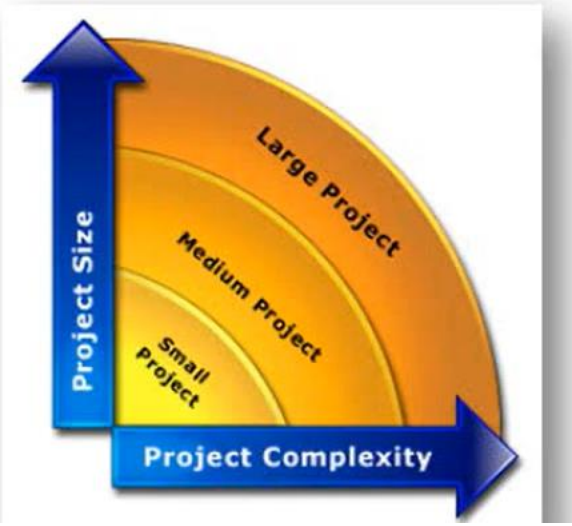
Estimation for Software Projects

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

Approaches of Software Sizing

Putnam and Myers suggest four different approaches of Software Sizing Problem:

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.

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.

2. 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

Examples of LOC & FP

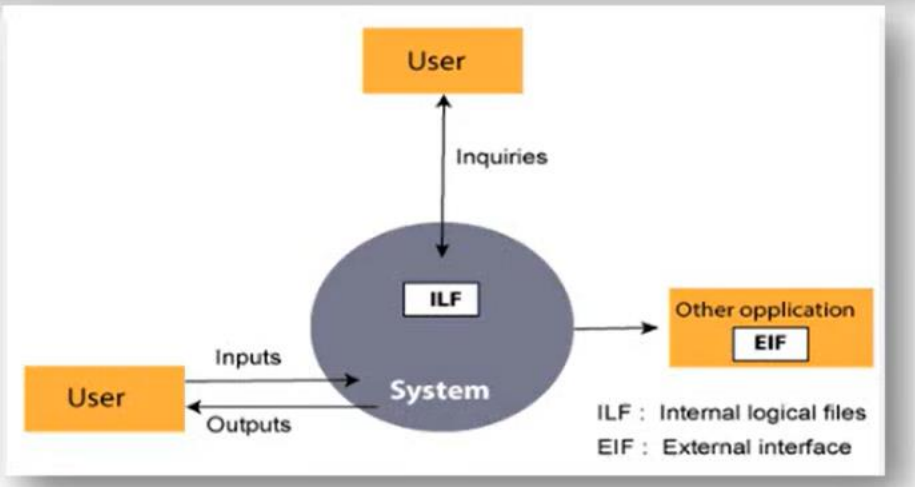
1. LOC (Lines of Code):

Project	LOC	Cost $K	Efforts (Persons/ Month)	Documentation	Errors
A	10000	110	18	365	39
B	12000	115	20	370	45
C	15400	130	25	400	32

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*

2. FP(Functional Points)

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						*320*



Size Metrics : LOC

- Size-Oriented Metrics concentrates on the size of the program created.

1. LOC (Lines of Code):

- It is one of the earliest and simpler metrics for calculating the size of the computer program.
- It is generally used in calculating and comparing the productivity of programmers.
- LOC not count comment or blank line in code.

Example:

Total Line of Code (LOC) = 09

```
//Import header file
#include <iostream>
int main () {
  int num = 10;
//Logic of even number
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
  return 0;
}
```

Size Metrics : LOC

Advantages:

1. Most used metric in cost estimation.
2. It is very easy in estimating the efforts.

Disadvantages:

1. It cannot measure the size of the specification.
2. Bad software design may cause an excessive line of code
3. It is language dependent
4. Users cannot easily understand it

Project	LOC	Cost SK	Efforts (Persons/ Month)	Documentation	Errors
A	10000	110	18	365	39
B	12000	115	20	370	45
C	15400	130	25	400	32

