Software engineering *assignment 2 *

Ques 1: Describe any two software size estimation techniques.

Size estimation in software engineering is a crucial process that involves estimating the size of a software project before development begins. It provides insights into resource requirements, project duration, cost, and other important factors. Two common approaches for size estimation are Line of Code (LOC) estimation and Function Point (FP) analysis.

1. **Line of Code (LOC) Estimation**:
   LOC estimation involves predicting the size of a software project based on the number of lines of code that will be written. However, this method has limitations, as the number of lines can vary greatly depending on coding style, programming language, and other factors.

   Eg:-
   ```
   int main(){
           cout << "hello world"<<endl;
           return 0;
   }
   ```
   LOC is 4 in this example.

2. **Function Point (FP) Analysis**:
   FP analysis measures the functionality delivered by the software. It quantifies user interactions, data manipulation, and processing complexity, providing a more comprehensive measure of software size than LOC.

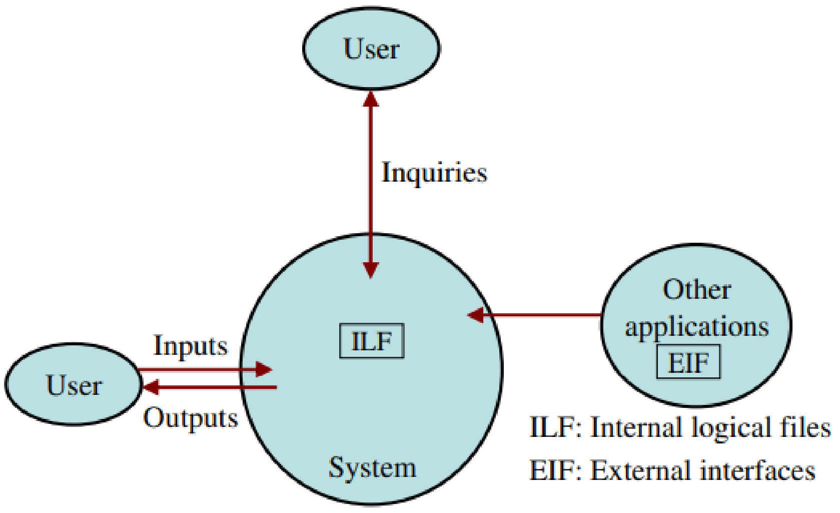


Fig. 3: FPAs functional units System

Steps for Function Point Analysis:

- **Identify Function Types**:
  - External Inputs (EI): User inputs that affect processing.
  - External Outputs (EO): Outputs generated for the user.

        - External Inquiries (EQ): User interactions that result in queries or data retrieval.
    - **Count Complexity Factors**:
        - Determine complexity factors (low, average, high) for each function type based on factors like data processing and user interaction.
    - **Calculate Unadjusted Function Points**:
        - Sum the weighted counts of each function type to get the Unadjusted Function Points (UFP).
    - **Apply Complexity Adjustment Factors**:
        - Apply complexity adjustment factors to UFP based on the complexity of the project.
    - **Calculate Adjusted Function Points (AFP)**:
        - Adjusted Function Points (AFP) = UFP * Complexity Adjustment Factor.

Function Point Analysis provides a more accurate estimate of software size by considering both the functionality delivered and the complexity involved. It is often used in cost estimation models and helps in planning resources, scheduling, and managing software projects. While both LOC estimation and Function Point Analysis have their uses, Function Point Analysis is generally considered more reliable and accurate, especially for larger and more complex projects.

Ques 2: what are size metrics? How is function point metric advantageous over loc metric explain ?

