Software Engineering

COCOMO

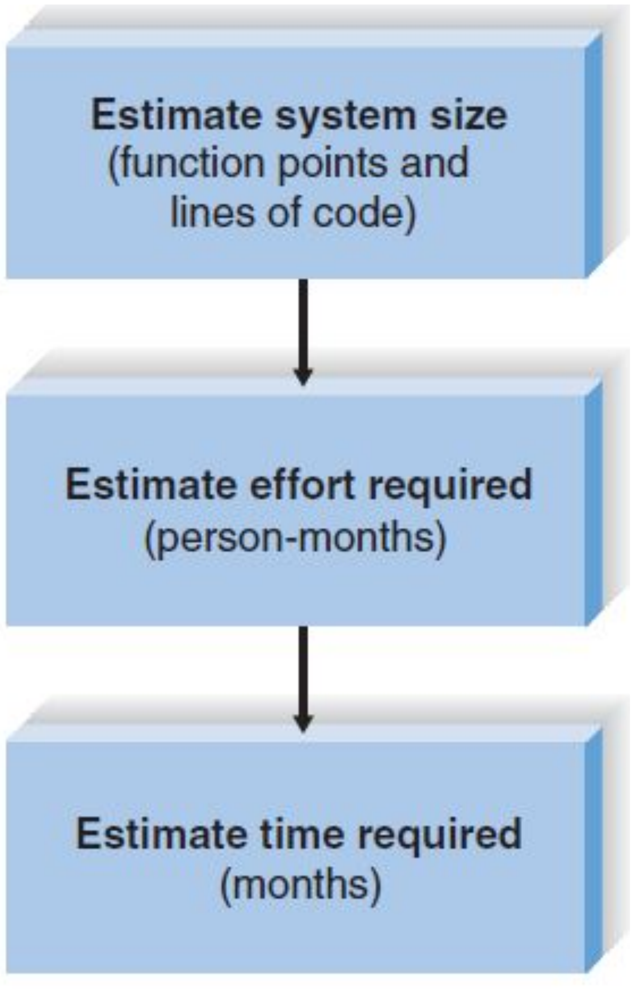
CONSTRUCTIVE

COST

MODEL

The function point approach

- Used to
  - Estimate the size of the new system
  - The effort that will be required to complete the system
  - The time the project will require
- It is a three-step process.



