Chapter 3 Project Scheduling and Tracking

SOFTWARE PROCESS AND PROJECT METRICS

- We are concerned primarily with productivity and quality metrics—**measures of software development "output"** **as a function of effort and time applied** and measures of the "fitness for use" of the work products that are produced.

- ***Metric is defined*** **as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."**

- **It is necessary to evaluate software development process on regular basis to check whether project proceeds in correction direction or not and hence metrics is used.**

- **Metrics gives an idea about what and how much progress has been made and what modifications and enhancements is req. to improve the s/w process.**

- A software engineer collects measures and develops metrics so that indicators will be obtained.

- An *indicator* is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself. An indicator provides insight that enables the project manager or software engineers to adjust the process, the project, or the process to make things better.

- ***Process indicators*** enable a software engineering organization to gain insight into the efficacy of an existing process.

- ***Project indicators*** enable a software project manager to (1) assess the status of an ongoing project, (2) track potential risks, (3) uncover problem areas before they go "critical," (4) adjust work flow or tasks, and (5) evaluate the project team's ability to control quality of software work products.

(4) metrices is used to measure following properties of s/w development process.

- No of s/w developers employed for the project
- overall cost of the project
- Resources required by the project
- Quality desired from the project
- Schedule planned for the project
- Overall productivity of the project

(4) s/w engineering metrics can also allow us to;

a) define success & failure, and/or degree of success & failure, for a product, a process, or a person.

b) identify & quantify improvement, lack of improvement or degradation in our product, process and people

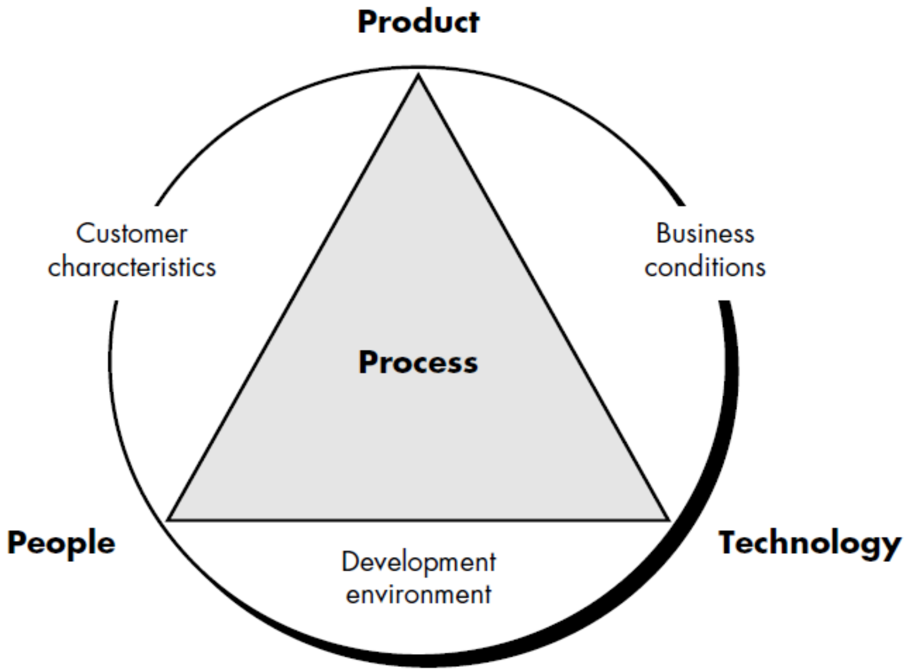
c) make meaningful & useful technical & managerial decisions.

d) make quantified & meaningful estimates.

Project Management Spectrum

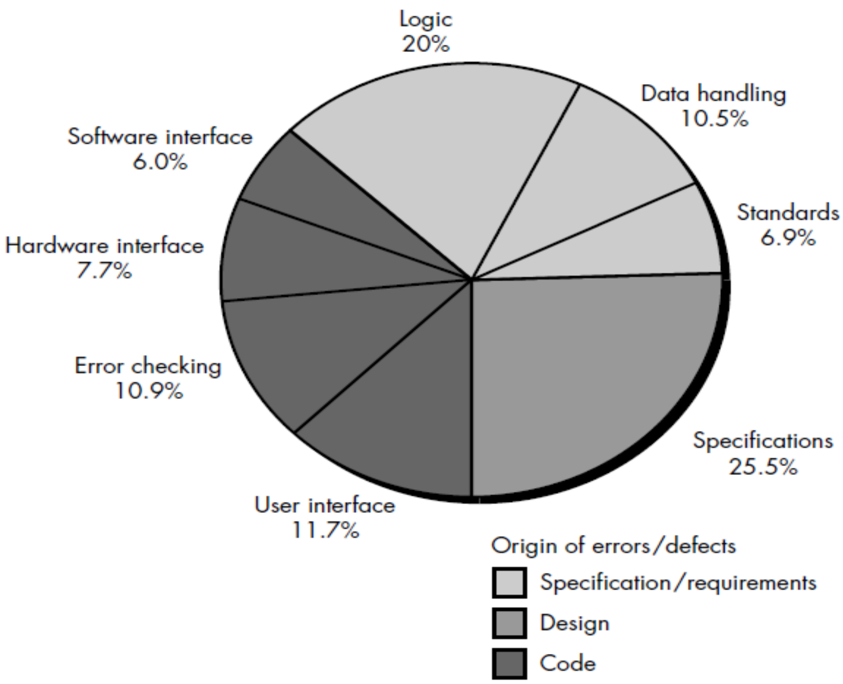
The only rational way to improve any process is to measure specific attributes of the process, develop a set of meaningful metrics based on these attributes, and then use the metrics to provide indicators that will lead to a strategy for improvement.

