SOFTWARE ENGINEERING & PROJECT MANAGEMENT

K. K. WAGH EDUCATION SOCIETY

K K Wagh

UNIT IV

Project Metrics and Estimation

Contents

Project Metrics

Software measurement

- Measurement can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control.
- Measurement can be used by software engineers to help assess the quality of work products and to assist in tactical decision making as a project proceeds.
- *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 measurement

1. Size-Oriented Metrics
2. Function-Oriented Metrics
3. Object-Oriented Metrics
4. Use-Case–Oriented Metrics
5. WebApp Project Metrics

1. Size-Oriented Metrics

A set of simple size-oriented metrics can be developed for each project:

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per KLOC
- Pages of documentation per KLOC

In addition, other interesting metrics can be computed:

- Errors per person-month
- KLOC per person-month
- $ per page of documentation

2. Function-Oriented Metrics

- Uses a measure of the functionality delivered by the application as a normalization value.
- The most widely used function-oriented metric is the function point (FP).
- Computation of the function point is based on characteristics of the software's information domain and complexity.

3. Object-Oriented Metrics

A set of metrics for OO projects: -

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Average number of support classes per key class
- Number of subsystems

4. Use-Case–Oriented Metrics

- Use cases describe user-visible functions and features that are basic requirements for a system.
- The use case is independent of programming language.
- In addition, the number of use cases is directly proportional to the size of the application in LOC and to the number of test cases that will have to be designed to fully exercise the application.
- As use cases can be created at vastly different levels of abstraction, there is no standard "size" for a use case.
- Researchers have suggested *use-case points* (UCPs) as a mechanism for estimating project effort.
- The UCP is a function of the number of actors and transactions implied by the use-case models and is analogous to the FP in some ways.

5. WebApp Project Metrics

- Number of static Web pages
- Number of dynamic Web pages
- Number of internal page links
- Number of persistent data objects
- Number of external systems interfaced
- Number of executable functions

Metrics for Software Quality

- Although many quality measures can be collected, the primary thrust at the project level is to measure errors and defects.
- Metrics derived from these measures provide an indication of the effectiveness of individual and group software quality assurance and control activities.
- Metrics such as work product errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric.
- Error data can also be used to compute the *defect removal efficiency* (DRE) for each process framework activity.

Measuring Quality

Although there are many measures of software quality, correctness, maintainability, integrity, and usability provide useful indicators for the project team.

- **Correctness:** Correctness is the degree to which the software performs its required function.
- **Maintainability:** Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.
- **Integrity:** Integrity measures a system's ability to withstand attacks to its security.
- **Usability:** Usability is an attempt to quantify ease of use

Defect Removal Efficiency (DRE)

- A quality metric that provides benefit at the project level is *defect removal efficiency* (DRE).
- When considered for a project as a whole, DRE is defined in the following manner:

$$\text{DRE} = \frac{E}{E + D}$$

- 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. That is, no defects are found in the software.

Metrics for Small Organizations

A small 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 change order to personnel, t_{eval}.
- Effort (person-hours) required to make the change, W_{change}.
- Time required (hours or days) to make the change, t_{change}.
- Errors uncovered during work to make change, E_{change}.
- Defects uncovered after change is released to the customer base, D_{change}.

Metrics for Small Organizations

- Once these measures have been collected for a number of change requests, it is possible to compute the total elapsed time from change request to implementation of the change and the percentage of elapsed time absorbed by initial queuing, evaluation and change assignment, and change implementation.
- The defect removal efficiency can be computed as –

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

- DRE can be compared to elapsed time and total effort to determine the impact of quality assurance activities on the time and effort required to make a change.

Estimation for Software Projects

Observation on Estimation

- Before the actual project work begins, 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

- A typical software project estimation consists of:
  - cost estimation
  - resources estimation
  - effort estimation

- The main reason for estimating is to avoid situation when the project exceeds the maximum budget or the expenses are higher than the profits expected from the end product.

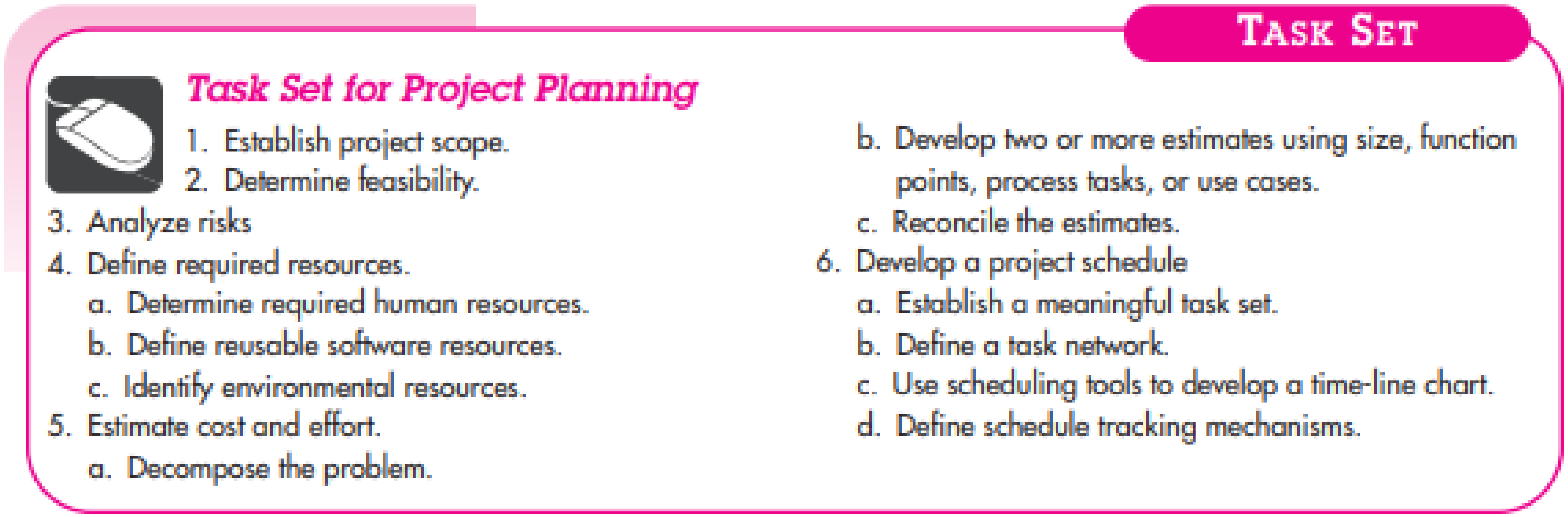
Steps for estimation

- Estimate the size of the development product
- Estimate the effort in person-months or person-hours
- Estimate the schedule in calendar month
- Estimate the project cost in agreed currency

The Project Planning Process

- Software project management begins with a set of activities that are collectively called project planning.
- The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost and schedule.
- The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking and monitoring a complex technical project.