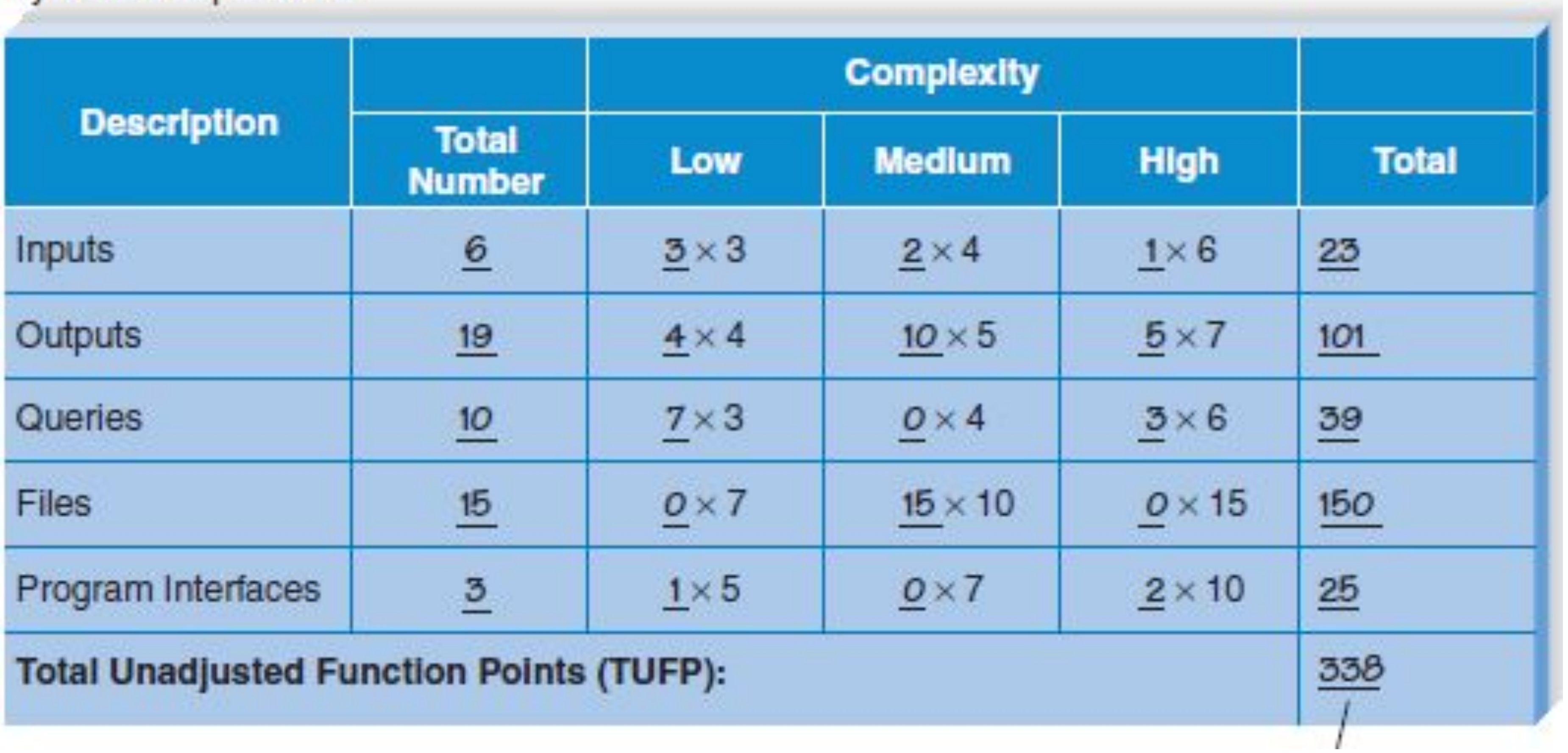
Step 1: Estimate System Size

- A function point is a measure of program size that is based on the system’s number and complexity of inputs, outputs, queries, files, and program interfaces.

