

# CHAPTER TWO

## **Software Metric**

- Measures
- Metric
- Indicators
- Software Management
- Metric for software quality
- Statistical Quality Control
- Metric for small Organizations

# Software Metrics

- Measure : A measure provides a **quantitative indication** of the extent, amount, dimension, capacity , or size of some attribute of a product or process
- Metric : Metric as “ a **quantitative measure** of the **degree to** which a system, component, or process possesses a given attribute.
- Indicator: An indicator is a metric or condition of the metrics that provides **insight** into the software process, a software project, or the product itself.

# Software Metrics

- Metrics strongly support software project **management activities**
- They relate to the four functions of management as follows:
  - **Planning**
  - **Organizing**
  - **Controlling**
  - **Improving**

# Size Estimation Metrics

- Size of a program is not the number **of bytes** that the source code occupies
- It is **not the size** of the executable code
- It is an indicator of the effort and time required to develop the program.
- Size of program indicates development complexity
- Estimating the problem size is fundamental to estimating **the effort, time, and cost** of planned software.

# Software Measurement

- **Direct measure and indirect measure.**
- Direct measures of the software include how many lines of **code (LOC) produced, execution speed, memory size, and defects reported.**
- Indirect measures include functionality, quality, complexity, efficiency, reliability, and maintainability of the software.

# Lines of Code

- The simplest among all metrics available to estimate project size
- Project size estimated by counting the number of source instructions
- Lines used for commenting, header lines ignored
- To find LOC at the beginning of a project divide module into sub modules and so on until size of each module can be predicted

# Disadvantages of LOC

- Gives a numerical value of problem size that vary widely with individual coding style

if( x>y )	x > y ? x++ : y++;
then x++;	
else	
y++;	

- Effort needed for analysis, design , coding, testing etc (not just coding)

# Disadvantages of LOC

- # Larger Code size → Better Quality?
- # Logical Complexity?

Complex Logic → More Effort

Simple Logic → Less Effort

```
while(i<4) {  
    printf("testing");  
}
```

```
printf("testing");  
printf("testing");  
printf("testing");  
printf("testing");
```



# Disadvantage of LOC

- Accurate computation of LOC only after project completion!!

# Lines of code

- **1. Cost per line of code** = Labor rate / Productivity
- **2. Estimated project cost** = Estimated line of code/cost per line of code
- **3. Estimate labor effort** = Estimated line of code / Productivity

# Example Of LOC

- Estimated line of code = 33,200
- Productivity = 620 LOC/PM
- Labor Rate = \$ 8000/PM
- Cost per line of code = ?
- Estimated Project cost = ?
- Estimate Labor Effort = ?

# Example of LOC

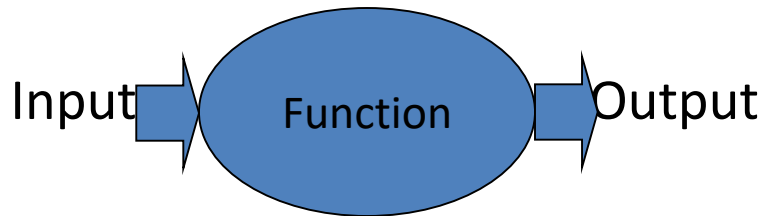
$$\square \text{Cost per line of code} = \$8000/620 \\ = \$13$$


$$\square \text{Estimated project cost} = 33,200 * 13 \\ = \$431,600$$

$$\square \text{Estimated Labor effort} = \text{LOC} / \text{productivity} \\ = 33,200/620 \\ = 54 \text{ PM}$$

# Function Point Metric

- Size of software product computed directly **from problem specification**
- Size of software = number of different functions/ features it supports



- Many features  Larger size
- Apart from that size depends on
  - number of files
  - number of interfaces
  - number of enquiries

# Functional Point

Size of Function Point (FP)= Weighted sum of these five problem characteristics

**1.Number of inputs:** Data items input by user

(Group of user inputs taken together)

» Employee

- Name
- Age
- Sex
- Address

Account

- Account Name
- Account Number
- Account Open Date

# Functional Point Metric

- 2. Number of Outputs:** Reports, Screen outputs, Error Messages
- 3. Number of inquiries:** Interactive queries made by users
- 4. Number of Files:** Logical files e.g. data structures, physical files
- 5. Number of interfaces:** Interfaces for exchanging information e.g. disk, tapes, communication links

# Functional Point

Measurement parameter	Count	Weighting factor					
		Simple Average			Complex		
Number of user inputs	<input type="text"/>	x	3	4	6	=	<input type="text"/>
Number of user outputs	<input type="text"/>	x	4	5	7	=	<input type="text"/>
Number of user inquiries	<input type="text"/>	x	3	4	6	=	<input type="text"/>
Number of files	<input type="text"/>	x	7	10	15	=	<input type="text"/>
Number of external interfaces	<input type="text"/>	x	5	7	10	=	<input type="text"/>
Count total	<input type="text"/>						<input type="text"/>



# Functional point

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
<i>Count total</i>						320

<b>Factor</b>	<b>Value</b>
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
<b>Value adjustment factor</b>	<b>1.17</b>

# Functional Point

- Expected Value for estimate variable size (s) = ?
- Make optimistic , most likely, pessimistic , estimate for each item, then compute expected value .

