# mood-book

# Testing Strategies



⇒ Software Testing Introduction:-

→ s/w testing is a critical element of s/w quality assurance and represents the ultimate review of specification, design & coding.

→ The purpose s/w testing is to ensure whether the s/w functions appears to be working according to specifications and performance requirements.

→ The objective of testing is a process of executing a program with the intend of finding an error.

→ A good test case is one that has high probability of finding an undiscoved error.

→ All tests should be traceble to customer requirements.

→ Generally, testing is a process that requires more effort than any other s/w engineering activity.

⇒

→ Various testing phases are

1. Test planning: The test plan or test script is prepared. These are generated from requirements analysis document and program code.

2. Test case Design:- The goal of test case design is to create a set of tests that are effective in testing.

3. Test execution: The test data is derived through various test cases in order to obtain the test results.

4. Data collection: The test results are collected and verified.

5. Effective evaluation: All the above test activities are performed on the s/w model and the maximum no. of errors are uncovered.

→ The testing strategy provides a process that provides describes for the developer, quality analysts and the customer the steps conducted as part of testing.

* → The strategic approach for sw testing can be -

1. Just before starting the testing process the "formal technical reviews" must be conducted. This will eliminate many errors before the actual testing process.
2. At the beginning, various components of the system are tested, then gradually each interface is tested & thus the entire computer based system is tested.
3. Different testing techniques can be applied at different point of time.
4. The developer of the sw conducts testing. For the large projects the Independent Test Groups (ITG) also assist the developers.
5. Testing & debugging are different activities that must be carried out in sw testing.
6. Debugging also lies within any testing strategy.

There are two levels specified in the testing strategy and those are "low level" and "high level".

→ The low level tests that are necessary to verify small source code segment has been correctly implemented.

→ The high level tests should be conducted that are validate major system functions against customer requirements.

→ Difference between verification & validation:-

| Verification | validation. |
|---|---|
| * Verification refers to set of activities that ensure s/w correctly implements the specific function. | * validation refers to the set of activities that ensure that the s/w that has been built is traceble to customer requirements. |
| * According to Boehm verification says " Are we building the product right"? | * According Boehm the validation says "Are we building the right product? |
| * After a valid and complete specification the verification starts. | * validation begins as soon as project starts. |
| * The verification is conducted to ensure whether s/w meets spe-fication or not. | * validation is conducted to show that the user requirements are getting satisfied. |

→ s/w testing is only one element of SQA.

→ Verification and validation involve large no. of s/w quality assurance activities such as-

1. Formal technical reviews
2. Quality & configuration measurements
3. performance monitoring
4. Feasibility study
5. Documentation review
6. Database review
7. Algorithmic analysis
8. Development testing
9. Installation testing.

## ⇒ Testing strategies

→ We begin by "testing-in-the-small" & move toward "testing-in-the large".

→ Various testing strategies for conventional s/w are

1. unit testing
2. Integration testing
3. validation testing
4. system testing.

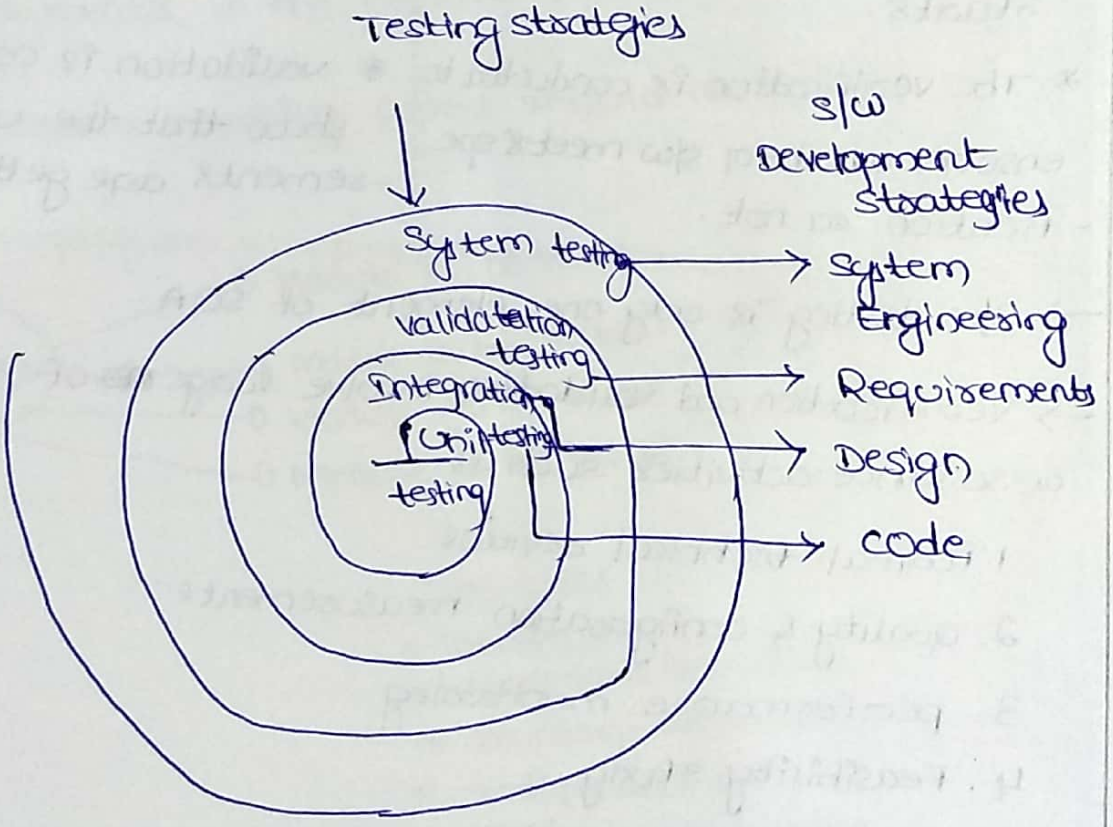



fig. testing strategy.

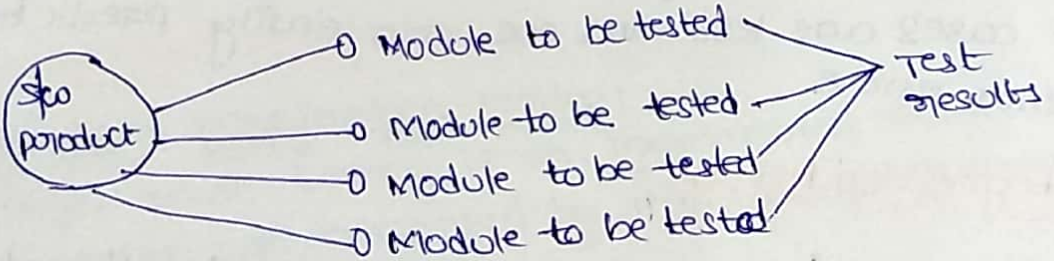
## ⇒ unit testing :-

→ In unit testing the individual components are tested independently to ensure that their quality.

→ The focus is to uncover the errors in design & implementation.

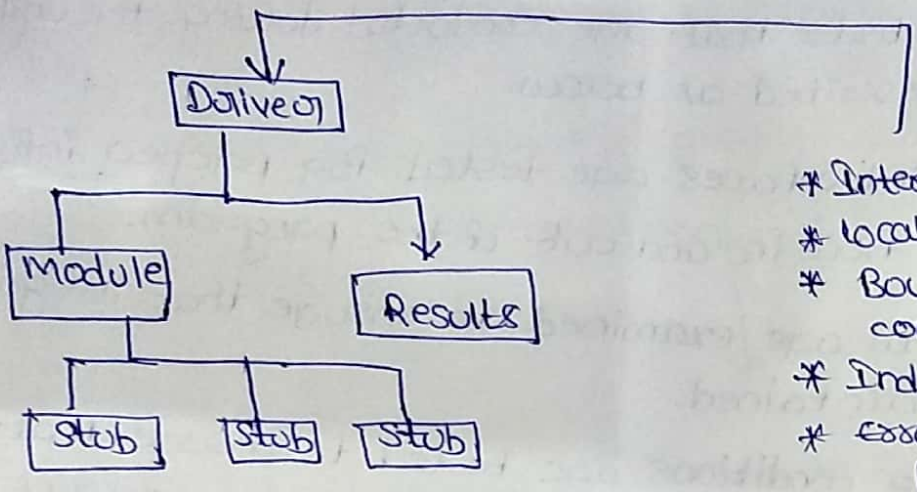
→ The various tests that are conducted during the unit test are described as below.

* Module interfaces are tested for proper info -rmation flow in and out of the program.
* Local data are examined to ensure that integri -ty is maintained.
* Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
* All the basis paths are tested for ensuring that all statements in the module have been executed only once.
* All error handling paths should be tested.




Test cases
* interfaces
* Local data structures
* Boundary conditions
* Independent path
* Error handling paths.

→ Drivers and stub s/w need to be developed to test incomplete s/w. The "driver" is a program that accepts the test data & prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.




* Interfaces
* local data structure
* Boundary conditions
* Independent paths
* error handling paths.


fig unit testing environment

→ The unit testing is simplified when a component with high cohesion is designed. In such a design the no. of test cases are less and one can easily predict or uncover errors

## ⇒ Integration Testing:-

→ A group of dependent components are tested together to ensure their quality of their integration unit.