The process triangle exists within a circle of environmental conditions that include the development environment (e.g., CASE tools), business conditions (e.g., deadlines, business rules), and customer characteristics (e.g., ease of communication).



Statistical software process improvement (SSPI) uses software failure analysis to collect information about all errors and defects3 encountered as an application, system, or product is developed and used. Failure analysis works in the following manner:

1. All errors and defects are categorized by origin (e.g., flaw in specification, flaw in logic, nonconformance to standards).
2. The cost to correct each error and defect is recorded.
3. The number of errors and defects in each category is counted and ranked in descending order.
4. The overall cost of errors and defects in each category is computed.
5. Resultant data are analyzed to uncover the categories that result in highest cost to the organization.
6. Plans are developed to modify the process with the intent of eliminating (or reducing the frequency of) the class of errors and defects that is most costly.



Because it is natural that individual software engineers might be sensitive to the use of metrics collected on an individual basis, these data should be private to the individual and serve as an indicator for the individual only. Examples of *private metrics* include defect rates (by individual), defect rates (by module), and errors found during development.

Public metrics generally assimilate information that originally was private to individuals and teams. Project level defect rates (absolutely not attributed to an individual), effort, calendar times, and related data are collected and evaluated in an attempt to uncover indicators that can improve organizational process performance.

Project Metrics:

Project metrics and the indicators derived from them are used by a project manager and a software team to adapt project work flow and technical activities.

The first application of project metrics on most software projects occurs during estimation. Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software work.

Measures of effort and calendar time expended are compared to original estimates.

The intent of project metrics:

1. These metrics are used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks.
2. Project metrics are used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.

SOFTWARE MEASUREMENT: (Software Size Estimation)

Direct measures of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.

Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "–abilities".

1. **Size-Oriented Metrics:**

Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the *size* of the software that has been produced.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value.

Size of the Software

Size

- Size serves as a primary cost factor in most models.
- Following are the 2 commonly used ways to measure s/w size.

1> Lines of Code (LOC)

2> Function points

1. Lines of Code

1> Lines of Code (LOC)

LOC is one of the most commonly-used measure of source code program length. The non-commented source line of code (NLOC) is also known as effective lines of code (ELOC). Apart from NLOC, the commented source line of code (CLOC) is also a valid measure.

By measuring NCLOC & CLOC seperately we can define:

LOC [Total length] = NCLOC + CLOC

KLOC → is used to denote thousand of lines of code.

2. Function Point

2> Function Points (FP) [function oriented metrics]

- FP is a s/w measure that calculate size in terms of functionality in ~~the~~ a system
- FP count are made for following categories

① No. of user inputs → Each user i/p that provides distinct application oriented data to the s/w is counted. i/p's should be distinguished from inquiries, which are counted seperately.

2) No. of user outputs → Each user output that provides application-oriented information to the user is counted. o/p's refers to reports, screens, error messages & so on.

3) No. of user enquiries → An inquiry is defined as an on-line i/p that results in generation of some immediate s/w response in the form of an on-line output. Each distinct inquiry is counted.

4) No. of files → Each logical master file [i.e. a logical grouping of data that may be one part of a large database or a separate file] is counted.

5) No. of external interfaces → All machine readable interfaces [e.g. data files on tape or disk] that are used to transmit information to another system are counted.

- Function points are counted by completing the table shown in fig. Five information domain characteristics are determined, and counts are provided in the appropriate table location

measurement parameter	Count		Weighting Factor (complexity factor) simple	average	complex		
1) No. of user i/p's	10	X	3	4	6	=	30
2) No. of user o/p's	0	X	4	5	7	=	
3) No. of user enquiries	0	X	3	4	6	=	
4) No. of files		X	7	10	15	=	
5) No. of external interfaces		X	5	7	10	=	

(EC)

∴ count = total →

Table (1)
fig : Computing function point metrics

Table: Computing function point metrics

- Organization that use function point methods develop criteria for determining whether a particular entry is simple, average or complex.
- The Components of complexity factor are listed in table below.

- Table (2): Components of the technical complexity factor

F_1 → Reliable back-up & recovery	F_2 → Data communications
F_3 → Distributed functions	F_4 → Performance
F_5 → Heavily used configuration	F_6 → Online data entry
F_7 → Operational ease	F_8 → Online update
F_9 → Complex interface	F_{10} → complex processing
F_{11} → Reusability	F_{12} → installation ease
F_{13} → multiple sites	F_{14} → Facilitate change

Table 2: Components of technical complexity

- Each component is rated from 0 to 5, where 0 means the component has no influence on the system & 5 means the component is essential.
- The complexity factor can be calculated as,

$$CF = 0.65 + 0.01\,(\text{sum}\,(F_i))$$

- The factor various from 0.65 (if each f_i is set to 0) to 1.35 (if each f_i is set to 5). The final function point calculation is;

