UNIT-IV TESTING

A strategic Approach for Software testing:

Software Testing is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specifies requirements or not.

The main objective of software testing is to design the tests in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for

the development of the software. The overall strategy for testing software includes:

1. **Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner.**

Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.

NRCM your roots to success...

2. Specifying the objectives of testing in a clear and detailed manner.

Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.

3. For the software, identifying the user's category and developing a profile for each user.

Use cases describe the interactions and communication among different classes of users and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.

4. Developing a test plan to give value and focus on rapid-cycle testing.

Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing.

5. Robust software is developed that is designed to test itself.

The software should be capable of detecting or identifying different classes of errors. Moreover, software design should allow automated and regression testing which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.

6. Before testing, using effective formal reviews as a filter.

Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.

7. Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.

The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.

8. For the testing process, developing a approach for the continuous development.

As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

Testing Strategies for Conventional Software

There are many strategies that can be used to test software.
At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors.
This approach simply does not work. It will result in buggy software.
At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed.
This approach, although less appealing to many, can be very effective.

Types:

1. Unit Testing
2. Integration Testing
3. Validation Testing and
4. System Testing

1. Unit Testing:

Unit testing is a type of software testing where individual units or components of a software are tested. It is concerned with functional correctness of the standalone modules. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

Why Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

your roots to success...

Here, are the key reasons to perform unit testing:

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the code base and enables them to make changes quickly
3. Good unit tests serve as project documentation

4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

5. Integration Testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

Types of Integration Testing

Integration testing can be classified into two parts:

1. **Incremental integration testing**
2. **Non-incremental integration testing**

NRCM
your roots to success...

Incremental Approach

In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

Incremental integration testing is carried out by further methods:

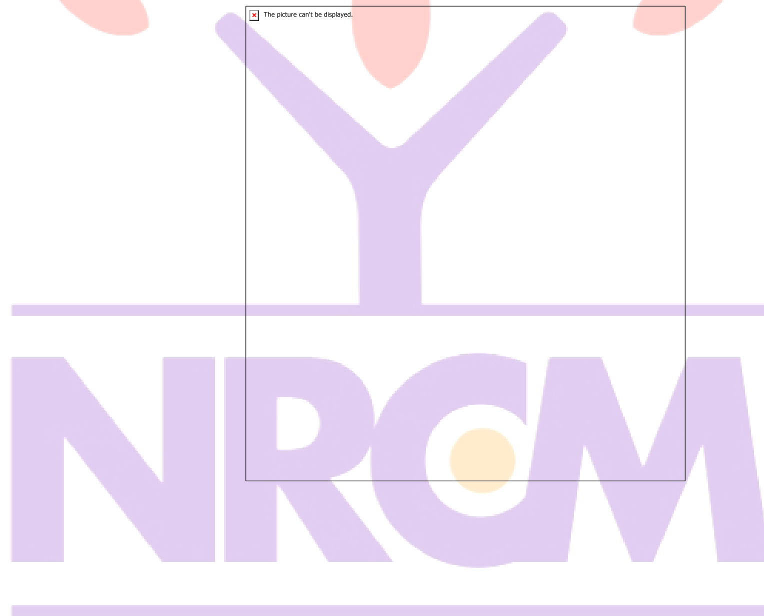
1. Top-Down approach
2. Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.

NRCM
your roots to success...

In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on as we can see in the below image:



Advantages:

3.Identification of defect is difficult.
4.An early prototype is possible.