→ The objective is to take unit tested components and build a program structure that has been dictated by s/w design.

→ The focus of integration testing is to uncover errors in

* Design and construction of s/w architecture.
* Integrated functions or operations at subsystem level.
* Interfaces and interactions b/w them.
* Resource integration.

→ The integration testing can be carried out using 2 approaches.

1. Non-Incremental Integration → Bigbang
2. Incremental Integration → Top-down testing
   → Bottom-up integration
   → Regression testing
   → Smoke testing.

Non-Incremental Integration

*** → The Non-incremental integration is given by the "bigbang" approach. All components are combined in advance.

→ the entire program is tested as a whole. And chos usually results.

→ A set of errors is tested as a whole. correction is diffi -cult because isolation of causes is complicated by the size of the entire program. once these errors are corre -cted new once appear. this process continues infinitely.

Incremental Integration:-

1) Top-down Integration-testing:

→ Top-down testing is an incremental approach in which modules are integrated by moving down through the control structure.

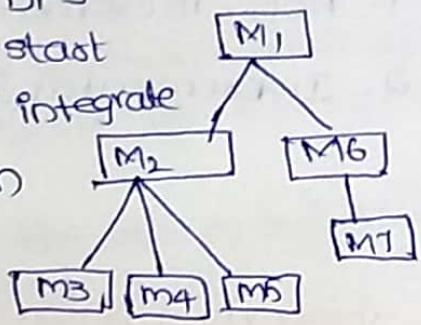
→ Module subordinate to the main control module are inco -rporated into the system in either DFS or BFS

→ In top-down Integration process can be performed us -ng following steps:

1. the main control module is used as a test driver & the stubs are substituted for all modules directly subordinate to the main control module.
2. subordinate stubs are replaced one at a time with actual modules using either DFS or BFS.
3. Tests are conducted as each module is integrated.
4. on completion of each set of tests, another stub is replaced with the real module.

→ Regression testing is conducted to prevent the introduction of new errors.

Eg: In top-down integration if DFS approach is adopted then we will start integration from M1 then we will integrate M2, then M3, M4, M5, M6 and then M7.




If BFS approach is adopted then we will integrate module M1 first then M2, M6, then we will integrate module M3, M4, M5 and finally M7.

M1

⇒ Bottom-Up-Integration Testing:

→ In bottom-up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

→ The bottom-up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific sw subfunction.
2. A driver program is written to co-ordinate test-case i/p & o/p.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

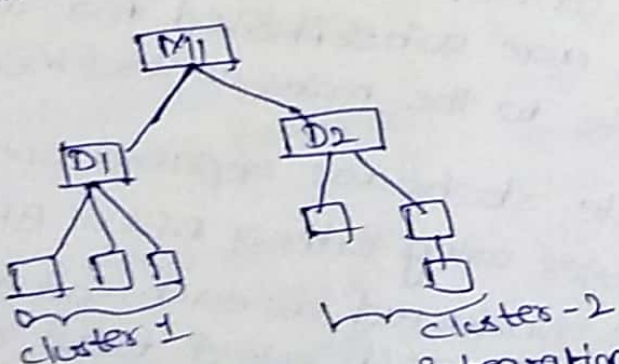



fig: Bottom-up integration testing

⇒ Following activities need to be carried out in smoke testing.

1) software components already translated into code are integrated into a "build." The "build" can be data files, libraries, reusable modules or program components.
2) A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.
3) The "build" is integrated with the other builds and the entire product is smoke tested daily.

⇒ Validation Testing:-

→ The integrated s/w is tested based on requirements to ensure that the desired product is obtained.

→ In validation testing the main focus is to uncover errors in

* System i/p or o/p
* System functions & information
* user interfaces
* system behaviour & performance

⇒ Acceptance Testing:-

The acceptance testing is a kind of testing conducted to ensure that the s/w works correctly in the user work environment. The acceptance testing can be conducted over a period of weeks or months.

Difference b/w Alpha & Beta testing:-

| Alpha testing | Beta Testing |
|---|---|
| 1. This testing is done by a developer or by a customer under the supervision of developer in company premises. | 1. This testing is done by the customer without any interface of developer and is done at customer's place. |
| 2. Sometime full product is not tested using alpha testing & only core functionalities are tested. | 2. The complete product is tested under this testing. such product is usually given as free trail version. |

→ First components are collected together to form cluster1 and cluster2. Then each cluster is tested using a driver program.

→ The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules

⇒ Regression Testing:-

* Regression testing is used to check for defects propaga -ted to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
* there are three different classes of test cases involved in regression testing.
  * Representative sample of existing testcases is used to exercise all s/w functions.
  * Additional test cases focusing s/w functions likely to be affected by the change.
  * tests cases that focus on the changed s/w compo -nents.

→ After product had been deployed, regression testing could be necessary because after a change has been made to the product an error that can be discovered and it should be corrected.

→ similarly for for deployed product addition of new features may be requested and implemented.

⇒ 4. smoke testing:-

→ the smoke testing is a kind of integration testing technique used for the time caitical projects wherein the project need to be assessed on frequent basis.

## Black-Box testing:-

→ Black-box testing is defined as a testing technique in which functionality of the Application under test(AUT) is testing.

→ In black-box-testing the application is tested without looking at the internal code structure, implementation details and knowledge of internal paths of s/w.

→ This type of testing is based on entirely on s/w requirements and specifications.


→ In black-box testing we just focus on inputs & outputs of the s/w system without bothering about internal knowledge of the s/w program.

Eg:- o/s like windows, website like google.

→ under black-box testing, you can test these applications by just focusing on the i/p's & o/p's without knowing their internal code implementation.

### How to do black-box testing:

1. Initially, the requirements & specifications of the system are examined.
2. Tester chooses valid i/p's (positive test scenario) to check whether system processes them correctly. Also, some invalid i/p's (negative test scenario) are chosen to verify that the system is able to detect them.

→ Tester determines expected o/p's for all those i/p's.
→ s/w tester constructs test cases with the selected i/p's.
→ Test cases are executed.
→ s/w tester compares the actual o/p's with the expected o/p's
→ Defects if any are fixed and re-tested.

Types of black-box testing:-

1) Functional testing: This type of testing is related to functional requirements of a system.

2) Non-functional testing: This type of testing is not related to testing of specific functionality. but non-functional requirements such as performance, scalability etc.

3) 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:-

i) Equivalence class testing:-
→ Equivalence partitioning is also known as equivalence class partitioning.
→ In equivalence partitioning, inputs to the s/w or system are divided into groups.
→ Each & every condition of particular partition works as same as other. If a condition in a partition valid, other conditions are valid too. If a condition in partition in invalid, other conditions are invalid too.
→ It helps to reduce the total no. of testcases from infinite to finite. The selected testcases from these groups ensure coverage all possible scenarios.

Guidelines for equivalence partitioning.

* If i/p condition specifies a range, one valid and two invalid equivalence age defined
* If an i/p condition specifies a specific value, one valid & two invalid equivalence classes are defined.
* If an i/p condition specifies a member of a set, one valid & one invalid equivalence class is defined.
* If an i/p condition is boolean, one valid & one invalid equivalence class is defined.

Eg: We have to test a field which accepts age 18-56.

Age [Enter age] * Accepts value 18 to 56

| Equivalence partitioning | | |
|---|---|---|
| Invalid | valid | Invalid |
| <=17 | 18-56 | >=57 |

Valid i/p : 18-56

Invalid i/p : <=17 & >=57

Valid class: 18-56 = pick any one i/p test data from 18-56.

Invalid class1: <=17 pick any one i/p test data less than or equivalence to 17

Invalid class 2 : >=57 pick any one i/p test-data greater than or equal to 57

2) Boundary value Analysis:-

→ Boundary value analysis is done to check boundary conditi-ons.

→ In this testing technique in which the elements at the edge of the domain are selected & tested.

→ using boundary value analysis, instead of focusing on i/p conditions only, the test cases from o/p domain are also derived.

⇒ Test cases for BVA accepting numbers b/w 1 & 1000 using BVA.

1) Test cases with test data exactly as the i/p boundaries of i/p domain i.e values 1 & 1000 in our case.

2) Test data with values just below the extreme edges of i/p domains i.e values 0 & 999.

3) Test data with values above the extreme edges i/p domain i.e values 2 & 1000|

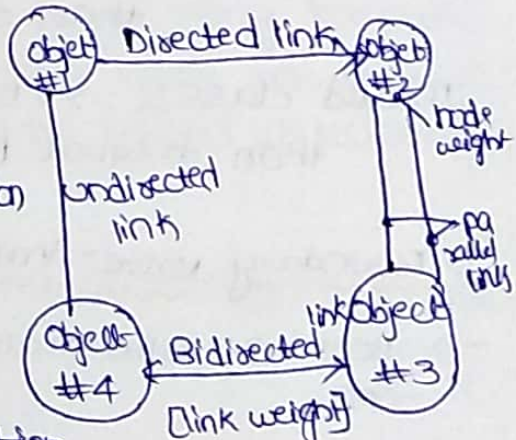
→ Boundary value analysis is often called as a part stress and negative testing.

3) Graph-based testing:-

→ In the graph based testing, a graph of objects present in the system is created.

