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.

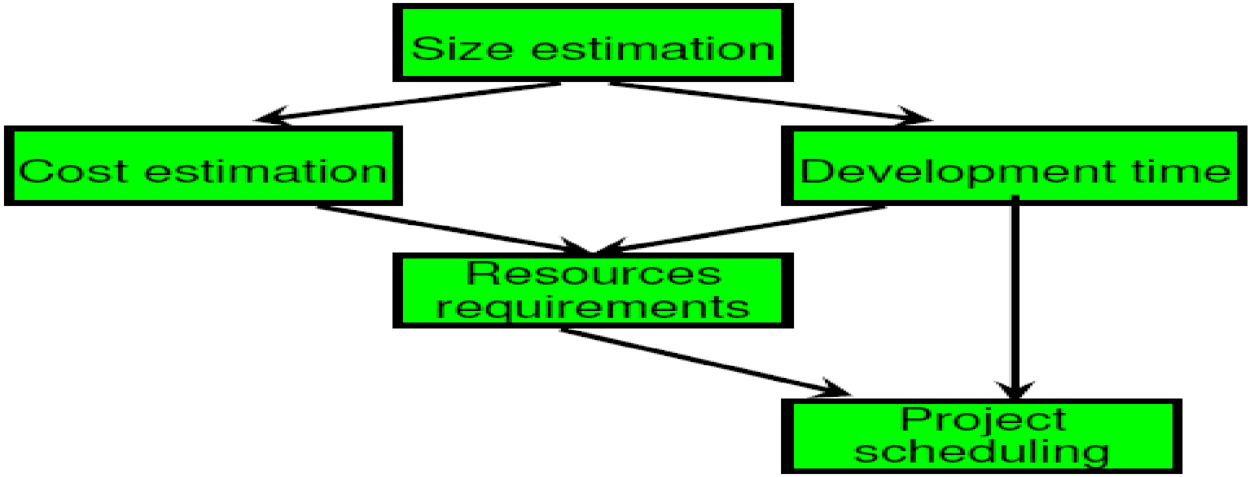
Cont……

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

Cont….

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



Activities during Software Project Planning

Project size estimation techniques

- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:
- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

Lines of Code (LOC)

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

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.	}

“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.

Function Count

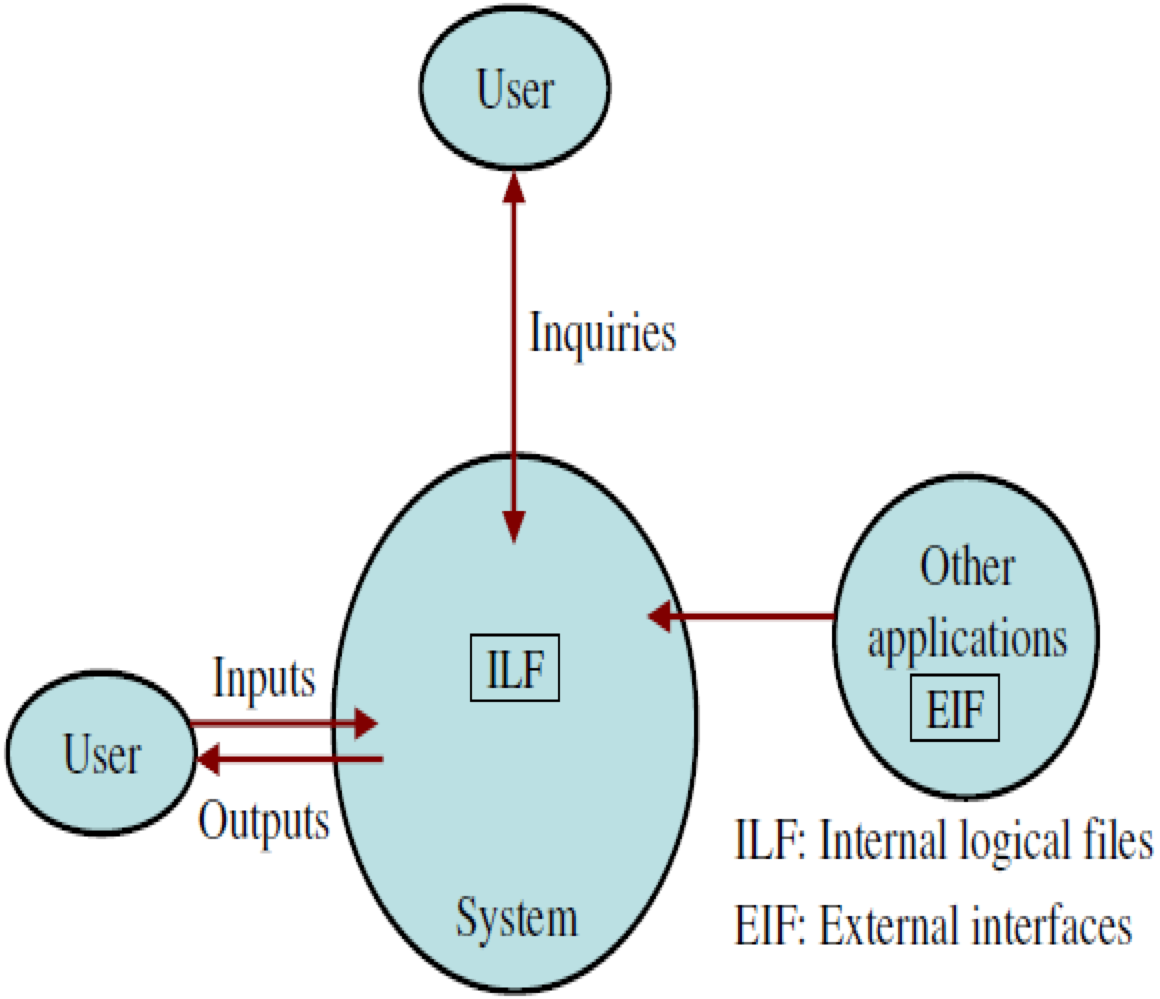
Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