Disadvantages:

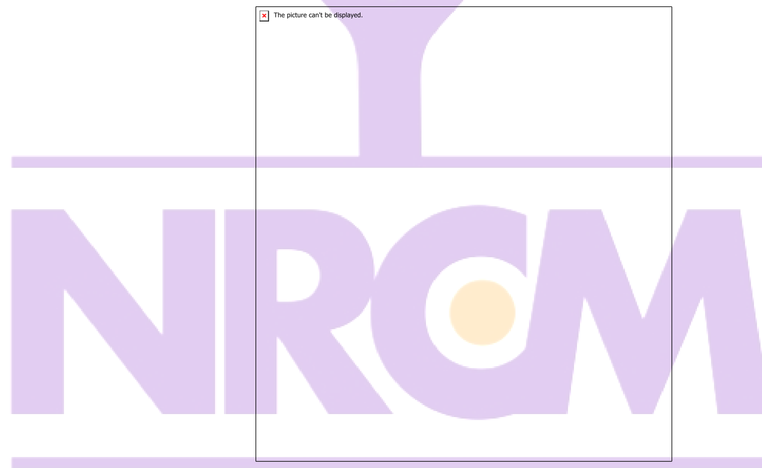
5.Due to the high number of stubs, it gets quite complicated.
6.Lower level modules are tested inadequately.
7.Critical Modules are tested first so that fewer chances of defects.

Bottom-Up Method

NRCM your roots to success...

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.

In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Advantages

8.Identification of defect is easy.
9.Do not need to wait for the development of all the modules as it saves time.

Disadvantages

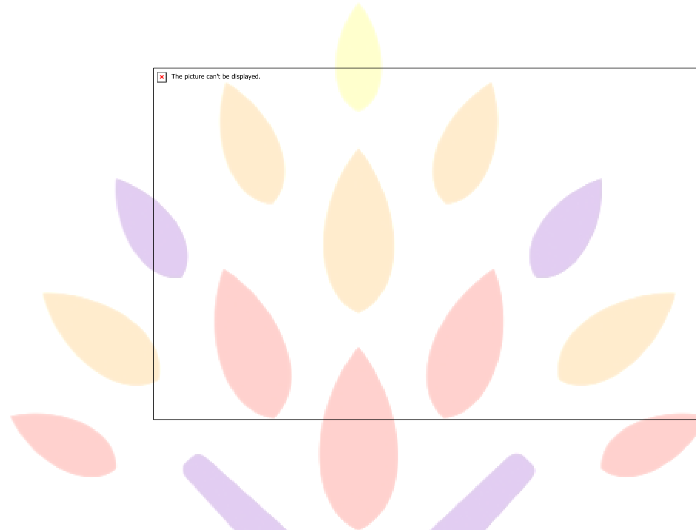
10. Critical modules are tested last due to which the defects can occur.
11. There is no possibility of an early prototype.

In this, we have one addition approach which is known as **hybrid testing**.

your roots to success...

Hybrid Testing Method

In this approach, both **Top-Down** and **Bottom-Up** approaches are combined for testing. In this process, top-level modules are tested with lower level modules and lower level modules tested with high-level modules simultaneously. There is less possibility of occurrence of defect because each module interface is tested.



Advantages

12. The hybrid method provides features of both Bottom Up and Top Down methods.
13. It is most time reducing method.
14. It provides complete testing of all modules.

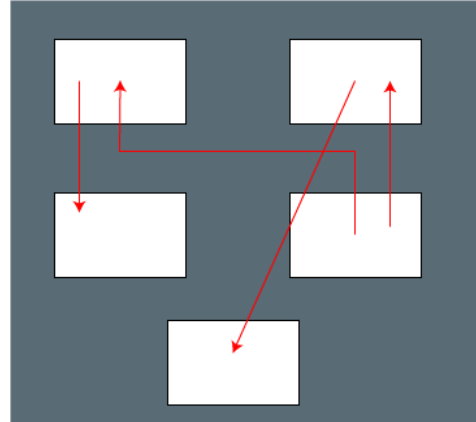
Disadvantages

15. This method needs a higher level of concentration as the process carried out in both directions simultaneously.
16. Complicated method.

Non- incremental integration testing

We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.

NRCM
your roots to success...



6. Validation Testing

Verification and Validation Testing

Verification testing

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.

Validation testing

Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).

Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.