→ The graph is basically a collection of nodes & links. Each node represents the object that is participating in the s/w system and links represents the relationship among these objects.

→ The node weight represents the properties of object & link weight represents the properties or characteristics of the relationship of the objects.




→ After creating the graph, important objects & their relationships are tested.

## ⇒ White-box testing :-

→ White-box testing is defined as the testing of a s/w solution's internal structure, design, and coding.

→ In this type of testing, coding is visible to the tester.

→ It focuses primarily on verifying the flow of i/p's & o/p's through the application, improving design & usability

→ White-box testing is also known as clear box testing, open-box testing, structural testing, glass-box testing, Transparent-box testing, code-based testing.

→ White-box testing involves the testing of the s/w code for the following:

* Internal security holes.
* Broken or poorly structured paths in the coding processes.
* The flow of specific inputs through the code.
* Expected output.
* The functionality of conditional loops.
* Testing of each statement, object, & function on an individual basis.

Testing techniques:-

i) Basis path testing:

→ In this method the procedural design using basis set of execution path is tested. This basis set ensures that every execution path will be tested at least once.

⇒ Flowgraph Notation:-

→ path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

→ Following are the steps that are carried out while performing path testing.

step1: Design the flowgraph for the program or a component.

step2: calculate the cyclomatic complexity.

step3: select a basis set of path.

step4: Generate testcases for these paths.

Eg:-

Step1: Design the flowgraph for the program or a component.

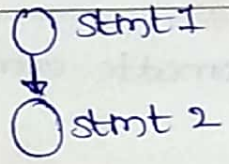
flowgraph is a graphical representation of logical control flow of the program. such a graph consists of circle called a flowgraph node which basically represents one or more procedural statements. and arrows called as edges or links. which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called regions.

→ Eg: program for searching a number using binary search method. Draw a flow graph for the same.

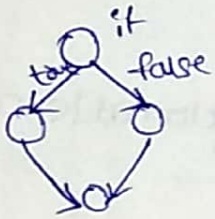
```
Void search (int key, int n, int a[])
{
    int mid;
1   int bottom = 0;
2   int top = n-1;
3   while (bottom ≤ top)
    {
4       mid = (top + bottom) / 2;
5       if (a[mid] == key)
        {
6           printf ("Element is present");
```

```
7 return ;
  } // end of IF
  else
  {
8 if (a[mid] < key)
9 bottom = mid + 1;
  else
10 top = mid - 1;
  } // end of else
  } // end of while
11 } // end of search
```

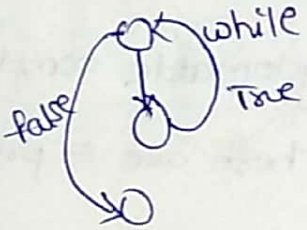
sequence



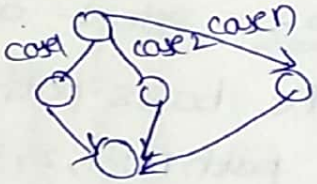

if-else



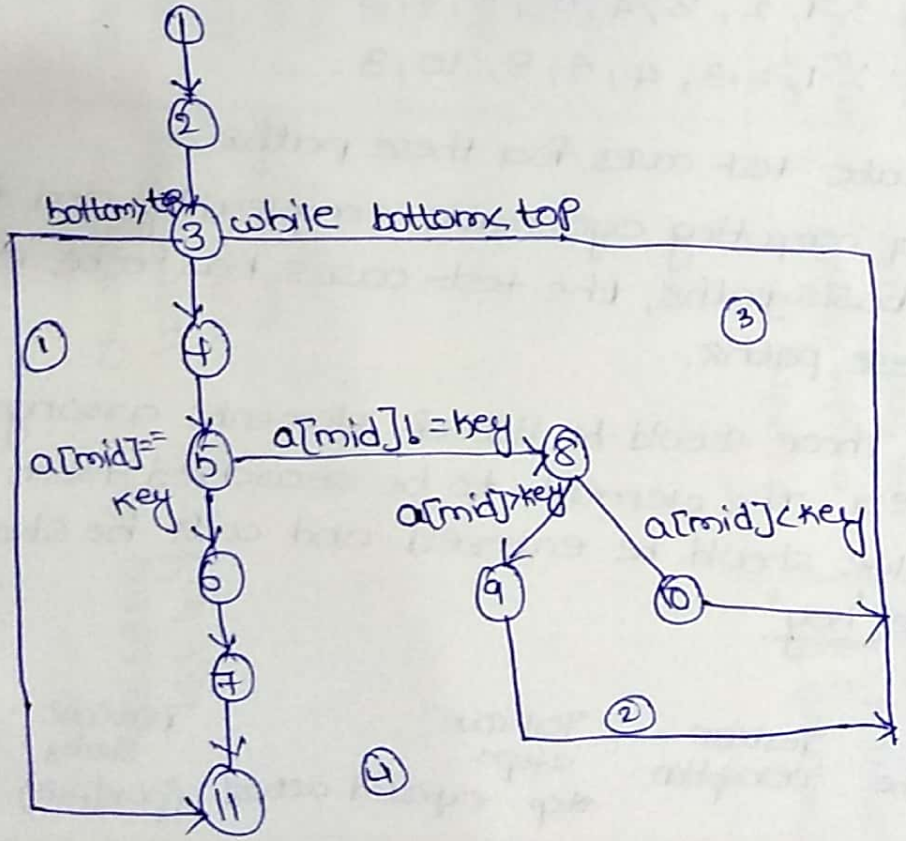

while




case




→ flow graph will be for binary search




Step 2: calculate the cyclomatic complexity.

→ The cyclomatic complexity can be computed by 3 ways

1) cyclomatic complexity = Total no. of regions in the flo -w graph = 4

2) cyclomatic complexity = E - N + 2
= 13 - 11 + 2
= 2 + 2
= 4

3) cyclomatic complexity = P + 1 = 3 + 1 = 4
There are 3 predicate nodes 3, 5, 8.

Step 3:- select a basis set of path.

The basis paths are

path 1 : 1, 2, 3, 4, 5, 6, 7, 11
path 2 : 1, 2, 3, 11
path 3 : 1, 2, 3, 4, 5, 8, 9, 3
path 4 : 1, 2, 3, 4, 5, 8, 10, 3 - -

Step 4: Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be execu -ted for these paths.

precondition: There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable "key".

| Test Case id | Test case name | Testcase Description | Test-case steps | | Testcase status (pass/fail) | test priority |
|---|---|---|---|---|---|---|
| | | | step | expected | actual | |

Defect severity.

| Test case id | Test case name | Test case Description | Test steps | | Test case status | Test stats (P/F) | TP | DS |
|---|---|---|---|---|---|---|---|---|
| | | | step | E A | | | | |
| 1. | validating the list boundary | checking the bottom & top values for the list of elements | set bottom=0 top = n-1 check if bottom<= top by while loop. This condition defines the length of the list from which the key searched. | Initially bottom≤ top will be true. But during iterations list's length will be reduced & if entire list gets scanned at one point (bottom≥ top) will be reached then return to main. | Design | | | |
| 2. | checking list element with key | checking if middle element of array is equal to key value | set mid = (top+bottom)/2 Then compare if a[mid] is equal to key. | If a[mid]=key value then print message "element is present" & return to main | Design | | | |

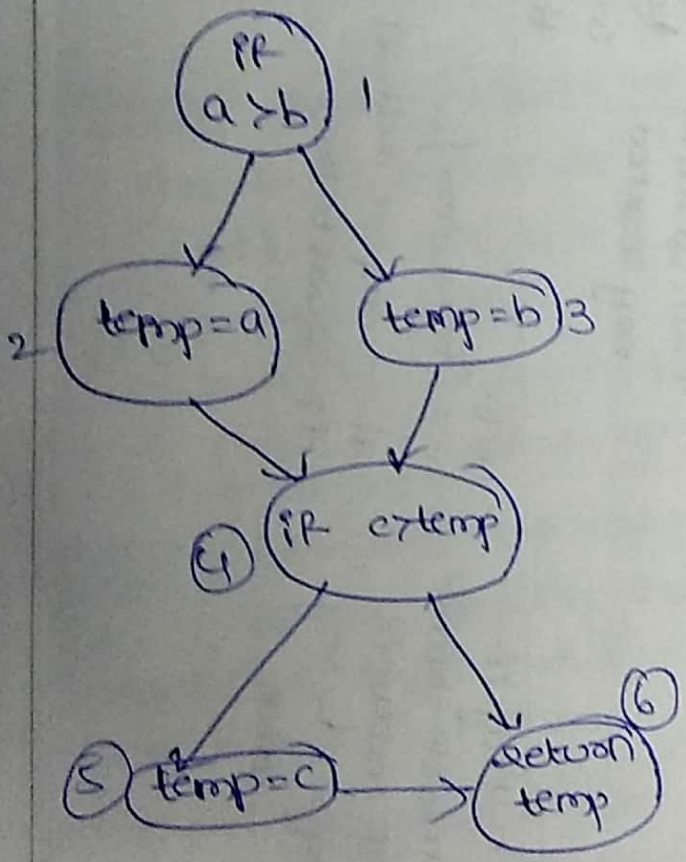
2) Graph Matrices:-

Graph matrix is a square matrix whose size is equal to no. of nodes of the flow graph.

→ For computing cyclomatic complexity following steps are adopted.

