## UNIT IV

**Testing Strategies:** A strategic approach to software testing, test strategies for conventionalsoftware, Black-Box and White-Box testing, Validation testing, System testing, the art of Debugging.

**Product metrics:** Software Quality, Metrics for Analysis Model, Metrics for Design Model,Metrics for source code, Metrics for testing, Metrics for maintenance.

**Metrics for Process and Products:** Software Measurement, Metrics for software quality.

## Testing Strategies

Software is tested to uncover errors introduced during design and construction. Testingoften accounts for

More project effort than other s/e activity. Hence it has to be done carefully using a testingstrategy.

The strategy is developed by the project manager, software engineers and testing specialists.

Testing is the process of execution of a program with the intention of finding errors Involves 40% of total project cost

Testing Strategy provides a road map that describes the steps to be conducted as part oftesting.

It should incorporate test planning, test case design, test execution and resultant data collection and execution

**Validation** refers to a different set of activities that ensures that the software is traceable tothe Customer requirements.

V&V encompasses a wide array of Software Quality Assurance

### A strategic Approach for Software testing

Testing is a set of activities that can be planned in advance and conducted systematically.Testing strategy

Should have the following characteristics:

-- usage of Formal Technical reviews(FTR)
-- Begins at component level and covers entire system
-- Different techniques at different points
-- conducted by developer and test group
-- should include debugging

Software testing is one element of verification and validation.

Verification refers to the set of activities that ensure that software correctly implements aspecific function.

( Ex: Are we building the product right? )

Validation refers to the set of activities that ensure that the software built is traceable tocustomer requirements.

( Ex: Are we building the right product ? )

### Testing Strategy

Testing can be done by software developer and independent testing group. Testing and debugging are different activities. Debugging follows testing

Low level tests verifies small code segments. High level tests validate major systemfunctions against customer requirements

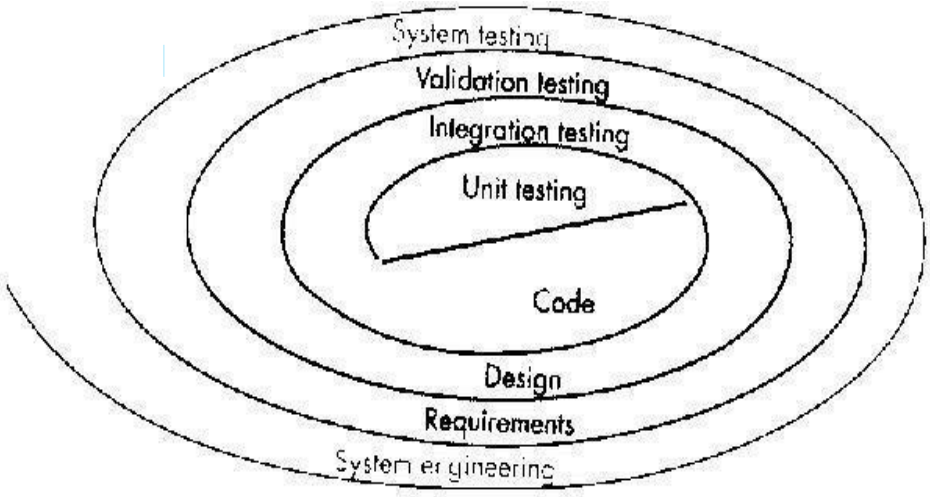
**Test Strategies for Conventional Software:**

Testing Strategies for Conventional Software can be viewed as a spiral consisting of four levelsof testing:

1) Unit Testing
2)Integration Testing
3)Validation Testing
and

4) System Testing

**Spiral Representation of Testing for Conventional Software**




**Unit Testing** begins at the vortex of the spiral and concentrates on each unit of software insource code.

It uses testing techniques that exercise specific paths in a component and its control structure to ensure complete coverage and maximum error detection. It focuses on the internal processing logicand data structures. Test cases should uncover errors.