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

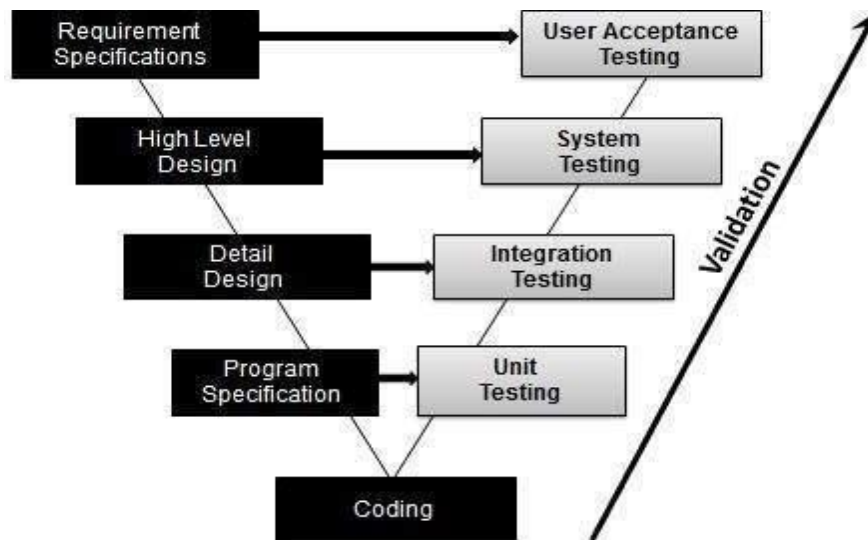
Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

NRCM
your roots to success...

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



7. System Testing

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as **System testing**. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

It is **end-to-end testing** where the testing environment is similar to the production environment.

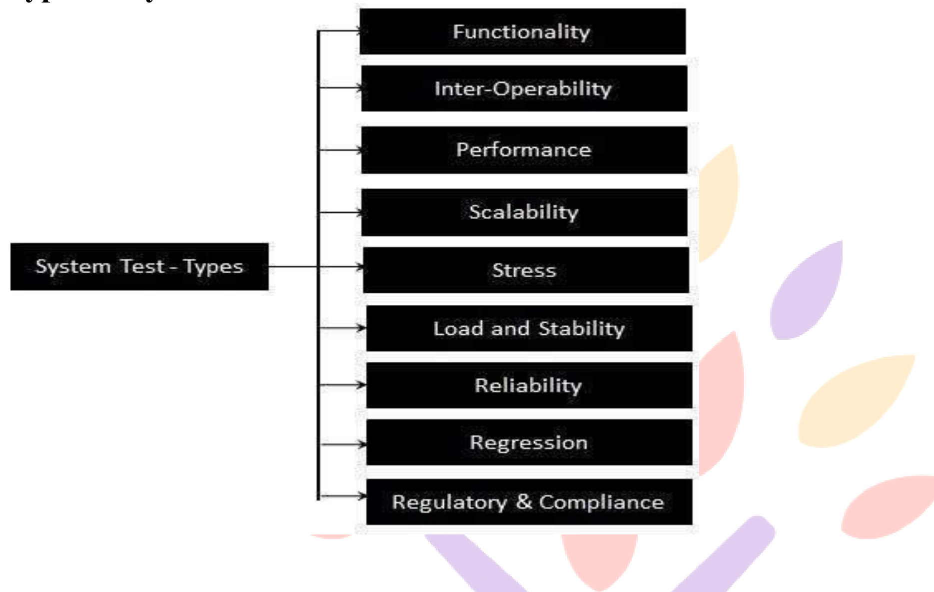
System Testing includes the following steps.

1. Verification of input functions of the application to test whether it is producing the expected output or not.

NRCM
your roots to success...

2. Testing of integrated software by including external peripherals to check the interaction of various components with each other.
3. Testing of the whole system for End to End testing.
4. Behavior testing of the application via a user's experience

Types of System Tests:



Software Testing

1. Two major categories of software testing
1. Black box testing
2. White box testing

Black box testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

How to do Black Box Testing?

Here are the generic steps followed to carry out any type of Black Box Testing.

1. Initially, the requirements and specifications of the system are examined.
2. Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
3. Tester determines expected outputs for all those inputs.
4. Software tester constructs test cases with the selected inputs.
5. The test cases are executed.

your roots to success...

