

Software Estimation and Scheduling

CO3: Student will be able to **Plan, schedule** and **track** the progress of the projects.

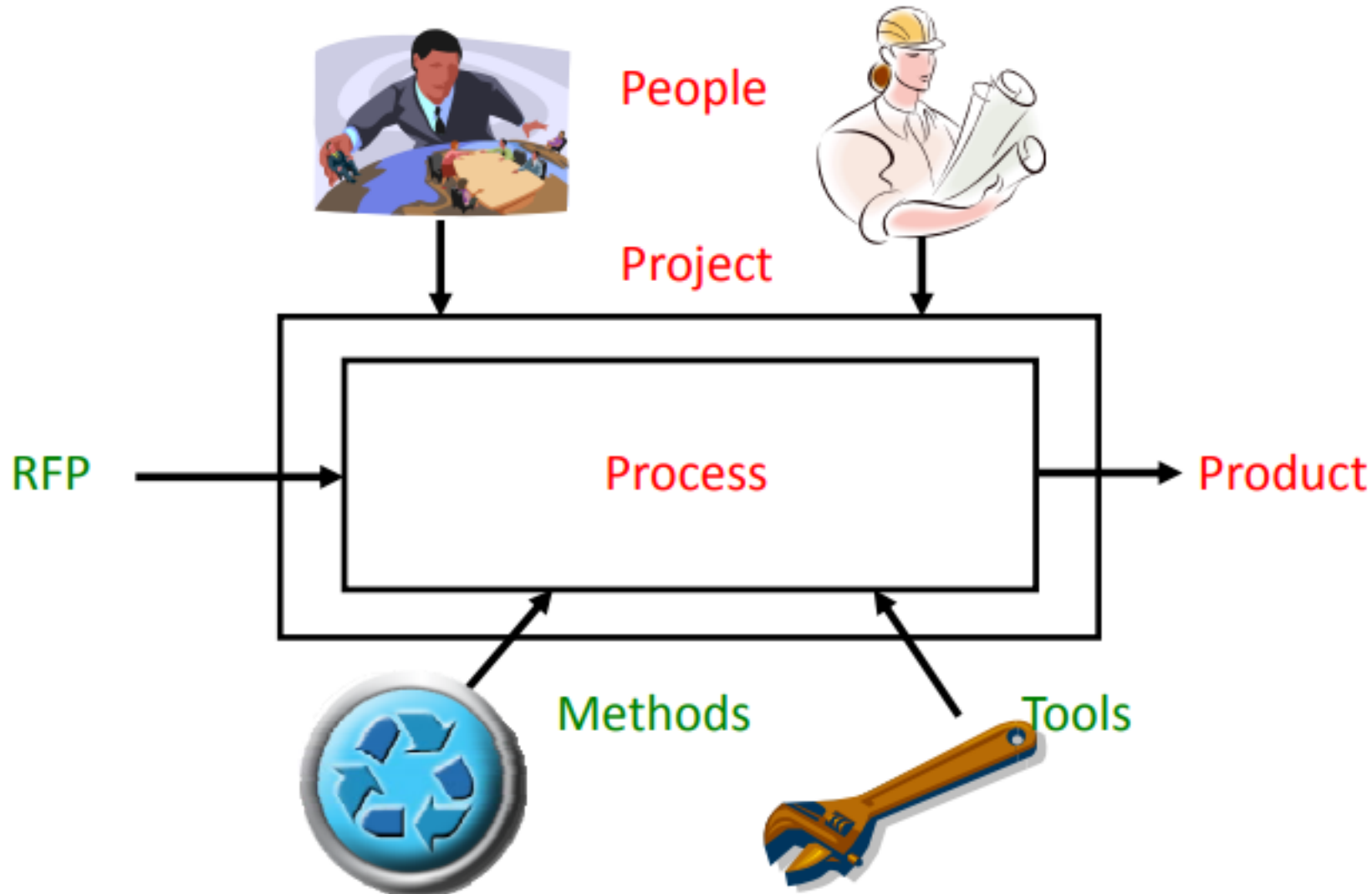
Contents

- Management Spectrum, 3Ps (people, product and process)
- Process and Project metrics
- Software Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model, Specialized Estimation Techniques, Object based estimation, use-case based estimation
- Project scheduling: Defining a Task Set for the Software Project, Timeline charts, Tracking the Schedule, Earned Value Analysis
- Self-learning Topics: Cost Estimation Tools and Techniques, Typical Problems with IT Cost Estimates.