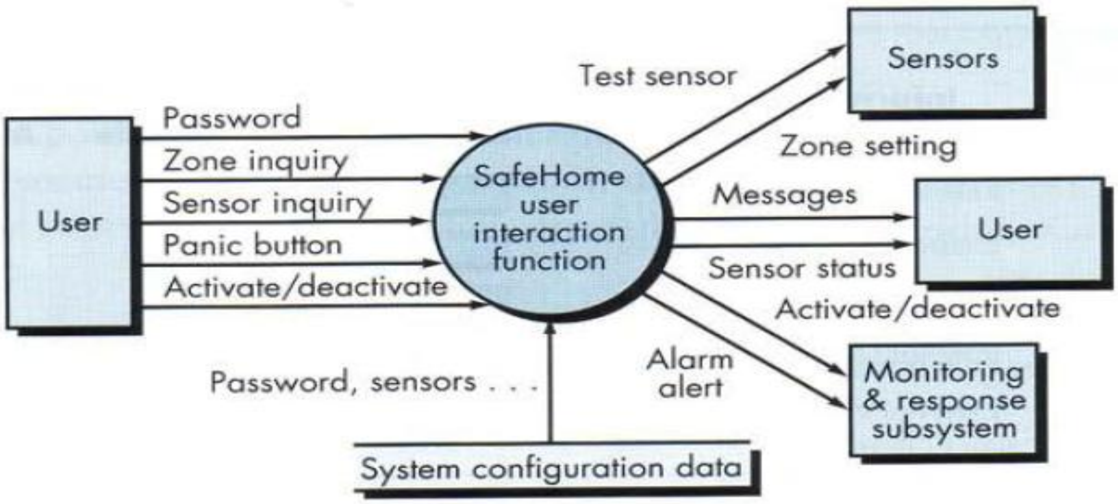
$$[FP = FC \times CF]$$

where; FP = function point
FC = function count (1)
CF = complexity factor

$$\therefore [FP = FC \times [0.65 + 0.01 \times (\text{sum}\,(F_i))]]$$

where FC = sum of all entries obtained from table (1)

Example:



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

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

Example 1:

1> An example of LOC-Based estimation

Consider a s/w package to be developed for a computer-aided design (CAD) application for mechanical components. A review of the system specification indicates that the s/w is to execute on engineering workstation & must interface with various computer graphics peripherals including a mouse, digitizer, high-resolution color display & laser printer.

- Following major s/w functions are identified

1> User interface & control facilities (UICF)
2> 2-dimensional geometric analysis (2DGA)
3> 3-dimensional geometric analysis (3DGA)
4> Database management (DBM)
5) Computer graphics display facilities (CGDF)
6) peripheral control (PC)
7) design analysis modules (DAM)

Function	Estimated LOC
UICF	2,300
2DGA	5,300
3DGA	6,800
DBM	3,350
CGDF	4,950
PC	2,100
DAM	3,400
estimated lines of code	33,200

Example 2:

2) An example of FP-based estimation

Information Domain Value	optimistic	most likely	pessimistic	estimated count	weight	FP count
no. of i/p's	20	24	30	24	4	96
no of o/p's	12	15	22	16	5	80
number of inquiries	16	22	28	22	4	88
number of files	04	04	05	04	10	40
number of external interfaces	02	02	03	02	7	14
Count - Total						318

Each of the complexity weighting factor is estimated

Factor	Value
Backup & recovery	4
Data communications	2
Distributed processing	0
performance critical	4
existing operating environment	3
on-line data entry	4
i/p transaction over multiple screens	5
master files updated on-line	3
information domain values complex	5
internal processing complex	5
code designed for reuse	4
conversion / installation in design	3

multiple installations 5

Application designed for change 5

∴ complexity adjustment factor

i.e. $CF = 0.65 + 0.01 \times \text{sum}(f_i)$

$= 0.65 + 0.01 \times 53$

$= 1.17$

∴ $FP_{estimated} = FC \times CF$

$= 318 \times 1.17$

$= 375$

Software Project Estimation

- Software is the most expensive element in most computer based system.
- An Empirical estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP.
- Estimation models provide direct estimates of the effort and duration that would be used in project.
- These models preferably have one main input (normally a measure of product size) and various secondary adjustment factors (called cost drivers)

1. **Cost Models:** Cost models provide direct estimates of efforts as they typically have a primary cost factor such as size and number of secondary adjustment factors or cost drivers.

a) COCOMO Model

① COCOMO Model

Barry Boehm introduces s/w estimation model named as cocomo, for Constructive Cost model

cocomo model takes 3 form:

① model 1 → The basic cocomo model computes s/w development effort (and cost) as a function of program size expressed in estimated lines of code.

② model 2 → The intermediate cocomo model computes s/w development effort as a function of program size & a set of "cost drivers" that include subjective assessments of product, hardware, personnel & project attributes

③ Model 3 → The advanced cocomo model incorporates all characteristics of the intermediate version with an assessment of the cost drives impact on each step (analysis, design etc) of the s/w engineering process

- cocomo models are defined for 3 classes of s/w projects

i> organic mode → relatively small, simple s/w projects in which small teams with good application experience

ii> Semi-detached mode → Refers to the mode that is used for an intermediate project in which mixed teams must work to set a set of rigid & less than rigid requirements (transaction processing system with fixed requirements for terminal hardware & database software) iii> Embedded mode → a s/w project that must be developed within a set of tight hardware, software & operational constraints (e.g. flight control s/w for aircraft).

