UNIT II

CHAPTER 6

Software Measurement and Metrics

Syllabus

Product Metrics – Measures, Metrics, and Indicators, Function-Based Metrics, Metrics for Object-Oriented Design, Operation-Oriented Metrics, User Interface Design Metrics, Metrics for Source Code, Halstead Metrics Applied to Testing, Metrics for Maintenance, Cyclomatic Complexity, Software Measurement - Size-Oriented, Function-Oriented Metrics, Metrics for Software Quality

Syllabus Topic : Product Metrics – Measures, Metrics and Indicators

6.1 Measures, Metrics and Indicators

Q. Explain the terms : measure, metric and indicator.

Measure

- A measure is a quantitative indication of the extent, capacity, or size of some attribute of a product or a process.

Metric

- A metric is the measurement of a particular characteristic of a program's performance or efficiency.
- A metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute.
- Metrics should be collected during and at the end of every phase in the SDLC. They are also a valuable input into process improvement.

Indicators

- An indicator is a combination of metrics that provides insight into the software process, project or product.

6.1.1 Need of Metrics

- Provides a means for control/status reporting
- Tracking Projects against plan - Provides a basis for estimation and facilitates planning for closure of the performance gap

- Identifies risk areas and take timely corrective actions
- Getting early warnings - Provides meters to flag actions for faster and more informed decision making
- Basis for setting benchmarks
- Basis for driving process improvements - Helps in identifying potential problems and areas of improvement
- Tracking process performance against business

6.1.2 Role of Software Metrics

- Software Development Life Cycle (SDLC) performs main role in case of creation of the software. There are primarily two things which are very important to measure at the time of software development which are ***quality*** of the software and ***progress*** of the software. Both things are measured by the SDLC.

Development or Management Team always focuses on the *quality and progress* parameters and for that purpose, it uses metrics for various purposes such as;

1. **Progress Checking :** As there is progress measure it is important to arrange some schedule for that. Actual Progress accessing is done at actual date as per mentioned in the schedule.
2. Making Quality software
3. **Costing and Scheduling :** are the important things which are concern with money and time respectively. Proper estimation of both make fruitful product with accuracy over time.

The Seven Core Metrics

Primarily there are seven core metrics which are classified on the basis of management issues and quality issues. Fig. 6.1.1 depicted metrics;

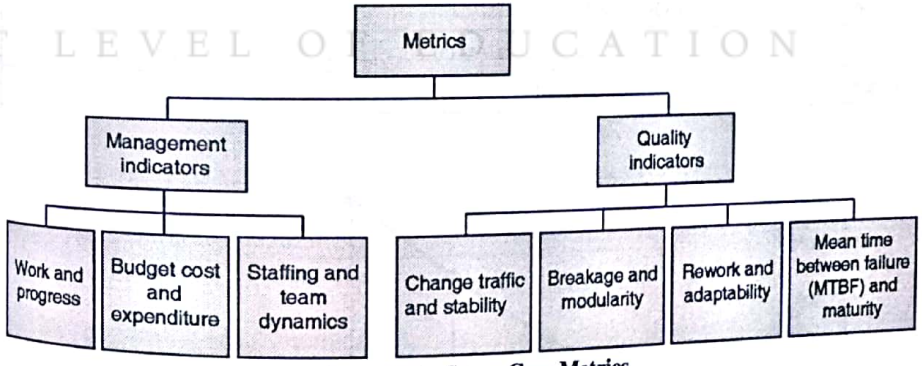

Fig. 6.1.1 : Seven Core Metrics

All the above metrics are classified into two different dimensions :

1. Static Value means objectives of the metrics
2. Dynamic trends concern with managing accomplishment of those objectives.

- All the above seven metrics are useful for managing the organization and project.
- Seven core metrics are based on the metric programs;
  - Common sense
  - Field experience

- Their Attribute
  - o They are very Simple to understand
  - o Each has definite objective
  - o Each time easy to collect, since automatic and nonintrusive collection is there
  - o Easy to interpret
  - o Hard to misinterpret
  - o Continuous assessment
  - o They are acts as communicating bridge between management and engineering personnel for the continuous progress and quality parameters.

I. Management Indicators

There are three important management parameters on which every manager can works for :

o Technical progress o Financial assets o Staffing progress

Project costing and scheduling is the critical job for most of the managers. At the time of the scheduling the project work assessing is again problematic for the managers.

A. Work and Progress

When any project starts then initial duty of the manager is scheduling the project. Planning is the most important task in the scheduling of iterative development of the project. The fact is when some things happen with plan then it is time bounded. Here the progress is most important fact which is carried out by the time scheduling.

The routine task allocated for this metrics for various teams is as follows :

Software teams	Duties
Architecture team	Use case demonstrated
Development team	SCOs closed
Assessment team	SCOs opened, test time executed, evaluation criteria congregate
Management team	Milestone targeted

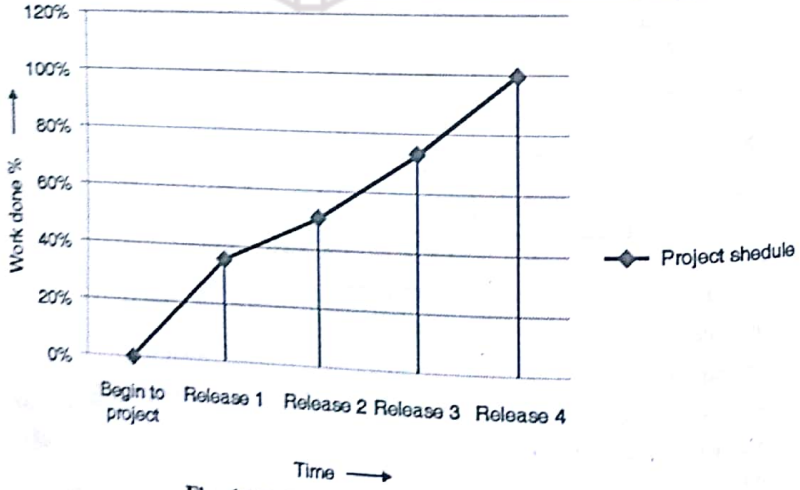

Fig. 6.1.2 : Progress against schedule time

B. Budgeted Cost and Expenditures

Work as per schedule time is important expenditure on the project as per budget cost is also important. Financial progress tracking is another major management indicator which is carried out by the specific standard format approved by the organization. Earned value system is the approach which is used to detailed information of cost and scheduling. All the parameters of the earn value system are indicated in the form of dollar. Some parameters mention below;

