Project Management and Metrics:-

- **The management Spectrum:- (Refer Roger B.page :- 647 - 660)**
- **Metric for process and project:- (Refer Roger B.page :- 675 - 676)**
- **Metric for software quality:- (Refer Roger B.page :- 679 - 682)**
- **Software Project Planning:-**
- **Objective:- (Refer Roger B.page :- 692 - 693)**
- **Scope And Resources:- (Refer Roger B.page :- 694 - 697)**
- **Project Estimation And Decomposition Technique:- (Line of Code(LOC) and Function Point(FP))**

  Line of Code:- *"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".*

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.	}

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.

Function Point:-

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.

*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:

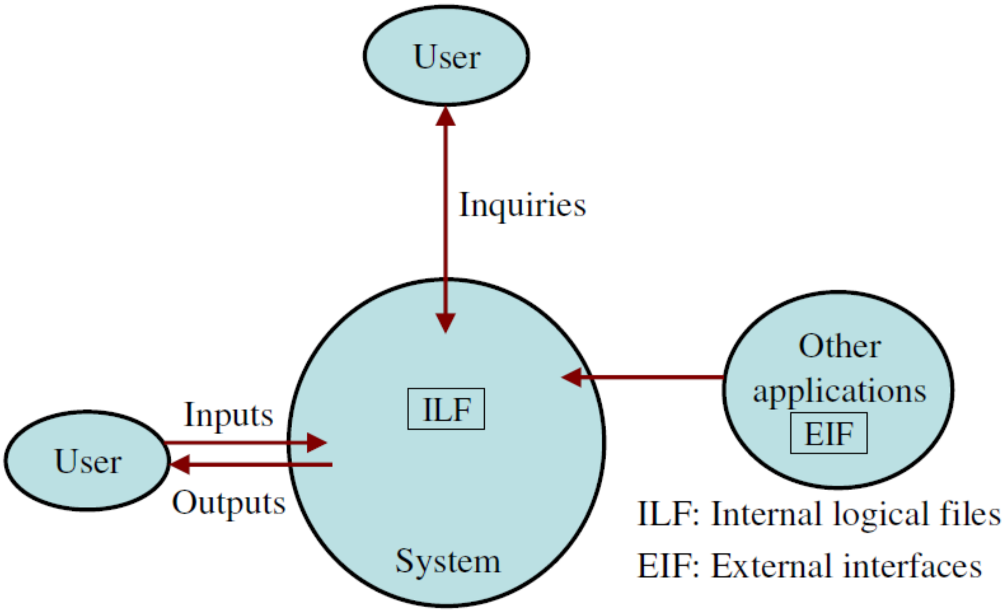

Fig. 3: FPAs functional units System

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.*

Software Project Planning

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.*

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.

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

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					

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.

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*.

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.

Table 3 : Computing function points.

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 ?

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.

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.

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

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 ?

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.

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

(*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

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

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

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

Empirical Estimation Model:-

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

Table 4: The comparison of three COCOMO modes

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

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:

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

Example: 4.5

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: 4.6

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

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

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

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

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

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

Intermediate Model

Cost drivers

(*i*) Product Attributes

- Required s/w reliability
- Size of application database
- Complexity of the product

(*ii*) Hardware Attributes

- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

(*iii*) Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(*iv*) Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

Multipliers of different cost drivers

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	--

Table 5: Multiplier values for effort calculations

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

Table 6: Coefficients for intermediate COCOMO

Detailed COCOMO Model

Detailed COCOMO

Phase-Sensitive effort multipliers

Cost drivers

design & test

Manpower allocation for each phase

Three level product hierarchy

Modules subsystem

System level

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

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

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

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

Distribution of software life cycle:

1. Requirement and product design
   (a) Plans and requirements
   (b) System design
2. Detailed Design
   (a) Detailed design
3. Code & Unit test
   (a) Module code & test
4. Integrate and Test
   (a) Integrate & Test

Example: 4.7

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used

Or

Developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

Solution

This is the case of embedded mode and model is intermediate COCOMO.