s/w 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

Table 1

1 • The basic cocomo equation take the form:

$$E = a_b \, KLOC^{b_b}$$

$$D = C_b E^{d_b}$$

where E = effort applied in person-months

D = Development time

KLOC = estimated no. of delivered lines of code for the project

a_b, C_b = coefficients } given in table 1

b_b, d_b = exponents }

(2) • The intermediate cocomo model takes the form:

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

where, E = effort applied in person - months

KLOC = estimated number of delivered lines of code for the project

a_i, b_i = coefficient are given in table (2)

Software Project	a_i	b_i
organic	3.2	1.05
semi - detached	3.0	1.12
embedded	2.8	1.20

EAF = effort adjustment factor

(Typical values for EAF range from 0.9 to 1.4)

- EAF (EAF) is calculated using 15 cost drivers.
- The cost drivers are grouped in to 4 categories:
  - i> product
  - ii> computer
  - iii> personnel
  - iv> project

- Following is the list of Cost-Drivers for cocomo model
  1. Required s/w reliability
  2. Database size
  3. product complexity
  4. Execution time constraint
  5. main storage constraint
  6. Virtual machine volatility
  7. Computer turnaround time
  8. Analyst capability
  9. Applications experience
  10. programmer capability
  11. virtual machine experience
  12. Language experience
  13. modern programming practices
  14. Software Tools

$\left.\begin{array}{l}1\\2\\3\end{array}\right\}$ product

$\left.\begin{array}{l}4\\5\\6\\7\end{array}\right\}$ computer

$\left.\begin{array}{l}8\\9\\10\\11\\12\end{array}\right\}$ personnel

$\left.\begin{array}{l}13\\14\end{array}\right\}$ project

Example 1:

15. Development Schedule → project

Example → Apply basic cocomo model to the CAD s/w.

LOC = 33,200

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

$= 2.4\,(33.2)^{1.05}$

$(E = 95 \text{ person - months})$

$\therefore D = C_b\, E^{d_b}$

$= 2.5\,(95)^{0.38}$

$(D = 12.3 \text{ months})$

The value for project duration enables the planner to determine a recommended no. of people N, for the project

$$N = E / D$$
$$= 95 / 12.3$$
$$\cong 8 \text{ people}$$

$\therefore$ Estimated cost $= 95 * 2000$
$=$ Rs

(assuming cost per person months $= 2000$)

Example 2:

Suppose you are developing a software product in the organic mode. You have estimated the size of the product to be 70695 lines of code, compute effort and development time. Assuming cost of 25,000 person month calculate total cost of product

(constants aa = 2.4. bb = 1.05, bc = 2.5, bd = 0.38) (10 Marks)

Ans.:

From the basic COCOMO estimation formula for organic software:

$$Effort = 2.4 * (70.695)^{1.05} = 202.92 \text{ PM}$$

$$Nominal\_Development\_Time = 2.5 * (202.92)^{0.38} = 19.06 \approx 20 \text{ months}$$

$$Cost\_Required\_to\_develop\_the\_product = 20 * 25000 = Rs.\ 500000$$

8) • Advanced Cocomo →

• Advanced cocomo model helps to calculate the effort as a function of program size as well as cost drives weighted on the basis of each phase of the s/w lifecycle.

• This model is applicable for the intermediate model at the component level, and further the phase-based approach is used to combine the estimate. The 4 phases namely

- → (1) Requirement planning & product Design (RPD)
- → (2) Detailed design (DD)
- → (3) Code & Unit test (CUT)
- → (4) Integration & test (IT)

b) COCOMO II

(2) CoCoMo II

• cocomo II model is useful for a much wider collection of techniques & technologies.

• This model provides up-to-date support for business software, object-oriented s/w, s/w created through spiral or evolutionary development models & s/w developed by using commercial-off-the-shelf application composition utilities.

• cocomo II includes the following models.

i> The Application Composition model → (used when s/w is composed from existing parts)

① Refers to the model that is used in prototyping to resolve potential high risk issues, such as user interfaces, software/system interaction, performance, technology maturity.

② Object points are used for sizing rather than the traditional LOC metric.

③ An initial size measure is determined by counting the no. of screens, reports & third generation components that will be used in the application.

④ Each object is classified as simple, medium, difficult.

ii> Early Design Stage Model → Used once requirements have been stabilized & basic s/w architecture has been established. i.e. used when requirements are available but design has not yet started.

iii> Post - Architecture stage model → Used during the construction of the s/w.
i.e. used one the system architecture has been designed & more information about the system is available.

- The cocomo II model require sizing information
  i.e.
  - ① object points
  - ② function points
  - ③ LOC

• ① object points → An indirect s/w measure that is computed using counts of the number of:

i> screen (at the user interface)

ii> reports

iii> components likely to be required to build the application.

• Each object instance is classified into one of the 3 complexity levels. i.e. simple, medium or difficult.

• Complexity is a function of number & source of the client & server data tables that are required to generate the screen or report & the no. of views or sections presented as part of the screen or report.

• Once complexity is determined, the no. of screens reports & components are weighted according to the table below.