Size metrics in software engineering are used to quantify the size of a software system or its components. The most common size metrics include:

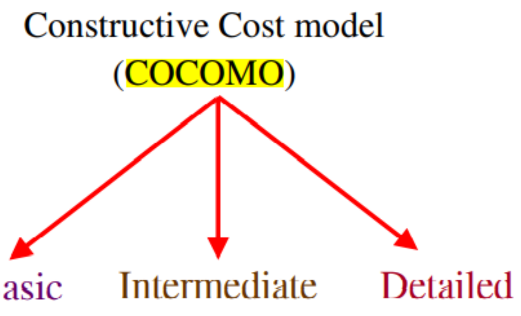
1. **Line of Code (KLOC)**: This measures the size of a software project by counting the number of thousands of lines in the source code.
2. **Function Points (FP)**: This measures the functionality delivered by the software by evaluating various aspects such as inputs, outputs, user interactions, and processing complexity.

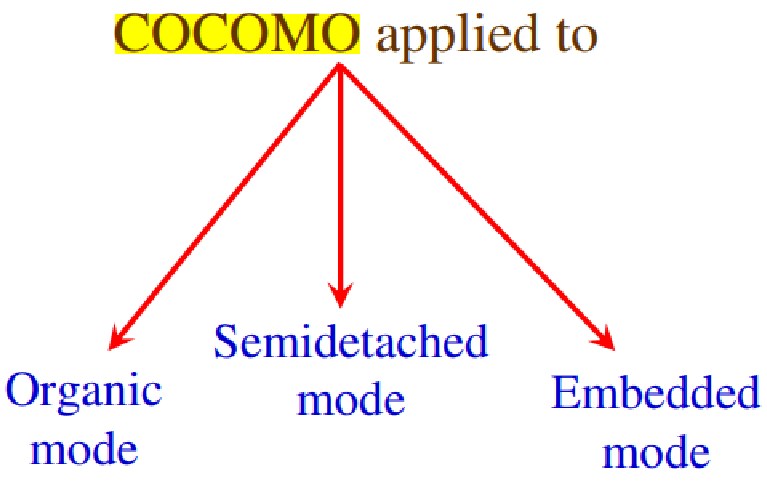
The main advantage of function points over the size metric of KLOC, the other commonly used approach, is that the definition of AFP depends only on information available from the specifications, whereas the size in KLOC cannot be directly determined from specifications. Furthermore, the AFP count is independent of the language in which the project is implemented. Though these are major advantages, another drawback of the function point approach is that even when the project is finished, the AFP is not uniquely known and has subjectivity. This makes building of models for cost estimation hard, as these models are based on information about completed projects (cost models are discussed further in the next chapter). In addition, determining the AFP—from either the requirements or a completed project—cannot be automated. That is, considerable effort is required to obtain the size, even for a completed project. This is a drawback compared to KLOC measure, as KLOC can be determined uniquely by automated tools once the project is completed.

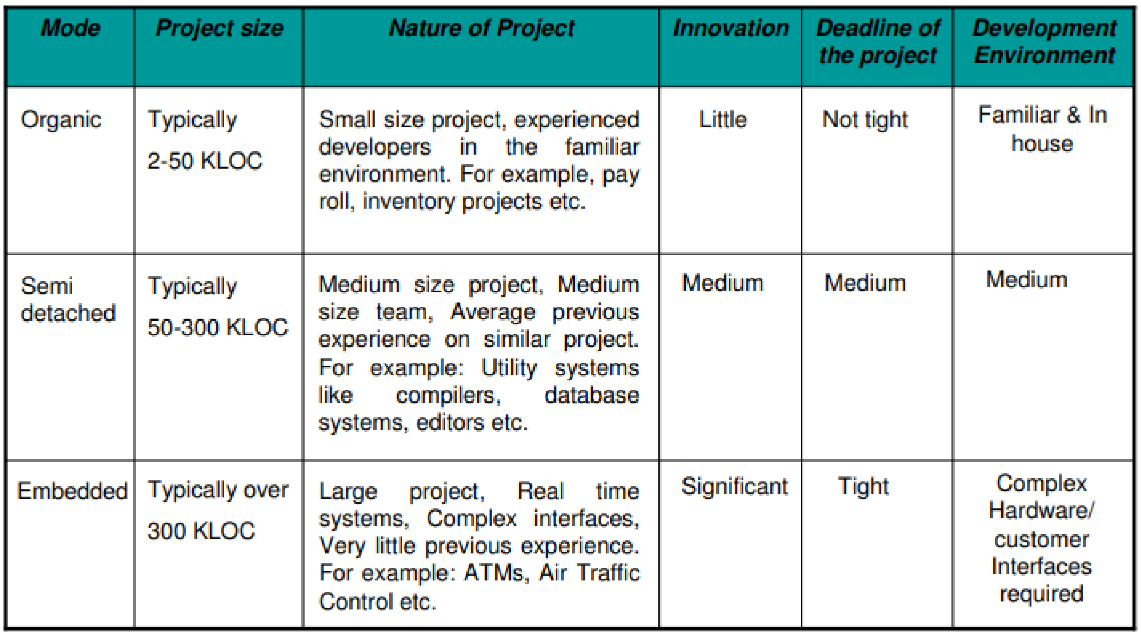
Ques 3 : describe various types of cococo model