The Management Spectrum

- Describes the management of a software project
- The management spectrum focuses on the four P's; people, product, process and project.
- The manager of the project has to control all these P's to have a smooth flow in the progress of the project and to reach the goal.
- The four P's of management spectrum are
 - People
 - Product
 - Process
 - Project

Process/Project/Product/People



The People:

- Includes from **manager to developer**, from **customer to end user**.
- Software Engineering Institute has developed a People Management Capability Maturity Model (PM-CMM),
“to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability”.
- Recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development.
- Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

The Product:

- Before a project can be planned, **product objectives and scope** should be established, **alternative solutions** should be considered, and **technical and management constraints** should be identified.

The Process:

- A software process provides the framework from which a comprehensive plan for software development can be established.
- A number of different tasks sets— **tasks, milestones, work products, and quality assurance points**—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.

The Project:

- The project is the complete software project that includes **requirement analysis, development, delivery, maintenance and updates.**
- The project manager of a project or sub-project is responsible for managing the people, product and process.
- A software project could be extremely complex and as per the industry data the failure rate is high. Its merely due to the development but mostly due to the steps before development and sometimes due to the lack of maintenance.

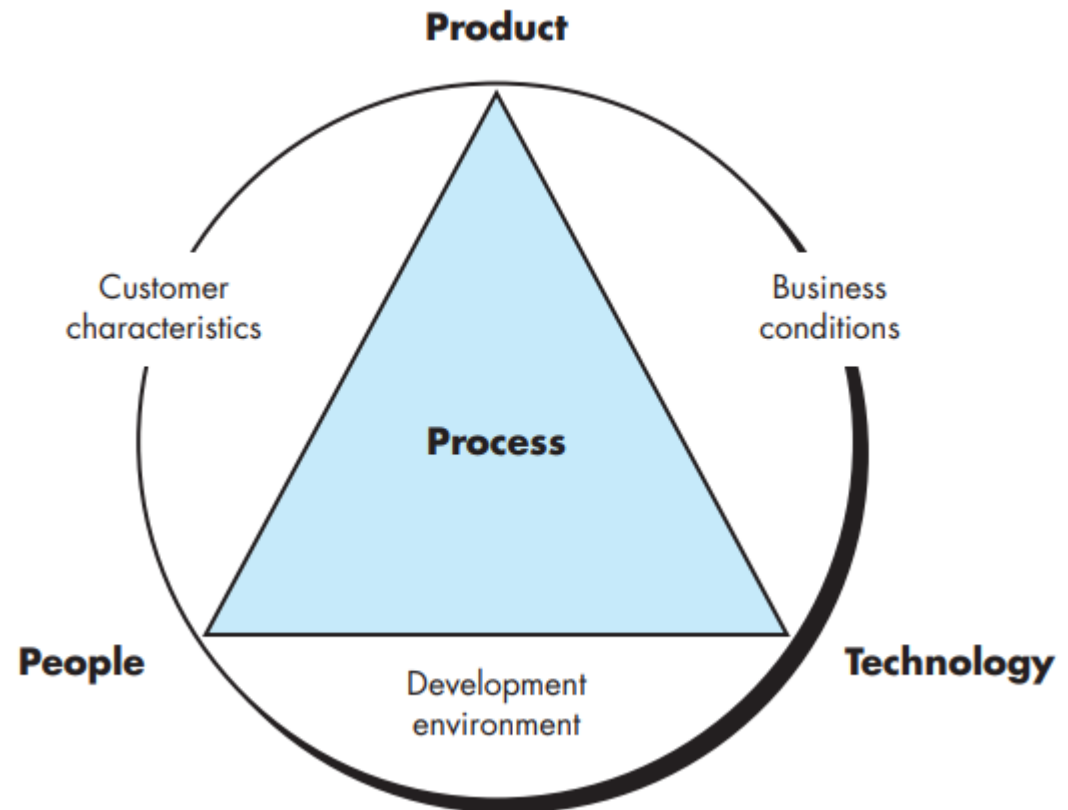
Process and Project metrics

Process metrics

- These are the metrics pertaining to the **Process Quality**. They measure efficiency and effectiveness of various processes.
- Process metrics are collected across all projects and over long periods of time.
- Their intent is to provide a set of process indicators that lead to long-term software process improvement.