Object Type	complexity weight		
	Simple	Medium	Difficult
screen	1	2	3
Report	2	5	8
3GL component			10

Table :- Complexity weighting for object types

- The object point count is then determined by multiplying the original no. of object instances by weighting factor & summing to obtain a total object point count.
- When component-based development or general s/w reuse is to be applied, the % of reuse is estimated & the object point count is adjusted.

$$NOP = (\text{object points}) \times [(100 - \%\text{reuse})/100]$$

where; NOP = new object points

- To derive an estimate of effort based on computed NOP, "productivity rate" must be derived.
- Table below represent productivity rate

$$PROD = NOP / \text{person-month}$$

$$\left[\therefore \text{estimated effort} = \frac{NOP}{PROD}\right]$$

Table ②: productivity rate for object points

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

Project Scheduling

Project Scheduling

- Project managers objective is to define all project tasks, build a n/w that depicts their interdependencies, identify the tasks that are critical within the n/w & then track their interdependencies progress to ensure that delay is recognized; To accomplish this the manager must have a schedule that has been defined at a degree of resolution that allows progress to be monitored & project to be controlled.

- principles of Project Scheduling
  1) Compartmentalization
  2) Interdependency
  3) Time allocation
  4) Effort validation
  5) Defined responsibilities
  6) Defined outcomes
  7) Defined milestone

- Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific s/w engineering tasks.

- During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major process framework activities & product functions to which they are applied.

- As the project gets underway, each entry on the macroscopic schedule is refined into a detailed schedule.

Task Network ▶ • A task n/w, also called an activity n/w, is a graphic representation of the task flow for a project.

• It is sometimes used as the mechanism through which task sequence & dependencies are input to an project scheduling tool.

▶ Objective of Project Scheduling

• The objective of project scheduling is to enable a project manager to define work tasks establish their dependencies, assign human resources to tasks, & develop a variety of graphs, chart & tables that helps in tracking & control of the s/w project.

• Program evaluation & Review technique (PERT) and Critical Path Method (CPM) are 2 project scheduling methods that can be applied to s/w development.

• (1) You need to consider the following factors before creating a project.

- i> identify the tasks to be performed
- ii> Allocating the resources (material, human resources) to each task.
- iii> Determining effort require for each task.
- iv> Determining which part of project should be allocated to whom.
- v> Determing the time required for project completion.

(2) Scheduling involves following activities:

i) identifying project goals

ii) working out work Breakdown structure (WBS) → Divide the project in to sub-tasks & it is necessary to complete these tasks in particular order for completion of project.

iii) Allocating Resources to the tasks

iv) Identifying Dependencies

v) Identifying Risks

vi) Estimating Costs (Budgets)

vii) Creating project plan or schedule.

1) Time Line Chart

i) Timeline Charts → (Gantt Chart) (Scheduling Techniques)

① Timeline chart can be developed for entire project or seperate charts can be developed for each project function.

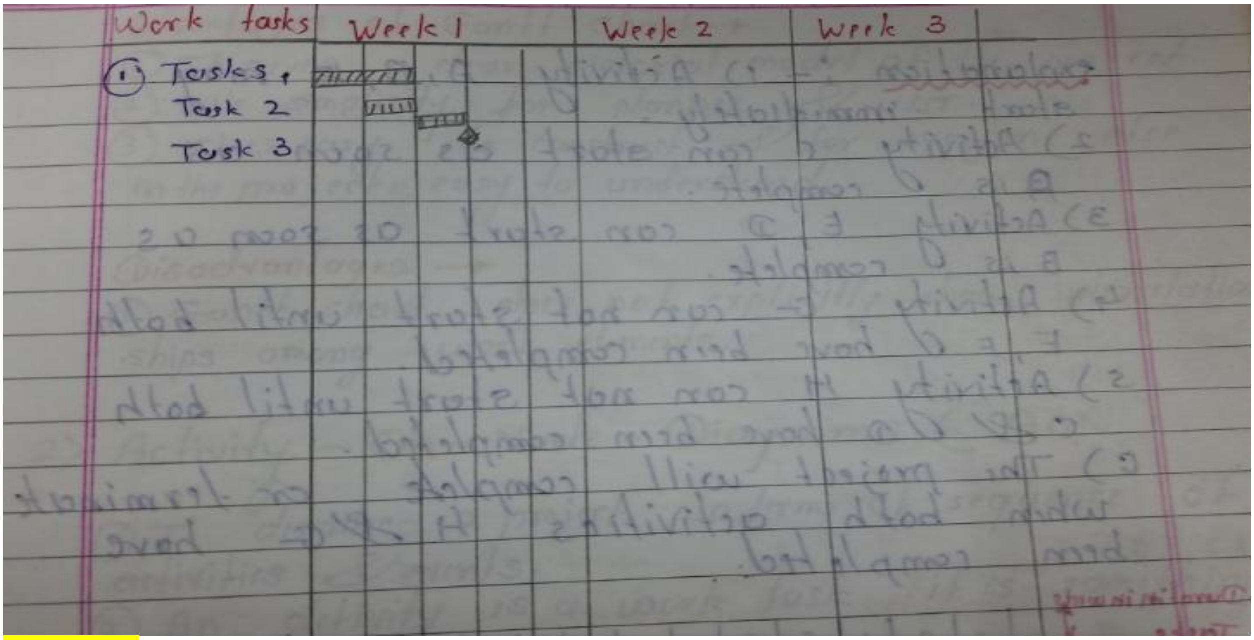
② Gantt chart, developed by Henry Gantt in 1917, is a type of bar chart that illustrates the entire project schedule.

③ Gantt chart consists of horizontal axis representing total time span of the project & vertical axis representing tasks that make up the project.

④ Horizontal bars of varying lengths represent sequence, timing & duration for each tasks.

⑤ The bar may overlap for tasks that are carried out during same time span.

