



Tribhuvan University
Institute of Engineering
Pulchowk Campus

Department of Electronics and Computer Engineering

Software Engineering

Cost Estimation

by

Er. Bishwas Pokharel

Lecturer, Kathford Int'l College of Engg. and Mgmt.



Project Planning:

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project.

Objectives :

- It prevents obstacles that arise in the project, such as changes in projects or organization's objectives, non-availability of resources.
- It helps in better utilization of resources and optimal usage of the allotted time for a project.



Objectives.....

- Define roles and responsibilities of the project management team members.
- Ensure that project management team works according to business objectives
- Check feasibility of schedule and user requirements



Project Scheduling

Project scheduling is concerned with determining the time limit required to complete the project. **An appropriate project schedule aims to complete the project on time,** and also helps in avoiding additional cost that is incurred when software is not developed on time.



There are various factors that delay project schedule:

- **Unrealistic deadline:** Project schedule is affected when the time allocated for completing a project is impractical and not according to the effort required for it.
- **Changing user requirements:** Sometimes, project schedule is affected when user requirements are changed after the project has started. This affects the project schedule, and thus more time is consumed both in revision of project plan and implementation of new user requirements.



- **Difficulties of team members:** Software project can also be delayed due to unforeseen difficulties of the team members. For example, some of the team members may require leave for personal reasons.
- **Lack of action by project management team:** project management team does not recognize that the project is getting delayed. Thus they do not take necessary action to speed up the software development process and complete it on time.
- **Under-estimation of resources:** This under-estimation of resources leads to delay in performing tasks of the project.



BASICS OF COST ESTIMATION

- Cost estimation is the process of approximating the costs involved in the software project.
- Cost estimation **should be done before software development** is initiated since it helps the project manager to know about resources required and the feasibility of the project.
- There are many parameters (also called factors), such as complexity, time availability, and reliability, which are considered during cost estimation process. However, **software size is considered as important parameters for cost estimation.**



Software Sizing:

- Before estimating cost, it is necessary to estimate the accurate size of software.
- This is a cumbersome task as many software are of large size. Therefore, software is divided into smaller components to estimate size.
- This is because it is easier to calculate size of smaller components, as the complexity involved in them is less than the larger components.
- These small components are then added to get an overall estimate of software size.



Various approaches can be followed for estimating size. These include direct and indirect approaches.

In **direct approach**, size can be measured in terms of lines of code (LOC)

In an **indirect approach**, size can be measured in terms of functional point (FP)



1. Lines Of Code(LOC): LOC can be defined as the number of delivered lines of code in software excluding the comments and blank lines.

Delivered lines of code include:

a. Variables Declaration

Ex: `int a , b;`

b. Actual code include logic and computation.

Ex: `if ..else` or any arithmetic computation



So Why include comments and blank lines ?

a. Blank lines:

Included to improve readability of code.

b. Comments

Including to help in code understanding as well as during maintenance.

But, these blank lines and comments do not contribute to any kind of the functionality so not considered in LOC for size estimation.



Advantages

1. Very easy to count and calculate from the developer code

Disadvantages

1. LOC is language depended.
Ex: same computation in python may have smaller code than C++.
2. Varies from one organization of code to another organization of code



