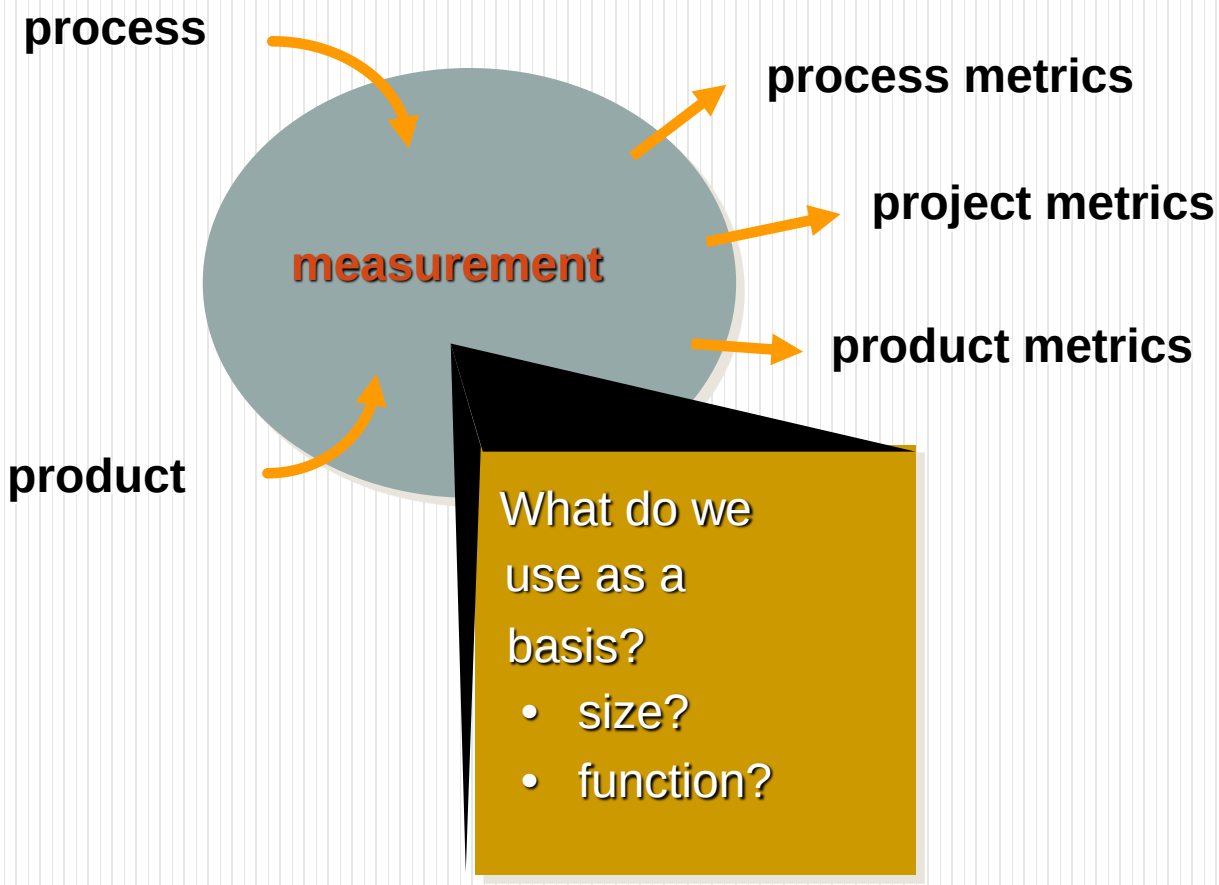
Process and Project Metrics
&
Estimation of Software Projects