⑥ A vertical line is used to represent report date.



Example:

example: - Draw Gantt chart for the following

Activity	Week Duration	Precedent (Predecessor)
A) Hardware specification	6	—
B) Software design	4	—
C) Install Hardware	3	A
D) code & test s/w	4	B
E) File take on	3	B
F) write user manuals	10	—
G) User Training	3	E, F
H) Install and Test	2	C, D

explanation :- 1) Activity A, B, F may start immediately.

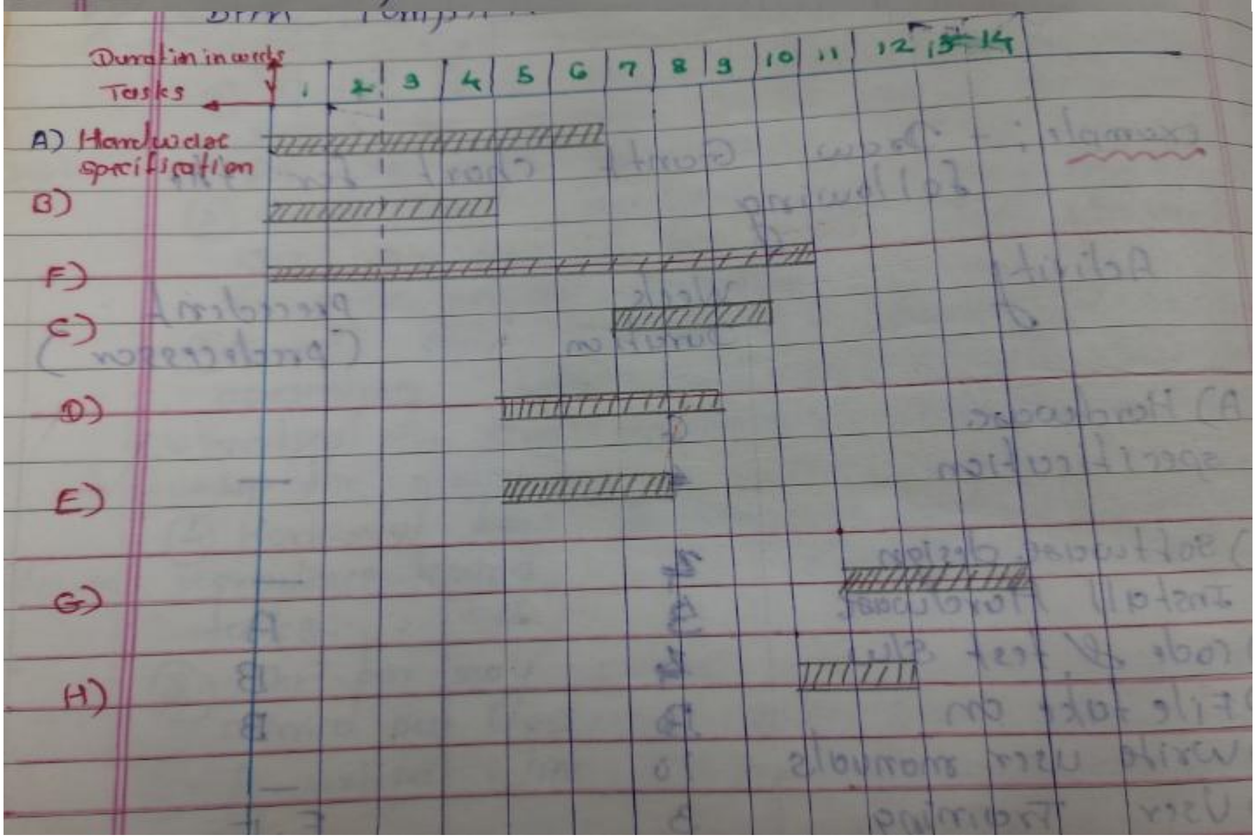
2) Activity C can start as soon as A is complete.

3) Activity E, D can start as soon as B is complete.

4) Activity G can not start until both E, F have been completed.

5) Activity H can not start until both C & D have been completed.

6) The project will complete or terminate when both activities H & G have been completed.



Advantages of Gantt chart →

1. It gives a clear pictorial model of the project.
2. It's simplicity for planner & user.
3. It is easy to construct & for everyone else in the project, easy to understand.

Disadvantages →

1. Gantt chart does not explicitly show interrelationships among work elements.

2) Critical Path Method: (CPM)

(3) CPM (Critical Path method) →

1. The expected project duration Te is determined by finding the longest path through the N/w.
2. Path is any route comprised of one or more arrows (activities) connected in sequence.
3. Longest path from original node to terminal node is called the Critical Path; this gives the expected project duration.