1. **Expenditure Plan :** The financial plan schedule spends for the project activity over the previous planned schedule is nothing but expenditure plan.
2. **Actual Progress :** Technically completed plan over the planned progress is actual progress.
3. **Actual Cost :** Whatever actually spent cost for the project profile over the actual schedule.
4. **Earned Value :** Planned cost of the actual progress.
5. **Cost Variance (CV)** = Actual cost – Earned value
   - Positive CV indicates – Over Budget
   - Negative CV indicates – Under Budget
6. **Scheduled Variance (SV)** = Planned cost – Earned Value
   - Positive SV indicates – Behind schedule situation
   - Negative SV indicates – ahead schedule situation

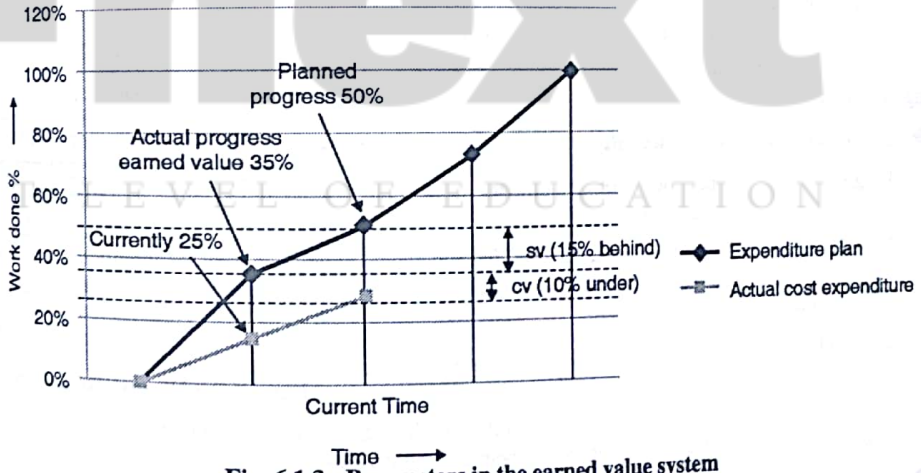

Fig. 6.1.3 : Parameters in the earned value system

C. Staffing and Team Dynamics

Quality staff is essential factor for developing quality software. When quality staff comes together and forms a dynamic team, they produce a long lived and continuously improving software product. It tracks the actual staffing against the planned staffing.

If new staff increases then work progress becomes slow. It affects on the overall project speed. New staff can take the productive time of exiting staff for speed. Low attrition of quality staff is the sign of the success. As the staff like software engineer is continuously working in the same company then they make good returns for the company by their experience.

Now days this kind of staff migrates from one place to another place because of the high and attractive salary package. In this case motivation to such staff is essential.

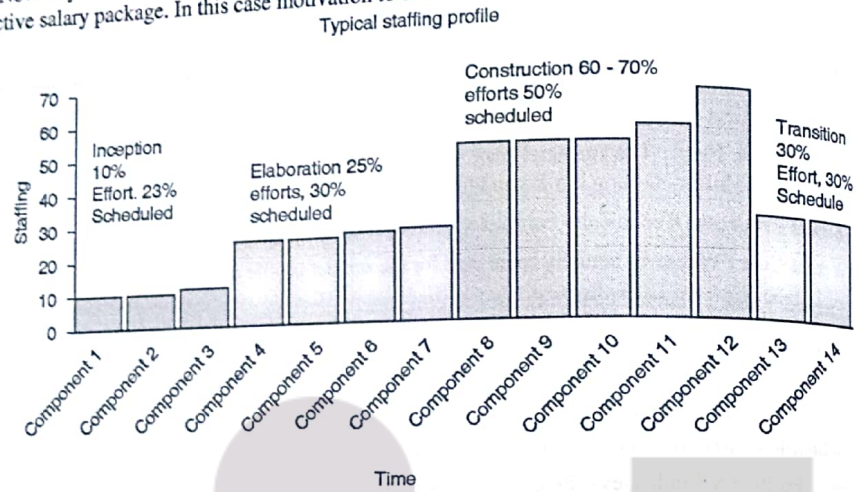

Fig. 6.1.4 : Typical Staffing Profile

II. Quality Indicators

There are four quality indicators which stand on the measurement of the change across evolving baseline engineering data.

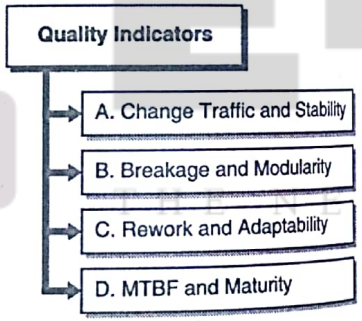

Fig. C6.1 : Quality Indicators

➜ A. Change Traffic and Stability

Progress and quality parameters of the software measure by the change traffic indicator.

1. **Change Traffic :** The number of the SCO's open and closed over the whole life cycle is known as ***Change Traffic***.
2. **Stability :** Relationship between open and closed SCO's is known as ***Stability***.

The information relates to the metrics is collected through overall change, its release type, all number of release, by team, by all its components, by all its subsystems, and so on.

➜ B. Breakage and Modularity

1. **Breakage :** The average extents of change, which is the amount of software baseline that needs rework is known as ***Breakage***.
2. **Modularity :** The average breakage trend over time is known as Modularity.

As time increase breakage trend is also increase, this indicates that product maintainability is suspect.

➜ C. Rework and Adaptability

1. **Rework :** The average cost of change, which is the effort to analyze, resolve and reset all changes to the software baseline is known as ***rework***.

2. **Adaptability :** The rework trend over time is known as adaptability.

➜ D. MTBF and Maturity

1. **Mean time Between Failures :** The average usage time between software faults is known as ***Mean Time Between Failures***. It is calculated by dividing the test hours by the number of type 0 -1 SCOs.
2. **Maturity :** The MTBF trend over time is known as Maturity.

As an error comes in the software projects then that are revised and prevent from those errors. Such errors are categorized into two types: Deterministic errors and non deterministic errors.