Hence $$E = a_i (KLOC)^{d_i}$$

$$= 2.8\,(400)^{1.20} = 3712 \text{ PM}$$

Case I: Developers are very highly capable with very little experience in the programming being used.

EAF = 0.82 x 1.14 = 0.9348

E = 3712 x .9348 = 3470 PM

D = 2.5 $(3470)^{0.32}$ = 33.9 M

Case II: Developers are of low quality but lot of experience with the programming language being used.

EAF = 1.29 x 0.95 = 1.22

E = 3712 x 1.22 = 4528 PM

D = 2.5 $(4528)^{0.32}$ = 36.9 M

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very litter experience.

Example: 4.8

Consider a project to develop a full screen editor. The major components identified are:

I. Screen edit

II. Command Language Interpreter

III. File Input & Output

IV. Cursor Movement

V. Screen Movement

The size of these are estimated to be 4k, 2k, 1k, 2k and 3k delivered source code lines. Use COCOMO to determine

1. Overall cost and schedule estimates (assume values for different cost drivers, with at least three of them being different from 1.0)
2. Cost & Schedule estimates for different phases.

Solution

Size of five modules are:

Screen edit	= 4 KLOC
Command language interpreter	= 2 KLOC
File input and output	= 1 KLOC
Cursor movement	= 2 KLOC
Screen movement	= 3 KLOC
Total	**= 12 KLOC**

Let us assume that significant cost drivers are

i. Required software reliability is high, i.e.,1.15

ii. Product complexity is high, i.e.,1.15

iii. Analyst capability is high, i.e.,0.86

iv. Programming language experience is low,i.e.,1.07

v. All other drivers are nominal

$$EAF = 1.15 \times 1.15 \times 0.86 \times 1.07 = 1.2169$$

(a) The initial effort estimate for the project is obtained from the following equation

$$E = a_i \, (KLOC)^{b_i} \times EAF$$

$$= 3.2(12)^{1.05} \times 1.2169 = 52.91 \text{ PM}$$

Development time $D = C_i(E)^{d_i}$

$$= 2.5(52.91)^{0.38} = 11.29 \text{ M}$$

(b) Using the following equations and referring Table 7, phase wise cost and schedule estimates can be calculated.

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Since size is only 12 KLOC, it is an organic small model. Phase wise effort distribution is given below:

System Design	= 0.16 x 52.91 = 8.465 PM
Detailed Design	= 0.26 x 52.91 = 13.756 PM
Module Code & Test	= 0.42 x 52.91 = 22.222 PM
Integration & Test	= 0.16 x 52.91 = 8.465 Pm

Now Phase wise development time duration is

System Design	= 0.19 x 11.29 = 2.145 M
Detailed Design	= 0.24 x 11.29 = 2.709 M
Module Code & Test	= 0.39 x 11.29 = 4.403 M
Integration & Test	= 0.18 x 11.29 = 2.032 M

COCOMO-II:- (Do Yourself)

Project Scheduling:- Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

Basic Principles

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

Compartmentalization:- The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.

Interdependency:- The interdependency of each compartmentalized activity or task must be determined.

Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.

Time allocation:-Each task to be scheduled must be allocated some number of

work units (e.g., person-days of effort):- In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation:- Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort4). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.

Defined responsibilities:- Every task that is scheduled should be assigned to a specific team member.

Defined outcomes:-Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.

Defined milestones:-Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

Each of these principles is applied as the project schedule evolves.

- **The Relationship Between People and Effort(Refer Roger B.page :- 725 - 727)**
- **Earned Value Analysis(Refer Roger B.page :- 739 - 741)**

27.12. Assume you are a software project manager and that you've been asked to compute earned value statistics for a small software project. The project has 56 planned work tasks that are estimated to require 582 person-days to complete. At the time that you've been asked to do the earned value analysis, 12 tasks have been completed. However the project schedule indicates that 15 tasks should have been completed. The following scheduling data (in person-days) are available:

Task	Planned Effort	Actual Effort
1	12.0	12.5
2	15.0	11.0
3	13.0	17.0
4	8.0	9.5
5	9.5	9.0
6	18.0	19.0
7	10.0	10.0
8	4.0	4.5
9	12.0	10.0
10	6.0	6.5
11	5.0	4.0
12	14.0	14.5
13	16.0	—
14	6.0	—
15	8.0	—

Compute the SPI, schedule variance, percent scheduled for completion, percent complete, CPI, and cost variance for the project.

