Module 4

Software Project Planning

After the finalization of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalization of the SRS.

Software Project Planning

In order to conduct a successful software project, we must understand:

- Scope of work to be done
- The risk to be incurred
- The resources required
- The task to be accomplished
- The cost to be expended
- The schedule to be followed

Software Project Planning

Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.

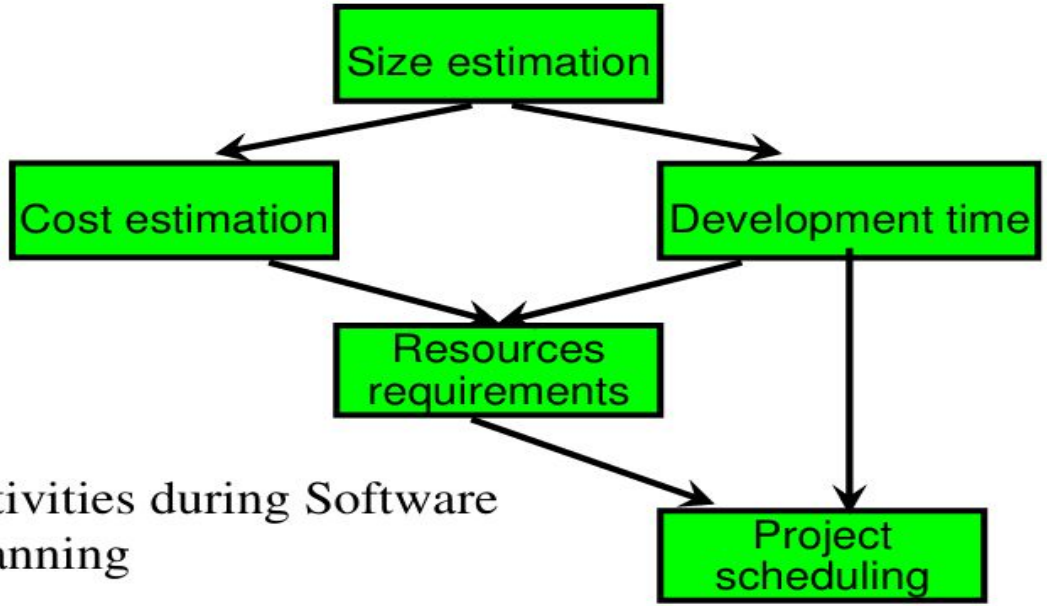


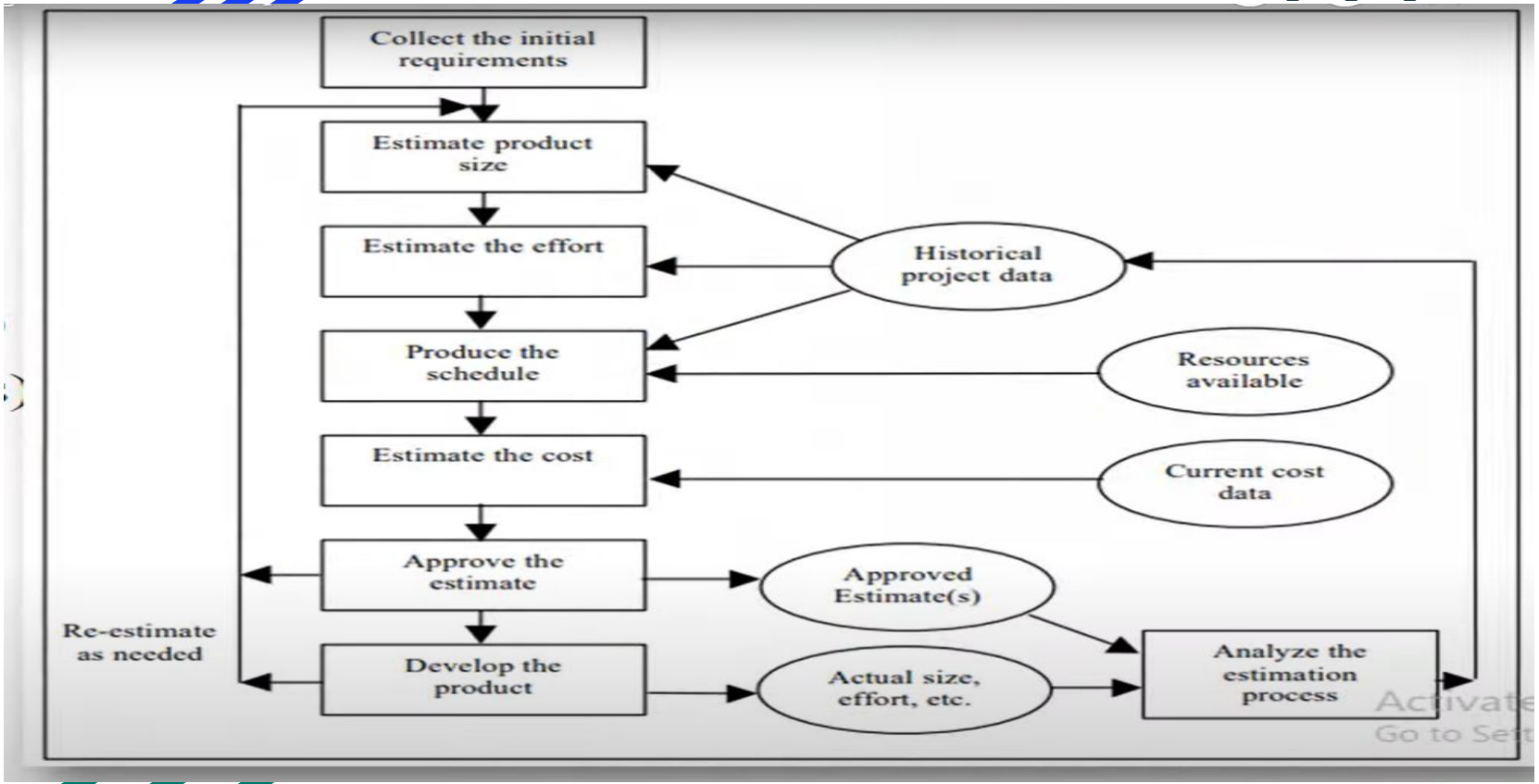
Fig. 1: Activities during Software Project Planning