LOC Table

Refer problems given in class notes

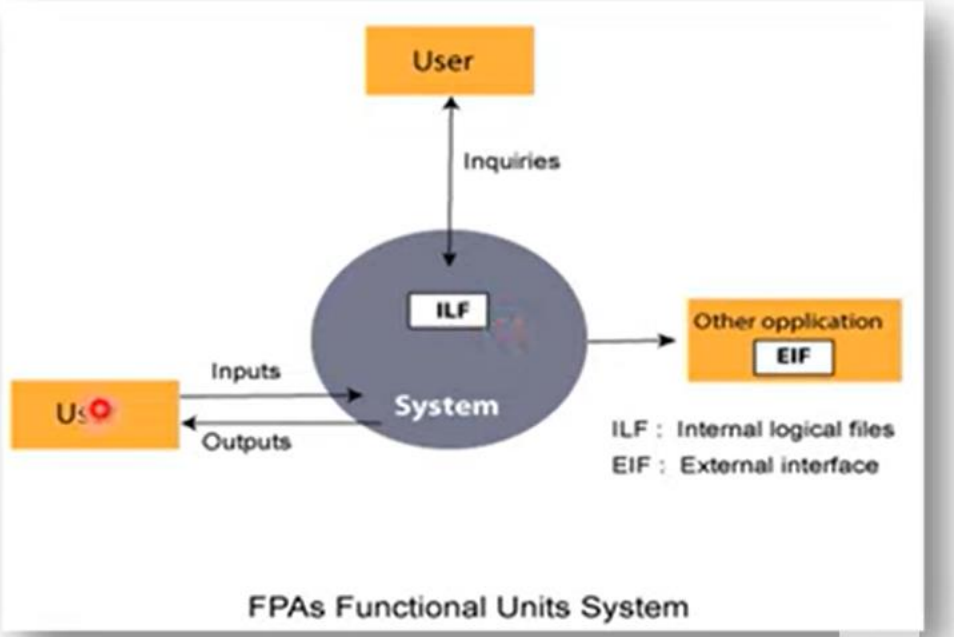
Size Metrics : FP

2. FP (Functional Point):

- Function-oriented software metrics measure functionality, what the system performs, is the measure of the system size.
- It find out by counting the number and types of functions used in the applications

FP Attributes

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.



FPAs Functional Units System

Size Metrics : FP

Advantages:

1. Need detail specification.
2. Not restricted to code.
3. Language Interdependent.

Disadvantages:

1. Ignore quality issues.
2. Subjective counting depend on estimation.

	A	B	C	D	E	F	G
1	Info Domain	Optimistic	Likely	Pessim.	Est Count	Weight	FP count
2	# of inputs	22	26	30	26	4	104
3	# of outputs	16	18	20	18	5	90
4	# of inquiries	16	21	26	21	4	84
5	# of files	4	5	6	5	10	50
6	# of external interf	1	2	3	2	7	14
7	UFC: Unadjusted Function Count						342
8			Complexity adjustment factor				1.17
9						FP	400

- The function point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system.
- Using historical data, the FP metric can then be used to

  (1) estimate the cost or effort required to design, code, and test the software;

  (2) predict the number of errors that will be encountered during testing; and

  (3) forecast the number of components and/or the number of projected source lines in the implemented system

- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity

- Information domain values are defi ned in the following manner:

Number of external inputs (EIs). Each *external input* originates from a user or is transmitted from another application and provides distinct application-oriented data or control information. Inputs are often used to update *internal logical files* (ILFs). Inputs should be distinguished from inquiries, which are counted separately.

Number of external outputs (EOs). Each *external output* is derived data within the application that provides information to the user. In this context external output refers to reports, screens, error messages, and the like. Individual data items within a report are not counted separately.

- **Number of external inquiries (EQs).** An external inquiry is defi ned as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF).

Number of internal logical files (ILFs). Each *internal logical file* is a logical grouping of data that resides within the application's boundary and is maintained via external inputs.

Number of external interface files (EIFs). Each *external interface file* is a logical grouping of data that resides external to the application but provides data that may be of use to the application.

FIGURE 30.1

Computing function points

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

Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)] \quad (30.1)$$

where count total is the sum of all FP entries obtained from Figure 30.1.

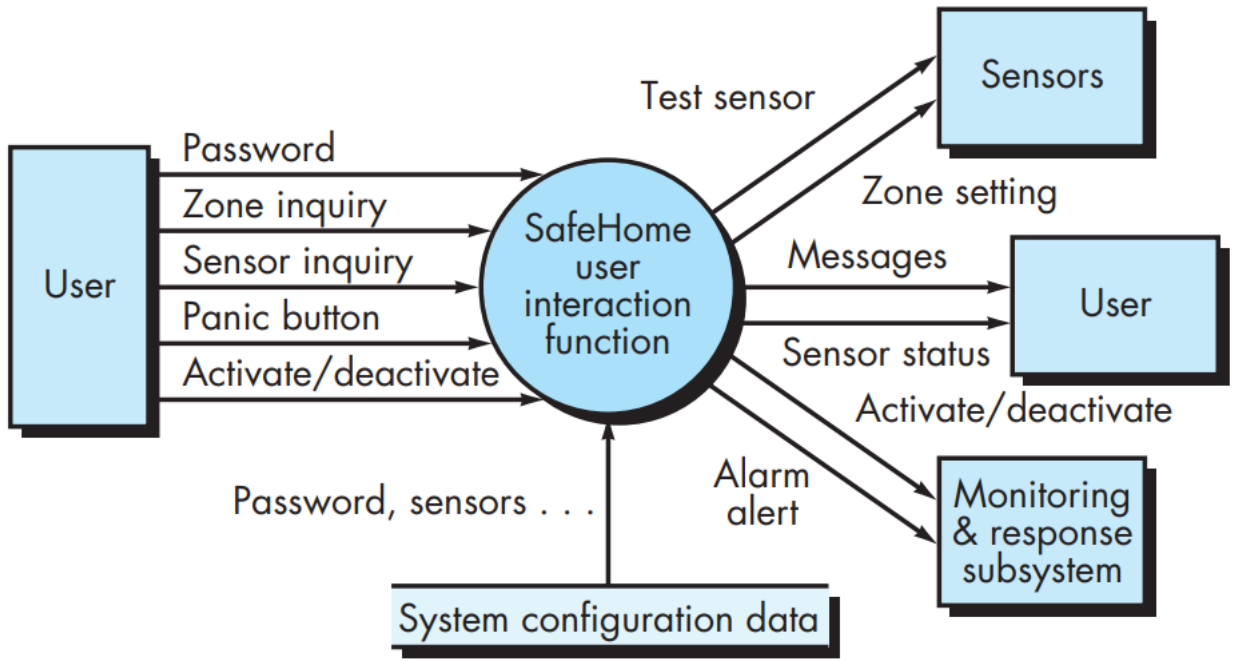
The F_i (i = 1 to 14) are *value adjustment factors* (VAF) based on responses to the following questions [Lon02]:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?