Chapter 25 & 26

Roger Pressman – 7th Edition

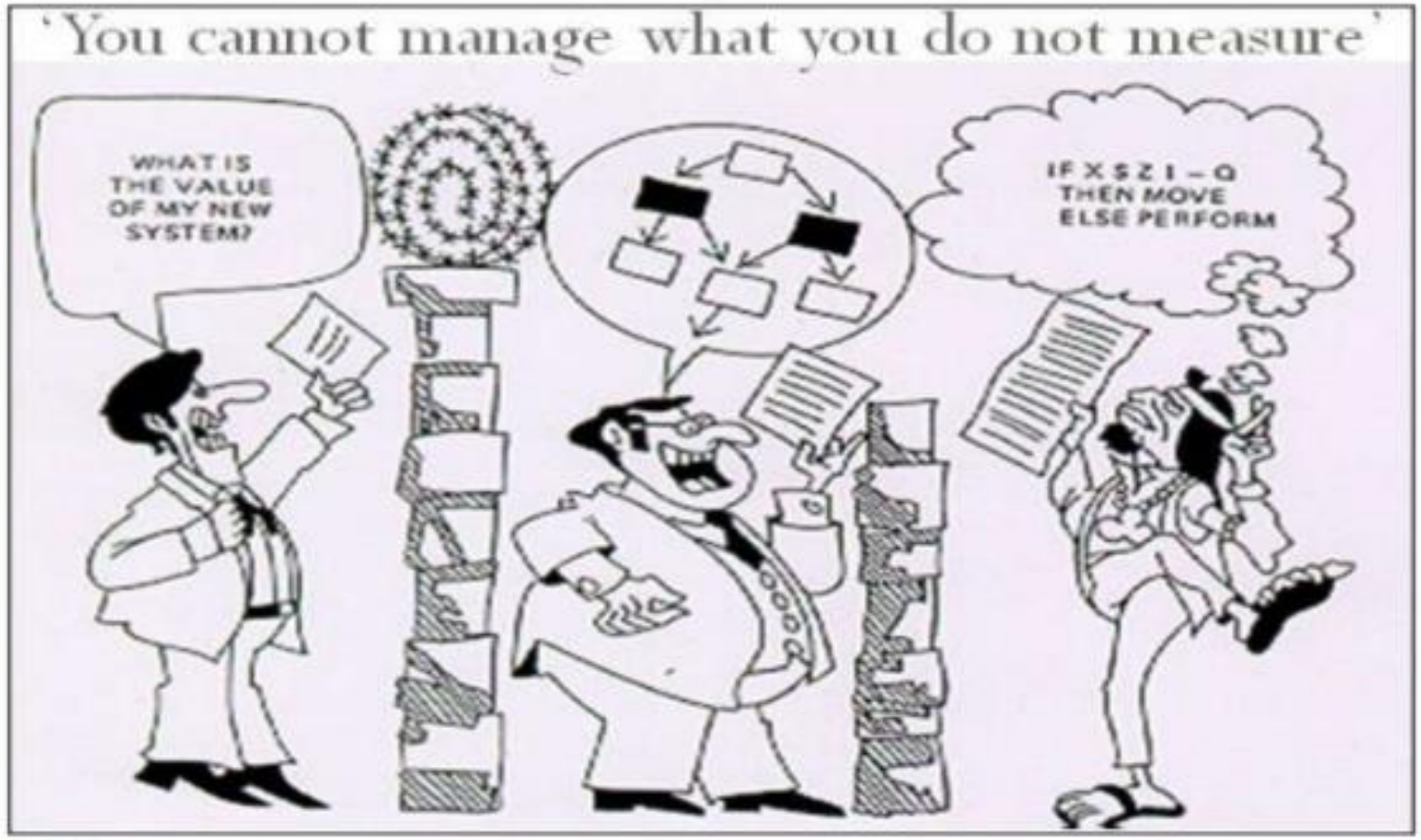
A Good Manager Measures



Software Metrics

- Software metrics can be classified into three categories –
- **Product metrics** – Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** – These characteristics can be used to improve the development and maintenance activities of the software.
- **Project metrics** – This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Why do we measure?



What do you see on the surface?

The image represents the tip of an iceberg. The real issue is not the tip, but what is under the surface of the water and can not be seen. The same is true when you design a software application.

Airline surface

- This appears on the surface to be a simple inquiry, but this is extremely complex. The process actually includes 1,000's of elementary processes, but the end user is only exposed to a very simple process.

- All possible routes are calculated, city names are converted to their international three characters, interfaces are sent to all the airline carriers (each one being unique), this is an extremely complex and robust process! When we size software applications we want to understand what is exposed and what is under the surface.