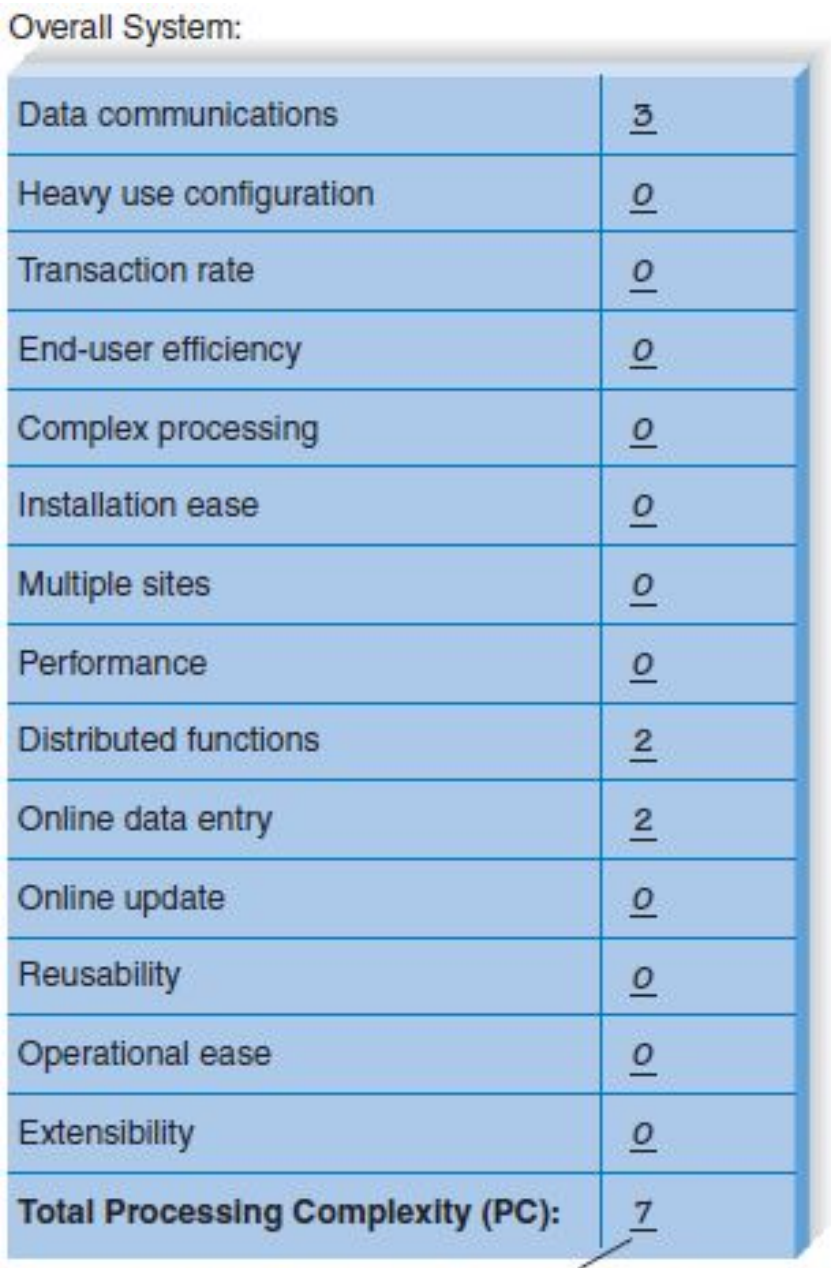
- The project manager records the total number of each component that the system will include, and then breaks down the number to show the number of components that have low, medium, and high complexity.