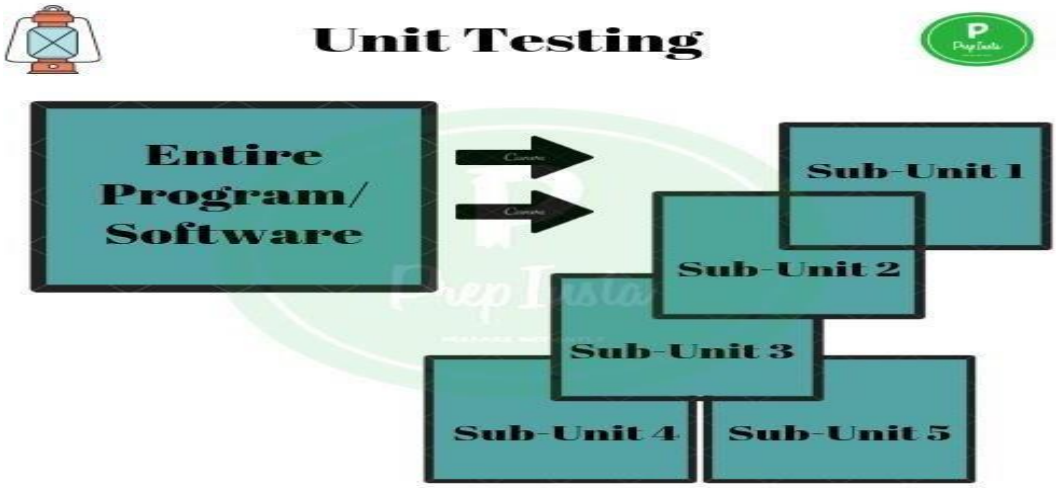



Fig: Unit Testing

Boundary testing also should be done as s/w usually fails at its boundaries. Unit tests can be designed before coding begins or after source code is generated.

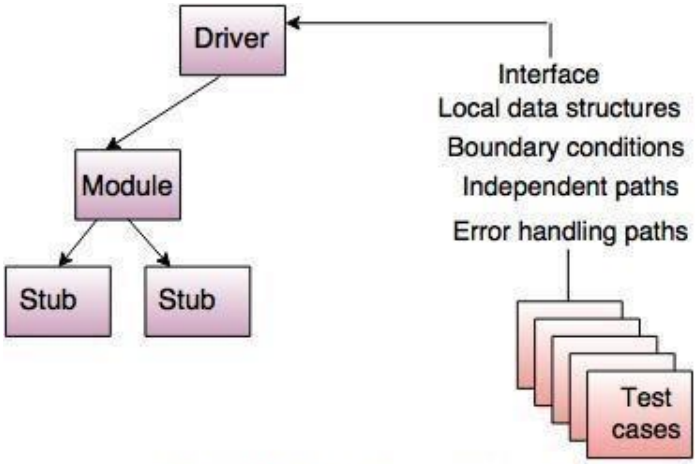



**Fig. - Unit test environment**

**Integration testing**: In this the focus is on design and construction of the software architecture. It addresses the issues associated with problems of verification and program construction by testing inputs and outputs. Though modules function independently problems may arise because of interfacing. This technique uncovers errors associated with interfacing. We can use top-down integration wherein modules are integrated by moving downward through the control hierarchy, beginning with the main control module. The other strategy is bottom –up which begins construction and testing with atomic modules which are combined into clusters as we move up the hierarchy. A combined approach called Sandwich strategy can be used i.e., top- down for higher level modules and bottom-up for lower level modules.

**Validation Testing**: Through Validation testing requirements are validated against s/wconstructed. These are high-order tests where validation criteria must be evaluated to assure that s/w meets all functional, behavioural and performance requirements. It succeeds when the software functionsin a manner that can be reasonably expected by the customer.

1)Validation Test Criteria2)Configuration Review 3)Alpha And Beta Testing

The validation criteria described in SRS form the basis for this testing. Here, Alpha and Beta testing is performed. Alpha testing is performed at the developers site by end users in a natural setting and with a controlled environment. Beta testing is conducted at end-user sites. It is a "live" application and environment is not controlled.

End-user records all problems and reports to developer. Developer then makes modifications and releases the product.

**System Testing:** In system testing, s/w and other system elements are tested as a whole. This is the last high-order testing step which falls in the context of computer system engineering. Software is combined with other system elements like H/W, People, Database and the overall functioning is checked by conducting a series of tests. These tests fully exercise the computer based system. The types of tests are:

1. Recovery testing: Systems must recover from faults and resume processing within a prespecified time.

It forces the system to fail in a variety of ways and verifies that recovery is properly performed. Here the Mean Time To Repair (MTTR) is evaluated to see if it is within acceptable limits.