Step1 Calculate

- BAC - Budget of completion
  = 582 (given in question)

- PV or BCWS {Budget cost work schedule} (task should have been completed is 15)
  planned value

  Sum of 15 task = 156.5

- EV or BCWP = Sum of 12 tasks = 126.5
  Earned value — Budget cost work performed

- AC or ACWP = 127.5 (Sum of Actual effort)
  actual cost — actual cost of work perform

Step2 Calculate & Analyze

- SPI = $\frac{BCWP}{BCWS} = \frac{126.5}{156.5} \times 100 = 80.8\%$
  Scheduled performance index

- SV = BCWP - BCWS = 126.5 - 156.5 = -30
  Scheduled Variance

- CPI $\frac{BCWP}{ACWP} = \frac{126.5}{127.5} \times 100 = 99.2\%$
  Cost performance index

- CV = BCWP - ACWP
  Cost Variance
  126.5 - 127.5
  = -1

1) Suppose you have a budgeted cost of a project at $ 900,000. The project is to completed in 9 months. After a month, you have completed 10 percent of the project at a total expense of $100,000. The planned completion should have been 15%.

Sol^n

BAC = 900,000$
AC = 100,000$

PV = planned completion (%) * BAC =
15% * 900,000
= $135000

EV = Actual completion (%) * BAC = 10% * 900,000
= 90,000

CPI = $\frac{EV}{AC} = \frac{90,000}{100,000} = 0.90$. This means for every $1 spent, the project is producing only 90% in work.

SPI = EV/PV = 90,000 / 135,000 = ~~0.90~~
0.67

Since both CPI & SPI are less than 1, it means that the project is over budget and behind schedule.

Suppose you are managing a software development project. The project is expected to be completed in 8 months at a cost of $10,000 per month. After 2 months you realize that the project is 30 percent completed at a cost of $40,000. You need to determine whether the project is on time and on budget after 2 months.

Step 1 Calculate the planned value (PV) and earned value (EV) from the scenario

- Budget at completion (BAC) = 10,000 * 8 = 80,000
- Actual cost (AC) = $40,000
- Planned completion = 2/8 = 25%
- Actual completion = 30%

Planned value = Planned completion (%) * BAC
= 25% * 80,000
= $20,000

Earned value = Actual completion (%) * BAC
= 30% * 80,000
= 24,000 $

Step 2 $CPI = \frac{EV}{AC} = \frac{24000}{40,000} = 0.6$

$SPI = \frac{EV}{PV} = \frac{24000}{20000} = 1.2$

Since ~~cost~~ CPI is less than one, this means the project is over budget. and SPI is more than one, the project is ahead of schedule.

Cost Performance index :- Represents the amount of work being completed on a project for every unit of cost spent. CPI is computed by Earned value / Actual cost. A value of above 1 means that the project is doing well against the budget.

Schedule Performance index Represent how close actual work is being completed compared to the schedule. SPI is computed by Earned value / planned value. A value of above one means that the project is doing well against the schedule.

EV = Earned value
PV = Budget at completion
AC - Actual cost.

PV = planned completion(%) * BAC
EV = Actual completion (%) * BAC

$CPI = \frac{EV}{AC}$ $SPI = \frac{EV}{PV}$

Categories of metrics.

① Product metrics:- describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability etc.

② Process Metrics:- describe the effectiveness and quality of the processes that produce the software product e.g.

→ effort required in the process.
→ time to produce the product.
→ effectiveness of defect removal during development.
→ Number of defects found during testing.
→ Maturity of the process.

③ project metrics:- describe the project characteristics and execution. e.g.

① Number of software developer.
② Staffing pattern over the life cycle of the software
③ Cost & schedule
④ productivity.