6. Software tester compares the actual outputs with the expected outputs.
7. Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

Functional testing - This black box testing type is related to the functional requirements of a system; it is done by software testers.

Non-functional testing - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

Regression testing - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

Equivalence Class Partitioning: It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

Boundary Value Analysis: Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.

Decision Table Testing: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

your roots to success...

Equivalence Partitioning Testing

Equivalence Partitioning is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc. also called as equivalence class partitioning. It is abbreviated as ECP. It is a software testing technique that divides the input test data of the

application under test into each partition at least once of equivalent data from which test cases can be derived.

An advantage of this approach is it reduces the time required for performing testing of a software due to less number of test cases.

Example:

The Below example best describes the equivalence class Partitioning:

Assume that the application accepts an integer in the range 100 to 999 Valid Equivalence Class partition: 100 to 999 inclusive.

Non-valid Equivalence Class partitions: less than 100, more than 999, decimal numbers and alphabets/non-numeric characters.

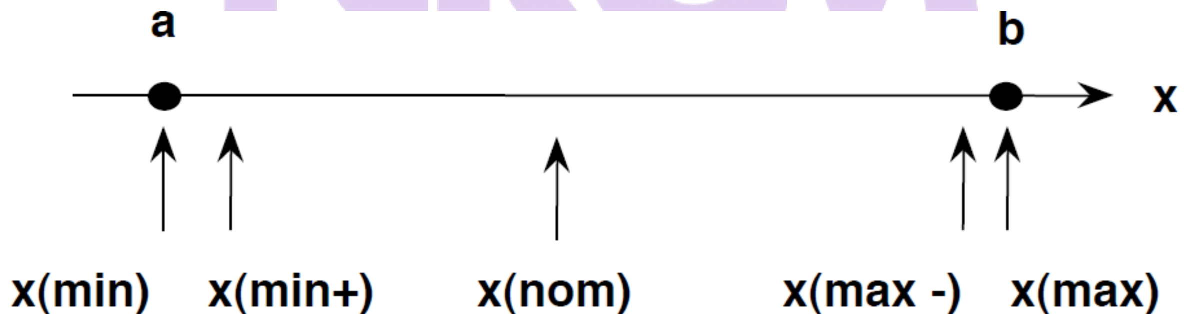
Boundary Value Analysis

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

1. So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".

2. The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



Example: Input Box should accept the Number 1 to 10

Here we will see the Boundary Value Test Cases

Test Scenario Description	Expected Outcome
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept

Interpretation:

1. Case 1 – Username and password both were wrong. The user is shown an error message.
2. Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
3. Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
4. Case 4 – Username and password both were correct, and the user navigated to homepage While converting this to test case, we can create 2 scenarios,
5. Enter correct username and correct password and click on login, and the expected result will be the user should be navigated to homepage

And one from the below scenario

6. Enter wrong username and wrong password and click on login, and the expected result will be the user should get an error message
7. Enter correct username and wrong password and click on login, and the expected result will be the user should get an error message
8. Enter wrong username and correct password and click on login, and the expected result will be the user should get an error message

White Box Testing:

White box testing is a testing technique that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

your roots to success...