2. Security Testing: This verifies that protection mechanisms built into a system will protect it from improper penetrations. Tester plays the role of hacker. In reality given enough resources and time it is possible to ultimately penetrate any system. The role of system designer is to make penetration cost more than the value of the information that will be obtained.

3. Stress testing: It executes a system in a manner that demands resources in abnormal quantity, frequency or volume and tests the robustness of the system.

4. Performance Testing: This is designed to test the run-time performance of s/w within the context of an integrated system. They require both h/w and s/w instrumentation.

**Testing Tactics**:

The goal of testing is to find errors and a good test is one that has a high probability of findingan error.

A good test is not redundant and it should be neither too simple nor too complex.Two major categories of software testing

- Black box testing: It examines some fundamental aspect of a system, tests whether eachfunction of product is fully operational.
- White box testing: It examines the internal operations of a system and examines theprocedural detail.

**Black box testing**

This is also called behavioural testing and focuses on the functional requirements of software. It fully exercises all the functional requirements for a program and finds incorrect or missing functions,interface errors, database errors etc. This is performed in the later stages in the testingprocess. Treatsthe system as black box whose behaviour can be determined by studying its input and related output Not concerned with the internal. The various testing methods employedhere are:

1) Graph based testing method: Testing begins by creating a graph of important objects and theirrelationships
and then devising a series of tests that will cover the graph so that each object and relationshipis exercised and errors are uncovered.

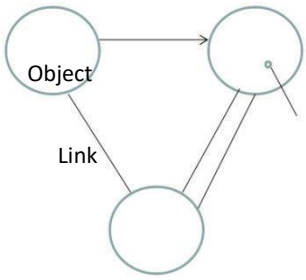



Fig: O-R graph.

2) Equivalence partitioning: This divides the input domain of a program into classes of datafrom which test
Cases can be derived. Define test cases that uncover classes of errors so that no. of test cases are reduced.This is based on equivalence classes which represents a set of valid or invalid states forinputconditions. Reduces the cost of testing

Example

Input consists of 1 to 10

Then classes are n<1,1<=n<=10,n>10

Choose one valid class with value within the allowed range and two invalid classes wherevalues are greater than maximum value and smaller than minimum value.

3) Boundary Value analysis

Select input from equivalence classes such that the input lies at the edge of the equivalence classes. Set of

data lies on the edge or boundary of a class of input data or generates the data that lies at the boundary of a class of output data. Test cases exercise boundary values to uncover errors at the boundaries of the input domain.

Example

If 0.0<=x<=1.0

Then test cases are (0.0,1.0) for valid input and (-0.1 and 1.1) for invalid input

4) Orthogonal array Testing

This method is applied to problems in which input domain is relatively small but too large for exhaustive testing

Example

Three inputs A,B,C each having three values will require 27 test cases**. O**rthogonal testing will reduce the number of test case to 9 as shown below

**White Box testing**

Also called glass box testing. It uses the control structure to derive test cases. It exercises all independent paths, Involves knowing the internal working of a program, Guarantees that all independent paths will be exercised at least once .Exercises all logical decisions on their true and false sides**,** Executes all loops,Exercises all data structures for their validity. White box testingtechniques

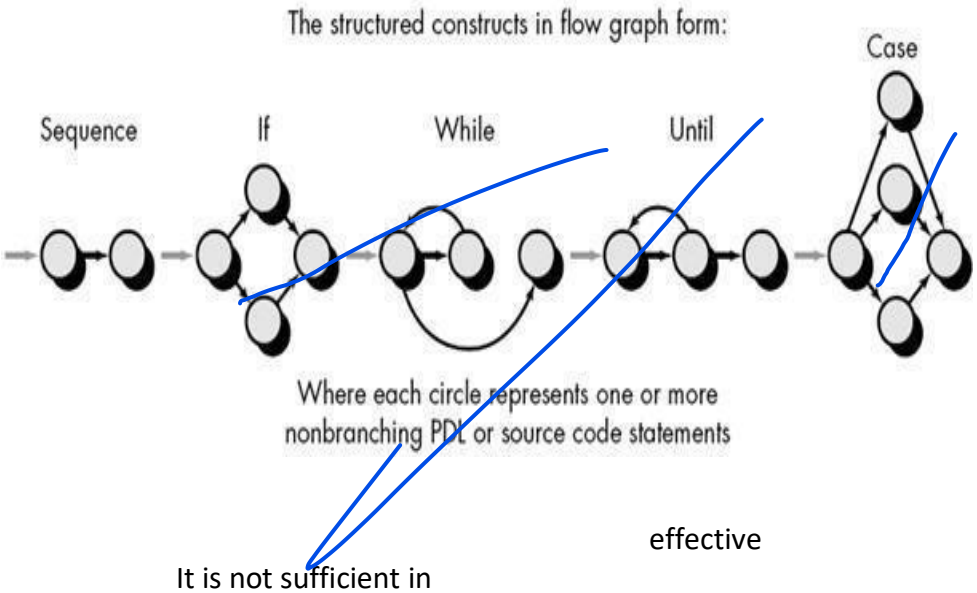
1. Basis path testing
2.Control structure testing1.Basis path testing