COCOMO (COnstructive COst MOdel): COCOMO is one of the earliest and most widely used cost estimation models. It was introduced by Barry Boehm in the 1980s and has evolved into different versions:

The Constructive Cost Model (COCOMO)



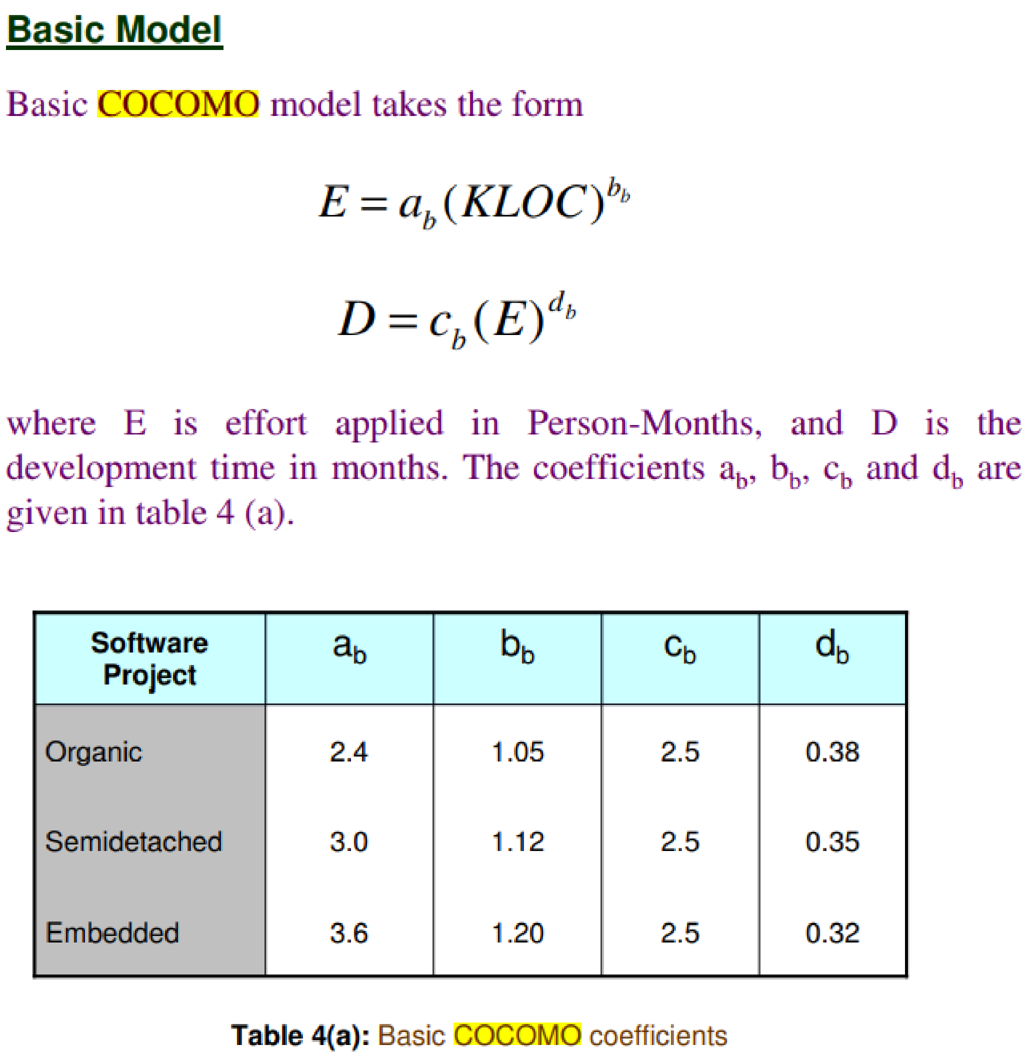




Mode	*Project size*	*Nature of Project*	*Innovation*	*Deadline of the project*	*Development Environment*
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

Table 4: The comparison of three COCOMO modes

Basic COCOMO is the simplest version of the model and estimates effort and duration based on the size of the project. It is suitable for small to medium-sized projects with straightforward development processes. The formula for estimating effort (E) in personmonths is:



Basic Model

Basic COCOMO model takes the form

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in table 4 (a).

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

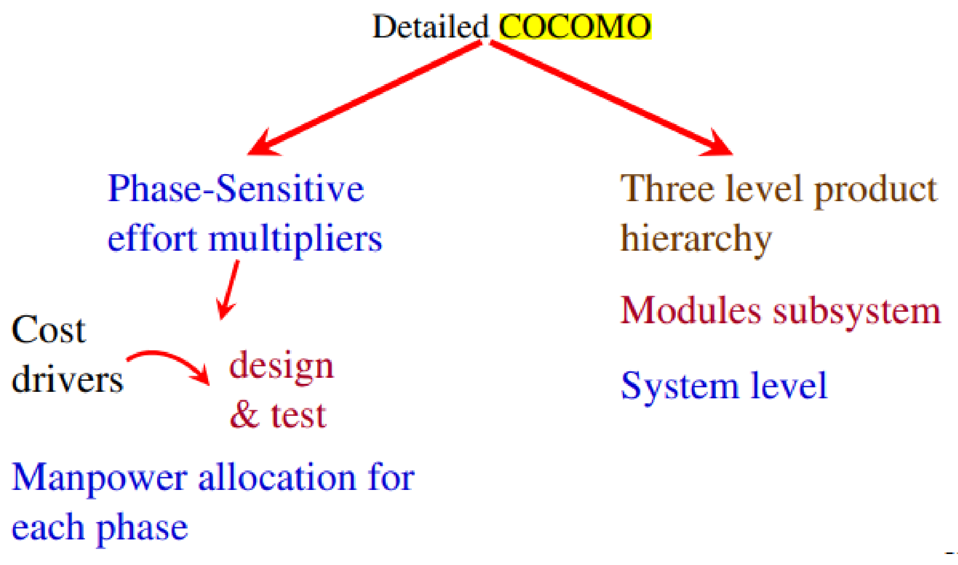
Table 4(a): Basic COCOMO coefficients

Intermediate COCOMO: Intermediate COCOMO builds upon the basic model by adding a set of 15 cost drivers that account for additional project characteristics and factors affecting effort and cost. These factors include team experience, development environment, complexity, and more. The effort estimation formula becomes:

E = a * (Size) ^ b * EAF Where EAF

(Effort Adjustment Factor) is calculated based on the 15 cost drivers.