The Project Planning Process



TASK SET

Task Set for Project Planning

1. Establish project scope.
2. Determine feasibility.
3. Analyze risks
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
   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

- 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 the user as a consequence of using the software; and
  - The performance, constraints, interfaces and reliability that bound the system
- Scope is defined using one of the two techniques
  - A narrative description of software scope is developed after communication with all stake-holders
  - A set of use-cases is developed by end–user

Feasibility

- Can we build software to meet this scope? Is the project feasible?"
- Software feasibility has four solid dimensions:
  1) Technology

     Is a project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  2) Finance

     Is it financially feasible? Can development be completed at a cost the software organization, its client, or the market can afford?
  3) Time

     Will the project's time-to-market beat the competition?
  4) Resources

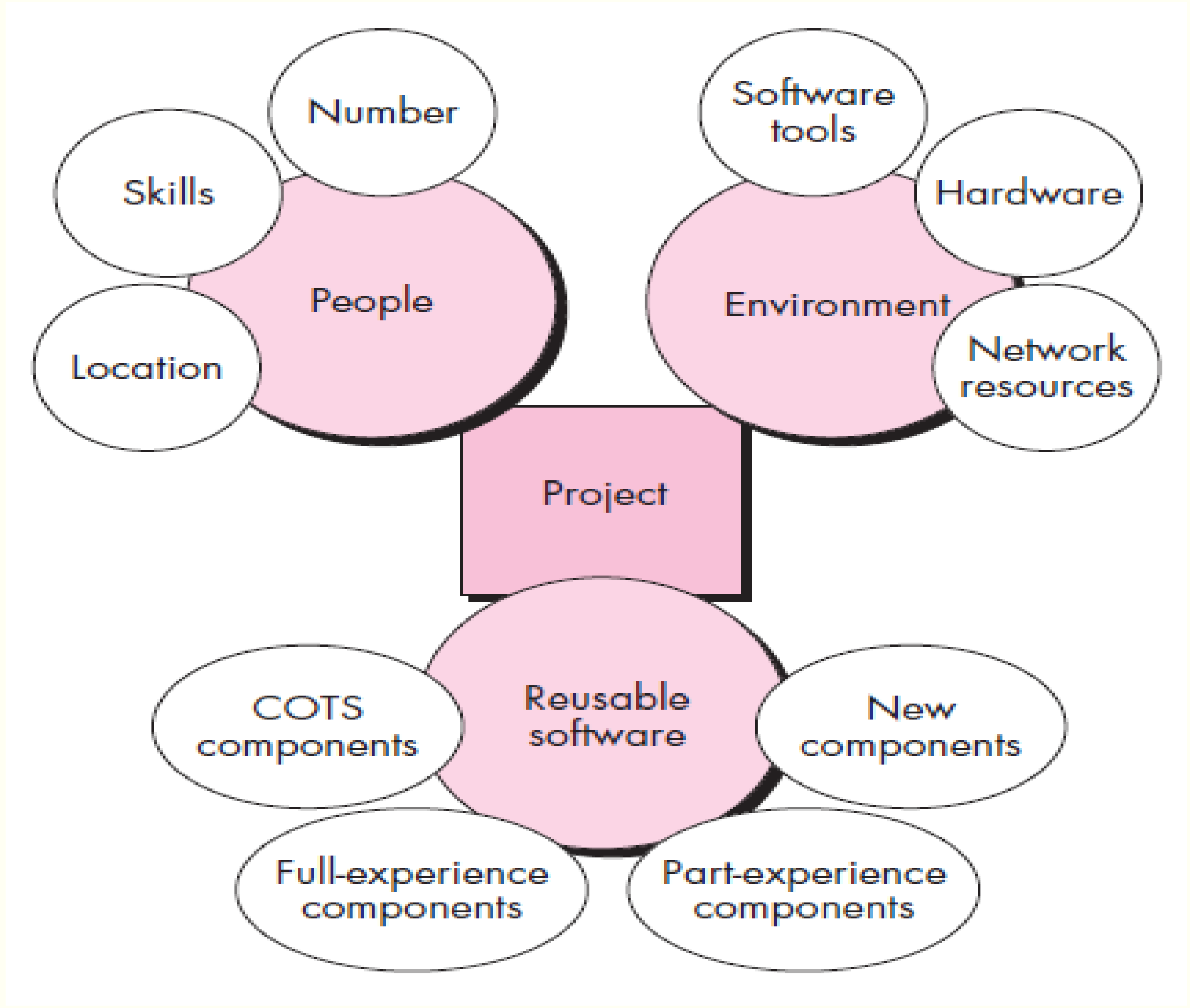
     Does the organization have the resources needed to succeed?

Resources

- The second planning task is estimation of the resources required to accomplish the software development effort.

- Each resource is specified with four characteristics:
    1. description of the resource
    2. a statement of availability
    3. time when the resource will be required
    4. duration of time that the resource will be applied

Project Resources



Software project estimation

- Software cost and effort estimation will never be an exact;
- Many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it
- To achieve reliable cost and effort estimates, a number of options are-
  1. Delay estimation until late in the project (obviously, we can achieve 100 percent accurate estimates after the project is complete!).
  2. Base estimates on similar projects that have already been completed.
  3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
  4. Use one or more empirical models for software cost and effort estimation.

Decomposition Techniques

- Decomposition techniques take a divide and conquer approach.
- Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.
- Decompose the problem, from two different points of view:
  - decomposition of the **problem** and decomposition of the **process**
  - Estimation uses one or both forms of partitioning
  - understand the scope of the software to be built
  - generate an estimate of its "size."

Decomposition Techniques

- Software Sizing
- Problem-Based Estimation
  - LOC-Based Estimation
  - FP-Based Estimation
- Process-Based Estimation
- Estimation with Use Cases
- Reconciling Estimates

Software Sizing

- The accuracy of a software project estimate is predicated on a number of things:

  (1) the degree to which you have properly estimated the size of the product to be built;

  (2) the ability to translate the size estimate into human effort, calendar time, and dollars

  (3) the degree to which the project plan reflects the abilities of the software team; and

  (4) the stability of product requirements and the environment that supports the software engineering effort.

- In the context of project planning, size refers to a quantifiable outcome of the software project.
  - If a direct approach is taken, size can be measured in lines of code (LOC).
  - If an indirect approach is chosen, size is represented as function points (FP).