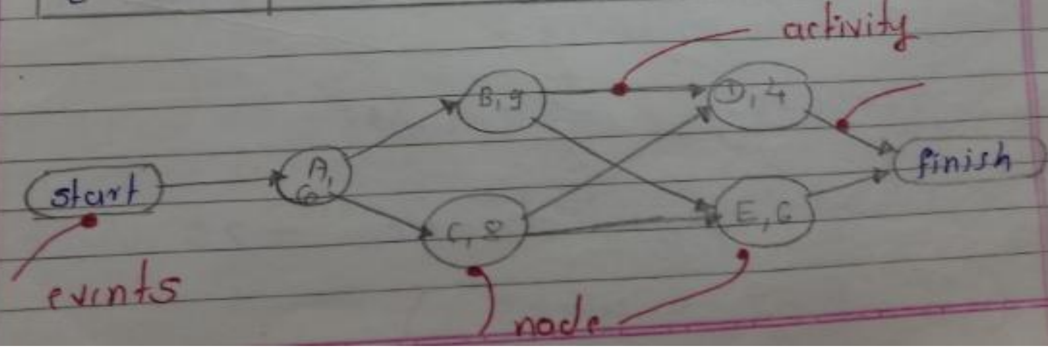
To fine critical path of network, first step is to create Activity On Node Diagram

2> Activity - On - Node - Diagrams (AON)

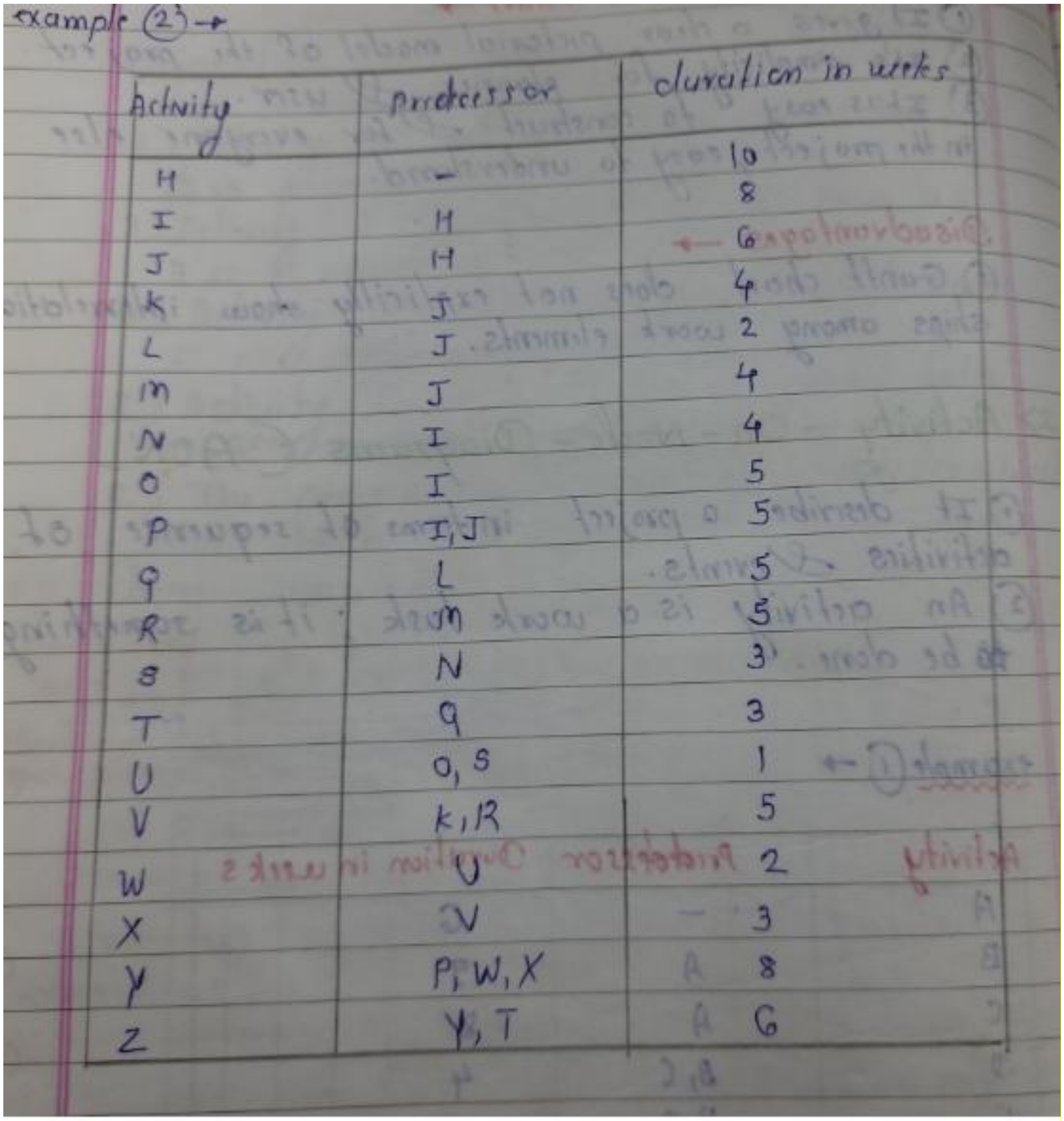
1. It describe a project in terms of sequence of activities & events.
2. An activity is a work task; it is something ~~to~~ to be done.