Ex:

```
for( int i=0;i<5;i++)  
    cout<<i;
```

Here, lines of code =2

```
for( int i=0;i<5;i++)  
{  
    cout<<i;  
}
```

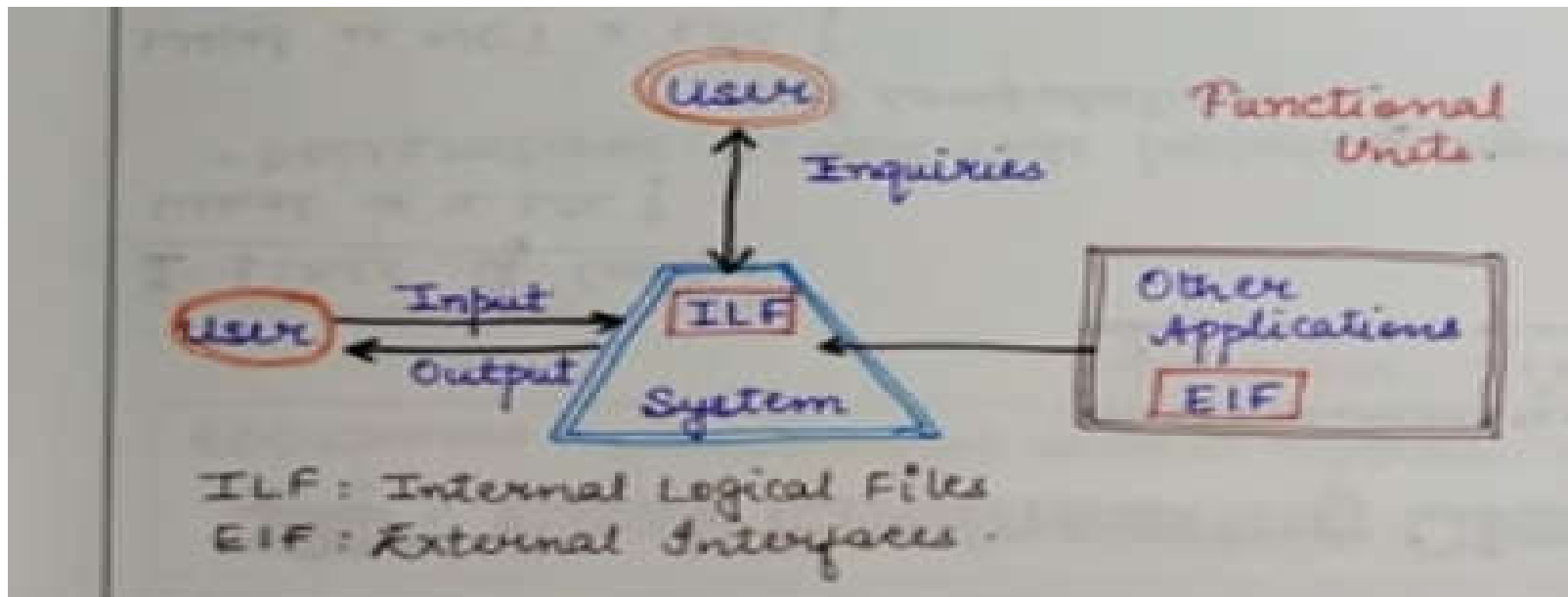
Here, lines of code =4

Same operation but differ in size of the code.



2. Function Point(FP): Function point metric is used to measure the functionality delivered by the system i.e If two different logic provides same function then we say, both have same size(not consider lines of code).

Some features FP considered to compute the size are:





- **Number of external inputs (EI):** Users and other applications act as a source of external inputs and provide distinct application oriented data or information.
- **Number of external outputs (EO):** Each external output provided by the application provides information to the user. External outputs refer to reports, screens, error message, and so on. Individual data items in reports or screens are not counted separately.



- **Number of external inquires (EQ):** online input that helps to generate immediate response in the form of online output.
- **Number of internal logical files (ILF):** Logical grouping of data that resides within the application boundary, such as database, is known as internal logical files. These files are maintained through external inputs.
- **Number of external interface files (EIF):** Logical grouping of data that resides external to the application, such as data files on tape or disk, is known as external interface file.

COUNTING FUNCTION POINTS

Step 1: Each Function Point is ranked according to complexity. There exists pre-defined weights for each F.P in each category.

Low
Average
High

Functional Units	Weighing Factors		
	Low	Average	High
EI			
EO			
EQ			
ILF			
EIF			

Step 2: Calculate Unadjusted Function Point by multiplying each F.P by its corresponding weight factor.

$$UFP = \sum_{i=1}^5 \sum_{j=1}^3 (Z_{ij} * W_{ij})$$

How To Assign weights or rank function points?
 ↳ Dependant on the organization.
 ↳ Based on past projects.