- To create a more realistic size for the project, a number of additional system factors such as end-user efficiency, reusability, and data communications are assessed in terms of their effect on the project’s complexity.

System Components:



Description		Complexity			
	Total Number	Low	Medium	High	Total
Inputs	6	3 × 3	2 × 4	1 × 6	23
Outputs	19	4 × 4	10 × 5	5 × 7	101
Queries	10	7 × 3	0 × 4	3 × 6	39
Files	15	0 × 7	15 × 10	0 × 15	150
Program Interfaces	3	1 × 5	0 × 7	2 × 10	25
Total Unadjusted Function Points (TUFP):					338



Overall System:

Data communications	3
Heavy use configuration	0
Transaction rate	0
End-user efficiency	0
Complex processing	0
Installation ease	0
Multiple sites	0
Performance	0
Distributed functions	2
Online data entry	2
Online update	0
Reusability	0
Operational ease	0
Extensibility	0
Total Processing Complexity (PC):	7

(0 = no effect on processing complexity; 3 = great effect on processing complexity)

APC factor has a baseline value of 0.65

Adjusted Project Complexity (APC):
.65 + (0.01 x 7) = **.72**

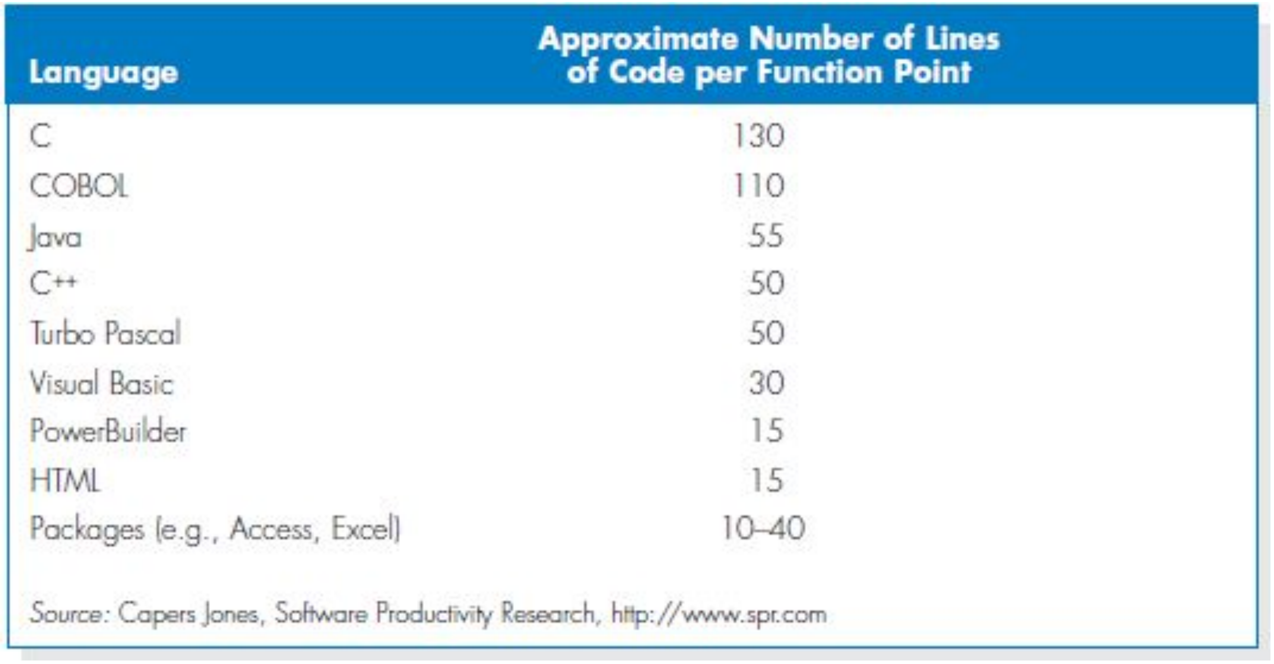
Total Adjusted Function Points (TAFP):
.72 (APC) x **338** (TUFP) = **243 (TAFP)**