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.

The FPA functional units are shown in figure given below:



FPAs functional units System

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.

(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.

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.

- 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.

Calculation of Function Point

Calculate Function Point.

$$FP = UFP * CAF$$

UFP=Unadjusted Function Point

Calculate Complexity Adjustment Factor (CAF)

CAF = 0.65 + (0.01 * F)

Where **F=14*Scale**

and scale can be calculated

0 - No Influence

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

Rate each factor on a scale of 0 to 5.

0	*1*	*2*	*3*	*4*	*5*
No Influence	*Incidental*	*Moderate*	*Average*	*Significant*	*Essential*

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 operations ?
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 ?

Calculate Unadjusted Function Point (UFP).

FUNCTION UNITS	LOW	AVG	HIGH
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

Ex-1(Calculation of Functional Point

Ex: Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

- User Input = 50
- User Output = 40
- User Inquiries = 35
- User Files = 6
- External Interface = 4

- **Step-1:** As complexity adjustment factor is average (given in question), hence,
- scale = 3. F = 14 * 3 = 42
- **Step-2:**CAF = 0.65 + (0.01 * 42) = 1.07
- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.
- UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628
- **Step-4:**Function Point = 628 * 1.07 = 671.96

Software Project Planning

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 ?

Solution

Unadjusted function point counts may be calculated using as:

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

$= 10 \times 3 + 12 \times 7 + 20 \times 7 + 15 + 10 + 12 \times 4$

$= 30 + 84 + 140 + 150 + 48$

$= 452$

FP $= UFP \times CAF$

$= 452 \times 1.10 = 497.2.$

Software Project Planning

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.