Eg: step 1: create a graph matrix. Mark the corresponding entry as 1 if node A is connected to node B.

step 2: count total no. of 1's from each row & subtract 1 from each corresponding each row.




step 3 Add cyclomatic complexity

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|
| 1 | | 1 | 1 | | | | 2-1=1 |
| 2 | | | | 1 | | | 1-1=0 |
| 3 | | | | 1 | | | 1-1=0 |
| 4 | | | | | 1 | 1 | 2-1=1 |
| 5 | | | | | | 1 | 1-1=0 |
| 6 | | | | | | | 3 |

step

## ⇒ System testing:-

→ The system test is a series of tests conducted to fully the computer based system.

→ Various types of system tests are

1. Recovery testing
2. security "
3. Stress "
4. performance "

→ The main focus of such testing is to test -

* system functions & performance.
* system reliability & Recoverability
* system installation
* system behavior in the special conditions
* system user operations.
* H/w & S/w integration & collaboration.
* Integration of external S/w & the system

## ⇒ 1) Recovery testing :-

→ Recovery testing is intended to check the system's ability to recover from failures.

→ In this type of testing the S/w is forced to fail and then it is verified whether the system recovers properly or not

→ For automated recovery then reinitialization, checkpoi -nt mechanisms, data recovery are restart are verified.

## ⇒ 2) security testing :-

→ security testing verifies that system protection mech -anism prevent improper penetration or data alterna -tion.

→ It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access.

→ system design goals is to make the penetration attempt more costly than the value of the information that will be obtained

⇒ 3. stress testing:-

→ Determines breakpoints of a system to establish maximum service level.

→ In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.

→ A variation of stress testing is a technique called sensitivity testing.

→ The sensitive testing is testing in which it is tried to uncover data from a large class of valid data that may cause improper processing.

⇒ 4. performance testing:-

→ performance testing evaluates run-time performance of s/w, especially real time s/w.

→ In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.

→ For big systems involving many users connecting to servers & performance testing very difficult.

→ Beta testing useful for performance testing.

⇒ control structure testing:

* The structural testing is sometime called as white-box testing.
* objective of structural testing is to exercise all program statements.

→ The conditional testing is used to test the logical conditio -ns in the program.

→ The condition can be boolean condition or relational Ex -pression.

→ The condition is incorrect, in following situations.

1) Boolean operator is incorrect, missing or extra.
2) Boolean variable is incorrect.
3) Boolean paranthesis may be missing.
4) Error in relational operator.
5) Error in arithmatic expression.

→ The condition testing, focuses on each testing condition in the program.

→ The branch testing is condition testing strategy in which for a compound condition each & every true or false branches are tested.

→ The domain testing is a testing strategy in which rela -tional expression can be tested using three or four tests.

⇒ Loop Testing:-

Loop testing is a white-box testing testing techniq -ue. which is used to test the loop constructs.

⇒ Basically there are 4 types of loops.

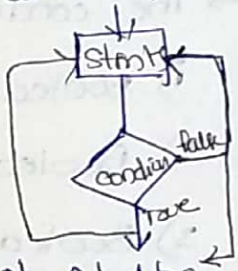
i) simple loop
ii) Nested loop
iii) concanterated loops
iv) unstructured loops.

1) Simple loops:-

→ The tests can be performed for n no. of classes.

i) n=0 that means skip the loop completely
ii) n=1 that means one pass through the loop is tested.
iii) n=2 that means two passes through the loop is tested.
iv) n=m that means testing is done when there are m passes where m<n.
v) perform the testing when no. of passes are n-1, n, n+1.

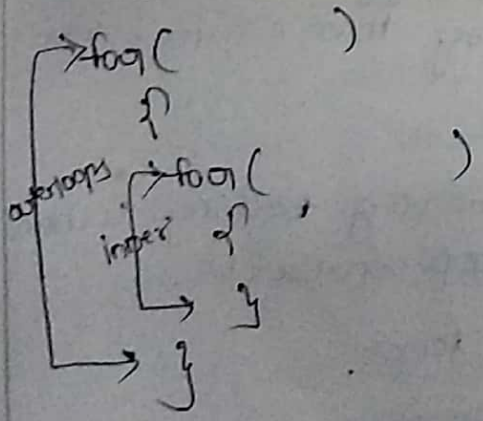
Eg foo ( )
{
y = .
}

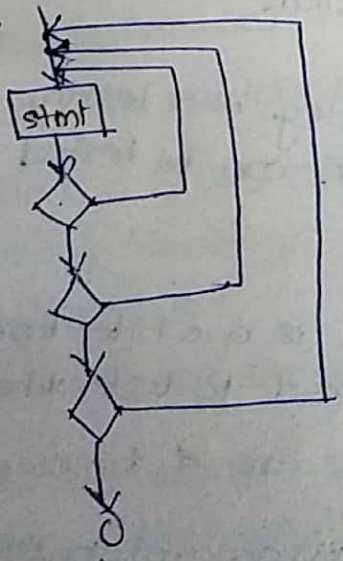



2) Nested loops:-

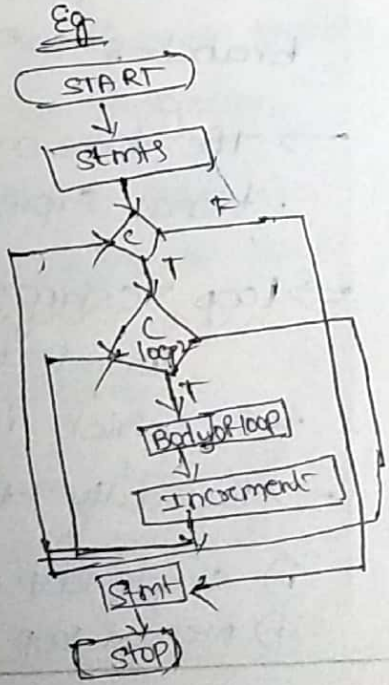
→ The nested loop can tested as follows.

1) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.
2) The simple loop test for innermost loop is done.
3) conduct the loop testing for the next loop by keeping the outer loops at the minimum values and other nested loops at some specified value.
4) This testing process is continued until all the loops have been tested.




Eg



3) Concanterated loops:-

The concanted loops can be tested is the same manner as simple loop tests.

4) unstructured loops:-

The testing can't be effectively conducted for a unstructu -red loops. Hence these types of loops needs to be redesign -ed.

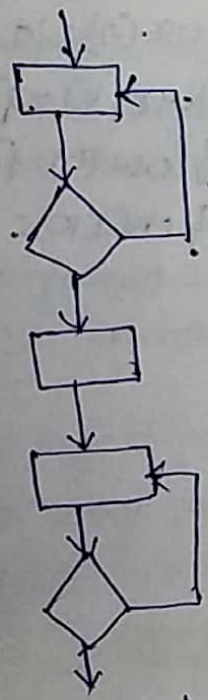


fig: concanted loops.

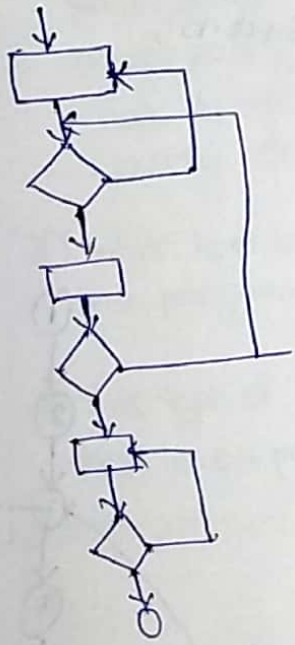


fig: unstructured loops.

⇒ Dataflow testing:-

→ The testing based on dataflow mechanism performs testing on definitions and uses of variables in the program.

→ In this method definition & use chain is required. the DU chain is obtained by identifying the def & use pairs from the program structure.

→ This testing is also called DU testing strategy.

* set DEF(n) contains variables that are defined at node n.
* set USE(n) contains variables that are read or used in at node n.