Why Do We Measure?

- Assess the status of an ongoing project
- Track potential risks
- Uncover problem areas before they go “critical”
- Adjust work flow or tasks
- Evaluate the project team’s ability to control quality of software work products

Measures and Metrics

- **A measure**
  - A measure is “an amount or degree of something.” E.g. We have two measures: “the number of customers acquired over a period of time”; and “funds invested in the marketing.”
- **A metric**
  - A metric is a standard of measurement.
  - **Metric is a derivative of measure.**
  - An example: we use two measures “investments in the marketing” and “the number of customers acquired” to calculate **a total cost to gain a new customer.** That’s a metric. It is a derivative of two measures.

How to use metrics?

- Metrics derived are used by a project manager and software team to adapt project workflow and technical activities.
- First comes estimation. Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software work.
- As the project proceeds, measures of effort and calendar time expended are compared to original estimates and the project schedule. The project manager uses these data to monitor and control progress.

Project Metrics

- Used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- Used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- Every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Metrics in the Process Domain

- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as
  - Errors uncovered before release of the software.
  - Defects delivered to and reported by the end users.
  - Work products delivered.
  - Human effort expended.
  - Calendar time(deadlines).
  - Conformance to the schedule.
  - Time and effort to complete each generic activity.

Categories of Software Measurement

- Two categories of software measurement
  - Direct measures of the
    - Software process (cost, effort, etc.)
    - Software product (lines of code produced, execution speed, defects reported over time, etc.)
  - Indirect measures of the
    - Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)
- Project metrics can be consolidated to create process metrics for an organization

Size-Oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value.
- Typical size oriented metrics are as foll
  - errors per KLOC (thousand lines of cod
  - defects per KLOC
  - $ per LOC
  - pages of documentation per KLOC
  - errors per person-month
  - errors per review hour
  - LOC per person-month
  - $ per page of documentation

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
...	...	...	...	...	...	...	...

LOC (lines of code)

- The LOC measure is used to measure size of the software.
- **Advantages of LOC**
  - Simple to measure
- **Drawbacks of LOC**
  - It is defined on code.
  - Lack of accountability – coding phase is only 30-35% of the total effort.
  - It characterises only one specific view of size, namely length, it takes no account of functionality or complexity
  - Bad software design may cause excessive line of code
  - It is language dependent
  - Users cannot easily understand it

Function-Oriented Metrics

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value.
- Most widely used metric of this type is the function point.
- A function point is a unit of measurement to express the amount of business functionality an information system provides to a user.
- Function points are a unit measure for software much like an hour is to measuring time, miles are to measuring distance.
- Typical function oriented metrics are as follows:
  - errors per FP
  - defects per FP
  - $ per FP
  - pages of documentation per FP
  - FP per person-month

Person - month

- Person Month : **It's mean amount of work performed by the average worker in one month.**
- If a project requires 12 persons-months of development time, then
- 4 developers will spend 3 months for 12 persons-months project.
  - **ProjectScopeInPersonsMonths / NumberOfDevelopers = NumberOfRequiredMonthsForProject**
- 4 months required for 3 developers to finish 12 persons-months project.
  - **ProjectScopeInPersonsMonths / NumberOfMonths = NumberOfRequiredDevelopersForProject**

Calculation of function point

- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity. Information domain values are defined in the following manner:
  - **Number of external inputs (EIs)**
  - **Number of external outputs (EOs)**
  - **Number of external inquiries (EQs)**
  - **Number of internal logical files (ILFs)**
  - **Number of external interface files (EIFs)**
- To compute function points (FP), the following relationship is used:
  - **FP = count total x [0.65 + 0.01 x ∑ (*Fi*)]** where count total is the sum of all FP entries

- **EI** - The number of external inputs. These are elementary processes in which derived data passes across the boundary from outside to inside. E.g. In an example library database system, enter an existing patron's library card number.

- **EO** - The number of external output. These are elementary processes in which derived data passes across the boundary from inside to outside. E.g. In an example library database system, display a list of books checked out to a patron.

- **EQ** - The number of external queries. These are elementary processes with both input and output components that result in data retrieval from one or more internal logical files and external interface files. E.g. In an example library database system, determine what books are currently checked out to a patron.