Determinants for software quality and organizational effectiveness

- The skill and motivation of people
- The complexity of the product
- The technology (i.e., the software engineering methods and tools)



Project metrics

- These are the metrics pertaining to the **Project Quality**. They measure **defects, cost, schedule, productivity and estimation of various project resources and deliverables**.
- Project metrics enable a software project manager to
 - (1) assess the status of an ongoing project,
 - (2) track potential risks,
 - (3) uncover problem areas before they go “critical,”
 - (4) adjust work flow or tasks, and
 - (5) evaluate the project team’s ability to control quality of software work products.

Software Project Estimation

Software Project Estimation

- Before the project can begin, the software team should estimate the **work to be done**, the **resources** that will be required, and the **time** that will elapse from **start to finish**.
- Once these activities are accomplished, the software team should establish a **project schedule** that defines software engineering tasks and milestones, identifies who is responsible for conducting each task, and specifies the inter-task dependencies that may have a strong bearing on progress.
- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation.

Important Factors

- Project complexity
- Project size
- Degree of structural uncertainty -refers to the degree to which requirements have been solidified, the ease with which functions can be compartmentalized

The Project Planning Process

TASK SET



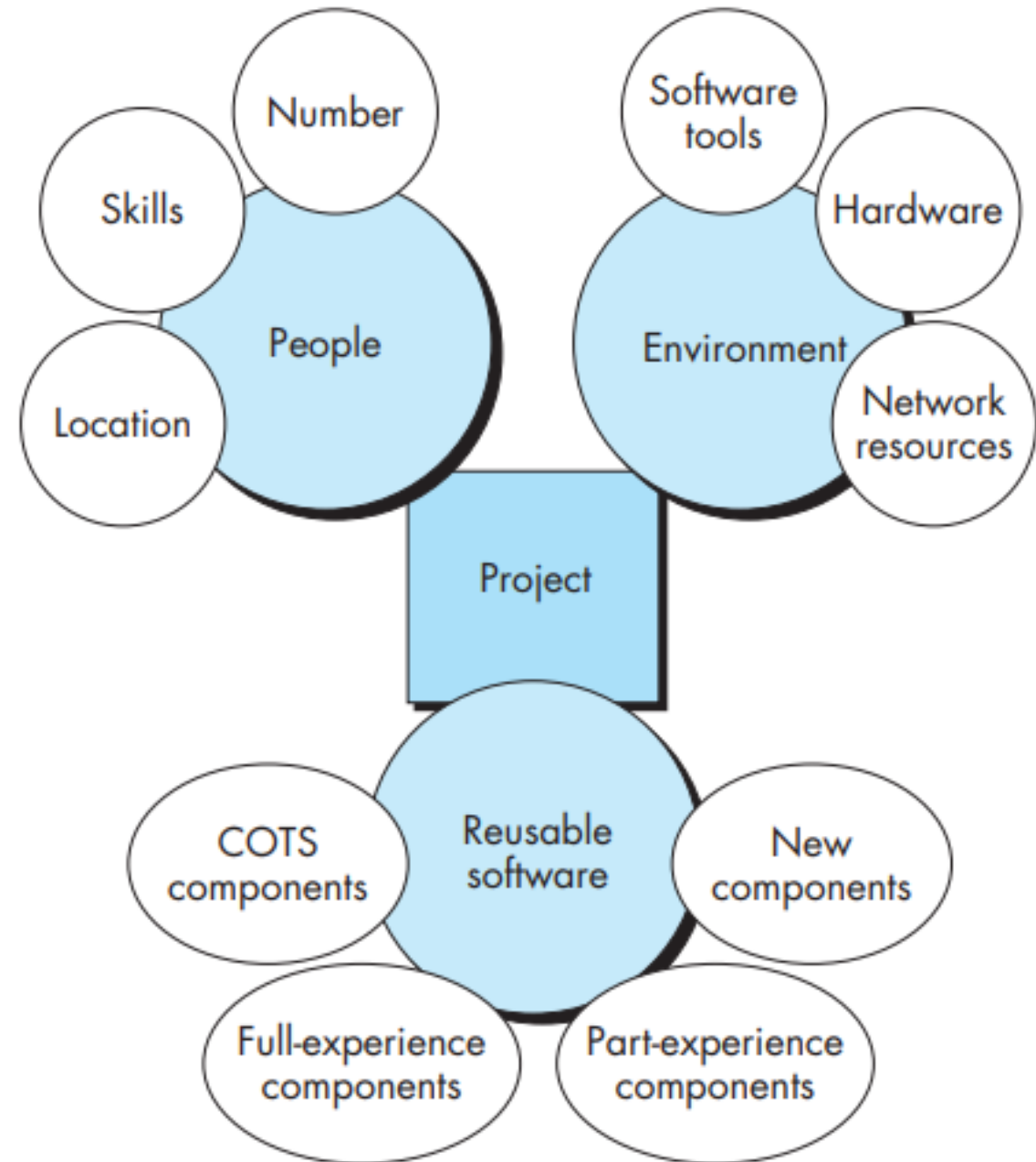
Task Set for Project Planning

1. Establish project scope.
2. Determine feasibility.
3. Analyze risks (Chapter 35).
4. Define required resources.
 - a. Determine required human resources.
 - b. Define reusable software resources.
 - c. Identify environmental resources.
5. Estimate cost and effort.
 - a. Decompose the problem.
 - b. Develop two or more estimates using size, function points, process tasks, or use cases.
 - c. Reconcile the estimates.
6. Develop a project schedule (Chapter 34).
 - a. Establish a meaningful task set.
 - b. Define a task network.
 - c. Use scheduling tools to develop a time-line chart.
 - d. Define schedule tracking mechanisms.

Software Scope and Feasibility

- Software **scope** describes the **functions and features that are to be delivered to end users**; the **data that are input and output**; the “**content**” that is **presented to users** as a consequence of using the software; and the **performance, constraints, interfaces, and reliability** that bound the system.
- Scope is defined using one of two techniques:
 1. A narrative description of software scope is developed after communication with all stakeholders.
 2. A set of use cases is developed by end users.

Project Resources



Reusable Software Resources

- **Off-the-shelf components**-existing software that can be acquired from a third party or from a past project,