Table 6.1.1 show the differentiation table of deterministic and non- deterministic errors;

Table 6.1.1 : Difference between deterministic and non deterministic errors

Sr. No.	Deterministic Errors	Non - deterministic Errors
1.	Physicists named it as Bohr – bugs.	Physicists names it as Heisen – bugs.
2.	Class of errors that always result when the s/w is simulated in certain way.	Probabilistic occurrence of given situation.
3.	Reasons of occurs – coding and change in the single component.	Reasons of occurs – design time errors.

6.1.3 Types of Testing Metrics

Q. List various types of metrics.

Metrics can be categorized into four types :

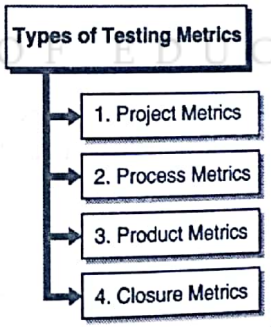

Fig. C6.2 : Types of Testing Metrics

➜ 6.2 Project Metrics

Project metrics are used by a project manager and software team to adapt project work flow and technical activities.

These metrics are useful in :

- minimizing the development schedule by making the necessary adjustment to avoid delays and mitigate problems.

- Assessing the product quality on an ongoing basis

1. Test coverage = (# of Test executed / # of Tests estimated) * 100
2. Defect Density = (Total number of defects found in all the phases + Post delivery defects)/Size

Defect arrival rate - This metric indicates the quality of the application/product under test.

Defect arrival rate = # Of Defects * 100 / # of Test Cases planned for Execution

6.3 Process Metrics

Process metric focuses on the quality achieved as a consequence of a repeatable or managed process. It is usually strategic and planned for long term.

1. Cost of Quality

☞ **% Cost of Quality = (green money + blue money + red money)*100/(Total efforts spent on project)**

where,

- Prevention Cost: (Green Money) : Cost of time spent by the team in implementing the preventive actions identified from project start date to till date.
- Appraisal Cost: (Blue Money): Cost of time spent on review and testing activities from the project start date to till date
- Failure Cost: (Red Money) : Cost of time taken to fix the pre and post-delivery defects. Customer does not pay for this

1. Delivered Defect Rate
2. Defect Injection Rate
3. Rejection Index - measures the Quality of the defects raised
4. Rejection Index = # of Defects rejected / # of Defects raised
5. Review Effectiveness = 100 * Total no. of defects found in review / Total no. of defects
6. Defect Removal Efficiency (DRE) = (No. of pre-delivery defects / Total No. of Defects) * 100

6.4 Product Metrics

Product metrics focus on the quality of deliverables. Product metrics are combined across several projects to produce process metrics

1. Test Case Design Productivity = # Of Test cases (scripts) designed/ Total Test case design effort in hours
2. Test Case Execution Productivity = # Of Test cases executed/ Total Test case executed effort in hours

6.5 Closure Metrics

1. Test Design Review Effort = (Effort spent on Test Case design reviews / Total effort spent on Test Case design) * 100

2. Test Design Rework Effort = (Effort spent on Test Case design review rework / Total effort for spent on Test Case design) * 100
3. KM Effort = (Total Effort spent on preparation of the KM artifacts / Total actual effort for the project) * 100

Syllabus Topic : Function-based Metrics

6.6 Function-based Metrics

Q. Explain Function point metric with an example.

The function point metric (FP) is a means for measuring the functionality delivered by a system.

Function point **data is used in two ways:**

1. as an estimation variable that is used to "size" each element of the software
2. as baseline metric collected from past projects

Baseline metric along with Estimation variable is used to develop cost and effort projections.

The basic units of the Universal Function Points (UFP) are :

- user inputs – user outputs – user inquiries – files
- external attributes i.e. the machine readable interfaces that are used to transmit information to another system.

Table 6.6.1 : Computing the UFP

Measuring Parameter	Count	Weighting Factor Simple	Average	Complex	
No. of user inputs		* 3	4	6	=
No. of user outputs		* 4	5	7	=
No. of user inquiries		* 3	4	6	=
No. of files		* 7	10	15	=
No. of external interfaces		* 5	7	10	=
CountTotal					

To compute UFP, the following formula is used;

$$FP = CountTotal * [0.65 + 0.01 * \sum (Fi)]$$

where, countTotal is the sum of all FP entries shown in the Table 6.6.1 and Fi (where i=1 to 14) are the 'complexity adjustment values' based on answers to the following question;

1. Does the system require reliable backup and recovery?
2. Are data communications required?

3. Are there distributed processing functions?
4. Is performance 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 transactions to be built over multiple operations?
8. Is the master file updated on-line?
9. Are the inputs, outputs, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code reusable?
12. Are installation details included in the design?
13. Is the system designed to adjust on various installations of different organizations?
14. Is the application designed to incorporate the changes and is it easy of use ?

LOC (Lines of Code) & Estimation

- These metrics are useful estimators when a solution is formulated and programming language is known.
- FP and LOC are the accurate predictors of software development effort and cost.
- The relation between LOC and FP depends upon the programming language that is used to implement the software and the quality of the design.
- The Table 6.6.2 provides rough estimates of the average number of LOC required to build a FP in various programming languages.

Table 6.6.2 : Various Comparisons of FPs to LOC based upon the programming language

Language	LOC per UFP
Assembly	320
C	128
Fortran 77	105
Cobol 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

Advantages

- The above data indicates that one LOC of C++ provides approximately 2.3 times the 'functionality' as one LOC of C.
- LOC and FP measures are used to derive productivity metrics.
- Use of higher level programming languages reduces the size, thus, the 'level of abstraction' also changes allowing more focus on architecture.

- The reduced size makes it easier to understand, reuse, maintain and import the packages of classes and objects.
- But, these higher-level abstractions often use high storages and communication bandwidths.

Example 6.6.1 :

Infosoft solutions Pvt. Ltd. uses the biometric system for keeping the attendance of the employees. This company takes the annual maintenance contract for their clients. The system is placed in this company's office that takes the details of the employee's database from the biometric. In the employees database system the employee id is stored, which is captured from biometric system. When the employee's thumb is scanned then the employee id is shown along with system data and time, which is also stored in database. The name of the employee, the time for which the employee has worked at client's locations along with the task they carried out at client's locations is even maintained in the database. The data which is generated from the same is taken into excel sheet by the staff from the office. The weekly report is generated by the staff where how many hours, the employee has worked is displayed. Even the monthly report is generated for the same. The number of hours in a week and average number of hours in a month for an employee is calculated. Explain at least 2 functional requirements given in this case study for each of the parameters required to calculate function point. Calculate unadjusted function point (UFP) assuming that the complexity level is "Simple" for this case study by taking the help of below given table.