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

# Functional Point

Consider a project with the following functional units :

- Number of user inputs = 50
- Number of user outputs = 40
- Number of user enquiries = 35
- Number of user files = 06
- Number of external interfaces = 04
- Assuming all complexity adjustment factors and weighing factors as average, the function points for the project will be;

# Functional Point

- Characteristics for weights are

**0 = No influence, 1= Incidental, 2= moderate, 3= Average, 4 = Significant, 5 = Essential**

# Functional Point

- **Function point ( FP )= UFP x VAF**

Where ,UFP = Unadjusted function point ,  
VAF =Value Adjustment Factor

# FP

- $FP_{\text{estimated}} = 672$
- Organizational productivity = 6.5 FP/PM
- Labor Rate = \$8000 /PM
- Cost per FP =  $\$8000/6.5 = \$1230$
- Effort =  $Fp_{\text{estimated}}/\text{Productivity} = 672/6.5 = 103 \text{ PM}$
- Total project cost =  $(672 * 1230) = \$826560$

# Categories of Metrics

- Product Metrics
- Process Metrics
- Project Metrics



# Characteristics of Metrics

- Product Metrics : Product metric describes the characteristics of the product, such as **size, complexity, performance, efficiency.**

# Characteristics of Metrics

**Process Metrics:** Process metric describe **effectiveness and quality** of the process. E.g

- **Effort** required in the process
- **Time** to produce the product
- Number of **defects** found during testing

# Categories of Metrics

**Project Metrics:** Project metrics describe the project characteristics and execution.

E.g

- Number of software developer
- Staffing pattern over the life cycle of the software
- Cost and schedule
- Productivity

# Attribute of Effective Software Metrics

1. Simple and computable
2. Empirically and Intuitively Persuasive (satisfy Engineers' intuitive)
3. Consistent and Objective
4. Consistent in the use of units and dimensions
5. Programming Language Independent
6. An Effective Mechanism for High Quality Feedback

# Metric for Software Quality

- Software quality can be measured through the software Engineering process, **before release** to customer and **after release** to the customer
- The main goal of software engineering is to produce a high-quality system.
- A good software Engineer and good software engineering must measure if **high quality** is to be realized .

# Metric for software quality

## Measuring Quality

- Indicators to measure the quality
  - Correctness
  - Maintainability
  - Integrity (loyally functioning or not )
  - Usability (user friendly or not)

# Defect Removal Efficiency

- A quality metric that provides **benefit at both the project and process** level is defect removal efficiency (DRE)
- DRE is computed as

$$DRE = E/E+D$$

Where E = Errors found before delivery of the software

D = Defects found after the delivery

# Metric for small Organization

Organization might select the following set of easily collected measures:

- Time (hours or days) elapsed from the time a request is made until evaluation is complete,  $t_{queue}$ .
- Effort (person-hours) to perform the evaluation,  $W_{eval}$ .
- Time (hours or days) elapsed from completion of evaluation to assignment of



# Metric for small Organization

- comprehensive software metrics programs
- software organizations of all sizes measure
- use the resultant metrics to help improve their local software.

# Metric for small Organization

- change order to personnel,  $teval$ .
- Effort (person-hours) required to make the change,  $Wchange$ .
- Time required (hours or days) to make the change,  $tchange$ .
- Errors uncovered during work to make change,  $Echange$ .
- Defects uncovered after change is released to the customer base,  $Dchange$ .

# Metric for Small organization

- Defect removal efficiency

$$DRE = \frac{E_{\text{change}}}{E_{\text{change}} + D_{\text{change}}}$$



# Project Estimation Technique

- Empirical estimation technique
- Heuristic Technique
- Analytic estimation technique

# Project Estimation Technique

- Empirical Estimation : making an educated guess of the project using past experience e.g Delphi and Expert judges
- Heuristic Techniques: Project parameter can be modeled by the mathematical expression.
- Analytic Estimation : Like Heuristic Technique but supports scientific facts .

# COCOMO II MODEL

- The constructive cost model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm
- The model uses a basic regression formula, with some parameters that are derived from historical project data and current project characteristic

# COCOMO II MODEL

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
- **Application composition model:** Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- **Early design stage model:** Used once requirements have been stabilized and basic software architecture has been established.
- **Post-architecture-stage model:** Used during the construction of the software.

# Basic COCOMO-II Model

- COCOMO applies to three classes of software projects:
  - **Organic:** Developing well understood application programs, small experienced team
  - **Semi Detached:** mix of experienced and non-experienced team
  - **Embedded:** strongly coupled to computer hardware



- Basic COCOMO
  - $\text{Effort} = a (\text{KLOC})^b \text{ PM}$
  - $\text{Time} = c (\text{Effort})^d \text{ Months}$
  - $\text{Number of people required} = \frac{(\text{Effort applied})}{(\text{Development time})}$

**Example:** The size of organic software is estimated to be 32,000 LOC. The average salary for software engineering is Rs. 15000/- per month. What will be effort and time for the completion of the project?

Solution:

- **Effort applied** =  $2.4 \times (32)^{1.05}$  PM = 91.33 PM (Since: 32000 LOC = 32KLOC) ▪ **Time** =  $2.5 \times (91.33)^{0.38}$  Month = 13.899 Months
- **Cost** = Time x Average salary per month =  $13.899 \times 15000$  = Rs. 208480.85
- **People required** = (Effort applied) / (development time) =  $6.57 = 7$  persons