Proposed by Tom McCabe. Defines a basic set of execution paths based on logicalcomplexity of a procedural design. Guarantees to execute every statement in the program at least once Steps of Basis Path Testing

1. Draw the flow graph from flow chart of the program
2.Calculate the cyclomatic complexity of the resultant flow graph3.Prepare test cases that will force execution of each path

Two methods to compute Cyclomatic complexity number
1.V(G)=E-N+2 where E is number of edges, N is number of nodes2.V(G)=Number of regions

The structured constructs used in the flow graph are:




effective

It is not sufficient in

itself2.Control

Structure testing

This broadens testing coverage and improves quality of testing. It uses the following methods:

a) Condition testing: Exercises the logical conditions contained in a program module.

Focuses on testing each condition in the program to ensure that it does not contain errorsSimple condition

E1<relation operator>E2 Compound condition

simple condition<Boolean operator>simple condition

Types of errors include operator errors, variable errors, arithmetic expression errors etc.

b) Data flow Testing

This selects test paths according to the locations of definitions and use of variables in aprogram Aims to ensure that the definitions of variables and subsequent use is tested

First construct a definition-use graph from the control flow of a program

DEF(definition):definition of a variable on the left-hand side of an assignment statement

USE: Computational use of a variable like read, write or variable on the right hand of

assignment statement Every DU chain be tested at least once.

c) Loop Testing

This focuses on the validity of loop constructs. Four categories can be defined

1.Simple loops

2.Nested loops

3.Concatenated
loops
4.Unstructured
loops

Testing of simple loops
N is the maximum number of allowable passes through the loop
1.Skip the loop entirely
2.Only one pass through the
loop3.Two passes through
the loop
4.m passes through the loop where
m>N5.N-1,N,N+1 passes the loop

## The Art of Debugging

Debugging occurs as a consequence of successful testing. It is an action that results in theremoval of errors.
It is very much an art.

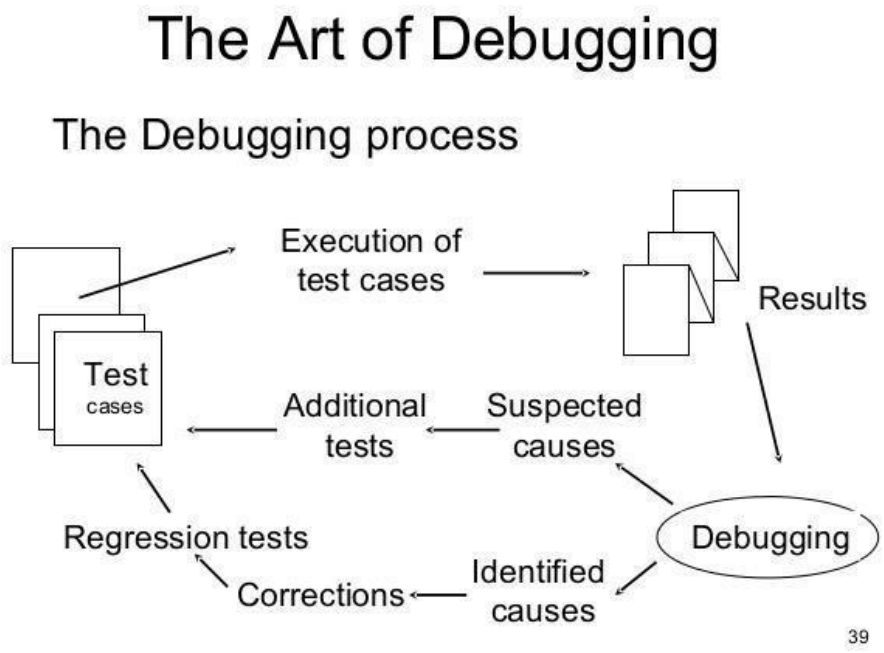



Fig: Debugging process

Debugging has two outcomes:
- cause will be found and corrected
- cause will not be
foundCharacteristics of
bugs:
- symptom and cause can be in different locations

- Symptoms may be caused by human error or timing problems

Debugging is an innate human trait. Some are good at it and some are

not.