Function point	Simple	Average	Complex
External input	3	4	6
External output	4	5	7
Logical internal file	7	10	15
External interface file	5	7	10
External inquiry	3	4	6

Solution :

Functional Requirements

1. **Enrol employee details :** Capture their fingerprints using image acquisition device such as the sensors and process them into usable form.
2. **Store images in database :** Distinctive measures of the enrolling images are extracted and stored in the database as a "template".
3. **Identifying and verifying those images** : These images are matched with the information by comparing a specific template with the templates already stored in the database so as to find if there is any match.
4. **Record the details of the matching identification otherwise generate an alert :** Record the entry date/time, employee id, exit date/ time of the matching identifications.
5. **Calculate the weekly worked hours / monthly worked hours** of every employee based on the recorded details where each record is identified upon the employee's fingerprints as an identification mark.

Calculating Unadjusted Function Point (UFP) for 'simple' complexity level from the given data :

The UFP calculation is broken into two categories with five sub-categories:

- Data Functions :
  - Internal Logical Files
  - External Interface Files
- Transactional Functions :
  - External Outputs
  - External Inputs
  - External Inquiries

Add Employee
Add Employee fingerprints
Scan thumb
Store date and time
} External Inputs

Generate employee details
Show attendance data and time
Show working hours
Show employee tasks
Generate excel sheet
Generate weekly/monthly report
} External Outputs

Query employee
Query client locations
Query tasks
Query over time hours
} External Inquiries

Function type	Number	Weight Factors Simple			
External Input (EI)	4	×	3	=	12
External Output (EO)	6	×	4	=	24
External Queries (EQ)	4	×	3	=	12
Internal Datasets (ILF)	5	×	7	=	35
Interface (EIF)	3	×	5	=	15
	Unadjusted function points (UFP)			=	98

Syllabus Topic : Metrics for Object-Oriented Design

6.7 Metrics for Object-Oriented Design

- OO methodology supports continuous integration of subsystems, classes, interfaces thus, increasing the chances of early detection of risks and incremental corrections without hampering the stability of the development process.

- OO methodology allows 'Architecture first approach' in which integration is an early and continuous life-cycle activity that brings stability in development, allows development and configuration of components in parallel.
- OO architecture creates a clear separation between the unrelated elements of a system, and includes firewalls that prevent a change in one part of the system that may be caused due to the errors in other part of the system thus, rending the structure of the entire architecture.
- OO metrics are used to evaluate the quality of the software. The software engineering metrics are usually associated with coupling, cohesion, reliability, maintainability and fault-proneness. But the OO metrics are specially concerned with;

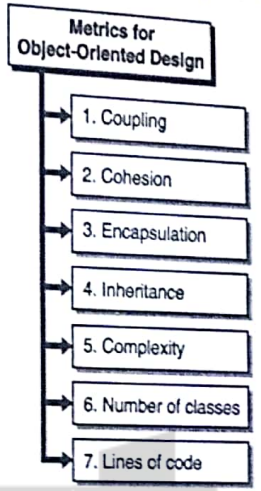

Fig. C6.3 : Metrics for Object-Oriented Design

1. Coupling

It refers to the association between modules. Coupling is a measure of interdependence of two objects.

Example : Objects X and Y are said to be coupled if a method of X calls or accesses a method or variable of Y.

$$\text{Coupling Factor (CF)} = \frac{\text{number of non-inheritance couplings}}{\text{maximum number of inheritance and non-inheritance couplings in a system}}$$

where, , Inheritance couplings arise from derived classes, inherited methods and attributes form its base class.

Desirable value of CF is lower.

2. Cohesion

It refers to how closely the operations in a class are related to each other. There are two ways of measuring cohesion :

i. Calculate the percentage of methods in a class that used the data field - Average the percentages then subtract from 100%. Lower percentages mean greater cohesion of data and methods in the class.

ii. Count the number of disjoint sets produced from the intersection of the sets of attributes used by the methods.

Desirable value of Cohesion is higher - High cohesion increases complexity, thereby increasing the likelihood of errors

3. Encapsulation

It refers to hiding of information means all that is seen of an object is its interface. There are two types of encapsulation measures.

i. **Attribute Hiding Factor (AHF) :** It measures the percentage of invisibilities of attributes in classes i.e. An attribute is said to be visible if it can be accessed by another class or an object. Attributes should be "hidden" within a class by declaring them as private. Desirable value of AHF is higher.

$$AHF = \frac{\text{sum of the invisibilities of all attributes defined in all classes}}{\text{total number of attributes defined in the project}}$$

ii. **Method Hiding Factor (MHF) :** It measures the percentage of invisibilities of methods in classes i.e. the percentage of total classes from which the method is not visible. Desirable value of MHF is higher.

$$MHF = \frac{\text{sum of the invisibilities of all methods defined in all classes}}{\text{total number of methods defined in the project}}$$

➜ **4. Inheritance**

It decreases the complexity by reducing the number of operations and operators, but on the contrary, it makes the maintenance and design difficult. There are two metrics to measure the amount of inheritance in terms of depth and breadth of the inheritance hierarchy.

i. **Depth of Inheritance Tree (DIT) : :** It is the depth from the current class node to the parent class node in the hierarchy tree and is measured by the number of ancestor classes The classes involved in multiple inheritance have maximum DIT. The deeper the class is within the hierarchy, the greater is the potentiality of reusing the inherited methods but increases the complexity. Desirable value of DIT is low.

ii. **Number of Children :** It measures the number of direct subclasses for each class. A class with large number of children is difficult to modify and test. Desirable value of NOC is low.

➜ **5. Complexity**

A class with more functions than its parent class is considered to be more complex and more error prone thereby limiting the possibility of reuse.

➜ **6. Number of Classes**

The projects having more number of classes are better abstracted. Desirable value of Number of Classes is higher.

➜ **7. Lines of Code**

The projects with fewer lines of code have superior design and require less maintenance. Desirable value of LOC is lower.

Syllabus Topic : Operation-Oriented Metrics

6.7.1 Operation-oriented Metrics

Q. What is the need of OO metrics and explain in brief.

- Average Operation Size (OSavg) = # of messages sent by the operation
- Operation Complexity (OC)
- Average number of Parameters per Operation (NPavg)

6.7.2 Use Case Point (UCP) Estimation Method

The UCP estimation method was introduced by Gustav Karner.