Software Project Estimation

Factors affect on project estimation

1. **Cost:** The project will fail if you do not have sufficient funds to complete it. So, At early stage estimate project cost & ensure you have enough money to complete the work.
2. **Time:** Estimate both overall project duration & timing of individual tasks. It also enables you to manage client expectations for key deliverables.
3. **Size & Scope:** All the tasks that must be completed in order to deliver product on time. You can ensure that you have right materials & expertise on the project by estimating how much work is involved and exactly what tasks must be completed.
4. **Risk:** Estimating predicting risk, what events will occur during the project's life cycle and how serious they will be. Create risk management plans.
5. **Resource's:** Resource management ensures that you have all the resources you require & make best use of them. Like, tools, people, materials, hardware, software & other resources

Basic project estimation process



Steps of software project estimation

1. Estimate Product Size (LOC & FP):

- It is the very first step to make an effective estimation of the project.
- A Customer's requirements, SRS Document & System design document used for estimating the size of a software.
- Estimate project size can be through similar project developed in past. This is called estimation by Analogy.
- The system is divided into several subsystems depending on functionality & size of each subsystem is calculated.

Steps of software project estimation

2. Estimate Project Efforts:

- The estimation of effort can be made from the organizational specifics of SDLC.
- Software development project involves Design, Coding, Testing, Writing, Reviewing documents, Implementing prototypes & Deliverable it decide overall project efforts.
- The project effort estimate requires you to identify and estimate & then sum up all the activities you must perform to build a product of the estimated size.

There are two main ways:

1. The best way to estimate effort is based on the organization's own historical data of development process. Suppose you have similar SDLC, Project size, Development methodology, Tools, Team with similar skills and experience for the new project.
2. If the project is in different nature which requires the organization to adopt different strategy or different models based on algorithmic approach. Ex. COCOMO Model.

Steps of software project estimation

3. Estimate Project Schedule:

- It involves estimating number of people who will work on the project & what they will work on (the Work Breakdown Structure).
- Also when they will start working on project & when they will finish.
- Once you have this information, you need to lay it out into a calendar schedule.

Steps of software project estimation

4. Estimate Project Cost:

- The cost of a project is derived not only from the estimates of person efforts and size.
- Other parameters such as purchase hardware, software, travel for meeting, telecommunication costs (long distance phone calls, video-conferences), training , office space etc.
- Exactly how you estimate total project cost will depend on how your organization allocates costs.
- The simplest labor cost can be obtained by multiplying the project's effort estimate (in hours) by a general labor rate ($ per hour).
- A more accurate labor cost would result from using a specific labor rate for each staff position (e.g., Technical, QA, Project Management, Documentation, Support, etc.)

Decomposition Techniques

- **Software Project Estimation** is a form of problem solving (To estimate cost, time & efforts in software project.)
- **Decomposition Technique** is divide & conquer approach of Software Project Estimation.
- By decomposition a project into major functions like software engineering related activities, cost, schedule & efforts estimation can be performed in stepwise manner.

➢**Decomposition Techniques are:**

1. Software Sizing
2. Problem based Estimation
3. Process based Estimation

1.Software Sizing

- Sizing represents the project planner's first major challenge.

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

1. The degree to which the planner has 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.
4. The stability of product requirements & environment that supports software engineering effort.