- **Full-experience components**-existing specifications, designs, code, or test data developed for past projects that are similar to the software to be built for the current project,
- **Partial-experience components**-existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification,
- **New components**-components built by the software team specifically for the needs of the current project.

Metrics

- Numerical measures that quantify the degree to which software, a process or a project possesses a given attribute
- Metrics help the followings
 - Determining software quality level
 - Estimating project schedules
 - Tracking schedule process
 - Determining software size and complexity
 - Determining project cost
 - Process improvement

Software Metrics

The five essential, essential, fundamental metrics:

- Size (LOC, etc.)
- Cost (in dollars)
- Duration (in months)
- Effort (in person-month)
- Quality (number of faults detected)

Product Size Metrics

- Conventional metrics
 - Size-oriented metrics
 - Function-oriented metrics
 - Empirical estimation models
- Object-Oriented metrics
 - Number of scenario scripts
 - Number of key classes
 - Number of support classes
 - Average number of support classes per key classes
- User-Case oriented metrics

Product Size Metrics (cont'd)

- Web engineering product metrics
 - Number of static web pages
 - Number of dynamic web pages
 - Number of internal page links
 - Number of persistent page links

Size Estimation

The methods to achieve reliable size and cost estimates:

- LOC-based estimation
- FP-based estimation
- Empirical estimation models
 - COCOMO

LOC-based Estimation

- The SLOC technique is an objective method of estimating or calculating the size of the project.
- The project size helps determine the resources, effort, cost, and duration required to complete the project.
- It is also used to directly calculate the effort to be spent on a project.
- We can use it when the programming language and the technology to be used are predefined.
- This technique includes the calculation of lines of codes (LOC), documentation of pages, inputs, outputs, and components of a software program.

LOC-based Estimation

The problems of lines of code (LOC)

- Different languages lead to different lengths of code
- It is not clear how to count lines of code
- A report, screen, or GUI generator can generate thousands of lines of code in minutes
- Depending on the application , the complexity of code is different

LOC-based Estimation - Example

• Function	• Estimated LOC
– User interface	2,300
– 2-D geometric analysis	5,300
– 3-D geometric analysis	6,800
– Database management	3,500
– Graphic display facilities	4,950
– I/O control function	2,100
– Analysis function	8,400
• Total estimated LOC	33,350

Example:

- Assume estimated lines of code of a system is: 33,200 LOC
 - Average productivity for system of this type is: 620 LOC/person-month
 - There are 6 developers
 - Labor rate is: \$ 800 per person-month
- Calculate the total effort and cost required to complete the above project.