This method is implemented using a spread sheet (excel sheet).

- Each actor and use case is categorized according to complexity and is assigned a weight.
- *Complexity* of a use case is measured in number of transactions.
- The *unadjusted use case points* are calculated by adding weights for each actor and use case based on *13 technical factors* and *8 environmental factors*.

13 Technical Factors		
Sr. No.	**Factor description**	**Weight**
1.	Distributed system	2
2.	Response or throughput performance objective	1
3.	End-user efficiency	1
4.	Complex internal processing	1
5.	Code must be reusable	1
6.	Easy to install	0.5
7.	Easy to use	0.5
8.	Portable	2
9.	Easy to change	1
10.	Concurrent	1
11.	Includes special security features	1
12.	Provides direct access for third parties	1
13.	Special user training facilities are required	1
8 Environmental Factors		
Sr. No.	**Factor description**	**Weight**
1.	Familiar with RUP	1.5
2.	Application experience	0.5
3.	Object-oriented experience	1
4.	Lead analyst capability	0.5
5.	Motivation	1
6.	Stable requirements	2
7.	Part-time workers	−1
8.	Difficult programming language	−1

- The unadjusted use case weights (UUCW) by adding weights for each use case.
- The unadjusted actor weight (UAW) is calculated by adding weights for each actor.
- The unadjusted use case points (UUCP) is calculated as :

$$UUCP = UAW + UUCW.$$

- Calculate technical factor, $TF = .6 + (.01 * \Sigma 1..13Tn * Weightn)$.
- Calculate environmental factor,
- $EF = 1.4 + (-.03 * \Sigma 1..8Fn * Weightn)$.
- Calculate Use Case Points as: $UCP = UUCP * TF * EF$
- Finally, calculate UCP * Productivity factor

Syllabus Topic : Other Testing Metrics

6.8 Other Testing Metrics

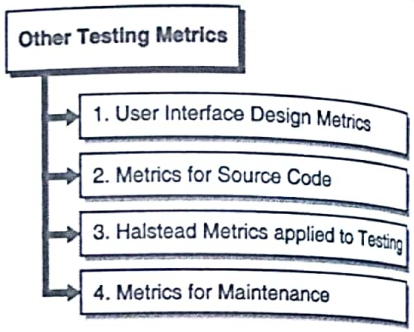

Fig. C6.4 : Other Testing Metrics

➜ **1. User Interface Design Metrics**

Layout appropriateness: It is a function of layout entities, the geographic position and the "cost" of making transitions among entities.

➜ **2. Metrics for Source Code**

Halstead's Software Science is a collection of all metrics predicated based on the number of operators and operands within a component.

➜ **3. Halstead Metrics applied to Testing**

Following design metrics have a direct influence on the "testability" of an OO system.

- Lack of cohesion in methods (LCOM).
- Public access to data members (PAD).
- Fan-in (FIN).
- Percent public and protected (PAP).
- Number of root classes (NOR).
- Number of children (NOC) and depth of the inheritance tree (DIT).

➜ **4. Metrics for Maintenance**

Q. Brief about metric of maintenance.

- Software Maturity Index (SMI) = $[M_T - (F_a + F_c + F_d)] / M_T$

where,

M_T : Total # of modules in the current release

F_a : # of modules added in the current release

F_c : # of modules changed in the current release

F_d : # of modules deleted in the current release

Syllabus Topic : Cyclomatic Complexity

6.9 Cyclomatic Complexity

- Cyclomatic Complexity testing is a process that measures the complexity of a program.
- Steps to calculate the Cyclomatic Complexity are as follows:

Step 1 : A program structure is first represented through a flowchart. It is a traditional means for pictorially describing a program's logic because describing the program code from a detailed flow chart is a very simple process.

Step 2: Flow charts are then converted in to a control flow graph containing only the nodes and edges. *Steps to convert the flow chart into flow graph are listed down:*

Step 2.1 : Identify the predicates (decision points) in the flow chart.

Step 2.2 : Ensure that the predicates are simple else break up a condition into simple predicates.

Step 2.3 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate into one node.

Step 2.4 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate check into one node and have two edges emanating from this one node. Such nodes with two edges emanating from them are called predicate nodes.

Step 2.5 : Make sure that all the edges terminate at some same node. Add an extra node at the end to represent that all the sets of sequential statements reach to that node.

Step 3: Calculate Cyclomatic Complexity (CC) using one of the either two methods:

Method 1 : From flow Chart, CC = P + 1 where P is number of predicates.

Method 2 : From Flow Graph, CC = E-N+2 where E is no. of edges, N is no. of nodes.

Method 3 : From Flow Graph, CC = # of closed Regions + 1 open Region

Example 6.9.1 :

1.	Wait for Card to be inserted	2.	IF card is valid THEN
3.	display "Enter PIN number"	4.	IF PIN is valid THEN
5.	select transaction	6.	ELSE
7.	display "PIN invalid"	8.	ELSE
9.	Reject card	10.	END

Solution :

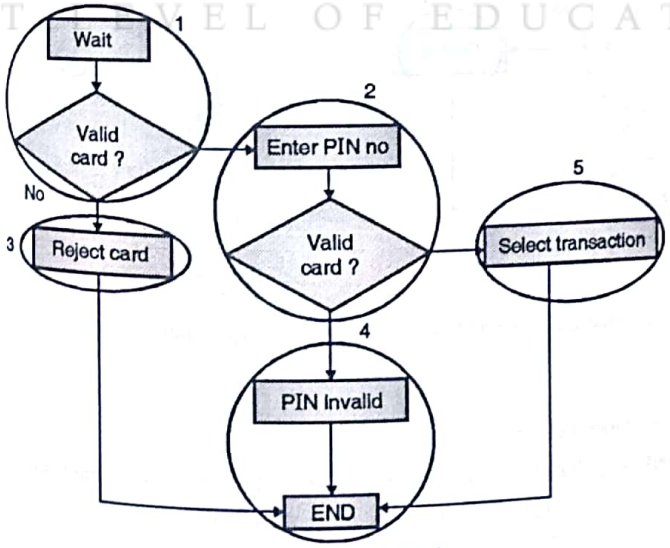

Fig. P. 6.9.1

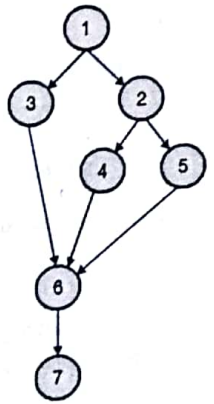

Fig. P . 6.9.1(a)