Detailed COCOMO Model



Detailed COCOMO
Phase-Sensitive effort multipliers
Three level product hierarchy
Cost drivers
design & test
Modules subsystem
System level
Manpower allocation for each phase

Detailed COCOMO (COCOMO II) further refines the model by introducing a set of 17 cost drivers that provide even more detailed estimation. It takes into account factors such as personnel attributes, reuse, and documentation. The effort estimation formula remains similar to Intermediate COCOMO, but with more refined EAF calculation

Ques4 explain cocomo -2 model in detail .what type of categories of project are identifiable?

COCOMO-II

The following categories of applications / projects are identified by COCOMO-II and are shown in fig. 4 shown below:

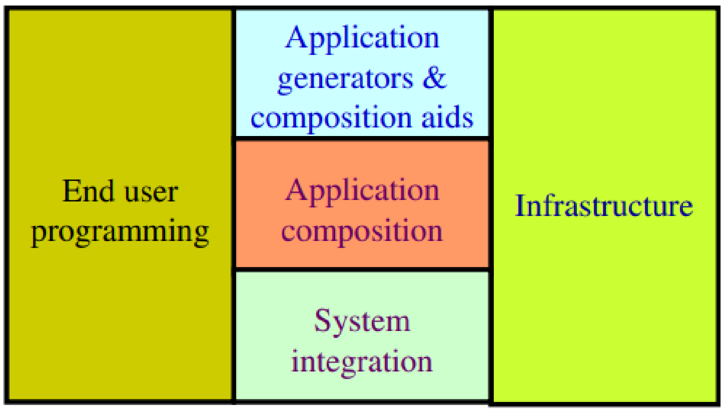
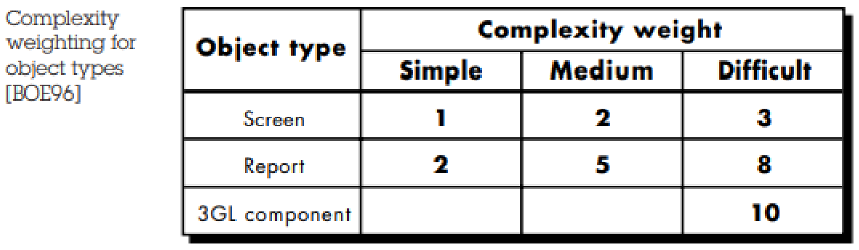


Fig. 4 : Categories of applications / projects

COCOMO has evolved into a more comprehensive estimation model called **COCOMO II**. Like its predecessor, COCOMO II is a hierarchy of estimation models that address different stages of the software development process. These models include:

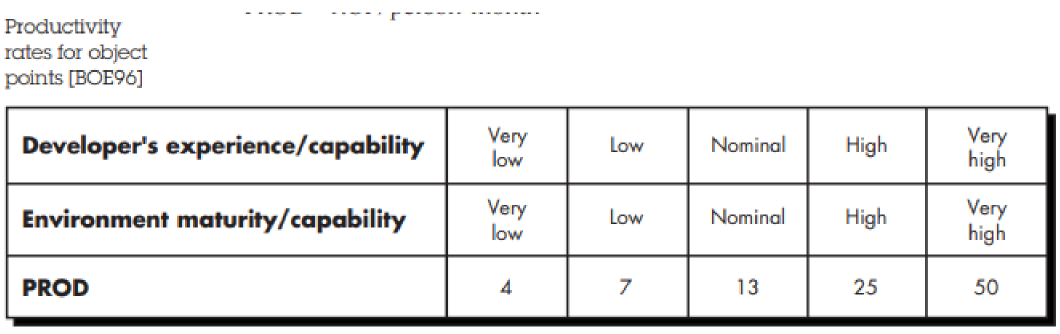
Stage No	Model Name	Application for the types of projects	Applications
Stage I	Application composition estimation model	Application composition	In addition to application composition type of projects, this model is also used for prototyping (if any) stage of application generators, infrastructure & system integration.
Stage II	Early design estimation model	Application generators, infrastructure & system integration	Used in early design stage of a project, when less is known about the project.
Stage III	Post architecture estimation model	Application generators, infrastructure & system integration	Used after the completion of the detailed architecture of the project.