Solution

+Way1

⇒ Total Effort = Total LOC/Productivity = $33200/620=53.54 \approx \underline{54 \text{ person-months}}$

⇒ 6 developers → Effort = Total Effort/6 = $54/6 = \underline{9 \text{ months}}$

⇒ Total Cost = Total Effort * Labor Rate = $54 * 800 \approx \underline{\$43,200}$

+Way2

⇒ Cost per LOC = Labor Rate /Productivity= $800/620=\$1.29 \approx \underline{\$1.3}$

⇒ Total Cost = Total LOC * Cost per LOC = $33,200* 1.3=\$43160 \approx \underline{\$43,200}$

⇒ Total Effort = Total Cost / Labor Rate = $43,200/800 = \underline{54 \text{ person-months}}$

FP-based Estimation

- Based on FP metric for the size of a product
- Based on inputs, outputs, data files, inquiries, and external interfaces
- Step 1: Classify each component of the product as simple, average , or complex
 - Assign the appropriate number of function points
 - The sum of function pointers for each component gives UFP (unadjusted function points)

The steps in function point analysis are:

1. Count the number of functions of each proposed type.
2. Compute the Unadjusted Function Points(UFP).
3. Find Total Degree of Influence(TDI).
4. Compute Value Adjustment Factor(VAF).
5. Find the Function Point Count(FPC).

1. Count the number of functions of each proposed type

Find the number of functions belonging to the following types:

- **External Inputs:** Functions related to data entering the system.
- **External outputs:** Functions related to data exiting the system.
- **External Inquiries:** They leads to data retrieval from system but don't change the system.
- **Internal Files:** Logical files maintained within the system. Log files are not included here.
- **External interface Files:** These are logical files for other applications which are used by our system.

2. Compute the Unadjusted Function Points(UFP)

- Categorize each of the five function types as simple, average or complex based on their complexity.
- Multiply count of each function type with its weighting factor and find the weighted sum.
- The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

each of the parameters and select the one of the project complexity levels

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

3. Find Total Degree of Influence

- Use the '14 general characteristics' of a system to find the degree of influence of each of them.
- The sum of all 14 degrees of influences will give the TDI. The range of TDI is 0 to 70.
- The 14 general characteristics are: **Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.**
- Each of above characteristics is evaluated on a scale of 0-5.

0 = No influence,
 1 = Incidental,
 2 = Moderate,
 3 = Average,
 4 = Significant,
 5 = Critical

Question	0	1	2	3	4	5
1. Does the system require reliable backup and recovery?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. Are data communications required?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. Are there distributed processing functions?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. Is performance critical?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. Will the system run in an existing, heavily utilized operational environment?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. Does the system require on-line data entry?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. Are there master files updated on-line?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. Are the inputs, outputs, files, or inquiries complex?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. Is the internal processing complex?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
11. Is the code designed to be reusable?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
12. Are conversion and installation included in the design?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
13. Is the system designed for multiple installations in different organizations?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
14. Is the application designed to facilitate change and ease of use by the user?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Total Weighting Factor(ΣFi)						

The general characteristics and brief description of the 14 complexity adjustment values above are shown below:

Nº	Characteristics	Description
1	Operational Ease	The degree to which the application attends to operational aspects, such as backup, start-up, and recovery processes
2	Data Communication	The degree to which an application communicates with the other applications
3	Distributed Functions	The degree to which an application transfers or shares data among the component of applications
4	Performance	The degree of the response time and throughput performance of an application
5	Heavily Used onfiguration	The degree to which the computer resources, where the application runs, are used
6	On-line Data Entry	The percentage of data that is entered by using interactive transaction
7	Transaction Rate	The frequency of transactions that are executed, on a daily, weekly, or monthly basis
8	On-line Update	The degree to which the number of internal logical files is updated on-line
9	End-user Efficiency	The degree to which human factors and user friendliness are to be considered
10	Complex Processing	The degree to which the complexity of logic influences the processing logic, in turn, influences the development of the application
11	Reusability	The degree to which the application and code in the application are specifically designed, developed, and supported to be reused in other applications
12	Installation Ease	The degree to which conversion from a previous environment influences the development of the application
13	Multiple Sites	The degree to which the application is developed for multiple locations and User organizations
14	Facilitates Change	The degree to which the application is developed for easy modification of processing logic or data structure

4. Compute Value Adjustment Factor(VAF)

- Use the following formula to calculate VAF
$$\text{VAF} = (\text{TDI} * 0.01) + 0.65$$

5. Find the Function Point Count

- Use the following formula to calculate FPC
$$\text{FPC} = \text{UFP} * \text{VAF}$$

Estimation of LOC from FP

- Basing on the size of the project in FP, you can estimate or calculate the numbers of LOC by multiplying the average number of LOC/ FP for a given language (AVC) by the total number of function points of the project.
- To calculate the numbers of line of code, the following formula is used

$$\text{LOC} = \text{LOC per FP} * \text{FP}$$

OR

$$\text{LOC} = \text{AVC} * \text{FP}$$

where:

- AVC: is the average number of LOC/FP for a given language
 - LOC: is the numbers of line of code
 - FP : is the Total numbers of Function Point
-

-To define AVC, the following table is used to choose the type of programming languages that will be used when developing a sw project:

Programming Language	LOC/FP (average)	Select
Assembly Language	320	<input type="checkbox"/>
C	128	<input type="checkbox"/>
COBOL/Fortran	105	<input type="checkbox"/>
Pascal	90	<input type="checkbox"/>
Ada	70	<input type="checkbox"/>
C++	64	<input type="checkbox"/>
Visual Basic	32	<input type="checkbox"/>
Object-Oriented Languages	30	<input type="checkbox"/>
Smalltalk	22	<input type="checkbox"/>
Code Generators (PowerBuilder)	15	<input type="checkbox"/>
SQL/Oracle	12	<input type="checkbox"/>
Spreadsheets	6	<input type="checkbox"/>
Graphical Languages (icons)	4	<input type="checkbox"/>




Note: You may use the similar one from the table for other languages.

Function Point Example

Information		Weighting Factor				
<u>Domain Value</u>	<u>Count</u>		<u>Simple</u>	<u>Average</u>	<u>Complex</u>	
External Inputs	3	x	3	4	6	= 9
External Outputs	2	x	4	5	7	= 8
External Inquiries	2	x	3	4	6	= 6
Internal Logical Files	1	x	7	10	15	= 7
External Interface Files	4	x	5	7	10	= 20
Count total						50

- $FP = \text{count total} * [0.65 + 0.01 * \text{sum}(F_i)]$
- $FP = 50 * [0.65 + (0.01 * 46)]$
- $FP = 55.5$ (rounded up to 56)

Example 1: Assuming we have collected data for a sw project as shown in the two tables below. The development team uses SQL Server for developing this sw project. **Calculate FP and LOC.**

Measurement Parameter	Count		Simple 	Average 	Complex 		Total
Number of user inputs	13	x	3	4	6	=	52
Number of user outputs	10	x	4	5	7	=	50
Number of user inquiries	3	x	3	4	6	=	12
Number of files	4	x	7	10	15	=	40
Number of external interfaces	2	x	5	7	10	=	14
Count Total							168

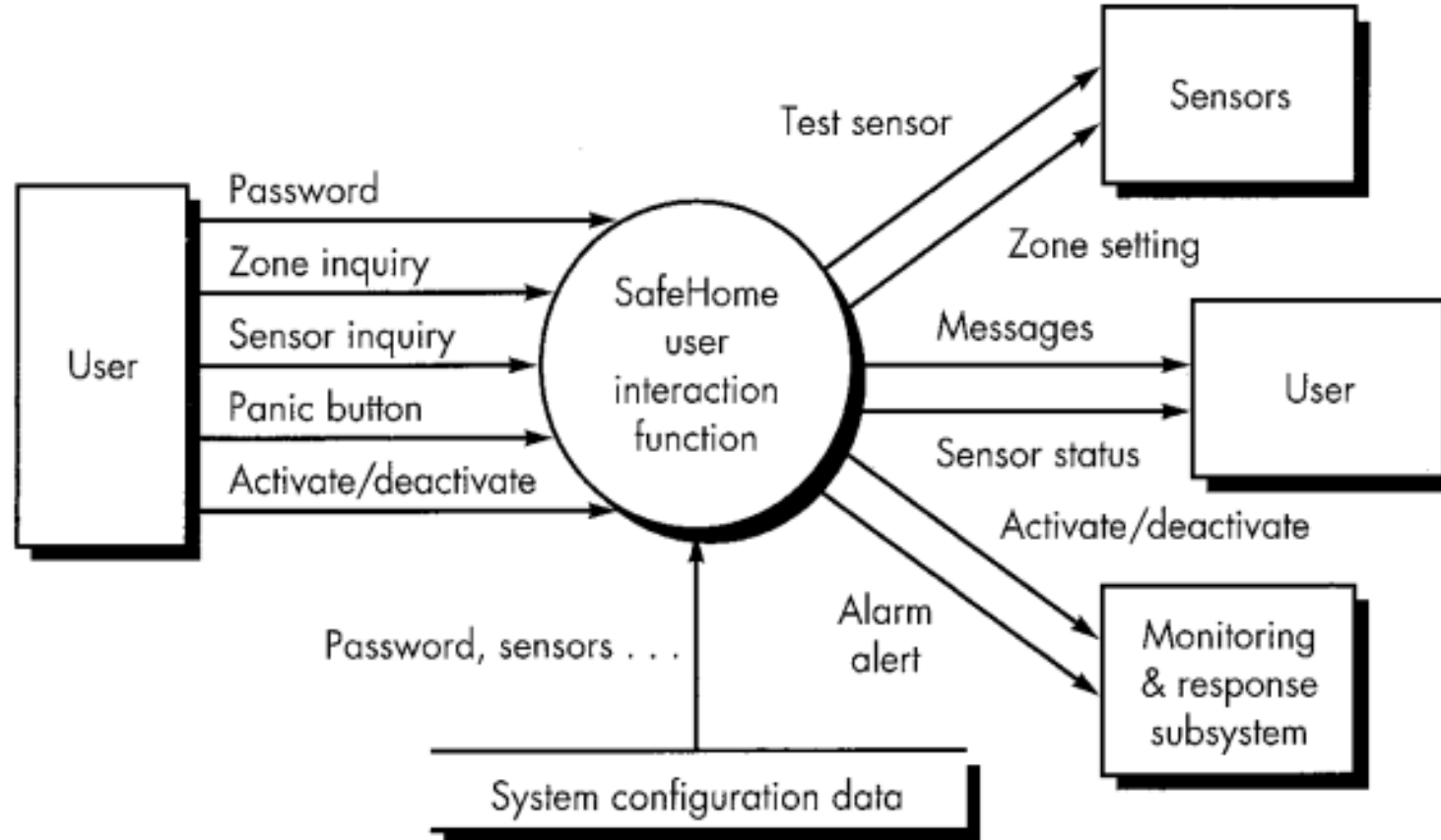
Example 1: cont'd.....