Number of Conditions (predicates) to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 Open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Example 6.9.2 :

Program with complex predicates	Same Program with simple predicates

Read (A)
If A > 0 and A < 5 Then
Print "A"
End If
	Read (A)
	If A > 0 Then
	If A < 5 Then
	Print "A"
	End If
	End If

Solution :

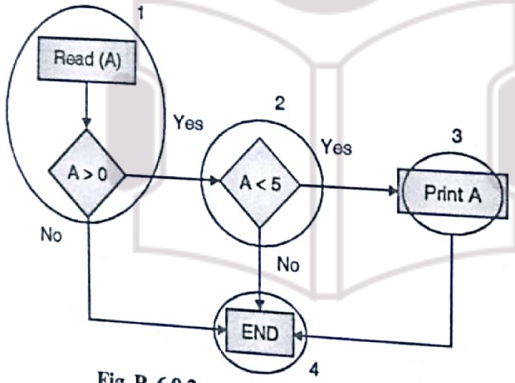

Fig. P. 6.9.2

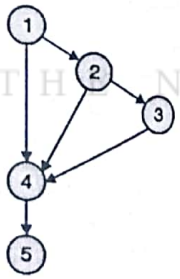

Fig. P. 6.9.2(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 Open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 4 5

Path 2: 1 2 4 5

Path 3: 1 2 3 4 5

Example 6.9.3 :

```
1.  Read A                  2.  Read B                  3.  IF A > 0 THEN
4.  IF B = 0 THEN           5.  Print "No values"       6.  ELSE
7.  Print B                 8.  IF A > 21 THEN          9.  Print A
10. ENDIF                   11. ENDIF                   12. ENDIF
```

Solution :

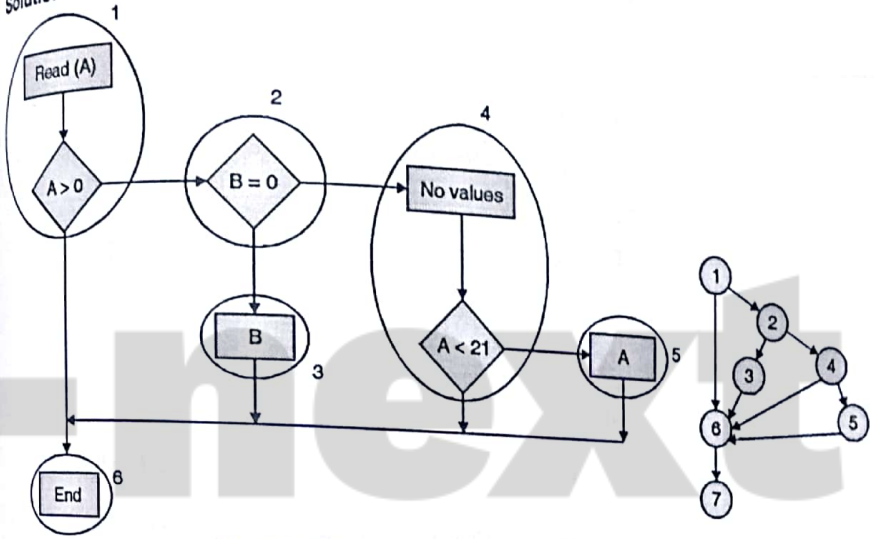

Fig. P. 6.9.3

Fig. P. 6.9.3(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 3

CC = P + 1 = 3 + 1 = 4

CC = # of Closed Regions + 1 open Region = 3 + 1 = 4

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 6 7 Path 2: 1 2 3 6 7 Path 3: 1 2 4 6 7

Path 4; 1 2 4 5 6 7

Example 6.9.4 :

```
1.  Read A                  2.  Read B                  3.  IF A < 0 THEN
4.  Print "A negative"      5.  ELSE                    6.  Print "A positive"
7.  ENDIF                   8.  IF B < 0 THEN           9.  Print "B negative"
10. ELSE                    11. Print "B positive"      12. ENDIF
```

Solution :

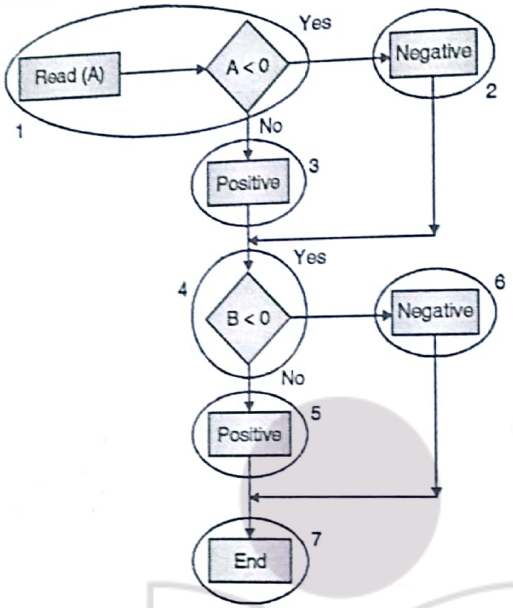

Fig. P. 6.9.4

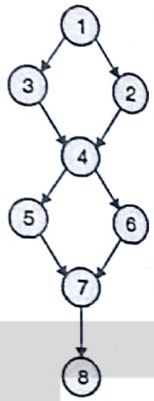

Fig. P. 6.9.4(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 4 5 7 8 Path 2: 1 3 4 6 7 8 Path 3: 1 2 4 5 7 8

Path 4; 1 2 4 6 7 8

Example 6.9.5 :

Check the cyclomateric complexity for a program of adding 100 integers. It should also check for valid boundaries and valid values. Design the Test Cases for such code.