- **ILF** - The number of internal log files. These are user identifiable groups of logically related data that resides entirely within the applications boundary that are maintained through external inputs. E.g. In an example library database system, the file of books in the library.

- **ELF** - The number of external log files. These are user identifiable groups of logically related data that are used for reference purposes only, and which reside entirely outside the system. E.g. In an example library database system, the file that contains transactions in the library's billing system.

Calculation of function point

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total	→						

Calculation of function point

- The *Fi* (*i* = 1 to 14) are *value adjustment factors* (VAF) based on responses to the following questions :
- **1.** Does the system require reliable backup and recovery?
- **2.** Are specialized data communications required to transfer information to or from the application?
- **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 online data entry?
- **7.** Does the online data entry require the input transaction to be built over multiple
- screens or operations?
- **8.** Are the ILFs updated online?
- **9.** Are the inputs, outputs, files, or inquiries complex?
- **10.** Is the internal processing complex?
- **11.** Is the code designed to be reusable?
- **12.** Are conversion 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?

Calculation of function point

- FP's can be used to determine the Size of the Software, also can be used to quote the price of the software, get the time and effort required to complete the software.
- **Effort in Person Month = FP divided by no. of FP's per month** (Using your organizations or industry benchmark)
- **Schedule in Months = 3.0 * person-month^1/3**
- For e.g. for a 65 person month project
- Optimal Schedule = 3.0 * 65^1/3 ~ 12 months
- Optimal Team Size = 65 / 12 ~ 5 or 6 persons.

Function Point

- Advantages of FP
  - It is not restricted to code
  - Language independent
  - The necessary data is available early in a project. We need only a detailed specification.
  - More accurate than estimated LOC

- Drawbacks of FP
  - Subjective counting
  - Hard to automate and difficult to compute
  - Ignores quality of output

Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the filtering ability of QA activities as they are applied throughout all process framework activities
  - It indicates the percentage of software errors found before software release
- It is defined as DRE = E / (E + D)
  - E is the number of errors found before delivery of the software to the end user
  - D is the number of defects found after delivery
- As D increases, DRE decreases (i.e., becomes a smaller and smaller fraction)
- The ideal value of DRE is 1, which means no defects are found after delivery
- DRE encourages a software team to institute techniques for finding as many errors as possible before delivery

Example: LOC Approach

Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2DGA)	5,300
three-dimensional geometric analysis (3DGA)	6,800
database management (DBM)	3,350
computer graphics display facilities (CGDF)	4,950
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
estimated lines of code	33,200

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate =$8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **$431,000 and the estimated effort is 54 person-months.**

Example: FP Approach

Information Domain Value	opt.	likely	pess.	est. count	weight	FP-count
number of inputs	20	24	30	24	4	97
number of outputs	12	15	22	16	5	78
number of inquiries	16	22	28	22	5	88
number of files	4	4	5	4	10	42
number of external interfaces	2	2	3	2	7	15
count-total						321

The estimated number of FP is derived:

$$FP_{estimated} = \text{count-total} \times [0.65 + 0.01 \times \sum (F_i)]$$

$$FP_{estimated} = 375$$

organizational average productivity = 6.5 FP/pm.

burdened labor rate = $8000 per month, approximately $1230/FP.