9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Each of these questions is answered using an ordinal scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). The constant values in Equation (30.1) and the weighting factors that are applied to information domain counts are determined empirically.

0 - No Influences or No Important
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential


Sensors
Test sensor
Zone setting
Password
Zone inquiry
Sensor inquiry
Panic button
Activate/deactivate
User
SafeHome user interaction function
Messages
Sensor status
User
Activate/deactivate
Alarm alert
Monitoring & response subsystem
Password, sensors . . .
System configuration data

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	3	3	4	6	=	9
External Outputs (EOs)	2	3	4	5	7	=	8
External Inquiries (EQs)	2	3	3	4	6	=	6
Internal Logical Files (ILFs)	1	3	7	10	15	=	7
External Interface Files (EIFs)	4	3	5	7	10	=	20
Count total							50

The count total shown must be adjusted using Equation (30.1). For the purposes of this example, we assume that S (Fi) is 46 (a moderately complex product). Therefore

$$\mathrm{FP} = 50 \times [0.65 + (0.01 \times 46)] = 56$$

Given the Following Values, calculate the Functional Point when complexity adjustment factors are significantly complex product and weighting factors are high.

User input = 55

User Output = 35

User Enquires = 40

User Files = 8

External Interfaces = 5

$$F.P = UFP \ X \ CAF$$

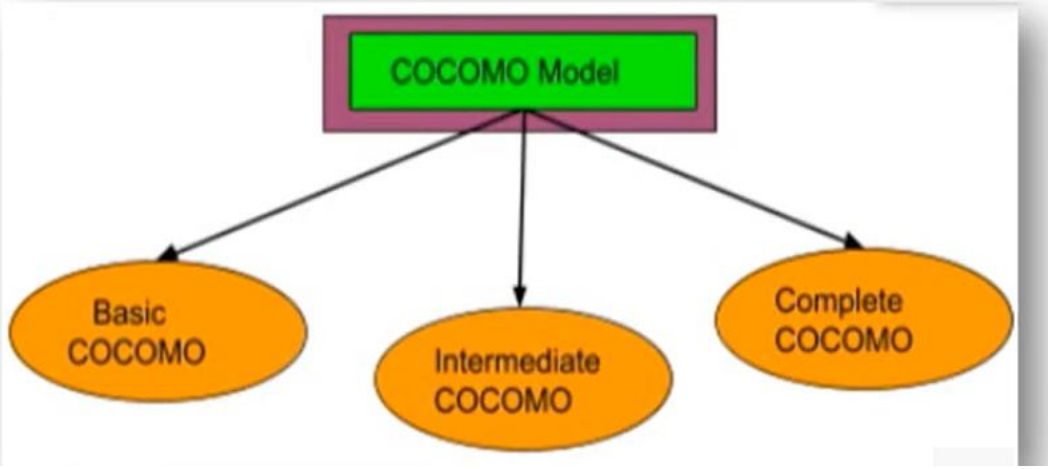
Where,

UFP = Sum of all the Complexities of all the EI's, EO's, EQ's, ILF's, and EIF's

$CAF = 0.65 + (0.01 \times \Sigma Fi)$

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.

Software Project Types

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.

Types of 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.

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

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

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.

Type 2: Intermediate COCOMO Model

- **EAF:** Effort Adjustment Factor, which is calculated by multiplying the parameter values of 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

Type 2: Intermediate COCOMO Model Example

❑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.

❑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)

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

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

Type 3: Detailed / Complete COCOMO Model

- 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

Type 3: Detailed COCOMO Model Example

❑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 of COCOMO Model

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 of COCOMO Model

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.