Problem-Based Estimation

- The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning.
- When **LOC** is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.
- For **FP estimates**, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external are estimated.

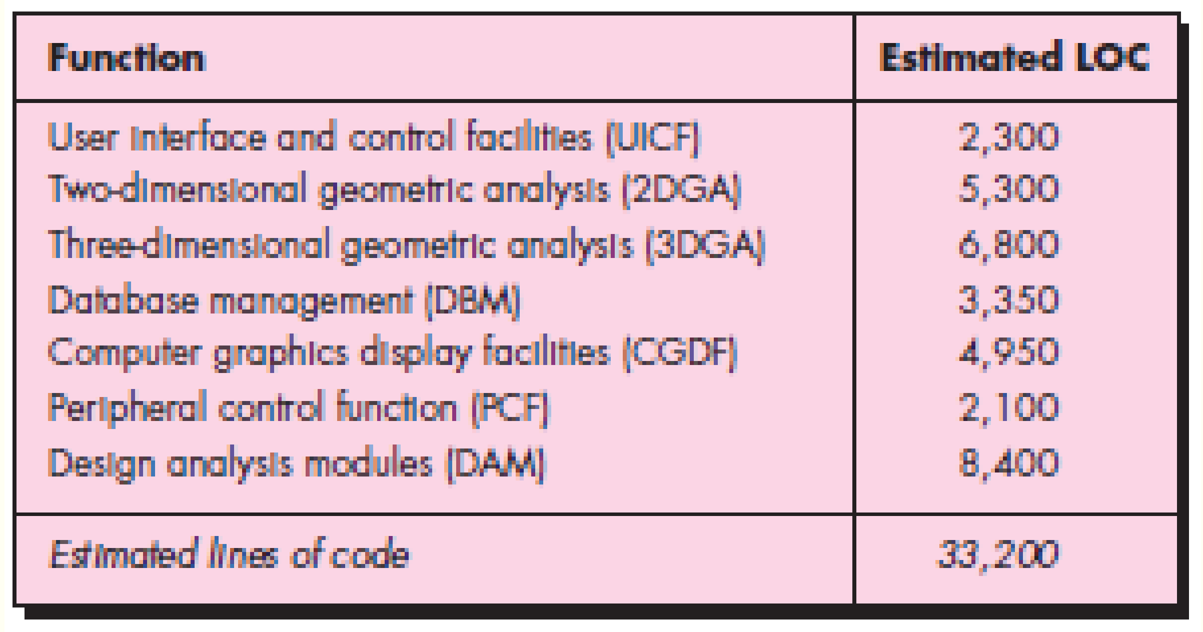
LOC-Based Estimation

- A preliminary statement of software scope can be developed:

  The mechanical CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

An Example of LOC-Based Estimation

1. A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm.
2. Based on a burdened labor rate of $8000 per month, the cost per line of code is approximately $13.
3. 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.

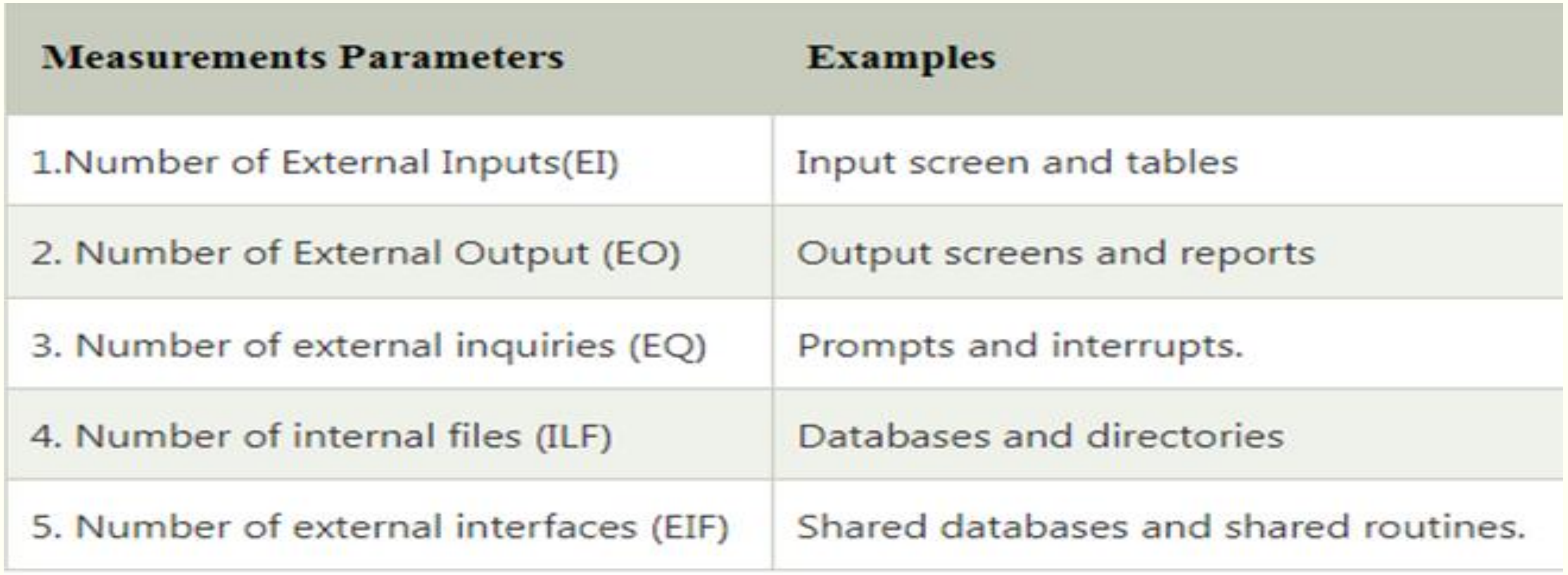


Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Figure: Estimation table for the LOC methods