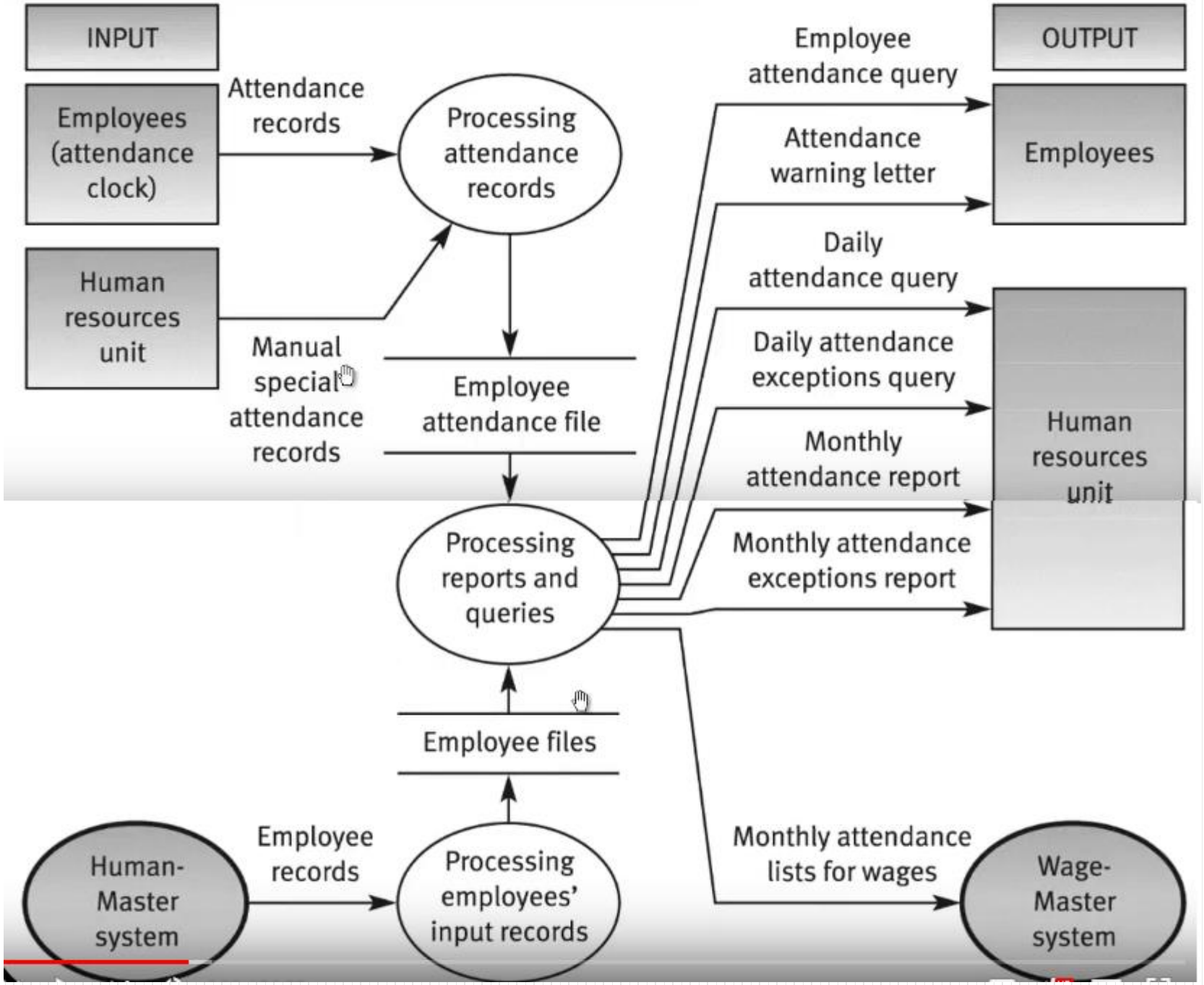
Based on the FP estimate and the historical productivity data, **total estimated project cost is $461,000 and estimated effort is 58 person-months.**

Exercise 1

- Consider the following system:
- User wants to maintain customer data and product data and needs to reference supplier data.
- User wants to add, change and delete customer data, wants to inquire on Customer and also requires four different reports on Customer with calculated data.
- User wants to add, change, delete Product data, wants to inquire on Product and also requires a report on Product with calculated data.
- User wants to inquire on Supplier using supplier number and also requires a report on Supplier with totaling results.
- All of these data are of average complexity and overall system is moderately complex i.e. assume sum of value adjustment factors is 46. Given the historical data that the organizational average productivity for systems of this type is 6.5 FP/pm. Also, labor rate is of $8000 per month, the cost per FP is approximately $1230. Based on the data provided, compute the following:

1. Compute FP for the system.
2. Total estimated project cost of the system.
3. Estimated effort in person months.

Exercise 2 Employee tracking system



Exercise 3

- A retail store has developed a new application, Frequent Buyer Program (FBF), to track customer purchases. The customer fills out a paper application and gives it to the store clerk. The clerk then adds the customer's information online. The clerk can also list customers, view a customer's detailed information and change a customer's info. A report is produced daily listed customers that were added with their addresses.
- Find the measures : EI, EO, EQ, ILF and EIF.