Debugging Strategies:

The objective of debugging is to find and correct the cause of a software error which is realizedby a combination of systematic evaluation, intuition and luck. Three strategies are proposed: 1)Brute Force Method.

2)Back Tracking
3)Cause
Elimination

Brute Force: Most common and least efficient method for isolating the cause of a s/w error.

This is applied

when all else fails. Memory dumps are taken, run-time traces are invoked and program is loaded with output statements. Tries to find the cause from the load of information Leads towaste of time and effort.

Back tracking: Common debugging approach. Useful for small programs

Beginning at the system where the symptom has been uncovered, the source code is tracedbackward until the site of the cause is found. More no. of lines implies no. of paths are unmanageable.

Cause Elimination: Based on the concept of Binary partitioning. Data related to error occurenec are organized to isolate potential causes. A "cause hypothesis" is devised and data is used to prove or disprove it. A list of all possible causes is developed and tests areconducted to eliminate each

Automated Debugging: This supplements the above approaches with debugging tools thatprovide semi-automated support like debugging compilers, dynamic debugging aids, test casegenerators, mapping tools etc.

**Regression Testing**: When a new module is added as part of integration testing the software changes.

This may cause problems with the functions which worked properly before. This testing is there-execution of some subset of tests that are already conducted to ensure that changes have not propagatedunintended side effects. It ensures that changes do not introduce unintended behaviour or errors. This can be done manually or automated. Software Quality Conformanceto explicitly stated functional andperformance requirements, explicitly documented development standards, and implicit characteristics that are expected of

All professionally developed software.

Factors that affect software quality can be categorized in two broad groups:
Factors that can be directly measured (e.g. defects uncovered during testing)
Factors that can be measured only indirectly (e.g. usability or maintainability)

**McCall's quality factors**

1.ProductoperationCorrectness

Reliability

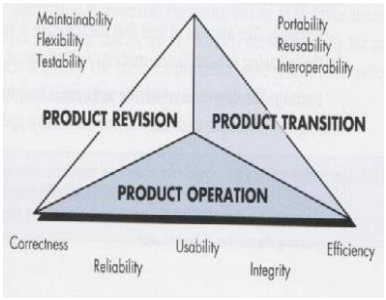
Efficiency

Integrity

Usability

2. Product Revision
Maintainability
Flexibility
3. Product
Transition
Portability
Reusability
Interoperability

**ISO 9126 Quality Factors**




1. Functionality
2.Reliability
3.Usability
4.Efficiency
5.Maintainability
6.Portability

## Product metrics

Product metrics for computer software helps us to assess quality. Measure Provides a quantitative indication of the extent, amount, dimension, capacity or size of someattribute of a product or processMetric(IEEE 93 definition)

A quantitative measure of the degree to which a system, component or process possess agiven attribute Indicator

A metric or a combination of metrics that provide insight into the software process, asoftware project or a product itself

**Product Metrics for analysis, Design, Test and maintenance**

**Product metrics for the Analysis model**

- Function point Metric
- First proposed by Albrecht

Measures the functionality delivered by the systemFP computed from the following parameters

1) Number of external inputs(EIS)
2) Number external outputs(EOS)

**Product metrics for the Analysis model**

Number of external Inquiries(EQS)

Number of Internal Logical Files(ILF)

Number of external interface files(EIFS)

Each parameter is classified as simple, average or complex and weights are assigned as follows

**Product metrics for the Analysis model**

- Information

| Domain | Count | Simple | avg | Complex |
|---|---|---|---|---|
| EIS | | 3 | 4 | 6 |
| EOS | | 4 | 5 | 7 |
| EQS | | 3 | 4 | 6 |
| ILFS | | 7 | 10 | 15 |
| EIFS | | 5 | 7 | 10 |

FP=Count total *[0.65+0.01*E(Fi)]