Step 3: Calculate Final Function Points

$$\text{Final F.P.} = UFP \times CAF$$

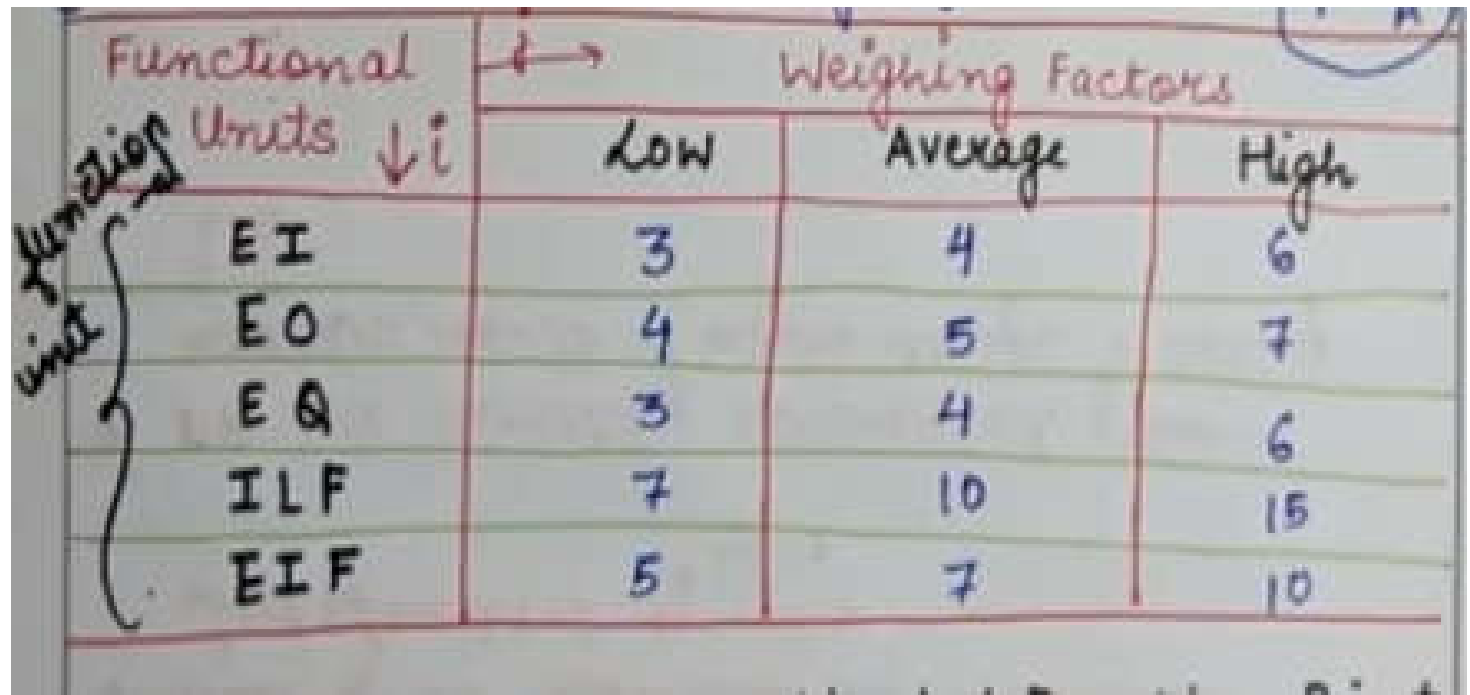
Complexity Adjustment Factor
 Calculated using 14 aspects of processing complexity

- 14 questions answered on a scale of 0 to 5
- 0 → NO Influence
- 1 → Incidental
- 2 → Moderate
- 3 → Average
- 4 → Significant
- 5 → Essential



Step 1:

- Each organization have their own criteria to defined weights based on their requirements.
- It is predefined value. (In examination, table is given)
- Ex:



A handwritten table on lined paper. The first column is labeled 'Functional Units' with a downward arrow and a bracket on the left side labeled 'function unit'. The next three columns are under the heading 'Weighing Factors' with an arrow pointing to them, and are labeled 'Low', 'Average', and 'High'. The rows contain the following data:

Functional Units ↓	Low	Average	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10



Step 2 :

- Calculate unadjusted FP.
- Ex: Suppose, if you have 5 external inputs of low category and 6 external output of high category only in whole system then,

$$\{5*3+6*7\}=\text{ans}(\text{unadjusted FP})$$



Step 3 :

- Calculate Final FP.
= U.F.P (unadjusted functional point)*
CAF(complexity adjustment factor)

where $CAF = [0.65 + 0.01 * \sum Fi]$

i=1 to 14 ie total number of questions where each question have answer with scale value 0 to 5.

- $\sum Fi$ sum of all scale value from 1 to 14.



The value adjustment factors are based on the response to 14 questions, which are listed below:

1. Is reliable backup and recovery required by the system?
2. Is data communication required to transfer the information?
3. Do distributed processing functions exist?
4. Is performance vital?
5. Does the system run under immensely utilised operational environment?
6. Is on-line data entry required by system?
7. Is it possible for the on-line data entry (that requires the input transaction) to be built over multiple screens or operations?
8. Is updation of internal logical files allowed on-line?
9. Are the inputs, outputs, files, or inquires complex?
10. Is the internal processing complex?
11. Is the code reusable?
12. Does design include conversion and installation?
13. Does system design allow multiple installations in different organizations?
14. Is the application easy to use and does it facilitate changes?