EXERCISE 4

Company XYZ plans to enhance its Accounts Payable (AP) application. The current application interfaces with existing banking, help, and purchase order (PO) applications. This is a menu-driven system. To enter the AP application, the user must make selections from a main menu. The menu has the following options:

- Invoices
  - Add an invoice
  - Display an invoice
  - Change an invoice
  - Delete an invoice
- Payments
  - Retrieve payments due
  - Record payments

Invoices and payments are maintained in the Invoice logical file in AP. The enhancement will allow users to maintain Vendor information in the AP application. The following is being added to the AP menu:

- Vendor
  - Add a vendor
  - Display vendor information
  - Change vendor information

The Vendor information will be maintained in a new Vendor logical file in the AP application.

Exercise 5

A new file is to be passed from the Accounts Payable (AP) application to the Banking application at the close of every business day. This file contains the payment date required, payment amount, PO number, vendor name, and vendor billing street address, city, state, and Zip Code. The Banking application must now be enhanced to process this incoming file and to generate the appropriate checks.

The Banking application will process the incoming file from the AP application without any edits or validation into two user-maintained logical files:

Checking Account and Disbursements.

The current process to generate checks to pay invoices is to be modified. Checks now will be generated with the PO number as a separate memo attribute by the banking system. Previously, checks contained the following information: preprinted name and address for the company, preprinted check numbers, payment date, payment amount, payee (same as vendor's name), and payee street address, city, state, and Zip Code. Checks previously did not include a memo attribute. These checks reference only the Checking Account logical file when they are created. The Checking Account logical file is updated internally to indicate payment as part of the check generation elementary process.

A printed report will be generated from the Checking Account logical file if checks were not produced because of an inadequate balance. The Report of Insufficient Funds will contain the following attributes: insufficient funds for payment date, payee, PO number, payment amount, total number of payees, and total payment amount (total attributes are calculated when the report is produced).

COCOMO 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 parameters that are derived from historical project data and current project characteristics.
- The Basic COCOMO model is a static, single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC).

Modes of COCOMO model

- The COCOMO models are defined for three classes of software projects. Using Boehm's terminology these are:
  - Organic projects - "small" teams with "good" experience working with "less than rigid" requirements.
  - Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements.
  - Embedded projects - developed within a set of "tight" constraints.

	Size	**Innovation**	**Deadline**	**Dev. Environment**
ORGANIC	Small	Little	Not Tight	Stable
SEMI-DITACHED	Medium	Medium	Medium	Medium
EMBEDDED	Large	Greater	Tight	Complex Hardware

Equation of COCOMO model

- The Basic COCOMO equations take the form:
  - $E = a_b (KLOC)^{b_b} PM$
  - $D = c_b (E)^{d_b}$ Months
- where E is the effort applied in person-months, D is the development time in chronological months and KLOC is the estimated number of delivered lines of code for the project (express in thousands). The coefficients a_b and c_b and the exponents b_b and d_b are given as follows:

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example

- **Problem:** Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

Solution

- As per the basic COCOMO estimation formula for organic software:

  Effort = $2.4 * (32)^{1.05}$ = 92 PM
  Nominal development time = $2.5 * (91)^{0.38}$ = 14 months
- Team Size = 92/14 = 6.5 = 7 people

  Cost required to develop the product = 7 * 15,000 * 14
  = Rs. 1,470,000/-

Object-Oriented Metrics

- Number 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 system
- Number of 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 software
  - It also indicates the potential amount of reuse to be applied during development
- Number 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)
  - This number indicates the amount of effort and potential reuse

Object-Oriented Metrics

- Average number of support classes per key class
  - Key classes are identified early in a project (e.g., 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
  - Non-GUI applications have between one and two times more support classes as key classes
- Number of subsystems
  - A subsystem is an aggregation of classes that support a function that is visible to the end user of a system

Thank You!!!

ANY QUESTIONS???