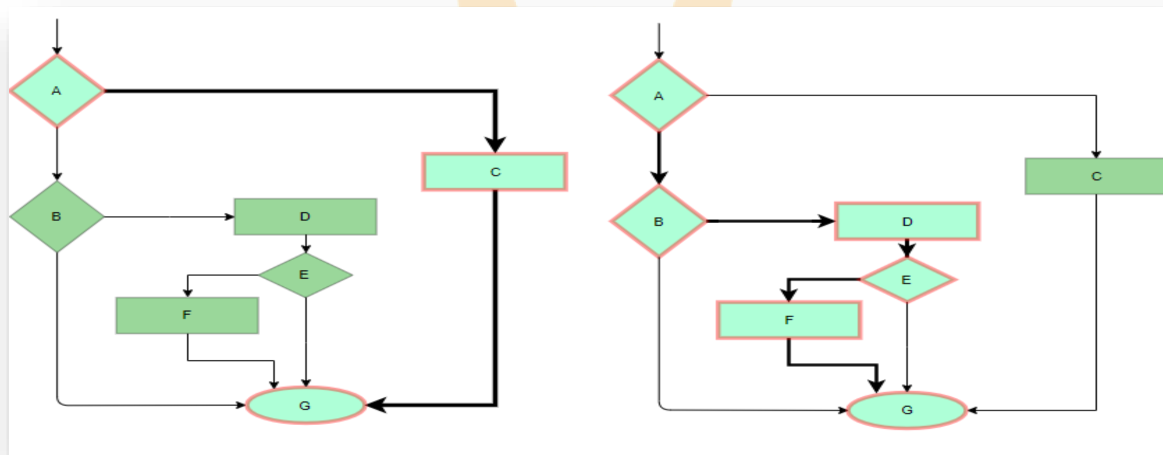
White Box Testing Techniques:

9. **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
10. **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.

11. **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch is covered.

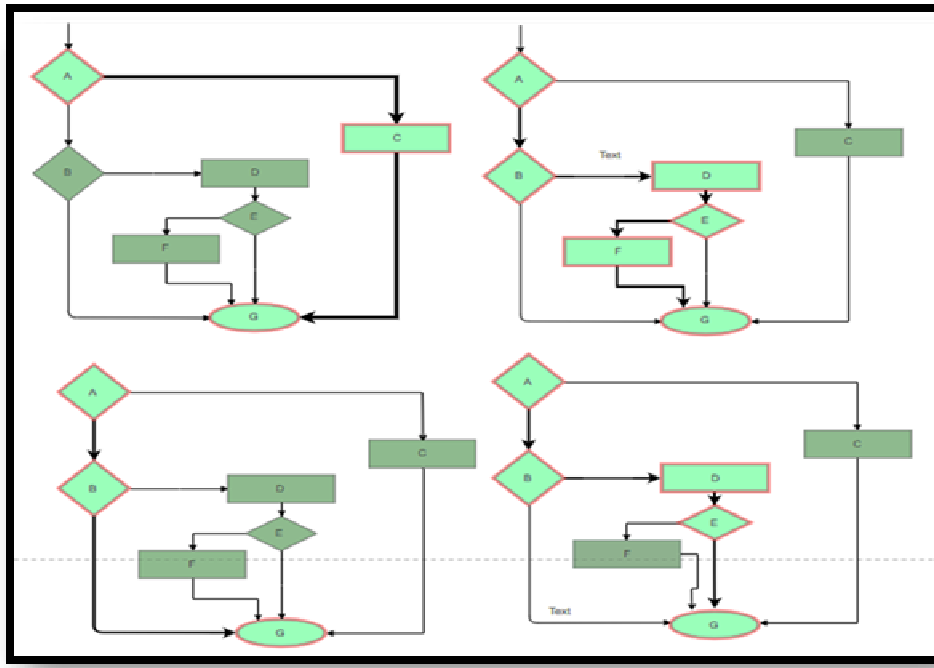
Statement coverage:

In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



Statement Coverage Example

Branch Coverage: In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



4 test cases required such that all branches of all decisions are covered, i.e, all edges of flowchart are covered

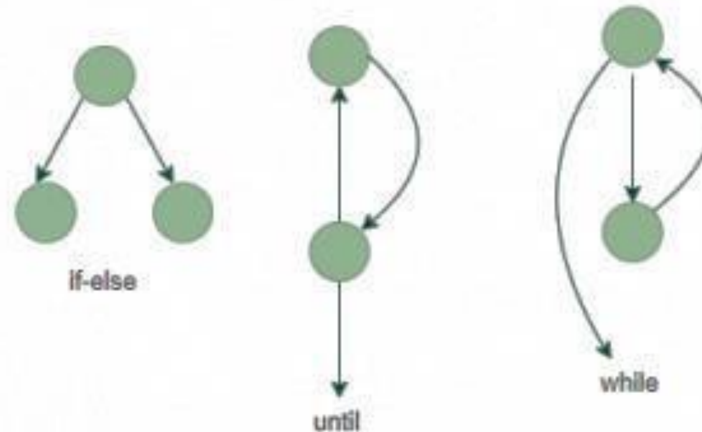
Basis Path Testing: In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

Steps:

1. Make the corresponding control flow graph
2. Calculate the cyclomatic complexity
3. Find the independent paths
4. Design test cases corresponding to each independent path

NRCM
your roots to success...