Eg:

```
1 S: =0;
2 a: =0;
3 while (a<b) {
4    a: =a+2;
5    b:= b-4;
6    if (a+b<20)
7    S: = S+a+b;
8    else
9       S: = S+a-b;
10 }
```

⟹

DU chain will be

DEF(1) = {S} USE(1) = {ϕ}

DEF(2) = {a} USE(2) = {ϕ}

DEF(3) = {ϕ} USE(3) = {a,b}

DEF(4) = {a} USE(4) = {a}

DEF(5) = {b} USE(5) = {b}

DEF(6) = {ϕ} USE(6) = {a,b}

DEF(7) = {S} USE(7) = {S,a,b}

DEF(8) = {S} USE(8) = {S,a,b}

DEF(9) = {ϕ} USE(9) = {ϕ}

DEF(10) = {ϕ} USE(10) = {ϕ}

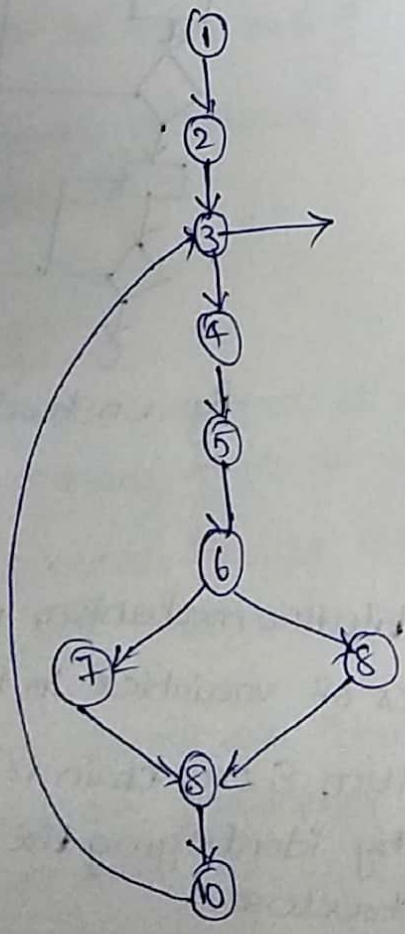



fig flowgraph

⇒ comparison b/w black box & white-box testing:-

| Blackbox testing | Whitebox testing |
|---|---|
| 1) Black-box testing is called behavioural testing. | 1) White box testing is called glass box testing. |
| 2) Black box testing examines some fundamental aspect of the system with little regard for internal logical structure of the soft. | 2) In white box testing the procedural details, all the logical paths, all the internal data structures are closely examined. |
| 3) During black-box testing the program cannot be tested 100%. | 3) White box testing lead to test the program thoroughly. |
| 4) This type of testing is suitable for large projects. | 4) This type of testing is suitable for small projects. |

⇒ list of debugging:-

→ Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

→ Debugging process starts with execution of test cases.

→ The actual test results are compared with the expected results.

→ The suspected causes are identified & additional tests or regression tests are performed to make the system to work as per requirement.

common approaches in debugging:

1) Brute-force method: The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues on error causes.

→ In this method " Let computer find the error" approach is used.

→ This is the best efficient method of debugging.

2) Back-tracking method :- This method is applicable to small program. In this method, the source code is examined by looking backwards from symptom to potential causes of errors.

3) cause elimination method :- this method uses binary partitioning to reduce the no. of locations where errors can exist.

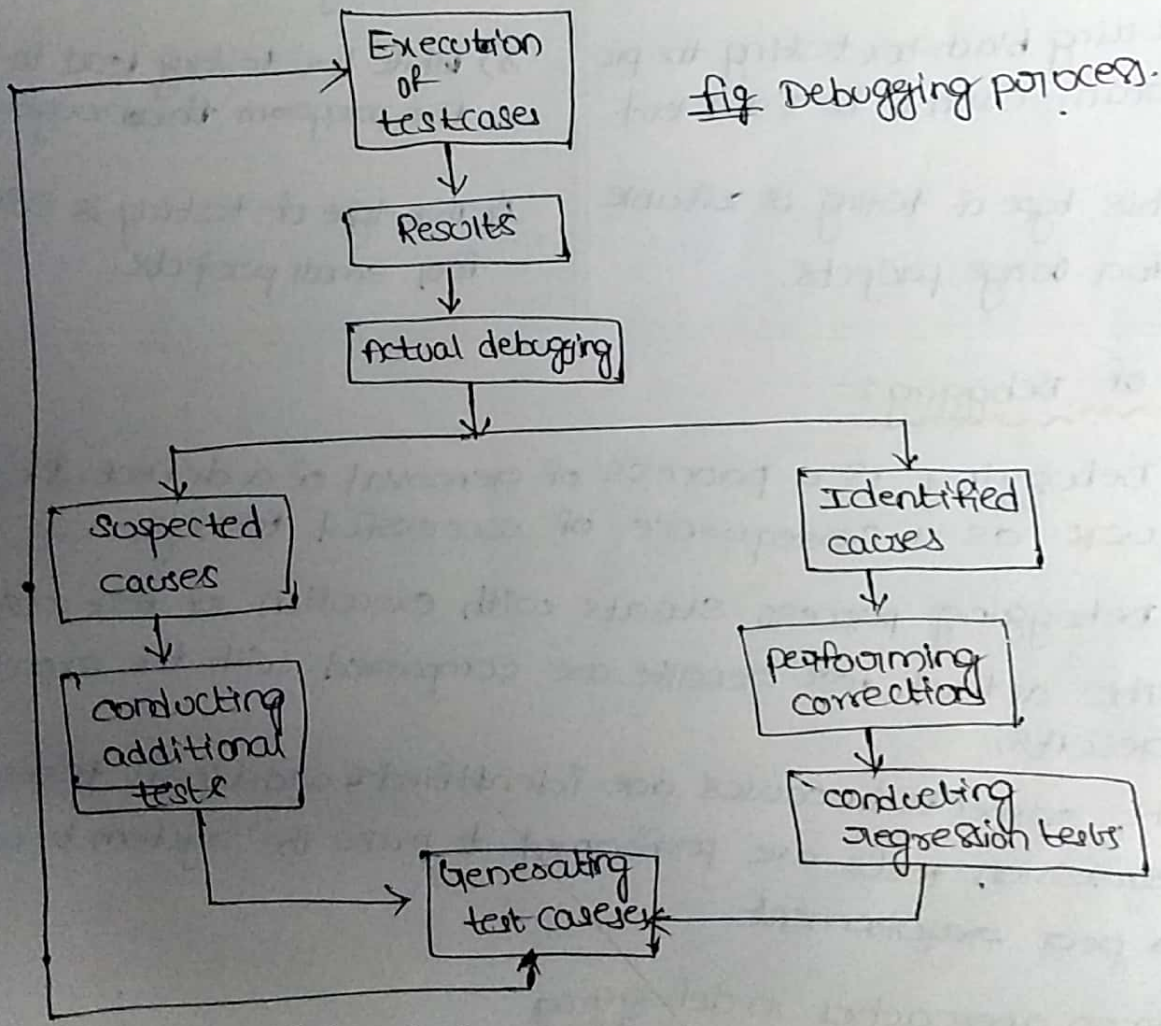



fig Debugging process.

→ Testing Vs Debugging :-

| Testing | Debugging |
|---|---|
| 1) Testing is a process in which the bug is identified | 1) Debugging is the process in which the bug or error is corrected by programmer. |

2) In testing process, it is identified where the bug occurs

3) Testing starts with the execution results from the testcases.

2) In debugging rootcause of error is identified.

3) Debugging starts only after the testing process.

# UNIT-IV (Part-I)

## Product metrics

* High quality software is an important goal in slw development.
* Slw quality is conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected to slw development.

## Mc Call's Quality factors:—

The factors that affect slw quality can be categorized into two broad categories.

1) factors that can be directly measured
   (eg: defects uncovered during testing)
2) factors that can be measured indirectly
   (eg: usability or maintainability)

* Mc Call's Richards and coalters propose a category of factors that affect slw quality.
  1) operational characteristics
  2) Ability to undergo change
  3) ability to undergo new environment.

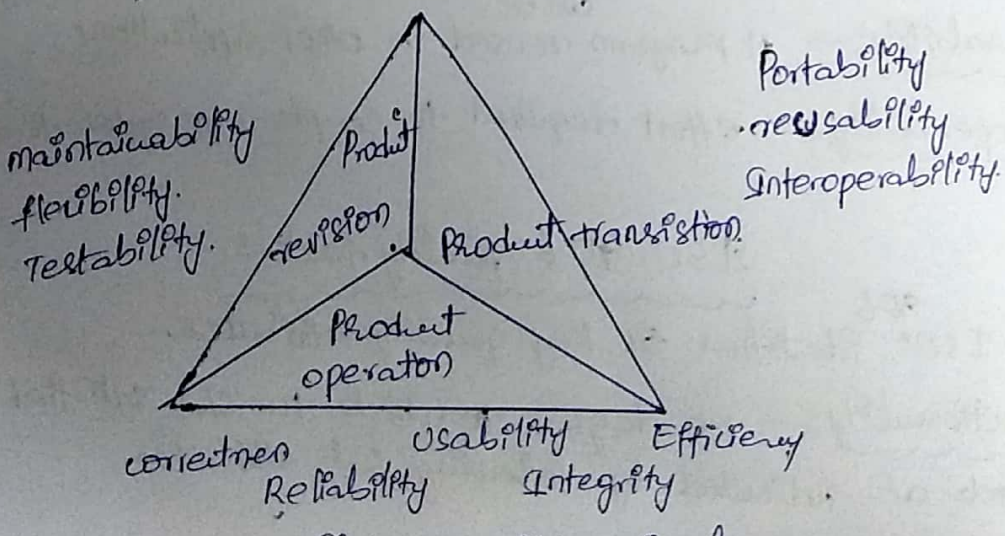



Fig: McCall's triangle.

* McCall's quality factors are as valid today as they were in 1970's. So it is clear that factors that affect slw quality do not change with time.

(i) Correctness :- the extent to which program satisfies its specification and fulfils the customer's objectives

(ii) Reliability :- the extent to which a program can be expected to perform its intended function with required precision.

(iii) Efficiency :- the amount of computing resources and code needed by a program

(iv) Integrity :- the extent to which slw or data controlled from unauthorized accen.

(v) Usability :- the effort required to learn, operate, Prepare input. and interpret output of a program.

(vi) Maintainability :- effort required to locate and fix an error in a program.

(vi) flexibility :- the effort required to modify an operational program

(vii) Testability :- the effort required to test and ensure correctness of the program.