Adjusted Project Complexity

- APC value that ranges from 0.65 for very simple systems to 1.00 for “normal” systems to as much as 1.35 for complex systems.
    - A very simple system that has 200 unadjusted function points would have a size of 130 adjusted function points (200 * .65 = 130).
    - If the system with 200 unadjusted function points were very complex, its function point size would be 270 (200 * 1.35 = 270).
- In the planning phase, the exact nature of the system has not yet been determined, so it is impossible to know exactly how many inputs, outputs, and so forth will be in the system. It is up to the project manager to make an intelligent guess.

Lines of code

Convert the number of function points into the lines of code that will be required to build the system.



Language	Approximate Number of Lines of Code per Function Point
C	130
COBOL	110
Java	55
C++	50
Turbo Pascal	50
Visual Basic	30
PowerBuilder	15
HTML	15
Packages (e.g., Access, Excel)	10–40

Source: Capers Jones, Software Productivity Research, http://www.spr.com

243 function points.

COBOL require approximately 26,730 lines of code to write it.

Visual Basic take 7290 lines of code.

Estimating Staff and Project size

COCOMO (Constructive Cost Model) was proposed by Boehm. This model estimates the total *effort* in terms of "person-months" of the technical project staff.

Boehm introduces three forms of COCOMO. It can be applied in three classes of software project:

Organic mode : Relatively simple , small projects with a small team are handled . Such a team should have good application experience to less rigid requirements.

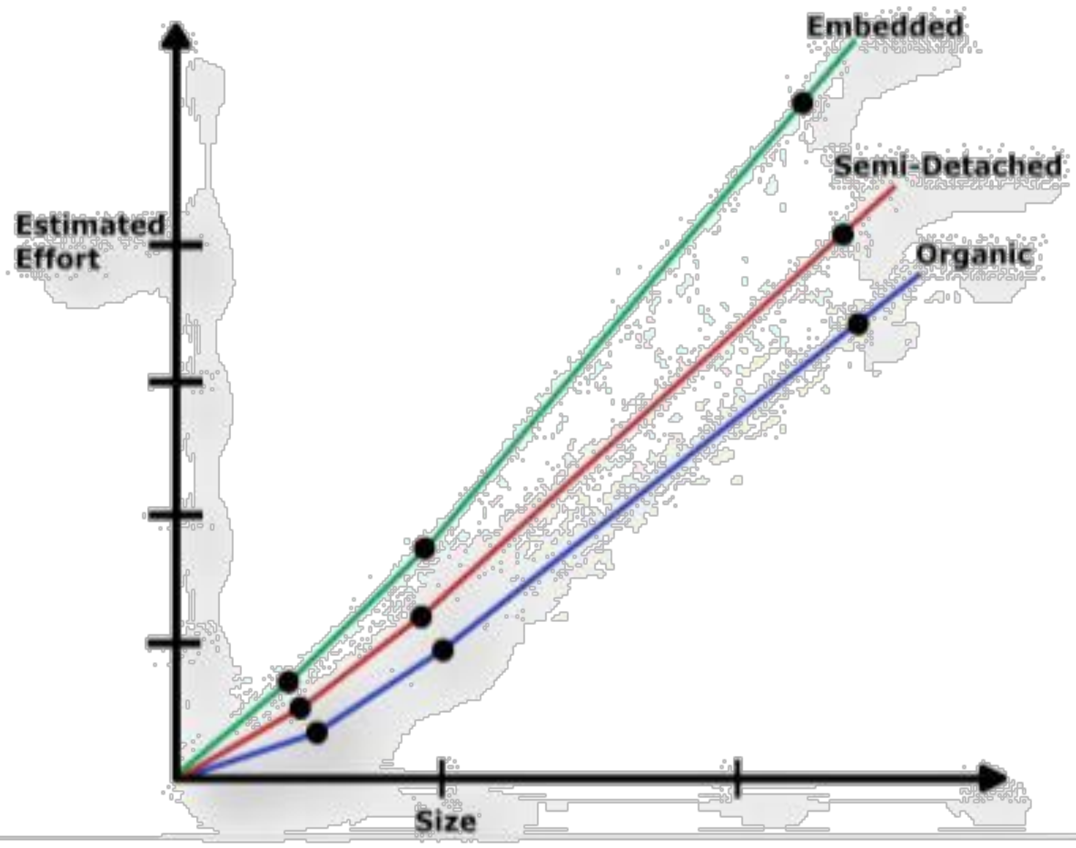
Semidetached mode: For intermediate software projects(little complex compared to organic mode projects in terms of size). Projects may have a mix of rigid and less than rigid requirements.