Nº	<i>General System Characteristics</i>	<i>Degree of Influence (Value)</i>
1	Operational Ease	2
2	Data Communication	5
3	Distributed Functions	4
4	Performance	5
5	Heavily Used Configuration	2
6	Transaction Rate	3
7	On-line Data Entry	4
8	On-line Update	2
9	End-user Efficiency	3
10	Complex Processing	4
11	Reusability	5
12	Installation Ease	2
13	Multiple Sites	3
14	Facilitates Change	4
Total complexity adjustment or Total Weighting Factor value		48

➔ $FP = 168 * (0.65 + 0.01 * 48) = 189.84 \approx \underline{190 \text{ FPs}}$

➔ $LOC = FP * 12 = 190 * 12 = \underline{2280 \text{ LOCs}}$

Example2: Part of analysis model for SafeHome software



- Number of **user inputs** = 3 (password, panic button, and activate/deactivate)
- Number of **user outputs** = 2 (messages and sensor status)
- Number of **user inquiries** = 2 (zone inquiry and sensor inquiry)
- Number of **file** = 1 (system configuration file)
- Number of **external interfaces** = 4 (test sensor, zone setting, activate/deactivate, and alarm alert)

Example 2: cont'd.....

Measurement parameter	Count		Weighting Factor			
			Simple	Average	Complex	
Number of user inputs	3	×	3	4	6	= 9
Number of user outputs	2	×	4	5	7	= 8
Number of user inquiries	2	×	3	4	6	= 6
Number of files	1	×	7	10	15	= 7
Number of external interfaces	4	×	5	7	10	= 20
Count total						50

+Assume Weighting Factor is simple:

$$\rightarrow CT = (3*3)+(2*4)+(2*3)+(1*7)+(4*5) = \underline{50}$$

+And assume $\sum Fi = 46$

$$\rightarrow FP = 50 \times [0.65 + (0.01 \times 46)] = \underline{55.50 \approx 56FP}$$

Using FP for Initial Estimation.....

+ Assuming that:

- To complete one FP of work, the project requires 10 hours (Productivity = 10hours/FP)
- The total FP estimated is 200 FP for the project
- The project team works 8 person-hours per working day
- There are 20 working days in a month

+ Calculate the effort required to complete the sw project

Solution

To complete a project of 200 FP, you require:

+Effort in person-hours:

$$\text{Effort} = 200 \text{ FP} * 10\text{hrs/FP} = \mathbf{2000 \text{ person-hours}}$$

+Effort in person-days:

$$\text{Effort} = 2000 \text{ hours} / 8 \text{ hours} = \mathbf{250 \text{ person-days}}$$

+Effort in person-months:

$$\text{Effort} = 250 \text{ person-days} / 20 \text{ days} = \mathbf{12.5 \text{ person-months}}$$

+Note:

- If there are two developers on the project, then you would require $= 12.5 \text{ months} / 2 \text{ developers} = \mathbf{6.25 \text{ months}}$ to complete the work.