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

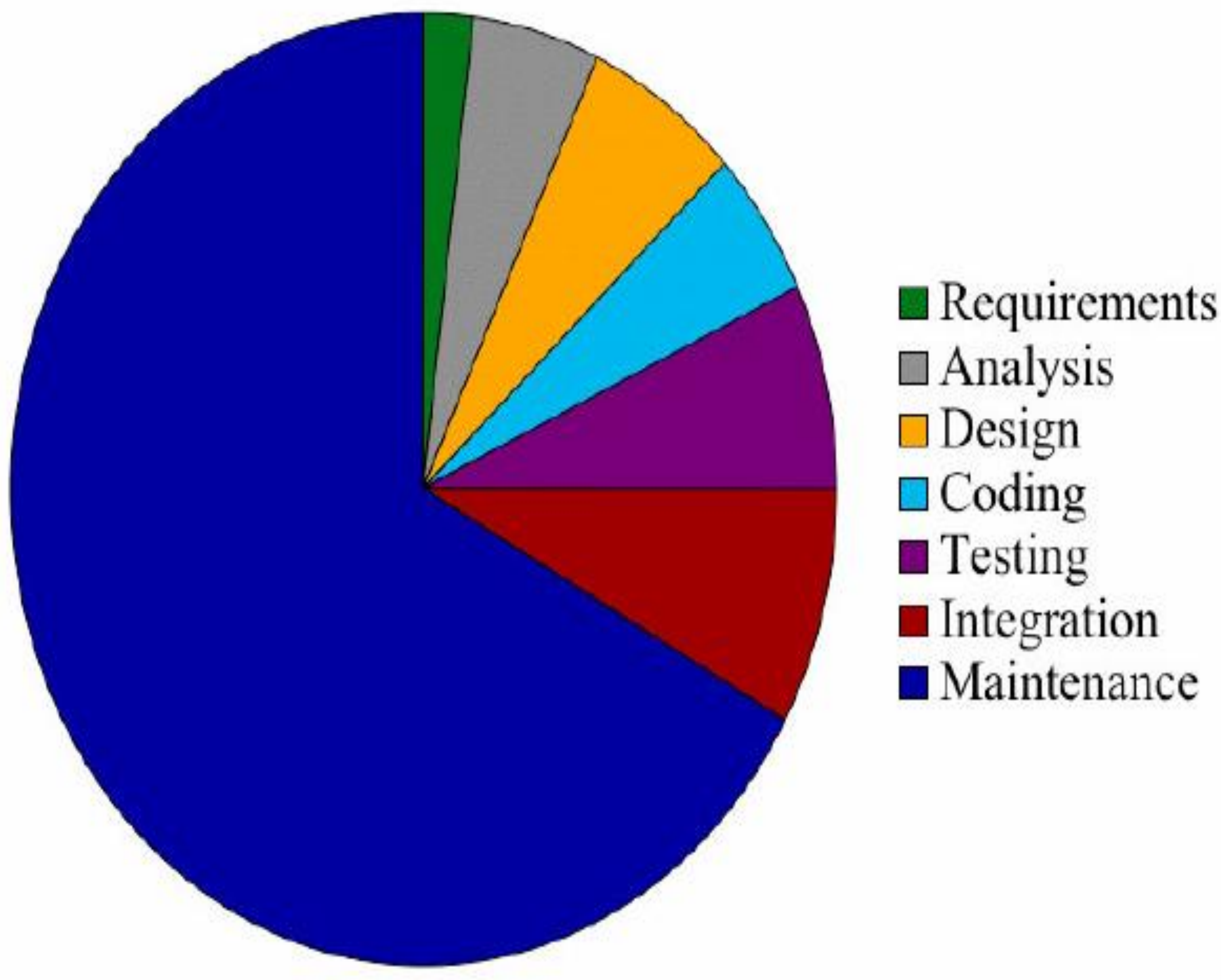
$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