FP-Based Estimation

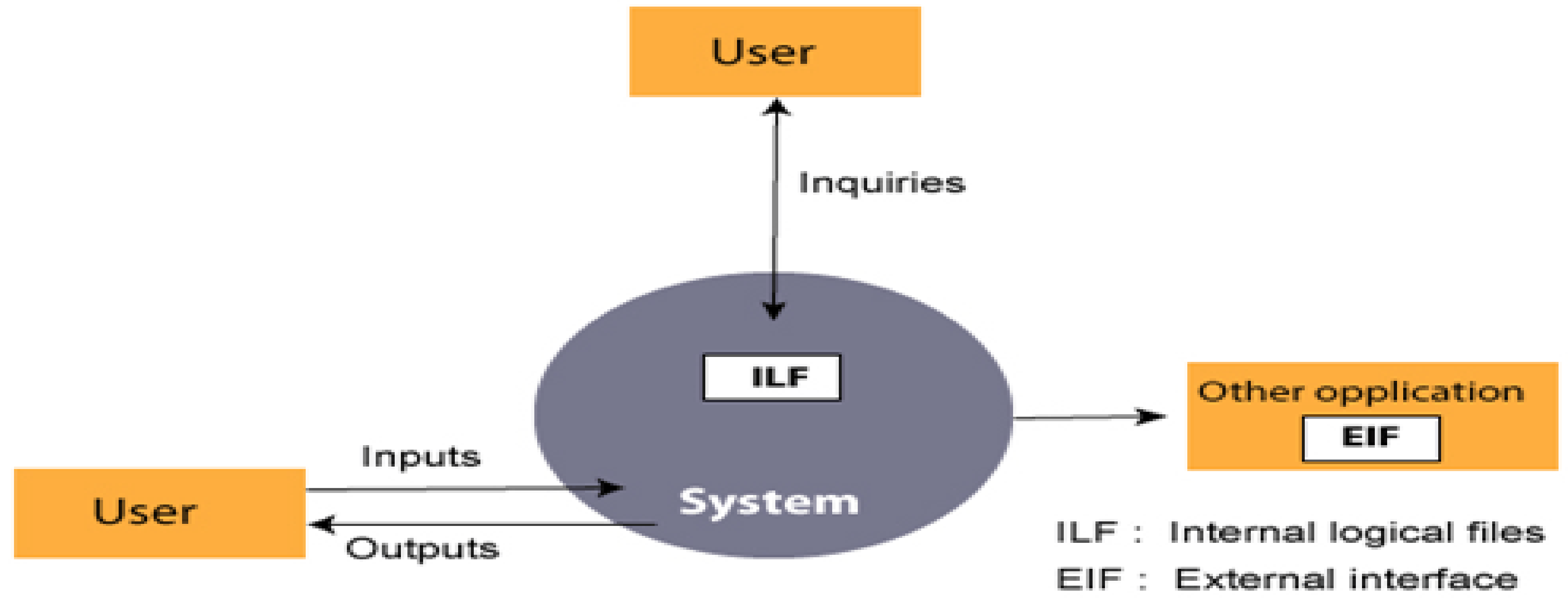
- The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client.
- FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types:



Measurements Parameters	Examples
1.Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.

FP-Based Estimation

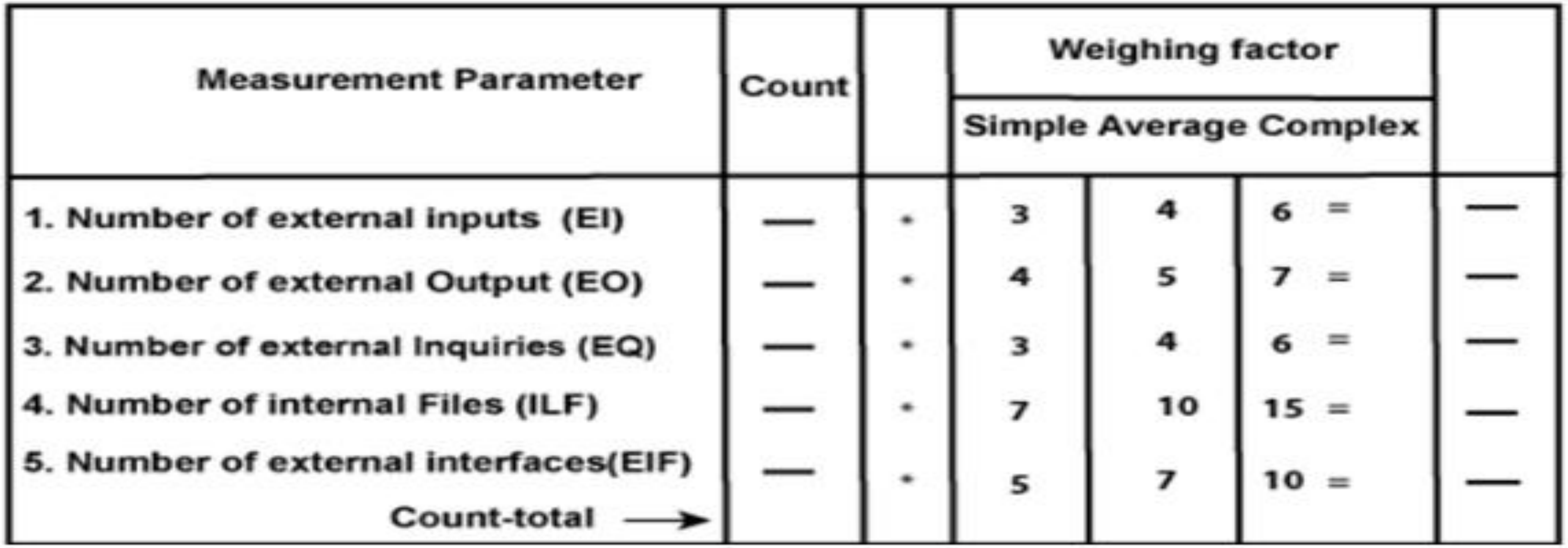
The FPA functional units are shown in Fig:



FP-Based Estimation

- All the parameters mentioned below are assigned some weights that have been experimentally determined and are shown in Table -

Computing FPs



Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	—	*	3	4	6 =	—
2. Number of external Output (EO)	—	*	4	5	7 =	—
3. Number of external Inquiries (EQ)	—	*	3	4	6 =	—
4. Number of internal Files (ILF)	—	*	7	10	15 =	—
5. Number of external interfaces(EIF)	—	*	5	7	10 =	—
Count-total →						

FP-Based Estimation

The Function Point (FP) is thus calculated as -

$$FP = \text{Count-total} * [0.65 + 0.01 * \sum(f_i)]$$
$$= \text{Count-total} * CAF$$

where Count-total is obtained from the above Table.