Solution :

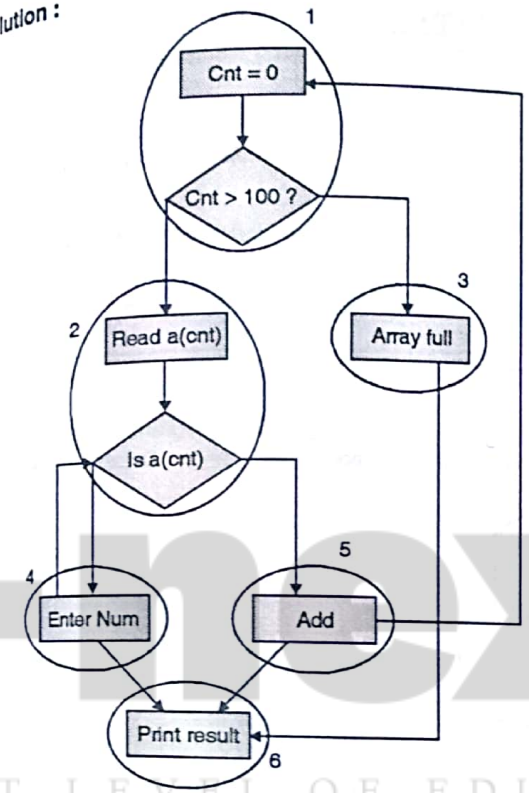

Fig. P. 6.9.5

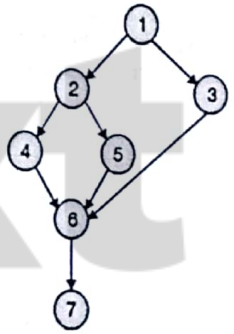

Fig. P. 6.9.5(a)

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

CC = P + 1 = 2 + 1 = 3

CC = # of Closed Regions + 1 open Region = 2 + 1 = 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Test Description :

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
No. of values in array must be 100 integers to perform addition	= 100 numbers	< 100 numbers > 100 numbers Non digit	100 numbers	99 numbers 101 numbers

Test case :

Test Case Id	Conditions	Input	Expected Output
1	No. of values in array must be 100 integers to perform addition	Values till count =100	Addition of Numbers
2		Values less than count =100	Accept more numbers
3		Values greater than count =100	Array Full Warning
4	Non digits not acceptable	Any character or symbol	Enter number warning

Example 6.9.6 :

A program reads three integer values as three sides of a triangle. The program prints a message indicating that whether the triangle is right angle triangle or equilateral triangle. Draw the flow graph, calculate cyclometric complexity.

Solution :

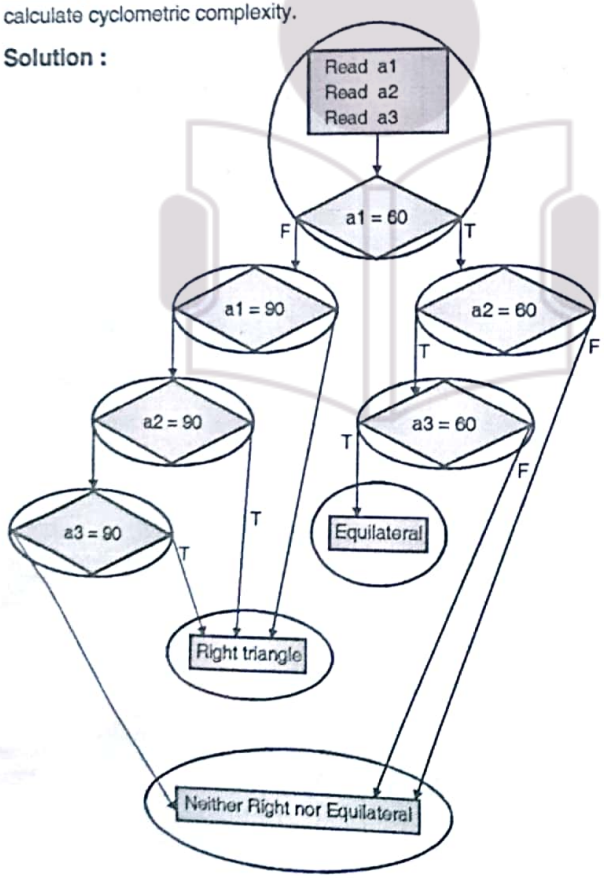

Fig. P. 6.9.6

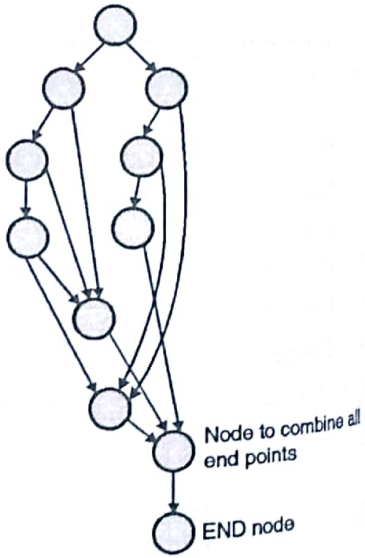

Fig. P. 6.9.6(a)

$CC = P + 1 = 6 + 1 = 7$

$CC = \#$ of Closed Regions + 1 open Region $= 6 + 1 = 7$

Example 6.9.7 :

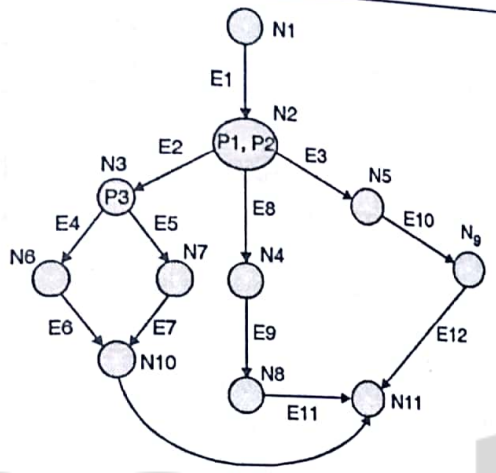

Fig. P. 6.9.7

Solution :

$CC = P + 1 = 3 + 1 = 4$

$CC = E - N + 2 = 13 - 11 + 2 = 2 + 2 = 4$

Syllabus Topic : Software Measurement

6.10 Software Measurement

Q. Based on which two parameters, the software is measured. Explain.

Organizations combine the metrics that come from different individuals or projects depending on the size and complexity.

1. Size oriented metrics (lines of code approach) such as errors per KLOC (thousand lines of code), defects per KLOC, documentation per KLOC, errors per person-month.
2. Function oriented metrics (function pointed approach) such as errors per FP, defects per FP, documentation per FP, FP per person-month

Syllabus Topic : Metrics for Software Quality

6.11 Metrics for Software Quality

Q. Explain metrics of software Quality.

The following are the software quality factors and the needed metrics and measurements for each :

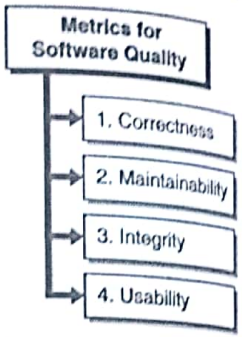

Fig. C6.5 : Metrics for Software Quality

→ **1. Correctness**

- Correctness is measured as the degree to which the software performs its required functionality.
- The most common measure for correctness is number of defects per KLOC (Kilo Lines Of Code) where a defect is defined as a verified lack of conformance to requirements.

