Identifying errors in a program code and then fix them up are known as debugging.

Debugging approaches

Brute Force Method:

This is the most common method of debugging but is the least efficient method. In this approach, the program is loaded with print statements to print the intermediate values with the hope that some of the printed values will help to identify the statement in error.

Backtracking:

In this approach, beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered.

Cause Elimination Method:

In this approach, a list of causes which could possibly have contributed to the error symptom is developed and tests are conducted to eliminate each.

Program Slicing:

This technique is similar to back tracking. Here the search space is reduced by defining slices. A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence value of that variable.

Debugging guidelines :

-- Many times debugging requires a thorough understanding of the program design.
-- Debugging may sometimes even require full redesign of the system.
-- One must be beware of the possibility that an error correction may introduce new errors. Therefore after every round of error-fixing, regression testing must be carried out.

Program Analysis Tool:

A program analysis tool means an automated tool that takes the source code or the executable code of a program as input and produces reports regarding several important characteristics of the program, such as its size, complexity, adequacy of commenting, adherence to programming standards, etc. We can classify these into two broad categories of program analysis tools:

Static Analysis tools :

Static analysis tool is also a program analysis tool. It assesses and computes various characteristics of a software product without executing it. The structural properties that are

www.jntukmaterials.com

usually analyzed are:
-- Whether the coding standards have been adhered to?
-- Certain programming errors such as uninitialized variables and mismatch between actual and formal parameters, variables that are declared but never used are also checked.

Code walk throughs and code inspections might be considered as static analysis methods. But, the term static program analysis is used to denote automated analysis tools. So, a compiler can be considered to be a static program analysis tool.

Dynamic program analysis tools :
Dynamic program analysis techniques require the program to be executed and its actual behavior recorded. A dynamic analyzer usually instruments the code (i.e. adds additional statements in the source code to collect program execution traces). The instrumented code when executed allows us to record the behavior of the software for different test cases. After the software has been tested with its full test suite and its behavior recorded, the dynamic analysis tool caries out a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete test suite for the program.

Integration testing

Introduction:

The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested.

Big-Bang Integration Testing :
It is the simplest integration testing approach, where all the modules making up a system are integrated in a single step. In simple words, all the modules of the system are simply put together and tested. However, this technique is practicable only for very small systems. The main problem with this approach is that once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. Therefore, debugging errors reported during big bang integration testing are very expensive to fix.

Bottom-Up Integration Testing :
In bottom-up testing, each subsystem is tested separately and then the full system is tested. A subsystem might consist of many modules which communicate among each

www.jntukmaterials.com

other through well-defined interfaces. The primary purpose of testing each subsystem is to test the interfaces among various modules making up the subsystem. Both control and data interfaces are tested. Large software systems normally require several levels of subsystem testing; lower-level subsystems are successively combined to form higher-level subsystems. A principal advantage of bottom-up integration testing is that several disjoint subsystems can be tested simultaneously.

Top-Down Integration Testing :

Top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested. Top-down integration testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test.

Mixed Integration Testing :

A mixed (also called sandwiched) integration testing follows a combination of top- down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level modules have been coded and unit tested. Similarly, bottom-up testing can start only after the bottom level modules are ready. The mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. In the mixed testing approaches, testing can start as and when modules become available. Therefore, this is one of the most commonly used integration testing approaches.

Testing Object-Oriented Programs:

What is suitable unit for Testing?
Do various OOP concepts makes testing easy?
Why are traditional techniques considered Unsatisfactory?

What is suitable unit for Testing?

-- Adequate testing of individual methods does not ensure that a class has been satisfactorily tested.
-- A method needs to be tested at all the states that the object can assume.
-- An object is considered as the basic unit of testing.

Do various OOP concepts makes testing easy?

-- Encapsulation: Encapsulation prevents the tester from accessing the data internal to the object.
-- Inheritance: Inherited methods may work in a new context (new method and new definitions). As a result, correct behavior of a method at an upper level, does not gaurenteee correct behaviour at lower level.
-- Dynamic binding: Dynamic binding was introduced to make the code compact, elegant, and easily extensible. However, as far as the testing is concerned all possible bindings of a mehtod call have to be identified and tested. This is not easy since the bindings take place at runtime.
-- Object States
-- Polymorphism

www.jntukmaterials.com

Why are traditional techniques considered Unsatisfactory?