Calculating Complexity Adjustment Factor (CAF) –

$$CAF = [0.65 + 0.01 * \sum(f_i)]$$

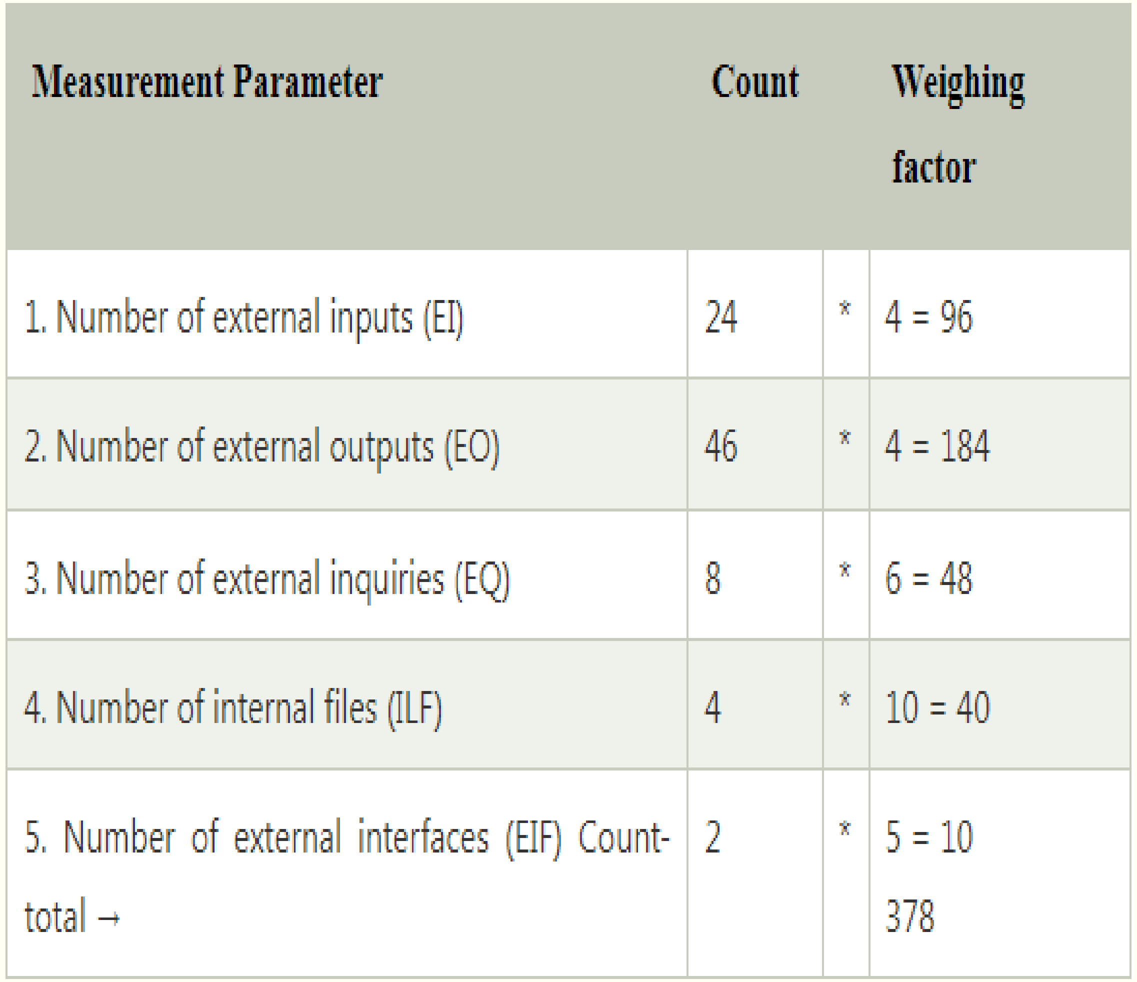
Where, f = 14 * scale (where, scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF))

(0 - No Influence 1 - Incidental 2 - Moderate 3 - Average 4 - Significant 5 - Essential)

FP-Based Estimation

- **Example** - Compute the function point, productivity, documentation, cost per function for the following data:
- Number of user inputs = 24
- Number of user outputs = 46
- Number of inquiries = 8
- Number of files = 4
- Number of external interfaces = 2
- Effort = 36.9 p-m
- Technical documents = 265 pages
- User documents = 122 pages
- Cost = $7744/ month
- Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

FP-Based Estimation



Measurement Parameter	Count		Weighing factor
1. Number of external inputs (EI)	24	*	4 = 96
2. Number of external outputs (EO)	46	*	4 = 184
3. Number of external inquiries (EQ)	8	*	6 = 48
4. Number of internal files (ILF)	4	*	10 = 40
5. Number of external interfaces (EIF) Count-total →	2	*	5 = 10 378

- So sum of all f_i ($i \leftarrow$ 1 to 14) =

4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43

- FP = Count-total * [0.65 + 0.01 * $\sum(f_i)$]
  = 378 * [0.65 + 0.01 * 43]
  = 378 * [0.65 + 0.43]
  = 378 * 1.08 = 408

- Total pages of documentation = technical document + user document
  = 265 + 122 = 387pages

- Documentation = Pages of documentation/FP
  = 387/408 = 0.94

FP-Based Estimation

$$\text{Productivity} = \frac{\text{FP}}{\text{Effort}} = \frac{408}{36.9} = 11.1$$

$$\text{Cost per function} = \frac{\text{cost}}{\text{productivity}} = \frac{7744}{11.1} = \$700$$

Estimate

- Size = Kilo Lines of Code (KLOC)
- Effort = Person/Month
- Productivity = KLOC / Person-Month
- Quality = Number of Faults / KLOC
- Cost = $ / KLOC
- Documentation = Pages of Documentation / KLOC

FP-Based Estimation

- **Example** - Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average. (User Input = 50, User Output = 40, User Inquiries = 35, User Files = 6, External Interface = 4)

FP-Based Estimation

- **Example** - Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average. (User Input = 50, User Output = 40, User Inquiries = 35, User Files = 6, External Interface = 4)

Solution:

complexity adjustment factor is average (that is 3)