**Metrics for Design Model**

DSQI(Design Structure Quality Index)US air force has designed the DSQI

Compute s1 to s7 from data and architectural design

S1:Total number of modules
S2:Number of modules whose correct function depends on the data inputS3:Number of modules whose function depends on prior processing S4:Number of data base items

S5:Number of unique database itemsS6: Number of database segments

S7:Number of modules with single entry and exit

Calculate D1 to D6 from s1 to s7 as follows:

D1=1 if standard design is followed otherwise D1=0D2(module independence)=(1-(s2/s1))
D3(module not depending on prior processing)=(1-(s3/s1))D4(Data base size)=(1-(s5/s4))
D5(Database compartmentalization)=(1-(s6/s4) D6(Module entry/exit characteristics)=(1-(s7/s1))

DSQI=sigma of WiDi

i=1 to 6,Wi is weight assigned to Di

If sigma of wi is 1 then all weights are equal to 0.167

DSQI of present design be compared with past DSQI. If DSQI is significantly lower than the average,further design work and review are indicated

METRIC FOR SOURCE CODE

HSS(Halstead Software science)

Primitive measure that may be derived after the code is generated or estimated once design is complete

$n_1$ = the number of distinct operators that appear in a program $n_2$ = the number of distinct operands that appear in a program $N_1$ = the total number of operator occurrences.
$N_2$ = the total number of operand occurrence.Overall program length N can be computed:

$N = n_1 \log2\ n_1 + n_2 \log2\ n_2$ $V = N \log_2(n_1 + n_2)$

$n_1$ = the number of distinct operators that appear in a program
$n_2$ = the number of distinct operands that appear in a program
$N_1$ = the total number of operator occurrences.
$N_2$ = the total number of operand occurrence.Program Level and Effort
PL = 1/[($n_1$ / 2) x ($N_2$ / $n_2$ l)]e = V/PL

## METRICS FOR MAINTENANCE

$M_t$ = the number of modules in the current release
$F_c$ = the number of modules in the current release that have been changed$F_a$ = the number of modules in the current release that have been added.
$F_d$ = the number of modules from the preceding release that were deleted in the currentrelease

The Software Maturity Index, SMI, is defined as:

$$SMI = [M_{t-(F_c + F_a + F_d)/} M_t ]$$

## Metrics for Process And Product

**Software Measurement**:
Software measurement can be categorized as
1)Direct Measure and
2)Indirect Measure

Metrics for Process And ProductDirect Measurement
Direct measure of software process include cost and effort
Direct measure of product include lines of code, Execution speed, memory size, defects per reporting time period.
Indirect Measurement
Indirect measure examines the quality of software product itself(e.g. :- Functionality,complexity, efficiency, reliability and maintainability)

Reasons for measurement
To gain baseline for comparison with future assessmentTo determine status with respect to plan
To predict the size, cost and duration estimate
To improve the product quality and process improvement

**Software Measurement**

The metrics in software Measurement areSize oriented metrics
Function oriented metricsObject oriented metrics
Web based application metric

**Size Oriented Metrics**

It totally concerned with the measurement of software.
A software company maintains a simple record for calculating the size of the software.It includes LOC, Effort,$$,PP document,Error,Defect ,People.

**Function oriented metrics**

Measures the functionality derived by the application
The most widely used function oriented metric is Function pointFunction point is independent of programming language
Measures functionality from user point of view

**Object oriented metric**

Relevant for object oriented programmingBased on the following

- Number of scenarios (Similar to use cases)
- Number of key classesNumber of support
- classesNumber of average support class per
- key class Number of subsystem

**Web based application metric**

Metrics related to web based application measure the following
1.Number of static pages(NSP)
2. Number of dynamic pages(NDP)
Customization(C)=NSP/NSP+NDPC should approach 1

MetricsforSoftware Quality
Measuring Software Quality
1.Correctness=defects/KLOC
2.Maintainability=MTTC(Mean-time to change)3.Integrity=Sigma[1-(threat(1-security))]

Threat : Probability that an attack of specific type will occur within a given time
Security : Probability that an attack of a specific type will be repelled Metrics for SoftwareQuality Usability: Ease of use

Defect Removal Efficiency(DRE) DRE=E/(E+D)

E is the no. of errors found before delivery and D is no. of defects reported after deliveryIdeal value of DRE is 1