Table 8: Stages of COCOMO-II



Complexity weighting for object types [BOE96]

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10



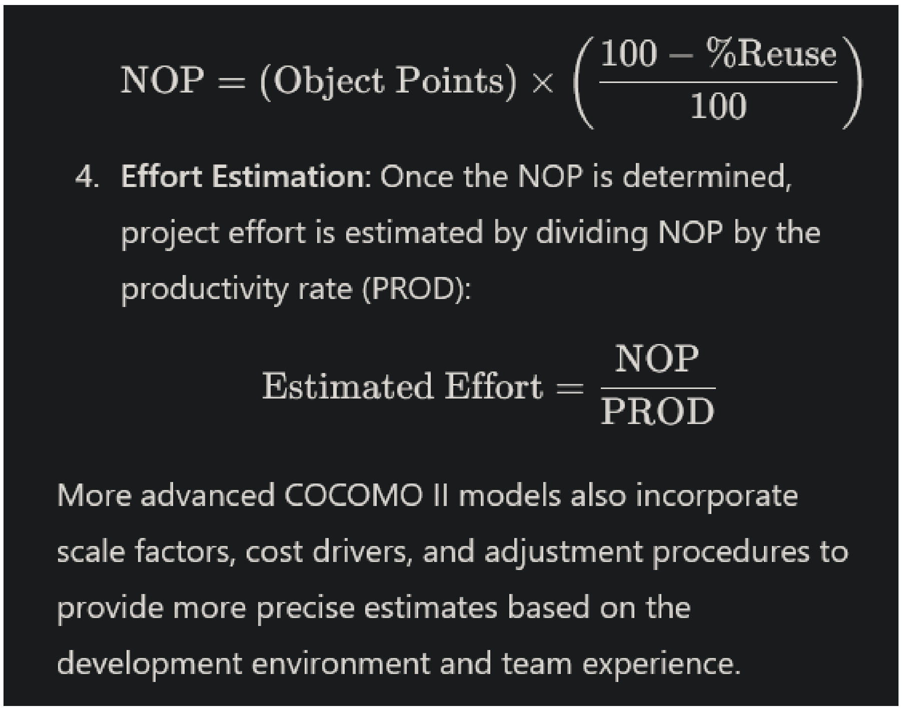
Productivity rates for object points [BOE96]

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Like all estimation models for software, the COCOMO II models require sizing information. Three sizing options are available: object points, function points (FP), and lines of source code (KLOC).

1. **Object Points**: The **COCOMO II application composition model** uses object points for estimation. Object points are an indirect measure of software size, calculated by counting the number of user interface screens, reports, and components. These object instances are classified as simple, medium, or difficult based on their complexity, which is determined by the number of data tables and sections involved in generating them.

2. **Complexity and Weights**: The complexity of each object instance is evaluated according to criteria defined by Boehm. Based on its complexity, each screen, report, or component is weighted according to predefined factors, as shown in a weighting table.
3. **New Object Points (NOP)**: To adjust for software reuse, the object point count is modified using the formula:



$$\text{NOP} = (\text{Object Points}) \times \left(\frac{100 - \%\text{Reuse}}{100}\right)$$

4. **Effort Estimation**: Once the NOP is determined, project effort is estimated by dividing NOP by the productivity rate (PROD):

$$\text{Estimated Effort} = \frac{\text{NOP}}{\text{PROD}}$$

More advanced COCOMO II models also incorporate scale factors, cost drivers, and adjustment procedures to provide more precise estimates based on the development environment and team experience.

The Early Design Model

The COCOMO-II models use the base equation of the form

$$PM_{nominal} = A * (size)^{B}$$

where

$PM_{nominal}$ = Effort of the project in person months

A = Constant representing the nominal productivity, provisionally set to 2.5

B = Scale factor

Size = Software size