Solve:

Given the following values, compute F.P when all complexity adjustment factors and weighting factors are average .

user i/p =50

user o/p=40

user enquiries =35

User files =6

External interface =4

Solution: F.P = 671.96

Solved Numerical

Q. Given the following values, compute F.P when all complexity adjustment factors and weighting factors are average

CAF

$$\text{User I/P} = 50$$

$$\text{User O/P} = 40$$

$$\text{User Enquiries} = 35$$

$$\text{User Files} = 6$$

$$\text{External Interfaces} = 4$$

$$\sum_{i=1}^{14} F_i \rightarrow \underline{14 \times 3}$$

$$4, 5, 4, 10, 7$$

$$UFP = \sum_{i=1}^5 \sum_{j=1}^3 Z_{ij} w_{ij}$$

$$\begin{aligned} UFP &= 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7 \\ &= 200 + 200 + 140 + 60 + 28 \\ &= 628 \end{aligned}$$

$$\begin{aligned} CAF &= 0.65 + (0.01 \times \sum F_i) = 0.65 + 0.01(14 \times 3) \\ &= 0.65 + 0.42 \end{aligned}$$

$$\begin{aligned} F.P &= UFP \times CAF \\ &= 628 \times 1.07 = 1.07 \end{aligned}$$



Advantages

1. Independent of Language and technical tool
2. Directly estimated from requirements before design and coding.

Disadvantages

1. More complex calculation than LOC.



Cost Estimation Models

Algorithmic models:

- Estimation in these models is performed with the help of **mathematical equations**, which are based on historical data or theory.
- In order to estimate cost accurately, various **inputs are provided to these algorithmic models**.
- These **inputs include software size and other parameters**.
- The various algorithmic models used are COCOMO, COCOMO II, and software equation.



Non-algorithmic models:

- Estimation in these models depends on the prior experience and domain knowledge of project managers.
- Note that these models do not use mathematical equations to estimate cost of software project.
- The various non-algorithmic cost estimation models are expert judgment, estimation by analogy, and price to win.

Note: We will discuss algorithmic models only



Constructive Cost Model(COCOMO):

- COCOMO is one of the most widely used software estimation models in the world.
- It was developed by Barry Boehm in 1981 .
- In this model, size is measured in terms of thousand of delivered lines of code (KDLOC).



In order to estimate effort accurately, COCOMO model divides projects into three categories :

1. Organic projects:

- These projects are small in size (not more than 50 KDLOC) and thus easy to develop.
- In organic projects, small teams with prior experience work together to accomplish user requirements, which are less demanding.



- Most people involved in these projects have thorough understanding of how the software under development contributes in achieving the organization objectives.
- Examples of organic projects include simple business system, inventory management system, payroll management system, and library management system.



2. Embedded projects:

- These projects are complex in nature (size is more than 300 KDLOC) and the organizations have less experience in developing such type of projects.
- Developers also have to meet stringent user requirements. These software projects are developed under tight constraints (hardware, software, and people). Examples of embedded systems include software system used in avionics and military hardware..



3. Semi-detached projects:

- These projects are less complex as the user requirements are less stringent compared to embedded projects.
- The size of semi-detached project is not more than 300 KDLOC.
- Examples of semi-detached projects include operating system, compiler design, and database design.



Constructive cost model is based on the hierarchy of three models, namely, **basic model, intermediate model, and advance model.**

A. Basic Model:

- In basic model, only the size of project is considered while calculating effort.
- To calculate effort, use the following equation (known as effort equation):

$$E = A \times (\text{size})^B \dots\dots\dots(i)$$

where E is the effort in person-months and size is measured in terms of KDLOC.



The values of constants 'A' and 'B' depend on the type of the software project. In this model, values of constants ('A' and 'B') for three different types of projects are listed in Table...

Project Type	A	B
Organic project	3.2	1.05
Semi-detached project	3.0	1.12
Embedded project	2.8	1.20

For example, if the project is an organic project having a size of 30 KDLOC, then effort is calculated using equation, $E = 3.2 \times (30)^{1.05}$

$$E = 114 \text{ PM}$$



B. Intermediate Model:

- In intermediate model, parameters like software reliability and software complexity are also considered along with the size, while estimating effort.
- To estimate total effort in this model, a number of steps are followed, which are listed below:
 - i. Calculate an initial estimate of development effort by considering the size in terms of KDLOC.
 - ii. Identify a set of 15 parameters, which are derived from attributes of the current project. All these parameters are rated against a numeric value, called **multiplying factor**.