scale = 3 and F = 14 * 3 = 42

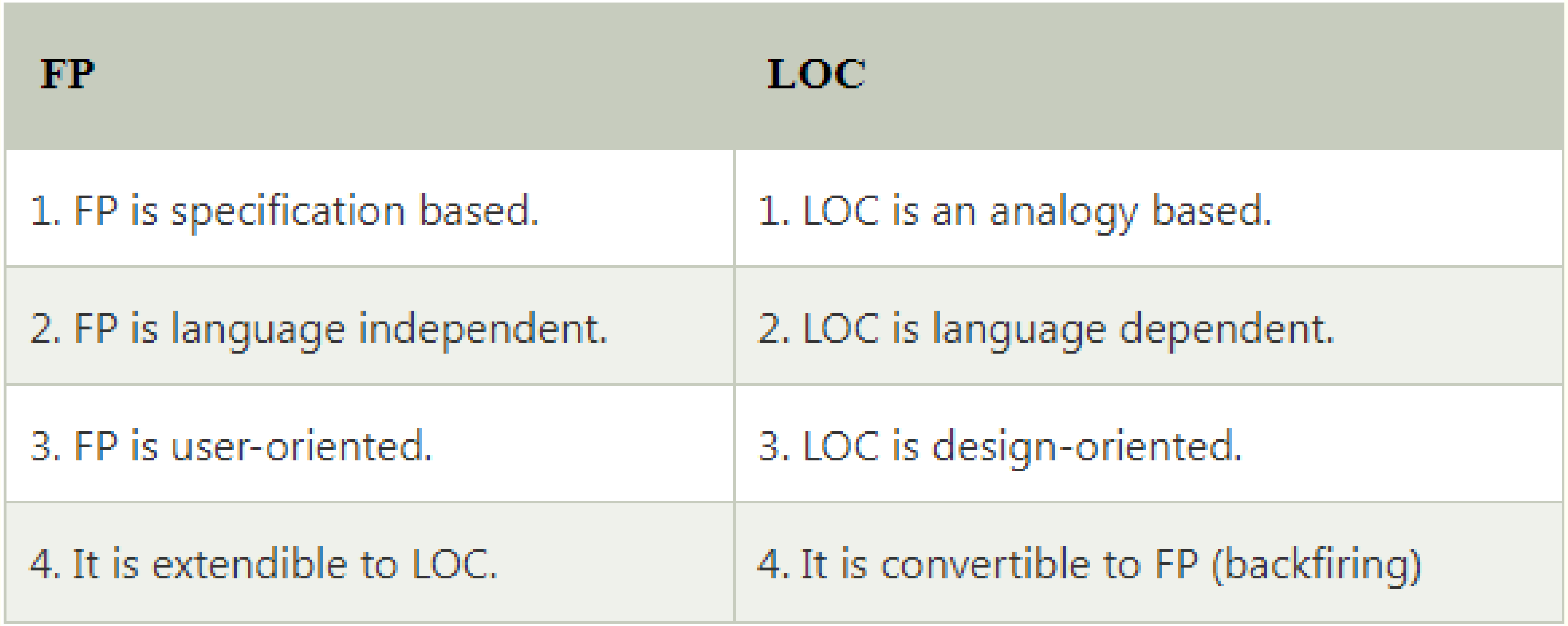
CAF = 0.65 + (0.01 * 42) = 1.07

weighting factors are also average

(50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628

Function Point = 628 * 1.07 = 671.96

FP Vs LOC



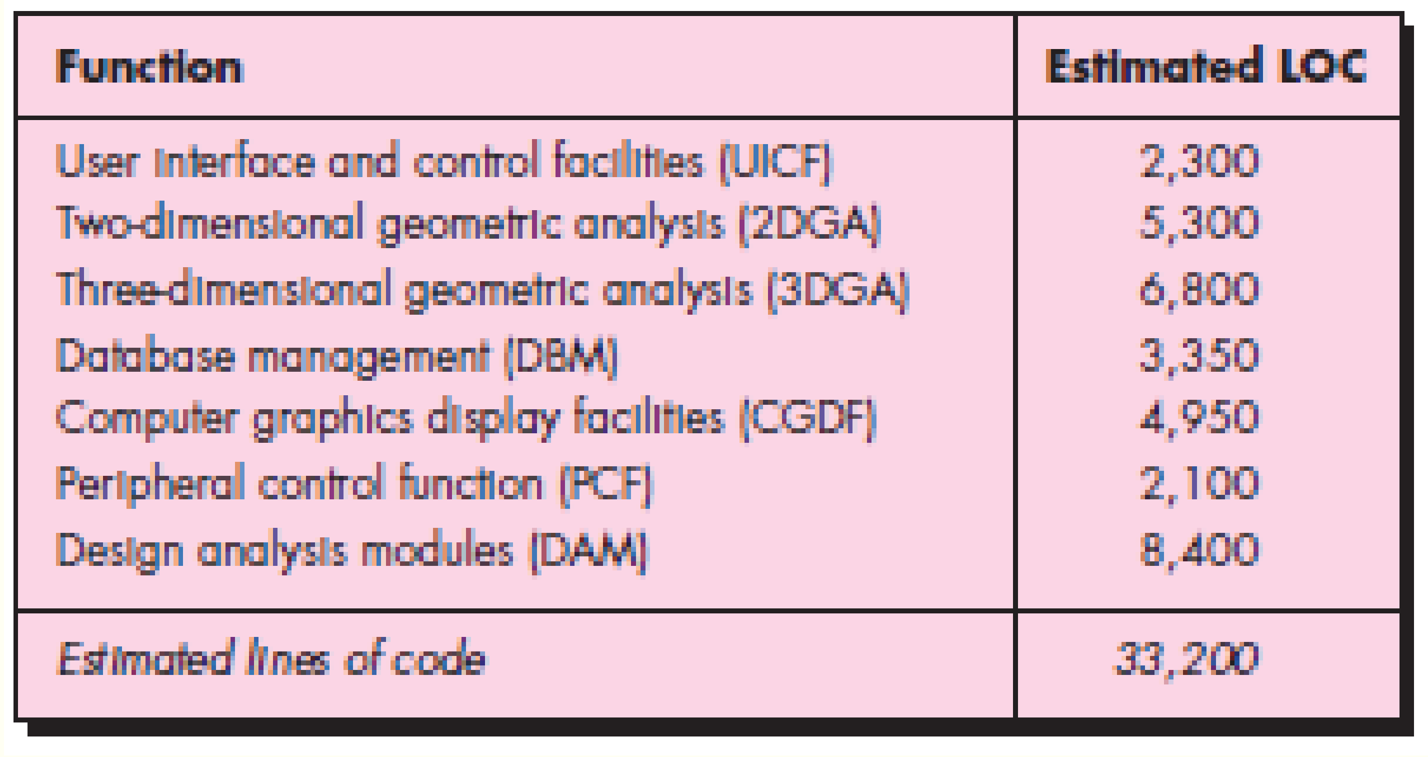
FP	LOC
1. FP is specification based.	1. LOC is an analogy based.
2. FP is language independent.	2. LOC is language dependent.
3. FP is user-oriented.	3. LOC is design-oriented.
4. It is extendible to LOC.	4. It is convertible to FP (backfiring)

Process-Based Estimation

- The process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.
- The most common technique for estimating a project is to base the estimate on the process that will be used.
- Process-based estimation begins with a delineation of software functions obtained from the project scope.
- A series of framework activities must be performed for each function.