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

Hence $\boxed{\text{FP} = 449}$

Software Project Planning

Relative Cost of Software Phases



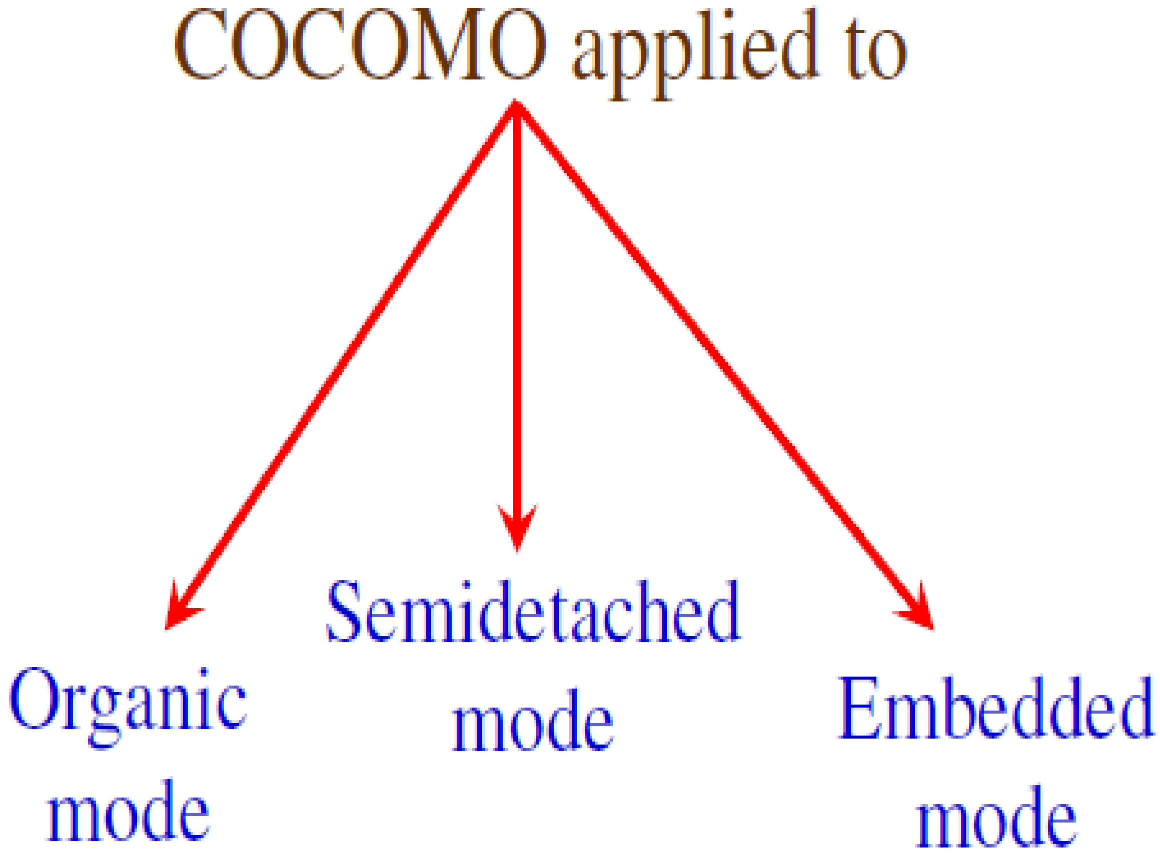
Software Project Planning

The Constructive Cost Model (COCOMO)

Constructive Cost model
(COCOMO)

Basic Intermediate Detailed

Model proposed by
B. W. Boehm's
through his book
Software Engineering Economics in 1981


COCOMO applied to
Organic mode
Semidetached mode
Embedded mode

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

Types of Models:
COCOMO consists of a hierarchy of three increasingly detailed and accurate forms.

Any of the three forms can be adopted according to our requirements.

These are types of COCOMO model:

- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

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. a_b, b_b, c_b and d_b are The coefficients

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

Basic COCOMO coefficients

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size } (SS) = \frac{E}{D} \, Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} \, KLOC / PM$$

Software Project Planning

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

Solution

The basic COCOMO equation take the form:

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

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

Estimated size of the project = 400 KLOC

(i) Organic mode

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \text{ PM}$$

(ii) Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{0.35} = 38.45 \text{ PM}$$

(iii) Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{0.32} = 38 \text{ PM}$$

Example

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{0.35} = 29.3$ PM

$$\text{Average staff size } (SS) = \frac{E}{D}\, Persons$$

$$= \frac{1133.12}{29.3} = 38.67\, Persons$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765\, KLOC / PM$$

$$P = 176\, LOC / PM$$

Intermediate Model

The basic COCOMO model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes

- ✓ **Product attributes -**

1:Required software reliability extent

2:Size of the application database

3:The complexity of the product

- ✓ **Hardware attributes -**

4:Run-time performance constraints

5:Memory constraints

6:The volatility of the virtual machine environment

7:Required turnabout time

- ✓ **Personnel attributes -**

8:Analyst capability

9:Software engineering capability

10:Applications experience

11:Virtual machine experience

12:Programming language experience

- ✓ **Project attributes -**

13:Use of software tools

14:Application of software engineering methods

15:Required development schedule

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications 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	
Programming language experience	1.14	1.07	1.00	0.95	

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Project Attributes					
Application of 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
Required developme nt schedule	1.23	1.08	1.00	1.04	1.10

Intermediate COCOMO equations

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

$$D = c_i (E)^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	..	..	1.00	1.11	1.30	1.66
STOR	..	..	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table.

These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor).

Detailed COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

Development Phase

Plan / Requirements

EFFORT : 6% to 8%

DEVELOPMENT TIME : 10% to 40%

% depend on mode & size

Design

Effort	:	16% to 18%
Time	:	19% to 38%

Programming

Effort	:	48% to 68%
Time	:	24% to 64%

Integration & Test

Effort	:	16% to 34%
Time	:	18% to 34%

Principle of the effort estimate

Size equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

$$A = 0.4\ DD + 0.3\ C + 0.3\ I$$

The size equivalent is obtained by

$$S\ (\text{equivalent}) = (S \times A) / 100$$

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Lifecycle Phase Values of μ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.06	0.16	0.26	0.42	0.16
Organic medium S≈32	0.06	0.16	0.24	0.38	0.22
Semidetached medium S≈32	0.07	0.17	0.25	0.33	0.25
Semidetached large S≈128	0.07	0.17	0.24	0.31	0.28
Embedded large S≈128	0.08	0.18	0.25	0.26	0.31
Embedded extra large S≈320	0.08	0.18	0.24	0.24	0.34

Lifecycle Phase Values of τ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.10	0.19	0.24	0.39	0.18
Organic medium S≈32	0.12	0.19	0.21	0.34	0.26
Semidetached medium S≈32	0.20	0.26	0.21	0.27	0.26
Semidetached large S≈128	0.22	0.27	0.19	0.25	0.29
Embedded large S≈128	0.36	0.36	0.18	0.18	0.28
Embedded extra large S≈320	0.40	0.38	0.16	0.16	0.30