(viii) Portability :- the ability of the slw to work properly even if the environment get changed

(ix) Reusability :- A program can be reused in other applications

(x) Interoperability :- effort required to couple one system to another.

## ISO 9126 quality factors

The ISO 9126 identifies six key quality attributes.

1) Functionality :- the degree to which the slw satisfied stated needs as indicated by following sub attributes.

(i) sustainability (ii) accuracy (iii) interoperability
(iv) compliance (v) security.

(2) Reliability ← maturity
fault tolerance
recoverability.

3) usability ← understandability
learnability
operability.

(4) Efficiency :— the degree to which the s/w makes optimal use of system resources as indicated by the following.
(i) time behavior (ii) resource behavior.

(5) maintainability :— the ease with which repair may be made to the s/w as indicated by the following.
→ Analyzability
→ changeability
→ stability
→ Testability.

(6) portability :— the ease with which the s/w can be transposed from one environment to another
→ adaptability
→ Instability
→ conformance
→ replaceability.

## A framework for Product metrics.

A set of basic principles for the measurement of Product metrics for S/w.

## Measures, metrics and Indicators.

**Measure:-** It is a quantitative indication of the extent amount, dimension, or size of the some attribute of a Product or process.

**Metric:-** It is the degree to which a system, component, or Process possess a given attribute. The s/w metrics relate several measures.

for eg:- Average no. of errors found per review.

**Indicators:-** Indicators mean combination of metrics that Provides insight into the s/w process, project or Product.

## Purpose of Product metrics:-

→ Aid in the evaluation of analysis and design models.
→ Provide an indication of the complexity of Procedural designs and source code.
→ facilitate the design of more effective testing techniques
→ Assess the stability of a s/w product

## Measurement principles or activities of a measurement process

In the measurement process, first we formulate, collect, analysis, Interpretation, & feedback.

1. formulation:-

* The appropriate s/w measures and metrics should be considered for the representation of the s/w.

2. Collection :- The mechanisms used collect the results or data obtained from the formulated metrics.

3. Analysis :- The analysis should be made on the computation of metrics and application of mathematical tool.

4) Interpretation :- The evaluation metrics provides the insight for the s/w quality.

5) feedback :- Interpretations obtained from product metrics must be submitted to the s/w team for review and feedback.

## Goal oriented s/w measurement

* Goal / question oriented s/w measurement (GQM) is a technique for identifying meaningful metrics for any part of s/w process.
* For applying this technique following are the requirements.

① The explicit measurement goals must be established which is based on process activity or process characteristics

② Prepare set of questionnaire which will help to find out measurement goals

③ ~~Finded~~ identify well formulated metrics that will help to answer the prepared set of questions.

Goal Definition Template :-

Analyze {name of activity to be measured}
Purpose of {object or purpose}
with respect to {activity or attribute that is to be considered}
from the viewpoint of {stakeholders performing measurement}
context of {environment in which the measurement takesplace}.

## Attributes of effective S/w metrics

Effective S/w metrics should have following attributes

(1) simple and computable:—

The derivation of metric should be easy to compute and should not be a time consuming activity.

(2) empirically and intuitively persuasive:—

* It should be immediate and can be derived based on observations

(3) Consistent and objective:—

* The metric should produce unambigous results anybody should get the same result by using metrics when same set of information is used.

(4) consistent in its use of units and dimensions:-

* the mathematical units and dimensions used for the metric should be consistent. And these should not be intermining of units.

(5) programming language independent:-

* The metric should be based on analysis model, design model and program structure. It should be independent of programming languages, syntax, or semantic of any programming language.

## Metrics for Analysis Model :-

* Metrics for the analysis model are useful in estimating the project. In order to determine the metrics in analysis model "size" of the s/w is used as a measure.

### Function point model :-

* The function point model is based on functionality of the delivered application.
* These are generally independent of the programming lang used.
* This method is developed by Albrecht in 1979.
* Using historical data, function points can be used to
  → estimate the cost or effort required to design, code and test the s/w
  → predict the number of errors encountered during testing.
  → Forecast the number of projected source code lines in the implemented system.

### How to calculate function point :-

The data for following information domain characteristics are collected to calculate function point.

1. Number of user inputs : Each user input that provides distinct data to the s/w is counted.
2. Number of user outputs : Each user output that provides information to the user is counted. (reports, o/p screens, error messages etc)
3. Number of user inquiries :-
   An inquiry is that an on-line input that results in the generation of some immediate s/w response

in the form of an on-line output. (eg: google search)

4. No. of files:- Each logical master file (i.e. large database or separate file) is counted.

5. Number of external interfaces:-

All machine readable interfaces (eg: data files on storage media) that are used to transmit information to another system are counted.

Function point computation:-

The process involved in function point computation is

1. Identify / collect the information domain values
2. Complete the table as shown below to get the count total.
3. * Associate a weighting factor (ie. Complexity Value) with each count based on criteria established by the s/w development organization.

3. Evaluate and sum up the adjustment factors.
"$F_i$" refers to 14 value adjustment factors, with each ranging in value from 0 (not important) to 5 (absolutely essential)

4. Compute the number of function points (FP)

$$Fp = \text{count total} * [0.65 + 0.01 * \text{Sum}(F_i)]$$

* cost total can be computed with the help of below table.

| Information Domain Value | count | weighting factor simple | Average | complex | |
|---|---|---|---|---|---|
| 1. External inputs (EIs) | ☐ x | 3 | 4 | 6 | = |
| 2. External outputs (EOs) | ☐ x | 4 | 5 | 7 | = |
| 3. External inquiries (EQs) | ☐ x | 3 | 4 | 6 | = |
| 4. Internal Logical files (ILFs) | ☐ x | 7 | 10 | 15 | = |
| 5. External interface files (EIFs) | ☐ x | 5 | 7 | 10 | = |
| | | cost total → | | | ☐ |

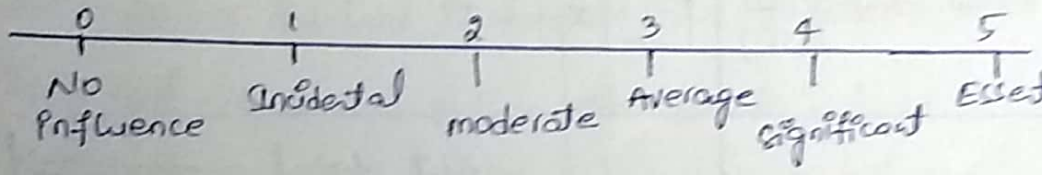
* The weighting factors should be determined by observations or by experiments.

* Now the sw complexity can be computed by answering following questions.

1. Does the system need reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance of the system critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations.

8. Are the master files updated on-line?
9. Are the Inputs, output, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code which is designed being reusable?
12. Are conversation and installation included in the design?
13. Is the system designed for multiple installations in different Organizations?
14. Is the application designed to facilitate change and ease of use by the user?

* Rate each of the above factors according to the following scale:




* once function point is calculated then we can compute various measures as follows.

→ Productivity = FP / person-month.

→ Quality = Number of faults / FP

→ Cost = $ / FP

→ Documentation = pages of documentation / FP.

Function point Example:—

| Domain value | count | | weighting factor simple | average | complex | |
|---|---|---|---|---|---|---|
| External Inputs | 3 | x | 3 | 4 | 6 | = 9 |
| External outputs | 2 | x | 4 | 5 | 7 | = 8 |
| External inquiries | 2 | x | 3 | 4 | 6 | = 6 |
| Internal Logical files | 1 | x | 7 | 10 | 15 | = 7 |
| External interface files | 4 | x | 5 | 7 | 10 | = 20 |
| | count total = 50 | | | | | 50 |

$FP = count\ total * [0.65 + 0.01 * sum(F_i)]$

$FP = 50 * [0.65 + (0.01 * 46)]$

$FP = 55.5$ (rounded up to 56)

=

Imp * **Metrics for specification quality**

To assess the quality of analysis model and corresponding requirements Davis and his colleagues has suggested some characteristics. These characteristics are.

* Completeness * correctness
* Understandability * verifiability
* Internal and external consistency
* Achievability
* Concision * Traceability
* modifiability * precision
* Reusability.

The total requirements in the specification can be specified as $n_r$

where $n_r$

~~where~~ $n_r = n_f + n_{nf}$

where

$n_r$ = Total number of requirements

$n_f$ = Total number of functional requirements

$n_{nf}$ = Total number of non functional requirements.

* Davis has suggested the metric for Specificity of req as

$Q_1 = n_{ui} / n_r$

where $n_{ui}$ is the number of requirements that have unique interpretation and $n_r$ is the total no. of requirements

* Completeness of functional requirements is given by.

$$Q_2 = n_u / [n_i * n_s]$$

where

$n_u$ = Number of unique functional requirements

$n_i$ = Number of input given in the specification

$n_s$ = Number of states specified.

Thus $Q_2$ gives the percentage of necessary functions but here it does not consider the non functional requirements.

* The overall metric ~~for~~ (even by considering the non functional requirements) for completeness can be obtained by validating the requirements.

$$Q_3 = n_c / [n_c + n_{nv}]$$

where

$n_c$ is number of requirements that are validated as correct

$n_{nv}$ is the number of requirements that are not been validated.

## Metrics for Design model :-

* metrics for design model focus on determining the measurement of design quality. These metric guides the SW design activity as design evolves.