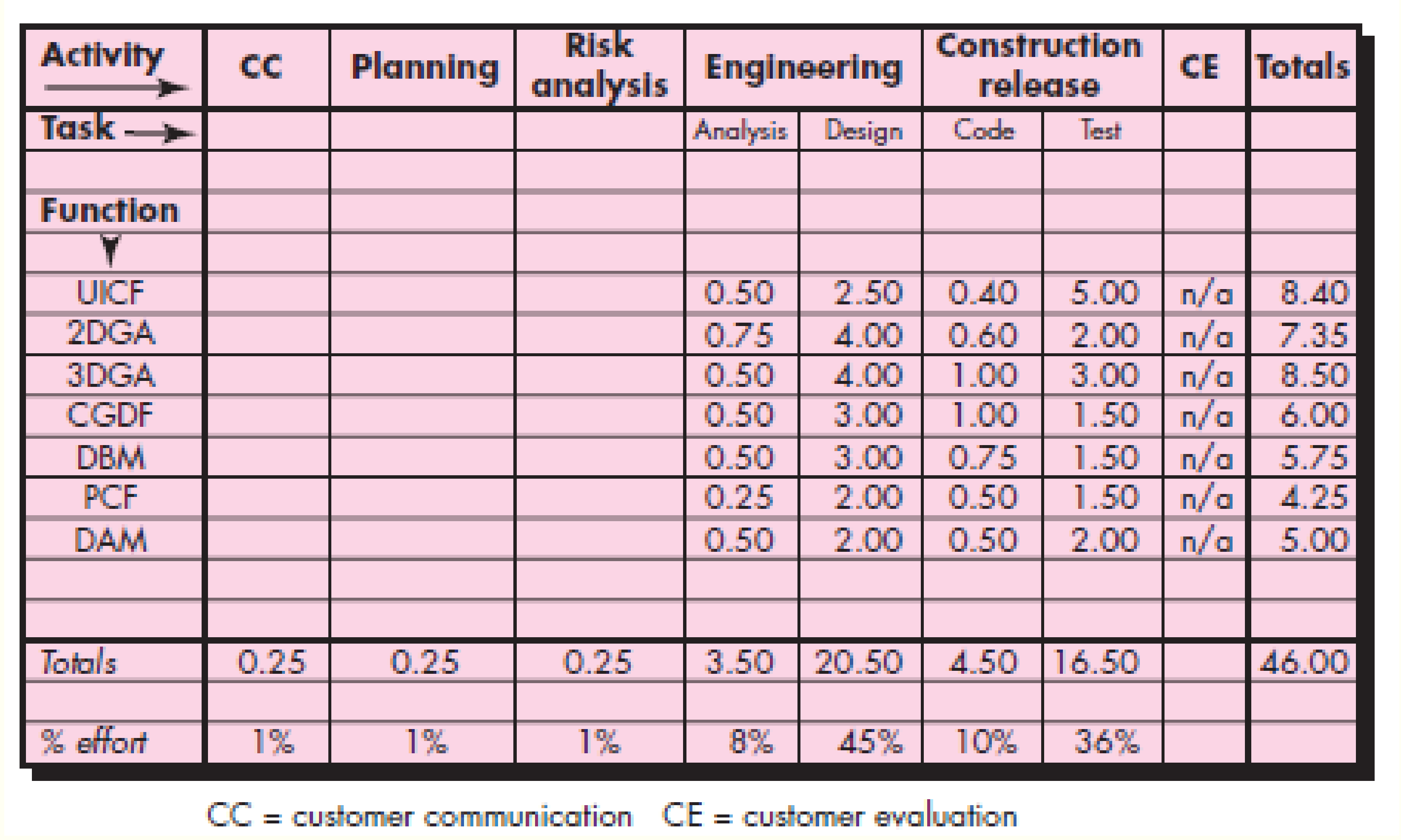
Process-Based Estimation

- The mechanical CAD software



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 function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	33,200

An Example of Process-Based Estimation



Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication   CE = customer evaluation

Figure: Process-based estimation table

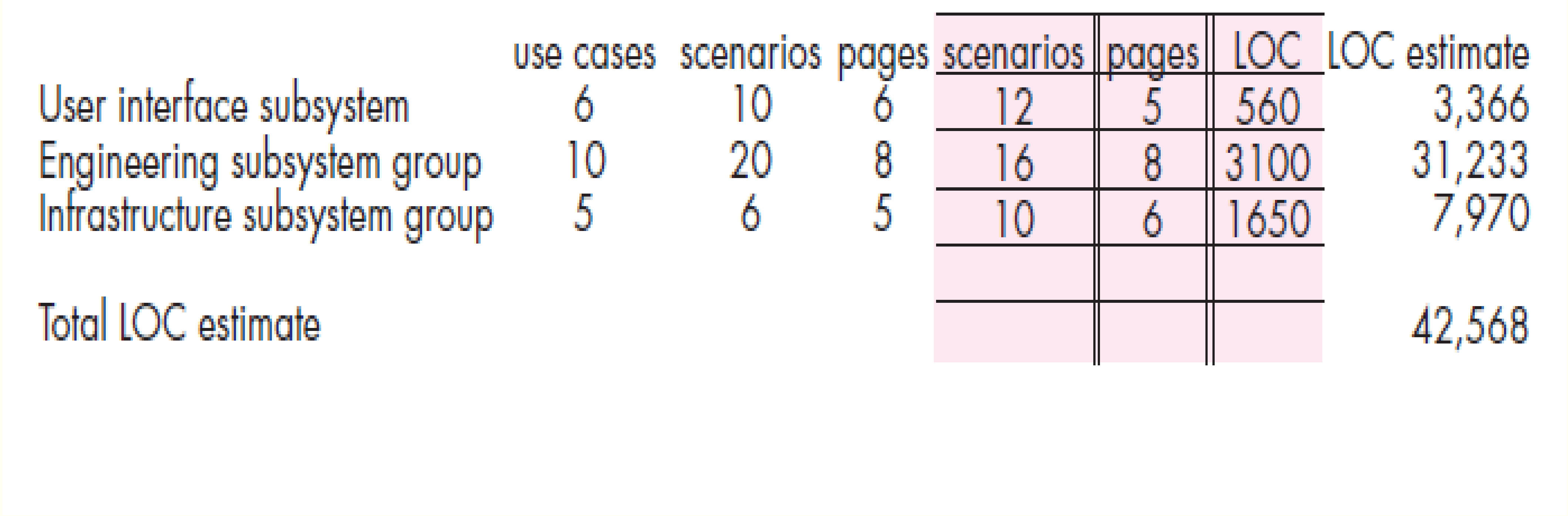
An Example of Process-Based Estimation

- Estimates of effort (in person-months) for each software engineering activity are provided for each CAD software function
- The engineering and construction release activities are subdivided into the major software engineering tasks shown.
- Gross estimates of effort are provided for customer communication, planning, and risk analysis.
- These are noted in the total row at the bottom of the table. Horizontal and vertical totals provide an indication of estimated effort required for analysis, design, code, and test.
- Based on an average burdened labor rate of $8000 per month, the total estimated project cost is $368,000 and the estimated effort is 46 person-months.