➢**Approaches of Software Sizing:**

1. "Fuzzy Logic" Sizing
2. Function Point Sizing
3. Standard Component Sizing
4. Change Sizing

1.Approaches of Software Sizing

1. "Fuzzy Logic" Sizing:

- To apply this approach, the planner must identify the type of application.
- Although personal experience can be used, the planner should also have access to historical database of projects so that estimates can be compared to actual experience.

2. Function Point Sizing:

- The planner develops estimates of the information domain characteristics.

1.Approaches of Software Sizing

3. Standard Component Sizing:

- Standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC and object-level instructions.
- Uses historical project data to determine the delivered size per standard component.

4. Change Sizing:

- This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.
- The planner estimates reuse, adding code, changing code, deleting code of modifications that must be accomplished.
- Using an "effort ratio" for each type of change, the size of the change may be estimated.

1.Problem based estimation

- Lines of code and function points were described as measures from which productivity metrics can be computed. **(LOC/pm & FP/pm)**

- **LOC and FP data are used in two ways during software project estimation:**

1. As an estimation variable to "size" each element of the software .
2. As baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

➢A three-point or expected value can then be computed

$S = (s_{opt} + 4s_m + s_{pess})/6$

- S = expected-value for the estimation variable (size)
- s_{opt} = optimistic value
- s_m = most likely value
- s_{pess} = pessimistic value

Software Project Planning

Size Estimation

Lines of Code (LOC)

If LOC is simply a count of the number of lines then figure shown below contains 18 LOC .

When comments and blank lines are ignored, the program in figure 2 shown below contains 17 LOC.

Fig. 2: Function for sorting an array

```
1.  int. sort (int x[ ], int n)
2.  {
3.  int i, j, save, im1;
4.  /*This function sorts array x in ascending order */
5.  If (n<2) return 1;
6.  for (i=2; i<=n; i++)
7.  {
8.  im1=i-1;
9.  for (j=1; j<=im; j++)
10. if (x[i] < x[j])
11. {
12. Save = x[i];
13. x[i] = x[j];
14. x[j] = save;
15. }
16. }
17. return 0;
18. }
```

Software Project Planning

Furthermore, if the main interest is the size of the program for specific functionality, it may be reasonable to include executable statements. The only executable statements in figure shown above are in lines 5-17 leading to a count of 13. The differences in the counts are 18 to 17 to 13. One can easily see the potential for major discrepancies for large programs with many comments or programs written in language that allow a large number of descriptive but non-executable statement. Conte has defined lines of code as:

Software Project Planning

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declaration, and executable and non-executable statements".

This is the predominant definition for lines of code used by researchers. By this definition, figure shown above has 17 LOC.

Software Project Planning

Function Count

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

Software Project Planning

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the system
▪ Outputs	:	information leaving the system
▪ Enquiries	:	requests for instant access to information
▪ Internal logical files	:	information held within the system
▪ External interface files	:	information held by other system that is used by the system being analyzed.

Software Project Planning

The FPA functional units are shown in figure given below:

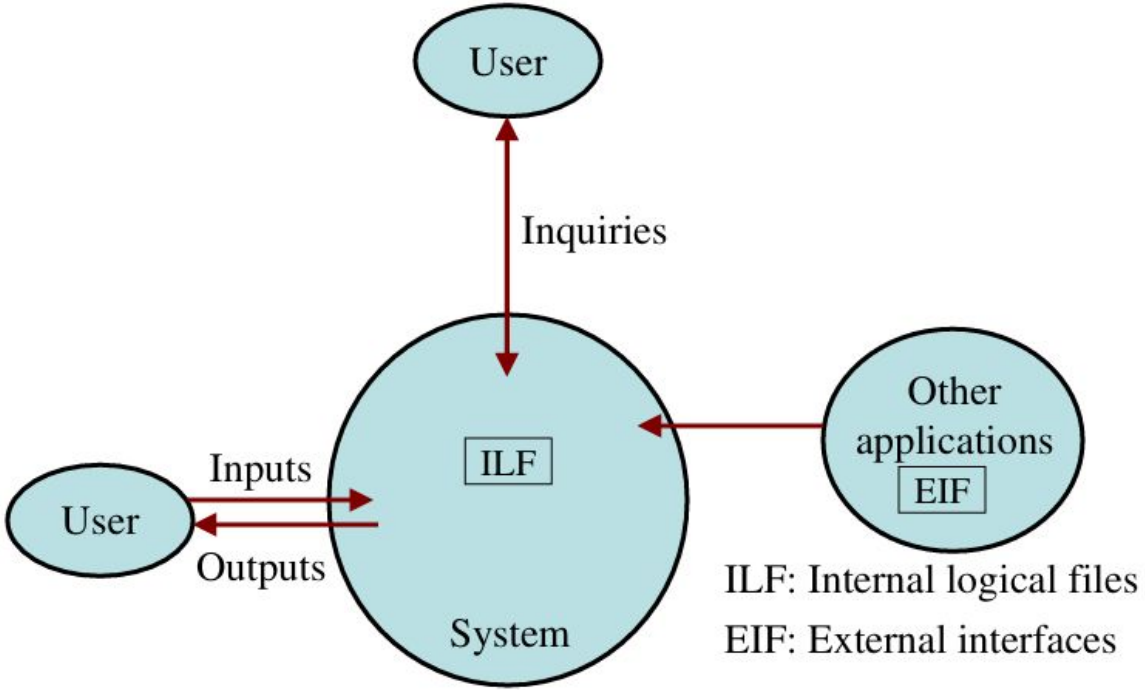


Fig. 3: FPAs functional units System

Software Project Planning

The five functional units are divided in two categories:

(i) Data function types

- Internal Logical Files (ILF): A user identifiable group of logical related data or control information maintained within the system.
- External Interface files (EIF): A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

Software Project Planning

(ii) Transactional function types

- External Input (EI): An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- External Output (EO): An EO is an elementary process that generate data or control information to be sent outside the system.
- External Inquiry (EQ): An EQ is an elementary process that is made up to an input-output combination that results in data retrieval.

Software Project Planning

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; i.e. they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.

Software Project Planning

- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate.
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

- **Lines of Code (LOC)**
  - Measures software size by **counting lines of source code**.
  - Language and coding-style dependent.
  - Example: Same program → 100 LOC in C, 40 LOC in Python.

Function Points (FP)

- Measures software size by **counting user-visible functionalities**.
- Language-independent.
- Considers: Inputs, Outputs, User Interactions, Files, Interfaces.
- Example: Library Management System = ~200 FP (same in any language).

- **Lines of Code (LOC)**
  - Measures software size by **counting lines of source code**.
  - Language and coding-style dependent.
  - Example: Same program → 100 LOC in C, 40 LOC in Python.

Function Points (FP)

- Measures software size by **counting user-visible functionalities**.
- Language-independent.
- Considers: Inputs, Outputs, User Interactions, Files, Interfaces.
- Example: Library Management System = ~200 FP (same in any language).

- **Lines of Code (LOC)**
  - Measures software size by **counting lines of source code**.
  - Language and coding-style dependent.
  - Example: Same program → 100 LOC in C, 40 LOC in Python.

Function Points (FP)

- Measures software size by **counting user-visible functionalities**.
- Language-independent.
- Considers: Inputs, Outputs, User Interactions, Files, Interfaces.
- Example: Library Management System = ~200 FP (same in any language).

LOC → “How much code?” → Best for productivity & maintenance tracking.

FP → “What functionality?” → Best for early estimation & cross-project comparison.

Industry practice:

- Use **FP for planning & estimation**.
- Use **LOC for productivity and quality analysis**.

Uses of LOC

- Productivity measurement (**LOC per person-month**).
- Defect density (**defects per KLOC**).
- Maintenance complexity analysis.
- Input for effort & cost estimation (e.g., **COCOMO I**).

Uses of Function Points

- Early project estimation (before coding).
- Effort & cost prediction (**COCOMO II**).
- Cross-project and cross-language comparison.
- Productivity benchmarking (**FP per staff-month**).
- Basis for vendor contracts in outsourcing.

Aspect	LOC	Function Points (FP)
Basis	Physical source code lines	Functionality delivered
Language	Dependent	Independent
Stage	After coding/design	Early in requirements
Focus	Developer-centric	User-centric
Accuracy	Varies with coding style	Standardized rules
Usage	Productivity, defect analysis	Estimation, benchmarking

Software Project Planning

Counting function points

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 1 : Functional units with weighting factors

Software Project Planning

Table 2: UFP calculation table

Functional Units	Count Complexity			Complexity Totals	Functional Unit Totals
External Inputs (EIs)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External Outputs (EOs)		Low x 4	=		
		Average x 5	=		
		High x 7	=		
External Inquiries (EQs)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External logical Files (ILFs)		Low x 7	=		
		Average x 10	=		
		High x 15	=		
External Interface Files (EIFs)		Low x 5	=		
		Average x 7	=		
		High x 10	=		
Total Unadjusted Function Point Count					

Software Project Planning

The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

Software Project Planning

The procedure for the calculation of UFP in mathematical form is given below:

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

Where i indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

Software Project Planning

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions noted in table 3.

Software Project Planning

Table 3 : Computing function points.

Rate each factor on a scale of 0 to 5.

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
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 on line data entry ?
7. Does the on line data entry require the input transaction to be built over multiple screens or operatior
8. Are the master files updated on line ?
9. Is 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 ?