* Design model metrics ~~to~~ considers three aspects

1) Architectural design
2) Object oriented design
3) User interface design.

## Architectural Design metrics :-

While determining the architectural design primarily characteristics of program architecture are considered. It does not focus on inner working of the system.

## Metrics by Card and Glass :-

Two scientists Card and Glass has suggested three design complexity measures as

### 1) Structural complexity :-

Structural complexity depends upon the fan-out for modules. It can be defined as

$$S(K) = f^2_{out}(K)$$

where fout represents fan-out for module k.
(fan out means number of modules that are subordinating module k).

### 2) Data Complexity :-

Data complexity is the complexity within the interface of internal module for some module k it can be defined as

$$D(K) = \frac{tot\_Var(K)}{[f_{out}(K)+1]}.$$

where tot_var is the total number of input and output variables going to and coming out of the module.

### 3) System complexity :-

System complexity is the combination of structural and data complexity. It can be denoted as

$$Sy(K) = S(K) + D(K)$$

* when structural, data and system complexity get increased the overall architectural complexity also get increased.

## Metrics for Object-oriented Design:-

Whitmire has suggested nine measurable characteristics of object oriented design and those are.

1) Size :- It can be measured using following factors.

2) Complexity:- It is a measure representing the characteristics that how the classes are interrelated with each other.

3) Coupling:- It is a measure stating the collaborations between classes or number of messages that can be passed between objects.

4) completeness :- It is the measure representing all the requirements of the design component

5) cohesion:- It is the degree by which the set of properties that are working together to solve particular property.

6) primitiveness:- the degree by which the operations are simple. In other words, the measure by which number of operations are independent upon other.

These Product metrics for object-oriented design is applicable to design as well as analysis model.

## CK Metrics Suite

CK have Proposed six class-based design metrics for object oriented systems.

1. weighted methods per class (wmc):-

* Assume that n methods of complexity $c_1, c_2, \ldots c_n$ are defined for a class C
* The specific complexity metric that is choosen (eg., cyclomatic complexity) should be normalized so that normal complexity for a method takes on a value of 1.0

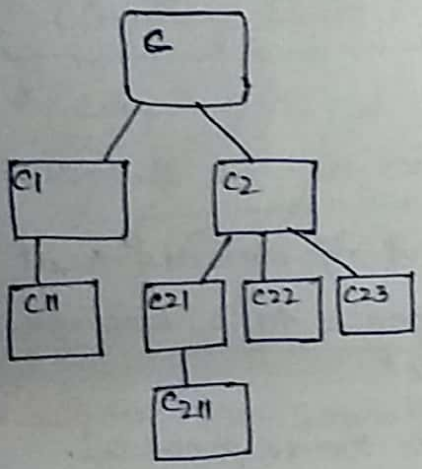
$$WMC = \Sigma c^i$$

for i = 1 to n. The number of methods and their complexity are reasonable indicators of the amount of effort required to implement and test a class.

* So. If no. of methods are increased, complexity of class also increased. Therefore, limiting potential reuse.

2. Depth of the Inheritance (DIT)

* This metric is "the maximum length from the node to the root of the tree".




* Refering to figure, the value of DIT for the class ~~hier~~ hierarchy shown is 4.
* As DIT grows, it is likely that lower-level classes will inherit many methods. This leads to potential ~~difficulties~~ difficulties when attempting to predict the behavior of a class.
* A Deep class hierarchy is also leads to greater design complexity.
* on the positive side, large DIT value imply that many methods may be reused.

3. Number of children (NOC) :-

* The Subclasses that are immediately sub ordinate to a class in the class hierarchy are termed as its children.
* refering to previous figure, class C2 has three children - Subclasses C21, C22 and C23.
* As the number of children grows, reuse increases, the abstraction represented by the Parent class can be diluted.

* As Noc increases, the amount of testing will also increase.

4. Coupling between object classes (CBO):— * CBO is no. of collaborations b/w the classes
* The CRC model may be used to determine the value for CBO
* CBO is the number of collaborations listed for a class on its CRC Index card.
* As CBO increases, it is likely that the ~~reuse~~ reusability of a class will decrease.
* If values of CBO is high, then modification get complicated.
* Therefore, CBO values for each class should be kept as low as is reasonable.

5. Response for a class (RFC)

* Response for a class is "a set of methods that can potentially be executed in response to a message received by an object of that class"
* RFC is the number of methods in the response set.
* As RFC increases, the effort required for testing also increases because the test sequence grows. as well as overall design complexity of the class increases.

Lorenz and Kidd metrics suite.

Lorenz and Kidd have proposed the conceptual division of class based metric into four distinct categories.

1) size 2) Inheritance 3) Internal 4) external.

* Size-oriented metrics for the OO class focus on counts of attributes and operations for an individual class.
* Inheritance-based metrics focus on the manner in which operations are reused through the class hierarchy.
* metrics for class internals look at cohesion and code oriented issues, and external metrics examine coupling and reuse.

Imp

# The MOOD Metrics Suite.

MOOD metrics suite is proposed by Harrison, Counsell and Nithi for object oriented design. It includes two metrics MIF and CF.

1. Method inheritance factor (MIF):-

$$MIF = \frac{\sum Mi(Ci)}{\sum Ma(Ci)}$$

* The degree to which the class architecture of an OO system makes use of inheritance for both methods. (operations) and attributes is defined.
* Value of MIF indicates impact of inheritance on the OOS/w.

* where Ci is a class within the architecture
Mi(Ci) is number of methods inherited in Ci
Md(Ci) number of method declared in Ci

2. Coupling factor (CF):-

* Coupling is an indication of the connections between elements of the OO design.

$$CF = \sum_i \sum_j is\_client(C_i, C_j) / (T_c^2 - T_c)$$

i and j varies from 1 to total number of classes in the architecture Tc

* where the summation occur over i=1 to Tc and j=1 to Tc.
* Function (is_client) = 1, if and only if a relationship exist between the client class, Cc, and the server class, Cs, and Cc ≠ Cs. = 0 otherwise.
* As CF increases the complexity of object oriented design get increased.

## Operation oriented metrics

* operation oriented metrics reside within a class.

(1) Average operation size:- (OSavg)

* Lines of code (Loc) could be used as an indicator for operation size.
* operation has some roles and responsibilities related to Product

* As the number of messages sent by a single operation increases, it is likely that responsibilities have not been well-allocated within a class.

2. Operation complexity (OC):-

* operations should be limited to a specific responsibility, the designer should strive to keep OC as low as possible.

3. Average number of parameters per operation (NPavg):-

* The larger the number of operation parameters, the more complex the collaborations blw objects.
* In general, NPavg should be kept as low as possible.

## Metrics for source code.

Halstead has proposed in "software science" some software science metrics. These metrics are based on

* common sense
* Information theory
* psychology.

* In the proposed metrics the used measures are

$n_1$ = The number of distinct operators in the program.

$n_2$ = The number of distinct operands in the program.

The paired block such as {---}, or begin .. end or repeat... until are treated as single operator. The program length N can be defined as.

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

The program volume can be defined as

$$V = N \log_2 (n_1 + n_2)$$

The program volume is heavily dependent upon the program length

Let $N_1$ = Total count for all the operations in the program.

$N_2$ = Total count for all the operands in the program

The program volume ratio L can be defined as $L = 2/n_1 \times n_2/N_2$

## Metrics for Testing.

Halstead's metrics for estimating the testing efforts are as given below.

The Halstead effort can be defined as

$$e = V/PL$$

where V is the program volume and PL is the Program Level.

The program level can be compated as

$$P_L = 1/[(n_1/2) \times (N_2/n_2)]$$

* The percentage of overall testing effort =

$$\frac{\text{testing effort of specific module}}{\text{testing efforts of all the module.}}$$

## Metrics for maintenance

* The stability of sw product is given by an IEEE standard which suggest a metrics software maturity index (SMI) for that matter.
* It is given as follows.

$$SMI = (M - (A + C + D))/M$$

where,

M = Number of modules in current version

A = Number of added modules in current version

C = Number of changed modules in current version

D = Number of deleted modules in current version compared to previous version.

When SMI reaches to the value 1.0 the product becomes more and more stabilized

Syllabus

Metrics for Process and products :- Software measurement, metrics for s/w quality.

## Product metrics outline

Product metrics

(I) Software Quality
- (i) mccall's Quality factors
- (ii) ISO 9126 Quality factors.

(II) A Framework for Product metrics
- (i) measures, metrics and Indicators
- (ii) The challenge of product metrics
- (iii) measurement principles
  - formulation
  - collection
  - Analysis
  - Interpreted
  - Feedback.
- (iv) Goal oriented s/w measurement
- (v) The attributes of effective s/w metrics.

(III) metrics for analysis model.
- FP (function point) model.
- metrics for specification quality

(IV) metrics for design model.
- → Architectural design metrics
  - metrics by card & Glan
    - Structural complexity
    - Data complexity
    - System complexity.
- → metrics for object-oriented design.
  - size
  - complexity
  - coupling
  - completeness
  - cohesion.
- → CLASS-Oriented metrics
  - → CK metrics suite.
    - WMC
    - DIT
    - NOC
    - CBO
    - RFC
  - → Lorenz and kidd metrics suite
  - → MOOD metrics suite
    - MIF
    - CF
  - (metrics for object oriented design)
  - → operation Oriented metrics
    - OSAvg
    - OC
    - NPavg.