→ **2. Maintainability**

- Maintainability is the ease with which a program can be :
- An application can be corrected if an error is encountered
- a defect can be fixed
- a application is adapted in a changed environment
  - a application is enhanced.
- A simple time oriented metric is Mean Time To Change (MTTC) i.e. the time it takes to analyze the changed request, design an appropriate modification, implement the changed modification, test it and distribute the change for all users. Below is the figure which gives the scenario of online detection and offline repair. The measures MTBF (Mean Time Between Failure), MTTD (Mean Time To Detect) and MTTR (Mean Time To Repair) are the average times to failure, to detection and to repair.

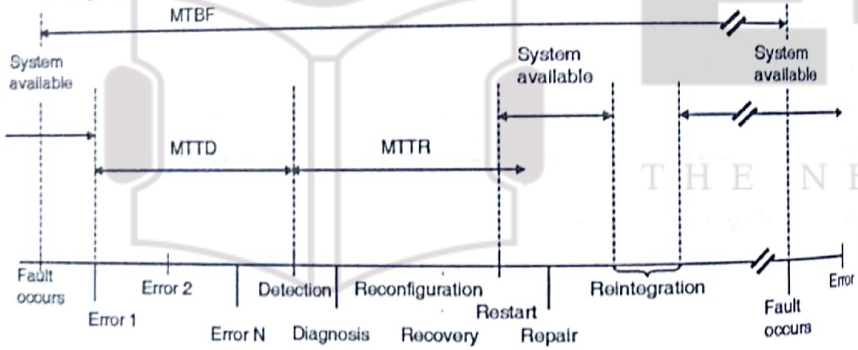

Fig. 6.11.1 : MTBF, MTTD and MTTR measures of maintainability

→ **3. Integrity**

- This attribute measures a system's ability to withstand attacks to its security. Attacks can be made on all three components of software: programs, data and documents.
- We cam measure integrity through following two additional attribute :
  1. Threat: It is the probability that an attack of a specific type will occur within a given time.
  2. Security: It is the probability that the attack of a specific type will be repelled. The integrity of a system van be then defined as :

  Integrity = Σ (1-(threat * (1-security)))

→ **4. Usability**

- If a program is not easy to use, it is often meant to failure. Even if the functions that it performs are valuable.

- Usability is an attempt to quantify ease-of-use and can be measured in terms of characteristics.

Defect Removal Efficiency

Q. What is Defect Removal Efficiency ?

- It is a quality metric that provides benefit at both the project and process level.
- DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- DRE is defined as: DRE = E/(E+D) where E is the number of errors found before delivery of the software to the end user and D is the number of defects found after delivery. The ideal value for DRE is 1 i.e. no defects are found in the software. Generally, D is greater than 0 but DRE can be 1 when, as E increases the overall value of DRE begins to approach 1.

Defect Detection Index

Defect Detection Index = # of defects detected in each phase / total # of defects planned to be detected in each phase

Defect Amplification and Removal

- A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design and coding steps of the software engineering process.

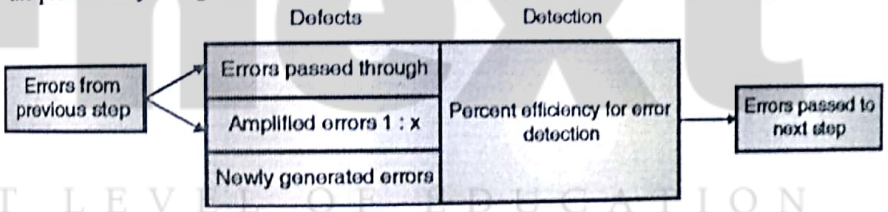

Fig. 6.11.2 : Defect Amplification model

- In case if the technical reviews fail to discover the newly generated defects, these may get amplified. The figure represents the percent of efficiency in detecting errors

Establishing a Software Metric Program

Following are the steps to conduct a goal-driven software metric program :

Step 1 : Identify the business goals
Step 2 : Identify what you want to learn
Step 3 : Identify the sub goals
Step 4 : Identify the related entities and attributes
Step 5 : Formalize your goals
Step 6 : Identify the quantifiable questions so as to achieve your goals.
Step 7 : Identify the data elements to help answer your questions.
Step 8 : Define the measures to be used
Step 9 : Identify the actions to implement the measures.
Step 10 : Prepare a plan for implementing the measures.

Review Questions

Q. 1 Explain the terms : measure, metric and indicator.

Q. 2 List various types of metrics.

Q. 3 What is Defect Removal Efficiency ?

Q. 4 Explain Function point metric with an example.

Q. 5 What is the need of OO metrics and explain in brief.

Q. 6 Brief about metric of maintenance.

Q. 7 Explain metrics of software Quality.

Q. 8 Based on which two parameters, the software is measured. Explain.

□□□

UNIT II

CHAPTER 7

Software Project Management Estimation

Syllabus :

Estimation in Project Planning Process – Software Scope and Feasibility, Resource Estimation, Empirical Estimation Models – COCOMO II, Estimation for Agile Development, The Make/Buy Decision

Syllabus Topic : Estimation in Project Planning Process

7.1 Estimation in Project Planning Process

- Planning and estimating are iterative processes which continue throughout the course of a project.
- Software costs often dominate computer system costs. The costs of software are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- The total cost required in developing a software product can be categorized as below :
  i. 60% of development costs.
  ii. 40% are testing costs.

Cost estimating problems occur due to following uncertainties

- Inability to accurately size a software project, Inability to accurately specify a software development and support environment.
- Improper assessment of staffing levels and skills, and Lack of well-defined requirements for the specific software activity being estimated.

Four types of cost estimates represent various levels of Reliability

- **Conceptual Estimate :** Often inaccurate because there are too many unknowns.
- **Preliminary Estimate :** Used to develop initial budget, more precise.
- **Detailed Estimate :** Serves as a basis for daily project control.
- **Definitive Estimate :** Accuracy should be within 10% of final cost.