14 General System Characteristics (GSCs) Data Communications

1. Distributed Data Processing
2. Performance
3. Heavily Used Configuration
4. Transaction Rate
5. On-Line Data Entry
6. End-User Efficiency
7. On-Line Update
8. Complex Processing
9. Reusability
10. Installation Ease
11. Operational Ease
12. Multiple Sites
13. Facilitate Change

Software Project Planning

Functions points may compute the following important metrics:

Productivity	=	FP / persons-months
Quality	=	Defects / FP
Cost	=	Rupees / FP
Documentation	=	Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Functions Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point method is also being developed.

Software Project Planning

Example: 4.1

Consider a project with the following functional units:

Number of user inputs	= 50
Number of user outputs	= 40
Number of user enquiries	= 35
Number of user files	= 06
Number of external interfaces	= 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Software Project Planning

Solution

We know

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

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = $(0.65 + 0.01\ \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

Software Project Planning

Example:4.2

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Software Project Planning

Solution

Unadjusted function point counts may be calculated using as:

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

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

Software Project Planning

Example: 4.3

Consider a project with the following parameters.

(*i*) External Inputs:
- (a) 10 with low complexity
- (b) 15 with average complexity
- (c) 17 with high complexity

(*ii*) External Outputs:
- (a) 6 with low complexity
- (b) 13 with high complexity

(*iii*) External Inquiries:
- (a) 3 with low complexity
- (b) 4 with average complexity
- (c) 2 high complexity

Software Project Planning

(*iv*) Internal logical files:

(a) 2 with average complexity

(b) 1 with high complexity

(v) External Interface files:

(a) 9 with low complexity

In addition to above, system requires

i. Significant data communication

ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Software Project Planning

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	
	15	Average x 4	=	60	
	17	High x 6	=	102	192
External Outputs (EOs)	6	Low x 4	=	24	
	0	Average x 5	=	0	
	13	High x 7	=	91	115
External Inquiries (EQs)	3	Low x 3	=	9	
	4	Average x 4	=	16	
	2	High x 6	=	12	37
External logical Files (ILFs)	0	Low x 7	=	0	
	2	Average x 10	=	20	
	1	High x 15	=	15	35
External Interface Files (EIFs)	9	Low x 5	=	45	
	0	Average x 7	=	0	
	0	High x 10	=	0	45
Total Unadjusted Function Point Count					424

Software Project Planning

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

$$\begin{aligned} CAF &= (0.65 + 0.01 \times \Sigma F_i) \\ &= (0.65 + 0.01 \times 41) \\ &= 1.06 \\ FP &= UFP \times CAF \\ &= 424 \times 1.06 \\ &= 449.44 \end{aligned}$$

Hence $\boxed{FP = 449}$

Proven Techniques for an Accurate Software Project Estimation

There's no universal method for estimation, but these tried-and-tested techniques have stood the test of time:

1. Analogous Estimation

This method involves using past projects as a reference point.

Example:
If a previous project took four months to develop a customer portal with similar features, you can use that timeline as a baseline for your current project.

- **Pros:** Quick and straightforward.
- **Cons:** Only works if the new project closely resembles the old one.

2. Bottom-Up Estimation

Here, you break the project into tasks, estimate each one individually, and add them up.

Example:

For a simple CRM system:

- Contact management: 30 hours
- Reporting: 40 hours
- User roles: 20 hours
- Total: 90 hours.

- **Pros:** Highly accurate for detailed projects.
- **Cons:** Time-intensive.

3. Three-Point Estimation

This technique accounts for uncertainty by considering three scenarios:

- **Optimistic (O):** Best-case scenario.
- **Pessimistic (P):** Worst-case scenario.
- **Most Likely (M):** Expected outcome.

Formula:

Estimate = (O+4M+P)/6

Example:

Developing a dashboard feature:

- Optimistic: 5 days
- Most Likely: 10 days
- Pessimistic: 20 days

Estimate = (5 + (4 × 10) + 20) / 6 = 11.67 days

- **Pros:** Accounts for risk and variability.
- **Cons:** Requires careful calculation.

4. Parametric Estimation

This method uses mathematical formulas and historical data to estimate effort.

Example:

If a developer takes 10 hours to complete one feature, and the project has 50 features, the total estimate is:

Estimate = 50 features × 10 hours per feature = 500 hours.

- **Pros:** Great for repetitive tasks.
- **Cons:** Assumes uniform complexity, which isn't always realistic.

Top Tools to Streamline Software Project Estimation

Technology makes estimation easier and more accurate. Here are my go-to **tools for an accurate software project estimation**:

1. **Jira**: Perfect for Agile teams. Use it to estimate story points, manage sprints, and track progress.
2. **Trello**: Great for visualizing tasks and timelines in smaller projects.
3. **Microsoft Project**: Ideal for complex projects with multiple dependencies.
4. **Clockify**: A time-tracking tool to gather historical data for future estimates.
5. **Function Point**: A specialized tool for estimating based on function points.