(V) metrics for source code.

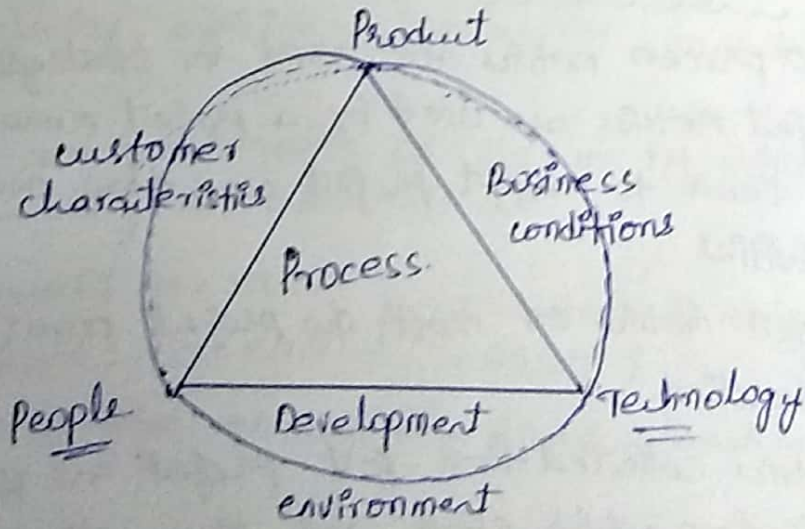
(VI) metrics for testing.

## Process metrics and Software Process Improvement

* Process metrics are collected across all projects and over long periods of time. their intent is to provide a set of process indicators that lead to long-term sw process improvement.

* Project metrics enable a sw project manager to
  1) assess the status of an ongoing project.
  2) track potential risk
  3) uncover problem areas before they go "critical"
  4) Adjust work flow or tasks, and
  5) evaluate the project team's ability to control quality of sw work products.

* In making improvements to any sw system, there are three basic quality factors to consider: product, people and technology.




* Fig: Determinants for sw quality and organizational effectiveness.

* Process at the center connecting 3 factors that have a profound influence on sw quality and organizational.

Performance.

* The skill and motivation of people has been shown to be the single most influential factor in quality and Performance.
* The complexity of the Product can have a substantial impact on quality and team performance.
* The technology that populate the process also has an impact
* Praess triangle within the circle, specifies the environmental conditions such as
  → customer characteristics ( communication and collaboration between user and developer)
  → Business conditions (organizational policies, Business rules)
  → Development Environment (use of new technologies, use of automated tools)

## Project metrics

* s/w process metrics are used for strategic purposes.
* project metrics are used by a project manager and a s/w team to adapt project work flow and technical activities
* Project metrics on most s/w projects occurs during estimation.
* metrics collected from past projects are used as a basis from which effort and time estimates are made for current s/w work
* As a project proceeds, measures of effort and calendar time expended are compared to original estimates.

Intent of project metrics is

1) used to minimize the development schedule
2) used to assen product quality.

## Software measurement

* measurement of slw can be classified into two categories.

1) Direct measures  2) Indirect measures.

### Direct measures:-

* Direct measure of the slw Process includes cost and effort applied.
* Direct measures of the product includes lines of code, (LOC) produced, execution speed, memory size and defects reported over some set period of time.

### Indirect measure:-

* Indirect measures of the product include functionality, complexity, efficiency, reliability, maintainability etc.
* The quality and functionality of slw or its efficiency or maintainability are more difficult.
* Team A found: 342 errors
  Team B found: 184 errors } which team is more efficient?
* It is depends on size or complexity (i.e functionality) of the projects.

### Size Oriented metrics:-

* Size-oriented slw metrics are derived by normalizing quality and/or productivity measures by considering the size of the slw that was been produced.

* S/w organization can maintains simple records as shaonin fig.
* The table lists each S/w development project that has been completed over the past few years and corresponding measures for that project.

| Project | LOC | Effort | $(cost) | Doc. (Pgs) | Errors | defects | people |
|---|---|---|---|---|---|---|---|
| ABC | 10,000 | 20 | 170 | 400 | 100 | 12 | 4 |
| PQR | 20,000 | 60 | 300 | 1000 | 129 | 32 | 6 |
| XYZ | 35,000 | 65 | 522 | 1290 | 280 | 87 | 7 |
| | | | | | | | |

Table : Size measure.

* A simple set of size measure that can be developed as
  - → Size = thousand Lines of code (kLOC)
  - → Quality = No. of faults/KLOC
  - → Effort = person/month
  - → cost = $/LOC
  - → pages of documentation/kLOC

* Size-oriented metrics are widely used but there is a slight debate about validity and applicability.
* Size-oriented metrics are programming language dependent
* It is difficult to estimate LOC in the early stages of development.

## Function oriented metrics

* It use a measure of functionality delivered by the application as a normalization value.
* Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures.
* Function Point (FP) is widely used as function oriented metrics
* FP is based on characteristic of s/w information domain.
* FP is programming language independent.

## Relationship between LOC and FP metrics

* Relationship between LOC and FP depends upon
  - → The programming language that is used to implement the s/w
  - → The quality of the design.
* FP and LOC have been found to be relatively accurate predicators of s/w development effort and cost.
  - → However, a historical baseline of information must first be established
* LOC and FP can be used to estimate object-oriented s/w projects.
  - → However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an incremental process.

| Programming language | Average | median | Low | high |
|---|---|---|---|---|
| Ada | 154 | - | 104 | 205 |
| Assembler | 337 | 315 | 91 | 694 |
| C | 162 | 109 | 33 | 704 |
| C++ | 66 | 53 | 29 | 178 |
| COBOL | 77 | 77 | 14 | 400 |
| Java | 55 | 53 | 9 | 214 |
| VB | 47 | 42 | 16 | 158 |

* The table above provides a rough estimate of the average LOC to one FP in various programming languages.

=

Object oriented metrics:-

1) No. of scenario scripts (i.e., use cases)

* This number is directly related to the size of an application and to the number of test cases required to test the program.

2) Number of ~~class~~ key classes (the highly independent components)

* Key classes are defined early in object-oriented analysis and are central to the problem domain.

* This number indicates the amount of effort required to develop the S/W.

* It also indicates the potential amount of reuse to be applied during development.

3) No. of support classes:-

→ Support classes are required to implement the system but are not immediately related to the problem domain. (e.g., user interface, database, computation)

4) Average number of support classes per key class :-

* Key classes are identified early in a project (eg., at requirements analysis)
* Estimation of the number of support classes can be made from the number of key classes.
* GUI applications have between two and ~~three~~ times more support classes as key classes.

5) Number of subsystems :-

* A subsystem is an aggregation of classes that support a function that is visible to the end user of a system.

## Metrics for S/w quality

* The goal of S/w engineering is to produce a high-quality system, application, or product within timeframe that satisfies the market need.
* To achieve this goal, S/w engineers must apply effective methods with modern tools within the context of a mature S/w process.

Measuring quality :-

There are many measures of software quality.

correctness,
Maintainability,
Integrity,
Usability

} useful indicators for the project team.

1. Correctness :-

* Correctness is a degree to which the s/w produces the ~~desig~~ desired functionality. The correctness can be measured as

$$\text{correctness} = \frac{\text{No. of Defects}}{\text{per KLOC}}$$

where defect means lack of conformance to requirements. Such defects are generally reported by the user of the Program.

2) Maintainability :-

* This describes the ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements.

* Mean time to change (MTTC) :- The time to analyze, design, implement, test and distribute a change to all users.

3) Integrity :-

* S/w Integrity has become increasingly important in the age of hackers and firewalls.

* This attribute measures a system's ability to withstand attacks to its security.

* Attacks can be made on all three components of s/w.
  - → programs
  - → Data
  - → Documents.

* To measure Integrity, two additional attributes must be defined.
  - → Threat
  - → Security.

## Measuring Defect Removal Efficiency (DRE):-

While developing the Slw project many work Products such as SRS, design doument, source code are being created.

* Along with these work products many errors may get generated. project manager has to identify all these errors to bring quality software.
* Error tracking is a process of assering the status of the Slw project.
* The Slw team performs the formal technical reviews to test the Slw developed. In this review various errors are identified and corrected. Any errors that remain uncovered and are found in later tasks are called defects.
* The defect removal efficiency can be defined as

$$DRE = E/(E+D)$$

where DRE represents Defect removal efficiency.
E is the error.
and D is defect.

* The DRE represents the effectiveness of quality anurance activities. The DRE also helps the project manager to asess the progress of Slw project as it gets developed through it Scheduled work task.
* During errors tracking activity following metrics are computed
  1. Errors per requirements specification page: denoted by Ereq
  2. Errors per Component - design level: denoted by Edesign
  3. Error per component - code level; ~~def~~ denoted by Ecode
  4. DRE - requirement analysis.

DRE - architectural design

DRE - component level design.

DRE - coding.

Project manager calculates current values for.

Ereq, Edesign, and Ecode.

* These values are then compared with past projects. If the current result differs more than 20% from the average, then there may be cause for concern and investigation needs to be made in this regard.

* These error tracking metrics can also be used for better target review and testing resources.