FP-Based Estimation.....

- Assume FP of a system is: 375
 - Average productivity for system of this type is: 6.5 FP/pm
 - Labor rate is: \$ 800 per month
-
- Cost per FP is: $800/6.5 = \$123$
 - Total Estimated Cost is: $\$123 * 375 = \$46,100$
 - Total Estimated Effort is: $46,100/800 = 57.62 = 58$ person-months

Using FP for Post-Project analysis...

- FP can be used to express factor such as **productivity**, **effort**, **defects**, and **cost** used in an already completed project.
- Expressing these factors in FP helps analyze a project effectively.
- Analysis of a project enables to apply the learning derived from a particular project to future projects

Example:

	Total number of defects reported	Total project size in FP
Project 1	10	150
Defect density	$10/150 = \underline{0.067}$ defects /FP	"14"
Project 2	20	200
Defect density	$20/200 = \underline{0.10}$ defects/FP	
Project 3	40	1000
Defect density	$40/1000 = \underline{0.04}$ defects /FP	,

Defect density = Total number of defects/total project size in FP

→ Project 3 is of superior quality because it reported least number of defects . per FP of development effort

Interpretation of the FP Number

- Assume that past project data for a software development group indicates that
 - One FP translates into 60 lines of object-oriented source code
 - 12 FPs are produced for each person-month of effort
 - An average of three errors per function point are found during analysis and design reviews
 - An average of four errors per function point are found during unit and integration testing
- This data can help project managers revise their earlier estimates
- This data can also help software engineers estimate the overall implementation size of their code and assess the completeness of their review and testing activities

COCOMO

- COnstructive COst Model
- Empirical model
 - Metrics such as LOC and FP are used as input to a model for determining product cost and duration
- Well documented, and supported by public domain and commercial tools;
- Widely used and evaluated
- Has a long pedigree from its first instantiation in 1981
 - COCOMO I (81)
 - COCOMO II

COCOMO types

There are three forms of the COCOMO

- Basic COCOMO (macro estimation) which gives an initial rough estimate of man months and development time
- Intermediate COCOMO which gives a more detailed estimate for small to medium sized projects
- Detailed COCOMO (micro estimation) which gives a more detailed estimate for large project

Basic COCOMO

- The basic COCOMO technique is used to estimate the effort and cost of a SW project by using only the lines of code.
- We use basic COCOMO when we need a rough estimate of effort, such as during maintaining a project.
- There are three steps involved in estimating the effort using basic COCOMO technique:
 - S1- Estimating the total delivered lines of code.
 - S2- Determine the effort constants based on the type of the project
 - S3- Substituting values for lines of code and effort constants in a formula

There are three types of projects to be calculated effort.

- **Organic Mode:** Relatively small, simple software projects in which a small team with good application experience work to a set of less than rigid requirement.
- The organic projects must have sufficient and defined objectives.
- These are simple businesses and applications, such as a banking system and inventory system

- **Embedded Mode:** A software project that must be developed within a set of tight hardware, software and operational constraints.
- The embedded projects must have stringent and specialized HW, SW, and human resources requirements.
- Organizations usually have less experience in developing such projects.
- Examples of such projects includes real-time operating systems, industrial automation systems, and sophisticated space and aviation systems

- **Semi-Detached Mode:** An intermediate (in size and complexity) software project in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements.
- Semidetached projects are combination of the preceding two types of SW projects.
- Examples: a new operating system, a database management system

Basic COCOMO

- Select the software project type and substitute the values of C and K from the following table into the formulas below the table:

Software Project Type	C	K	Select
Organic	3.2	1.05	<input type="checkbox"/>
Embedded	2.8	1.20	<input type="checkbox"/>
Semi-detached	3.0	1.12	<input type="checkbox"/>

$$E_i = C * (KLOC)^K$$

-Where E_i is the effort for a project, C and K are the effort or COCOMO constants depending on the type of project being developed.

Basic COCOMO Example:

- Estimate the effort of an application (organic type) with 4 KLOC
- Ans:
- $E_i = 3.2 * 4^{1.05} = 3.2 * 4.28 \approx 14$ person-months