Function Point Calculation Example

This document explains a simple step-by-step example of Function Point (FP) calculation for a Library Management System (LMS).

Step 1: Identify Components and Assign Weights

For the Library Management System (LMS), we identify the following functionalities:

1. Inputs: Add book, Register student, Issue book, Return book → 4 EI (External Inputs)

2. Outputs: Generate report of issued books, Overdue list → 2 EO (External Outputs)

3. Inquiries: Search book, Search student → 2 EQ (External Inquiries)

4. Files: Book database, Student database → 2 ILF (Internal Logical Files)

5. Interfaces: Payment gateway → 1 EIF (External Interface File)

Component	Count	Weight (Average)	Total
EI (Inputs)	4	4	16
EO (Outputs)	2	5	10
EQ (Inquiries)	2	4	8
ILF (Files)	2	7	14
EIF (Interfaces)	1	5	5
Total UFP (Unadjusted Function Points)			53

Step 2: Value Adjustment Factor (VAF)

There are 14 general system characteristics (e.g., reliability, performance, reusability). Each is rated on a scale of 0 to 5. Assume the total score is 30.

Formula: VAF = 0.65 + (0.01 × Total Score)

VAF = 0.65 + (0.01 × 30) = 0.95

Step 3: Final Function Points

Formula: FP = UFP × VAF

FP = 53 × 0.95 = 50.35 ≈ 50

What is LOC?

- LOC = **Number of lines in the source code**.
- Can be counted as:
  - **Physical LOC** → number of non-comment lines.
  - **Logical LOC** → number of executable statements.
- Used for: productivity, defect density, maintenance effort.

LOC Counting

- Total lines in code (including brackets & blank lines): **25**
- Exclude comments/blank lines → **20 physical LOC**
- Logical executable statements → **12 logical LOC**

Interpretation

- **Physical LOC** = 20 → measures size for maintenance.
- **Logical LOC** = 12 → better for productivity & defect density.
- Example usage:
  - If productivity = **10 LOC/hour**, effort = ~2 hours.
  - If defect density = **1 defect per 100 LOC**, expected defects = 0.12.

LOC = direct measure of code size.

Easy to calculate but **language and style dependent**.

Best used for:

- **Defect analysis** (defects per KLOC).
- **Productivity tracking** (LOC per staff-hour).
- **Maintenance estimation**.

LOC = direct measure of code size.

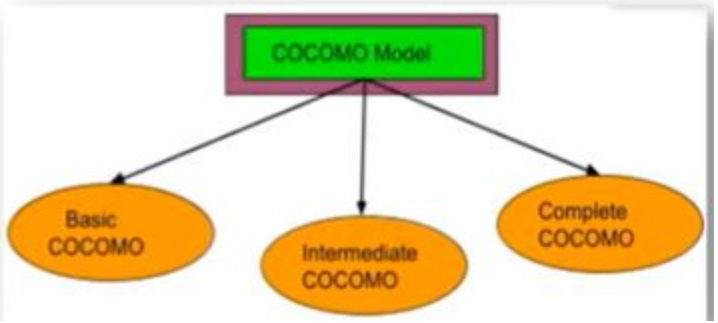
Easy to calculate but **language and style dependent**.

Best used for:

- **Defect analysis** (defects per KLOC).
- **Productivity tracking** (LOC per staff-hour).
- **Maintenance estimation**.

About COCOMO Model

- It was developed by a scientist Barry Boehm in 1981.
- The COCOMO **(Constructive Cost Model)** is one of the most popularly used software cost estimation models.
- This model depends on the size means number of lines of code for software product development.
- It estimates Effort required for project, Total project cost & Scheduled time of project.



Software Project Types

In COCOMO, projects are categorized into three types:

1. Organic Type:

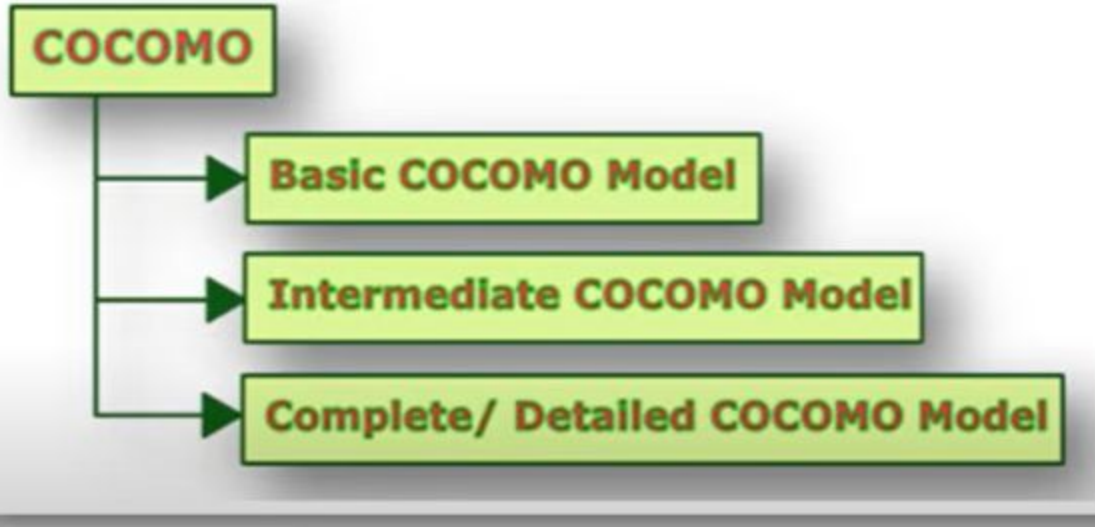
- Project is small and simple. (2-50 KLOC)
- Few Requirements of projects.
- Project team is small with prior experience.
- The problem is well understood and has been solved in the past.
- **Examples:** Simple Inventory Management Systems & Data Processing Systems.

2. Semidetached Type:

- Medium size and has mixed rigid requirements. (50-300 KLOC)
- Project has Complexity not too high & low.
- Both experienced and inexperienced members in Project Team.
- Project are few known and few unknown modules.
- **Examples:** Database Management System, Difficult inventory management system.

3. Embedded Type:

- Large project with fixed requirements of resources. (More then 300 KLOC)
- Larger team size with little previous experience.
- Highest level of complexity, creativity and experience requirement.
- **Examples:** ATM, Air Traffic control, Banking software.


COCOMO
Basic COCOMO Model
Intermediate COCOMO Model
Complete/ Detailed COCOMO Model

Type 1: Basic COCOMO Model

- It is type of static model to estimates software development effort quickly and roughly.
- It mainly deals with the number of lines of code in project.

Formula:

Effort (E) = a*(KLOC)b MM

Scheduled Time (D) = c*(E)d Months(M)

PROJECT TYPE	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Where,

- **E** = Total effort required for the project in Man-Months (MM).
- **D** = Total time required for project development in Months (M).
- **KLOC** = The size of the code for the project in Kilo lines of code.
- **a, b, c, d** = The constant parameters for a software project.

Mode	When to choose	a	b	c	d
Organic	Small teams, familiar domains, flexible requirements	2.4	1.05	2.5	0.38
Semi-detached	Mixed experience/constraints; some unfamiliarity	3.0	1.12	2.5	0.35
Embedded	Tight HW/SW/operational constraints; high criticality	3.6	1.20	2.5	0.32

equations:

- Effort (person-months): $E = a \times (KLOC)^b$
- Development time (months): $T = c \times (E)^d$
- Average staffing (persons): $P = E / T$
- Productivity: Prod = KLOC / E (KLOC per person-month)

Type 1: Basic COCOMO Model Example

❑Problem Statement:

➢Consider a software project using semi-detached mode with 300 Kloc.

Find out Effort estimation, Development time and Person estimation.

❑Solution:

Effort (E) = **a*(KLOC)b** = 3.0*(300)1.12 = 1784.42 PM

Development Time (D) = **c(E)d** = 2.5(1784.42)0.35 = 34.35 Months(M)

Person Required (P) = **E/D** = 1784.42/34.35 = 51.9481 Persons ~52 Persons

Example 1 — Organic project, 30 KLOC

Given: Mode = Organic, Size = 30 KLOC

Step 1: Effort E = 2.4 × (30)^1.05 = 85.347 PM

Step 2: Time T = 2.5 × (85.347)^0.38 = 13.55 months

Step 3: Avg staffing P = E / T = 6.30 persons

Step 4: Productivity = KLOC / E = 0.352 KLOC/PM

Example 2 — Embedded project, 100 KLOC

Given: Mode = Embedded, Size = 100 KLOC

Step 1: Effort E = 3.6 × (100)^1.20 = 904.279 PM

Step 2: Time T = 2.5 × (904.279)^0.32 = 22.08 months

Step 3: Avg staffing P = E / T = 40.96 persons

Step 4: Productivity = KLOC / E = 0.111 KLOC/PM

Type 2: Intermediate COCOMO Model

- Extension of Basic COCOMO model which enhance more accuracy to cost estimation model result.
- It include cost drivers (Product, Hardware, Resource & project Parameter) of project.