Flow graph notation: It is a directed graph consisting of nodes and edges. Each node represents



a sequence of statements, or a decision point. A predicate node is the one that represents a decision point thatcontains a condition after which the graph splits. Regions are bounded by nodes and edges.

Metrics for Process and Products

Software Measurement: A measurement is a manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process.

It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement:

There are 2 types of software measurement:

1. Direct Measurement:
In direct measurement the product, process or thing is measured directly using standard scale.

2. Indirect Measurement:
In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

NRCM your roots to success...

Metrics:

A metrics is a measurement of the level that any impute belongs to a system product or process. There are 4 functions related to software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

Characteristics of software Metrics:

1. Quantitative:
Metrics must possess quantitative nature. It means metrics can be expressed in values.

2. Understandable:
Metric computation should be easily understood, the method of computing metric should be clearly defined.

3. Applicability:
Metrics should be applicable in the initial phases of development of the software.

4. Repeatable:
The metric values should be same when measured repeatedly and consistent in nature.

5. Economical:
Computation of metric should be economical.

6. Language Independent:
Metrics should not depend on any programming language.

Metrics for software quality:

Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

NRCM
your roots to success...

Software quality metrics can be further divided into three categories −

1. Product quality metrics
2. In-process quality metrics
3. Maintenance quality metrics

Product Quality Metrics

This metrics include the following −

4.Mean Time to Failure
5.Defect Density
6.Customer Problems
7.Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**. PUM = Total Problems that customers reported (true defect and non-defect oriented problems) for a time period + Total number of license months of the software during

the period Where,

Number of license-month of the software = Number of install license of the software × Number of months in the calculation period

PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year.

Customer Satisfaction

Customer satisfaction is often measured by customer survey data through the five-point scale −

8. Very satisfied
9. Satisfied
10. Neutral
11. Dissatisfied
12. Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis. For example −

your roots to success...

13. Percent of completely satisfied customers
14. Percent of satisfied customers
15. Percent of dis-satisfied customers
16. Percent of non-satisfied customers Usually, this percent satisfaction is used.

In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes −

17. Defect density during machine testing
18. Defect arrival pattern during machine testing
19. Phase-based defect removal pattern
20. Defect removal effectiveness

Defect density during machine testing

Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.

This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following −

21. The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.
22. The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.
23. The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

Phase-based defect removal pattern

NRCM your roots to success...

This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management. **Defect removal effectiveness**

It can be defined as follows –

$$DRE = \frac{\text{Defect removed during a development phase}}{\text{Defects latent in the product}} \times 100\%$$

This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used for the front-end and **phase effectiveness** for specific phases. The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

24. Fix backlog and backlog management index
25. Fix response time and fix responsiveness
26. Percent delinquent fixes
27. Fix quality

Fix backlog and backlog management index

Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.

Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

your roots to success...

$$BMI = \frac{Number\ of\ problems\ closed\ during\ the\ month}{Number\ of\ problems\ arrived\ during\ the\ month} \times 100\%$$

If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

Fix response time and fix responsiveness

The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.

The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

Percent delinquent fixes

It is calculated as follows −

$$Percent\ Delinquent\ Fixes = \frac{Number\ of\ fixes\ that\ exceeded\ the\ response\ time\ criteria\ by\ ceverity\ level}{Number\ of\ fixes\ delivered\ in\ a\ specified\ time} \times 100\%$$

Fix Quality

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.

A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.

Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

your roots to success...