Embedded mode: When the software project must be developed within a tight set of hardware and software operational constraints. Ex of complex project: Air traffic control system

DEVELOPMENT MODE WITH PROJECT CHARACTERISTICS:

Development Mode	**Project Characteristics**			
	Size	**Innovation**	**Deadline**	**Dev. Environment**
ORGANIC	Small	Little	Not Tight	Stable
SEMI-DITACHED	Medium	Medium	Medium	Medium
EMBEDDED	Large	Greater	Tight	Complex Hardware

From the following figure which shows a plot of **estimated effort versus product size**. We can observe that the effort is somewhat superlinear in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Now the following figure plots the **development time versus the product size in KLOC** can be observed that the development time is a sublinear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately.

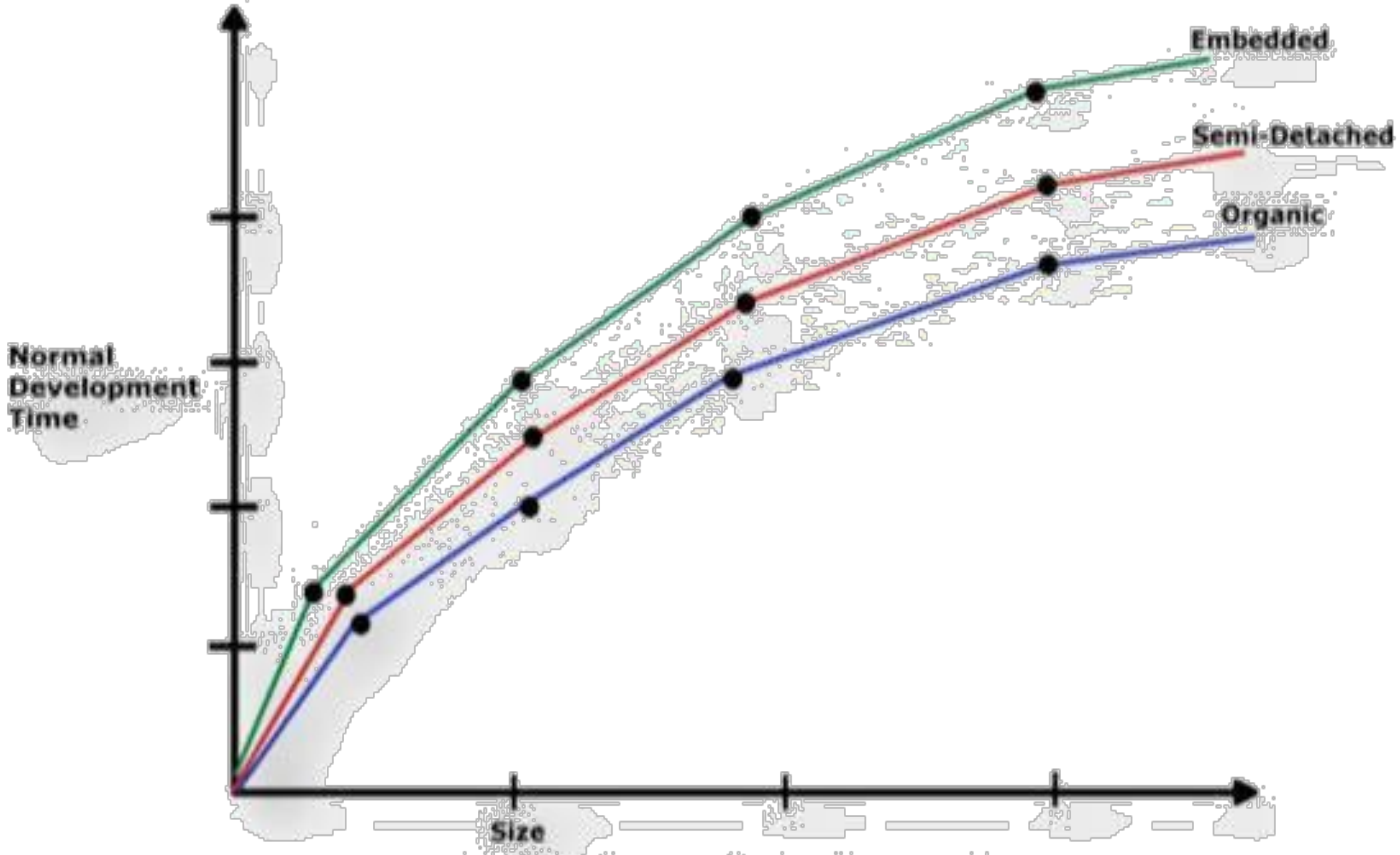


Fig. Development time Verses size

FORMS OF COCOMO MODEL

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Complete/Detailed COCOMO Model

Basic COCOMO:

Computes software development effort and cost as a function of programme size expressed in terms of Lines Of Code (LOC).

The basic cocomo model takes the following

form: **$E=a_b*((KLOC)^{\wedge}b_b)$** persons-months

$D=c_b*((E)^{\wedge}d_b)$ months

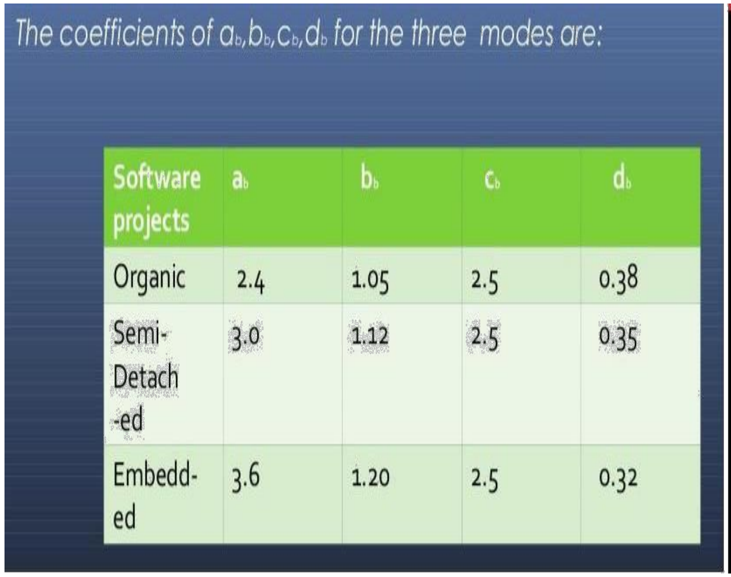
Where

E = Stands for the effort applied in terms of person months

D = Development time in chronological months

KLOC-Kilo lines of code of the project

ab, bb, cb,db are the coefficients for three modes



The coefficients of a_b,b_b,c_b,d_b for the three modes are:

Software projects	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-Detach-ed	3.0	1.12	2.5	0.35
Embedd-ed	3.6	1.20	2.5	0.32

MERITS AND DEMERITS OF BASIC COCOMO MODEL

Advantages:

1. Basic COCOMO model is good for quick, early, rough order of magnitude estimates of software project.
2. COCOMO is simple, because it requires a small amount of data (LOC) to determine the effort and cost. Hence it is a static single-valued model.

Limitations :

1. The accuracy of this model is limited because it does not consider certain factors for cost estimation of software. These factors are hardware constraints, personal quality and experiences, modern techniques and tools.
2. Not suitable for rapid , recuse based developments.

***Example*:** consider a software project using semi-detached mode with 30,000 lines of code . We will obtain estimation for this project as follows:

(1) Effort estimation

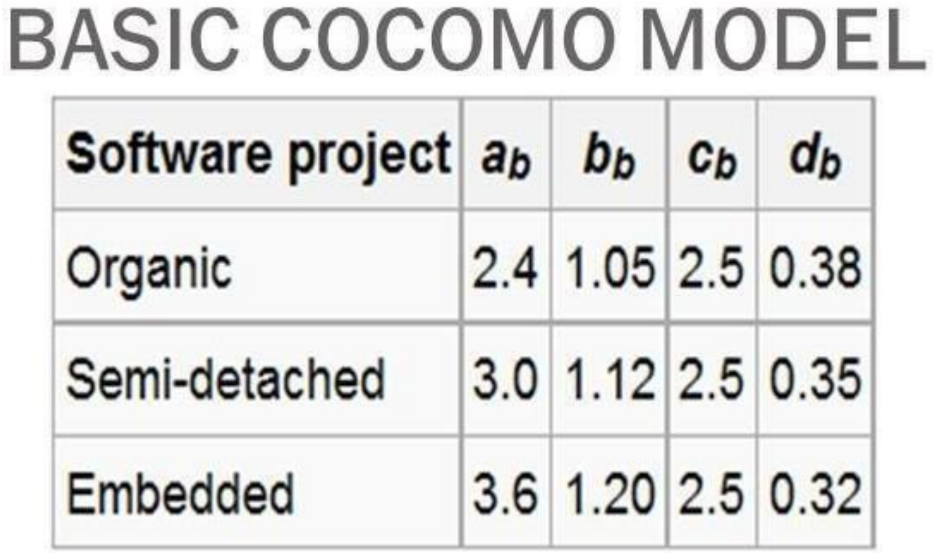
$E= a_b*((KLOC)^{\wedge} b_b)$ person-months
E=3.0*(30^1.12) , lines of code=30000=30000/1000 KLOC= 30 KLOC
E=135 person-month

(2) Duration estimation

$D= (c_b *(E^{\wedge}d_b))$ months
=2.5*(135^0.35)
D= 14 months

(3)Person estimation

N= E/D
=135/14
N=10 persons approx.



BASIC COCOMO MODEL

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Note: you have to memorize the value of these co-efficients

Intermediate COCOMO:

Computes effort as a function of programme size and a lot of cost drivers that includes subjective assessment of product attributes, hardware attributes , personal attributes and project attributes.

The basic model is extended to consider a set of cost driver attributes grouped into 4 categories:

(1) Product Attributes:

I. Required software reliability
II. Size of application software
III. Complexity of the product

(2) Hardware Attributes:

- I. Run-time performance constraints
- II. Memory constraints
- III. Required turn around time
- IV. Volatility of virtual machine

(3) Personal attributes:

- I. Analyst capability
- II. Software Engineer Capability
- III. Applications Experience
- IV. Programming language experience
- V. Virtual machine Experience

(4) Project Attributes:

- I. Use of software tools
- II. Required development schedule
- III. Application of software engineering methods

Now these 15 attributes get a 6-point scale ranging from “very low” to “extra high”. These ratings can be viewed as:

Very Low, Low, Nominal High, High, Very high, Extra high

Based on the rating effort multipliers is determined. The product of all effort Multipliers result in "Effort Adjustment Factor" (EAF).

The intermediate COCOMO takes the form.

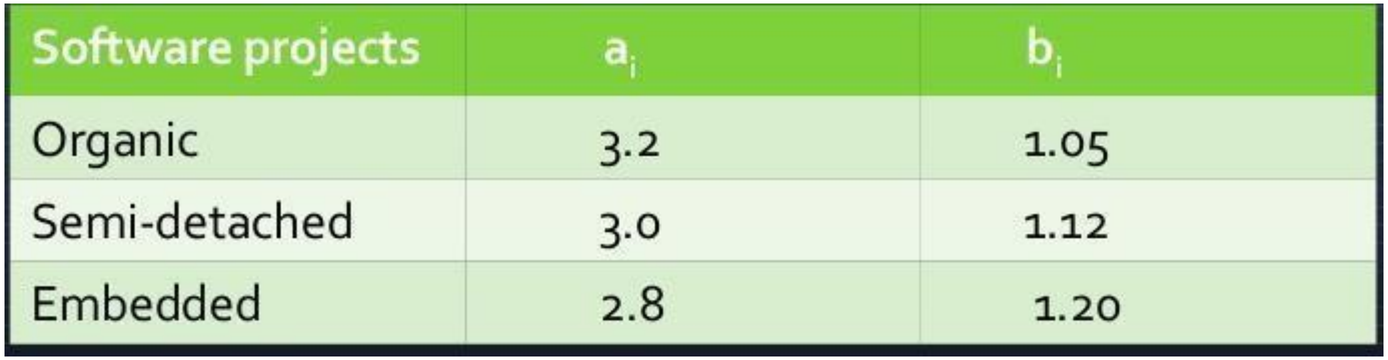
$E=a_i *(KLOC^{\wedge}b_i)*EAF$ where

E: Effort applied in terms of person-months

KLOC : Kilo lines of code for the project

EAF : It is the effort adjustment factor

The values of a_i and b_i for various class of software projects are:

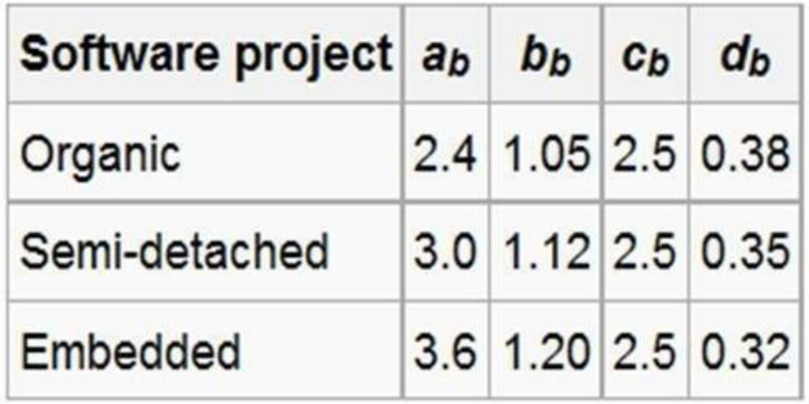


Software projects	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Memorize the table for maths

The duration and person estimate is same as in basic COCOMO model i.e;

$D=c_b*(E\text{^} d_b)$ months
i.e; use values of c_b and d_b coefficients. [Values of coefficients will not be given. Memorize it.]
N=E/D persons



Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Merits:

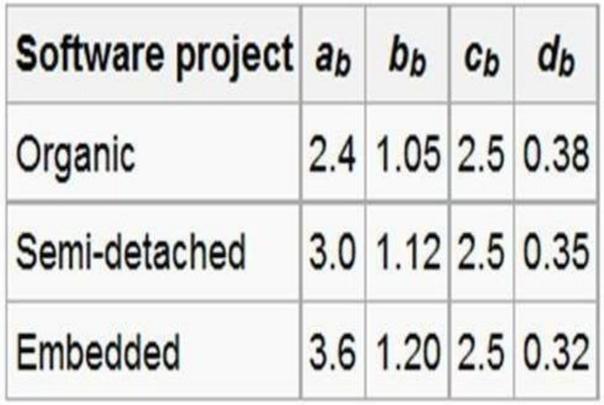
1. This model can be applied to almost entire software product for easy and rough cost estimation during early stage.
2. It can also be applied at the software product component level for obtaining more accurate cost estimation.

Limitations:

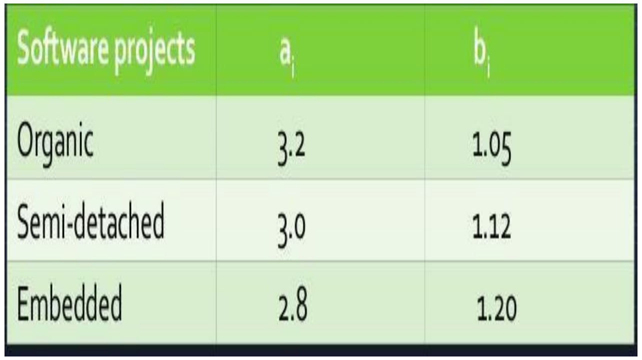
1. The effort multipliers are not dependent on phases.
2. A product with many components is difficult to estimate.

Example:

Consider a project having 30,000 lines of code which in an embedded software with critical area hence reliability is high (EAF=1.15).



Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32



Software projects	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

The estimation can be

E=a_i*(KLOC^b_i)*EAF

As reliability is high EAF=1.15 (product attribute)

a_i = 2.8

b_i = 1.20 [for embedded software which will not be given]

E=2.8*((30^1.20))*1.15

=191 person month

D=C_b*(E^d_b) = 2.5*(191^0.32)

=13.422 = 14 months approximately

N=E/D

=191/14

N=14 persons approx.

DETAILED/ADVANCED COCOMO MODEL:

A major shortcoming of both the basic and Intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics.
The Detailed COCOMO Model differs from the Intermediate COCOMO model in that it uses effort multipliers for each phase of the project. These phase dependent effort multipliers yield better estimates because the cost driver ratings may be different during each phase.

In Advanced COCOMO Model the cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

Example: 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

Of these, the communication part can be considered as **Embedded software**. The database part could be **Semi-detached software**, and 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.