Formula:

Effort (E) = $a*(KLOC)^b * EAF$ MM

Scheduled Time (D) = $c*(E)^d$ Months(M)

PROJECT TYPE	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Where,

- **E** = Total effort required for the project in Man-Months (MM).
- **D** = Total time required for project development in Months (M).
- **KLOC** = The size of the code for the project in Kilo lines of code.
- **a, b, c, d** = The constant parameters for the software project.

- **EAF:** Effort Adjustment Factor, which is calculated by multiplying the parameter values o different cost driver parameters. For ideal, the value is 1.

COST DRIVERS PARAMETERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Parameter					
Required Software	0.75	0.88	1	1.15	1.4
Size of Project Database	NA	0.94		1.08	1.16
Complexity of The Project	0.7	0.85		1.15	1.3
Hardware Parameter					
Performance Restriction	NA	NA	1	1.11	1.3
Memory Restriction	NA	NA		1.06	1.21
virtual Machine Environment	NA	0.87		1.15	1.3
Required Turnabout Time	NA	0.94		1.07	1.15

Personnel Parameter					
Analysis Capability	1.46	1.19	1	0.86	0.71
Application Experience	1.29	1.13		0.91	0.82
Software Engineer Capability	1.42	1.17		0.86	0.7
Virtual Machine Experience	1.21	1.1		0.9	NA
Programming Experience	1.14	1.07		0.95	NA

Project Parameter					
Software Engineering Methods	1.24	1.1	1	0.91	0.82
Use of Software Tools	1.24	1.1		0.91	0.83
Development Time	1.23	1.08		1.04	1.1

Cost Driver Parameters and Their Multipliers

1. Product Parameters

Parameter	Very Low	Low	Nominal	High	Very High
Required Software	0.75	0.88	1.00	1.15	1.40
Size of Project Database	N/A	0.94	1.08	1.16	1.30
Complexity of Project	0.75	0.88	1.00	1.15	1.30

2. Hardware Parameters

Parameter	Very Low	Low	Nominal	High	Very Hig
Performance Restriction	N/A	N/A	1.00	1.11	1.30
Memory Restriction	N/A	N/A	1.00	1.06	1.21
Virtual Machine Env.	0.87	0.94	1.00	1.10	1.30
Required Turnaround Time	N/A	0.94	1.00	1.07	1.15

3. Personnel Parameters

Parameter	Very Low	Low	Nominal	High	Very High
Analysis Capability	1.46	1.19	1.00	0.86	0.71
Application Experience	1.29	1.13	1.00	0.91	0.82
Software Engineer Capability	1.42	1.17	1.00	0.86	0.70
Virtual Machine Experience	1.21	1.10	1.00	0.90	N/A
Programming Experience	1.14	1.07	1.00	0.95	N/A

4. Project Parameters

Parameter	Very Low	Low	Nominal	High	Very High
Software Engineering Methods	1.24	1.10	1.00	0.91	0.82
Use of Software Tools	1.24	1.10	1.00	0.91	0.83
Development Time	1.23	1.08	1.00	1.04	1.10

❑Problem Statement:

- For a given Semidetached project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development by considering developer having high application experience and very low experience in programming.

PROJECT TYPE	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

❑Solution:

EAF = 0.82 *1.14 = 0.9348

Effort (E) = $a*(KLOC)^b * EAF = 3.0*(300)^{1.12} *0.9348 = 1668.07$ MM

Scheduled Time (D) = $c*(E)^d = 2.5*(1668.07)^{0.35} = 33.55$ Months(M)

COST DRIVERS PARAMETERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Personnel Parameter					
Analysis Capability	1.46	1.19	1	0.86	0.71
Application Experience	1.29	1.13		0.91	0.82
Software Engineer Capability	1.42	1.17		0.86	0.7
Virtual Machine Experience	1.21	1.1		0.9	NA
Programming Experience	1.14	1.07		0.95	NA

- The model incorporates all qualities of both Basic COCOMO and Intermediate COCOMO strategies on each software engineering process.
- The whole software is divided into different modules and then apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

❑Problem Statement:

A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- Database part
- Graphical User Interface (GUI) part
- Communication part

❑Solution:

✓The communication part can be considered as Embedded software.

✓The database part could be Semi-detached software,

✓The GUI part Organic software.

The costs for these three components can be estimated separately and summed up to give the overall cost of the system.

Advantages

1. Provides a systematic way to estimate the cost and effort of a software project.
2. Estimate cost and effort of software project at different stages of the development process.
3. Helps in identifying the factors that have the greatest impact on the cost and effort of a software project.
4. Provide ideas about historical projects.
5. Easy to implement with various factors.

Disadvantages

1. It ignores requirements, customer skills and hardware issues.
2. It limits the accuracy of the software costs.
3. It is based on assumptions and averages.
4. It mostly depends on time factors.
5. Assumes that the size of the software is the main factor that determines the cost and effort of a software project, which may not always be the case.