Estimation with Use-case

- Use cases provide a software team with insight into software scope and requirements.
- However, developing an estimation approach with use cases is problematic for the following reasons -
  - Use cases are described using many different formats and styles—there is no standard form.
  - Use cases represent an external view (the user's view) of the software and can therefore be written at many different levels of abstraction.
  - Use cases do not address the complexity of the functions and features that are described.
  - Use cases can describe complex behaviour (e.g., interactions) that involve many functions and features.

Example of use-case based estimation



	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

- Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the use-case estimate and the historical productivity data, the total estimated project cost is $552,000 and the estimated effort is 68 person-months.

Reconciling Estimates

- Considering the CAD software
- Low range 46 person-months(from process based estimation)
- High range 58 person-months (derived with use-case estimation)
- The average estimate is 56 person-months (using all four approaches)

Empirical Estimation models

- Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.
- These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions.
- It uses the size of the software to estimate the effort.

Structure of Estimation Models

- A typical estimation model is derived using regression analysis on data collected from past software projects.

$$E = A + B \times (e_v)^C$$

- Where:
  - E is effort in person-months
  - A, B, and C are empirically derived constants
  - e_v is the estimated variable (LOC or FP)
- the majority of estimation models have some form of project adjustment component that enables *E to be adjusted by other project characteristics*

Concrete Estimation Models

- Value of E for LOC based estimation

$E = 5.2 \times (\text{KLOC})^{0.91}$	Walston-Felix model
$E = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$	Bailey-Basili model
$E = 3.2 \times (\text{KLOC})^{1.05}$	Boehm simple model
$E = 5.288 \times (\text{KLOC})^{1.047}$	Doty model for KLOC > 9

- Value of E for FP-based estimation

$E = -91.4 + 0.355\ \text{FP}$	Albrecht and Gaffney model
$E = -37 + 0.96\ \text{FP}$	Kemerer model
$E = -12.88 + 0.405\ \text{FP}$	Small project regression model

COCOMO Model

- Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.
- COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.
- It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects.
- The key parameters are primarily **Effort & Schedule**:
  - **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
  - **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

COCOMO Models

- In COCOMO, projects are categorized into three types:
- Organic: Small, simple projects developed by small teams of software engineers, that work with less than rigid requirements; **Example** - simple business systems, simple inventory management systems, and data processing systems.
- Semi-detached: Medium in size and complexity projects, developed by teams with mixed experience, that work with mixed requirements; **Example** - developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.
- Embedded: projects that have to be developed under tight hardware and software constraints. **Example** - ATM, Air Traffic control.

COCOMO Models

- According to Boehm, software cost estimation should be done through three stages:

1. **Basic COCOMO Model:** The model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\text{Effort} = a_1{}^*(\text{KLOC})\ a_2\ \text{PM}$$
$$\text{Tdev} = b_1{}^*(\text{efforts})\ b_2\ \text{Months}$$

  - Where, **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code
  - a_1,a_2,b_1,b_2 are constants for each group of software products
  - **Tdev** is the estimated time to develop the software, expressed in months
  - **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.

COCOMO Models

Estimation of development effort: For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

- **Organic:** Effort = 2.4(KLOC) 1.05 PM
- **Semi-detached:** Effort = 3.0(KLOC) 1.12 PM
- **Embedded:** Effort = 3.6(KLOC) 1.20 PM

Estimation of development time: For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

- **Organic:** Tdev = 2.5(Effort) 0.38 Months
- **Semi-detached:** Tdev = 2.5(Effort) 0.35 Months
- **Embedded:** Tdev = 2.5(Effort) 0.32 Months

COCOMO Models

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution:

- The basic COCOMO equation takes the form:

$$\text{Effort}=a_1*(\text{KLOC})\ a_2\ \text{PM}$$

$$\text{Tdev}=b_1*(\text{efforts})b_2\ \text{Months}$$

Estimated Size of project= 400 KLOC

COCOMO Models

- (i)Organic Mode

  E = 2.4 * (400)1.05 = 1295.31 PM

  D = 2.5 * (1295.31)0.38=38.07 PM

- (ii)Semidetached Mode

  E = 3.0 * (400)1.12=2462.79 PM

  D = 2.5 * (2462.79)0.35=38.45 PM

- (iii) Embedded Mode

  E = 3.6 * (400)1.20 = 4772.81 PM

  D = 2.5 * (4772.8)0.32 = 38 PM

COCOMO Models

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution:

The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM

D=2.5(1133.12)0.35=29.3PM

COCOMO Models

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

COCOMO II Model

- **COCOMO-II** is the revised version of the original Cocomo (Constructive Cost Model) and was developed at the University of Southern California.
- It is the model that allows one to estimate the cost, effort, and schedule when planning a new software development activity.

COCOMO II Model

There are mainly 3 types of COCOMO 2 estimation models.

1. Application Composition Model

It is intended for usage with reusable components to create prototype development estimates and operates on the basis of object points. It is more suited to prototype system development.

2. Early Design Model

It generates estimates based on function points, which are subsequently converted into numerous lines of source code. The estimates at this level are based on the basic algorithm model formula.

3. Post Architecture Model

A more accurate software estimate may be generated after designing the system architecture, and it is regarded as the most detailed of all models capable of producing an accurate estimate.

Specialized Estimation Techniques

- Estimation for Agile Development
- Estimation for Web-App Projects

Estimation for Agile Development

Estimation for agile projects uses a decomposition approach that encompasses the following steps:

1. Each user scenario is considered separately for estimation purposes.

2. The scenario is decomposed into the set of software engineering tasks that will be required to develop it.

3a. The effort required for each task is estimated separately.

3b. The "volume" of the scenario can be estimated in LOC or FP

4a. Estimates for each task are summed to create an estimate for the scenario.

4b. The volume estimate for the scenario is translated into effort using historical data.

5. The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

Estimation for WebApp Projects

WebApp projects often adopt the agile process model. The following approach is suggested for adapting function points for WebApp estimation:

- *Inputs* are each input screen or form
- *Outputs* are each static Web page, each dynamic Web page script
- *Tables* are each logical table in the database
- *Interfaces* retain their definition as logical files
- *Queries* are each externally published or use a message-oriented interface.

Thank You