Effort adjustment factor (EAF) is derived by multiplying all the multiplying factors with each other.

- iii. Adjust the estimate of development effort by multiplying the initial estimate calculated in step 1 with EAF.

let us consider an example

For simplicity reasons, an organic project whose size is 45 KDLOC is considered.

- i. $E(i) = 3.2 \times (45)^{1.05} = 174 \text{ PM}$



Cost Drivers	Description	Very Low	Low	Rating Nominal	High	Very High	Extra High
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	–
DATA	Database size	–	0.94	1.00	1.08	1.16	–
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
TIME	Execution time constraint	–	–	1.00	1.11	1.30	1.66
STOR	Main storage constraint	–	–	1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility	–	0.87	1.00	1.15	1.30	–
TURN	Computer turnaround time	–	0.87	1.00	1.07	1.15	–
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	–
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	–
PCAP	Programmer capability	1.42	1.17	1.00	0.86	0.70	–
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90	–	–
LEXP	Language experience	1.14	1.07	1.00	0.95	–	–
MODP	Modern programming practices	1.24	1.10	1.00	0.91	0.82	–
TOOL	Software Tools	1.24	1.10	1.00	0.91	0.83	–
SCED	Development Schedule	1.23	1.08	1.00	1.04	1.10	

Fig: Standard value for cost driver



Cost Drivers	Rating	Multiplying Factor
Reliability	High	1.15
Complexity	Low	0.85
Application Experience	High	0.91
Programmer Capability	Nominal	1.00

Fig: cost drivers in project



ii. The multiplying factors of all cost drivers considered for the project are multiplied with each other to obtain EAF. For instance, using cost drivers listed in Table above, EAF is calculated as:

$$=0.8895(=1.15 \times 0.85 \times 0.91 \times 1.00)$$

iii. Once EAF is calculated, the effort estimates for a software project is obtained by multiplying EAF with initial estimate (E_i). To calculate effort use the following equation:

$$\text{Total effort} = \text{EAF} \times E_i (=0.8895 * 174)$$

For this example, the total effort will be 155 PM



C. Advance model: To estimate total effort in this model, a number of steps are followed, which are listed below:

- i. Calculate an initial estimate of development effort by considering the size in terms of KDLOC.
- ii. Four phases in advance COCOMO model namely, requirements planning and product design (RPD), detailed design (DD), code and unit test (CUT), and integration and test (IT). In advance model, each cost driver is rated as very low, low, nominal, high, and very high. For all these ratings, cost drivers are assigned multiplying factors.
- iii. Multiply result of i and ii.



Rating	RPD	DD	CUT	IT
Very Low	1.80	1.35	1.35	1.50
Low	0.85	0.85	0.85	1.20
Nominal	1.00	1.00	1.00	1.00
High	0.75	0.90	0.90	0.85
Very High	0.55	0.75	0.75	0.70

Fig: Multiplying Factors for ACAP in Different Phases



For example, software project (of organic project type), with a size of 45 KDLOC and rating of ACAP cost driver as nominal is considered (That is 1.00). To calculate effort for code and unit test phase in this example, only ACAP cost drivers are considered.

- i. Initial effort can be calculated by using equation
$$E_i = 3.2 \times (45)^{1.05} = 174 \text{ PM}$$
- ii. 1 (for nominal code and unit test)
- iii. Using the value of E_i , final estimate of effort can be calculated by using the following equation:

$$E = E_i \times 1$$

$$\text{That is, } E = 174 \times 1 = 174 \text{ PM}$$



Advantages	Disadvantages
<ul style="list-style-type: none">▪ Easy to verify the working involved in it.▪ Cost drivers are useful in effort estimation as they help in understanding impact of different parameters involved in cost estimation.▪ Efficient and good for sensitivity analysis.▪ Can be easily adjusted according to the organization needs and environment.	<ul style="list-style-type: none">▪ Difficult to accurately estimate size, in the early phases of the project.▪ Vulnerable to misclassification of the project type.▪ Success depends on calibration of the model according to the needs of the organization. This is done using historic data, which is not always available.▪ Excludes overhead cost, travel cost and other incidental cost.



!! Attendance please !!

Thank You!!!