-- Statement coverage based testing which is popular for testing procedural programs is not meaningful for OOP programs.
-- Different OOP features (Encapsulation, Inheritance ...) requires special testcases to be designed compared to the traditional testing.

System Testing:

Functionality Testing
Performance Testing

Functionality Testing:

System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:
• ***Alpha Testing:*** Alpha testing refers to the system testing carried out by the test team within the developing organization.
• ***Beta testing:*** Beta testing is the system testing performed by a select group of friendly customers.
• ***Acceptance Testing:*** Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

In each of the above types of tests, various kinds of test cases are designed by referring to the SRS document. Broadly, these tests can be classified into functionality and performance tests. The functionality tests test the functionality of the software to check whether it satisfies the functional requirements as documented in the SRS document.

Performance testing

Performance testing is carried out to check whether the system needs the non- functional requirements identified in the SRS document. There are several types of performance testing. Among of them nine types are discussed below. The types of performance testing to be carried out on a system depend on the different non-functional requirements of the system documented in the SRS document. All performance tests can be considered as black-box tests.

-- Stress testing
-- Volume testing
-- Compatibility testing
-- Regression testing
-- Recovery testing
-- Maintenance testing
-- Documentation testing
-- Usability testing
-- Error seeding

Stress Testing

Stress testing is also known as endurance testing. Stress testing evaluates system performance when it is stressed for short periods of time. Stress tests are black box tests which are designed to impose a range of abnormal and even illegal input conditions so as to stress the capabilities of the software. For example, suppose an operating system is supposed to support 15 multiprogrammed jobs, the system is stressed by attempting to run 15 or more jobs simultaneously.

www.jntukmaterials.com

Volume Testing
It is especially important to check whether the data structures (arrays, queues, stacks, etc.) have been designed to successfully extraordinary situations. For example, a compiler might be tested to check whether the symbol table overflows when a very large program is compiled.

Compatibility Testing :
This type of testing is required when the system interfaces with other types of systems. Compatibility aims to check whether the interface functions perform as required. For instance, if the system needs to communicate with a large database system to retrieve information, compatibility testing is required to test the speed and accuracy of data retrieval.

Regression Testing
This type of testing is required when the system being tested is an upgradation 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. However, 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.

Recovery Testing
Recovery testing tests the response of the system to the presence of faults, or loss of power, devices, services, data, etc. The system is subjected to the loss of the mentioned resources (as applicable and discussed in the SRS document) and it is checked if the system recovers satisfactorily. For example, the printer can be disconnected to check if the system hangs. Or, the power may be shut down to check the extent of data loss and corruption.

Maintenance Testing
This testing addresses the diagnostic programs, and other procedures that are required to be developed to help maintenance of the system. It is verified that the artifacts exist and they perform properly.

Documentation Testing
It is checked that the required user manual, maintenance manuals, and technical manuals exist and are consistent. If the requirements specify the types of audience for which a specific manual should be designed, then the manual is checked for compliance.

Usability Testing
Usability testing concerns checking the user interface to see if it meets all user requirements concerning the user interface. During usability testing, the display screens, report formats, and other aspects relating to the user interface requirements are tested.

Error seeding
Sometimes the customer might specify the maximum number of allowable errors that may be present in the delivered system. These are often expressed in terms of maximum number of allowable errors per line of source code. Error seed can be used to estimate the number of residual errors in a system.

www.jntukmaterials.com

Some General Issues Associated with Testing:

Test Documentation
Regression Testing

Test Documentation:

Software testing is an essential portion of Software Development Life Cycle. In current's testing process, project requires planned and serialized documentation for testing and development. Because of this most of companies concentrate on creating documentation of software development process.

Proper documentation is the only key thing that can make it possible and makes testing more accurate in an organization. A project's documentation makes testing process easy and organized, also saves company money and time spent on that project. Proper documentation makes easy for the client to review the software process.

Fact is that, careful documentation can save an organization's time, efforts and money. For any bigger company their product or software successfully get releases with their proper documentation that makes easy for any user to understand the product. So that proper documentation has become major reason of software development.

Regression Testing:

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

Need of Regression Testing

Regression Testing is required when there is a

- Change in requirements and code is modified according to the requirement
- New feature is added to the software
- Defect fixing
- Performance issue fix