example 1 →

Activity	Predecessor	Duration in weeks
A	—	6
B	A	9
C	A	8
D	B, C	4
E	B, C	6

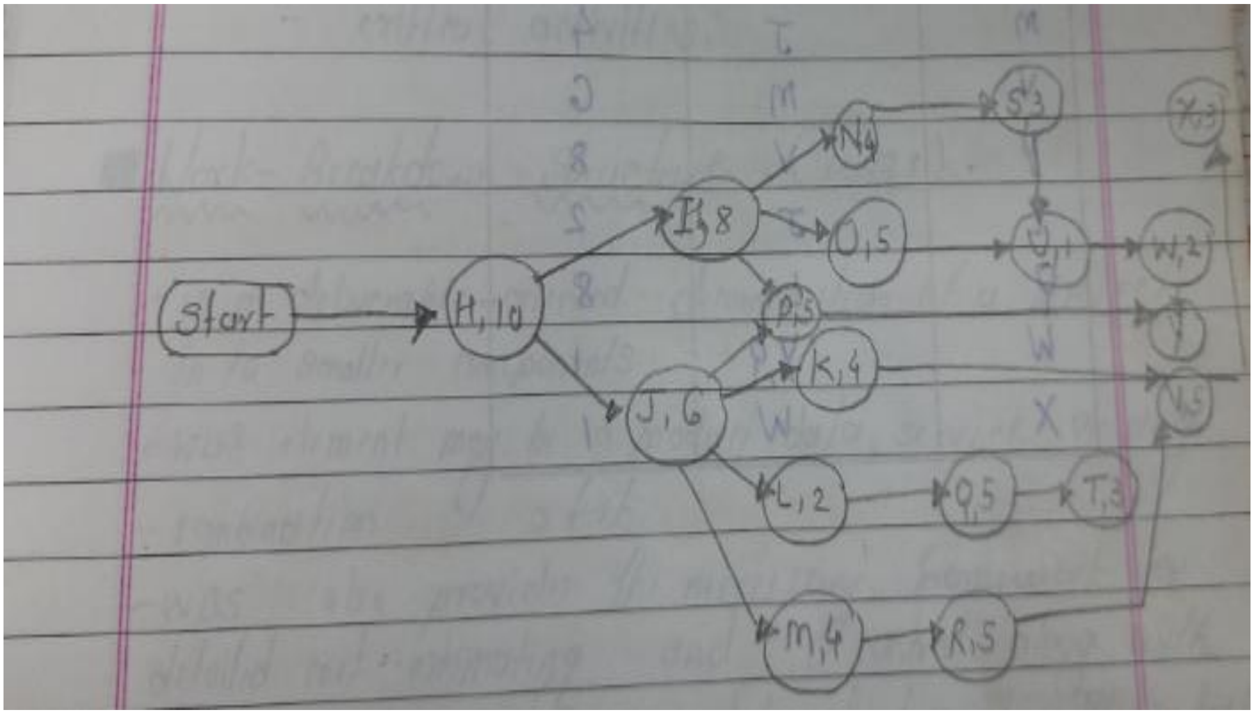


Example: Draw AON for given example



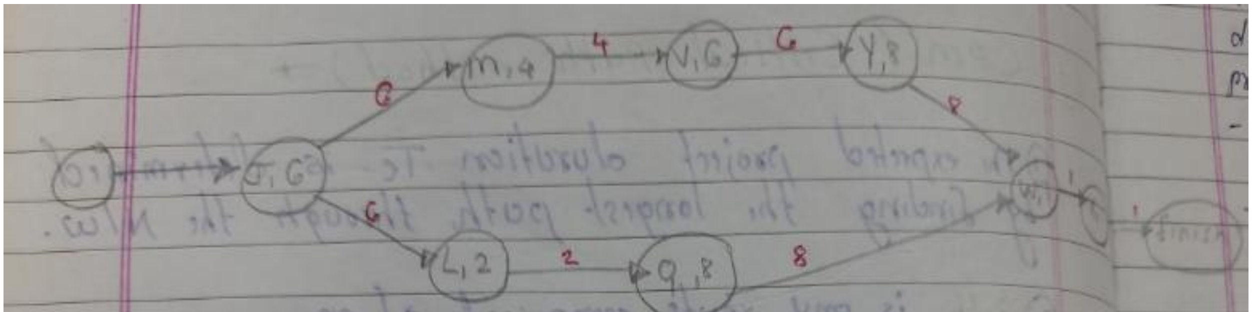
example (2) →

Activity	Predecessor	duration in weeks
H	–	10
I	H	8
J	H	6
K	J	4
L	J	2
M	J	4
N	I	4
O	I	5
P	I, J	5
Q	L	5
R	M	5
S	N	3
T	Q	3
U	O, S	1
V	K, R	5
W	U	2
X	V	3
Y	P, W, X	8
Z	Y, T	6



Example: Calculate critical path for given example

Activity	Immediate predecessor	Duration
J	–	6
M	J	4
V	M	6
Y	V	8
L	J	2
Q	L	8
W	Y,Q	1
X	W	1



∴ Two paths from start to end.

i) J-m-V-Y-W-X = 6+4+6+8+1+1 = 26 weeks

ii) J-K-q-W-X = 6+2+8+1+1 = 18 weeks

∴ Critical path is ;

J-m-V-Y-W-X = 26 weeks

∴ Estimated project duration = 26 Weeks

(∴ Te = 26 Weeks)

Example:

Draw activity diagram and Gantt-chart for the following:

Activity	Predecessor	Duration (in weeks)
A	–	2
B	A	3
C	–	2
D	A, C	3
E	D	4
F	A, D	3
G	B, E	2
H	G	3

Find duration required for the project

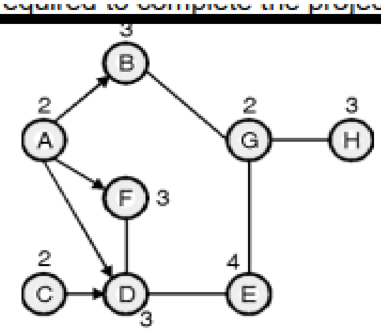


Fig Activity diagram

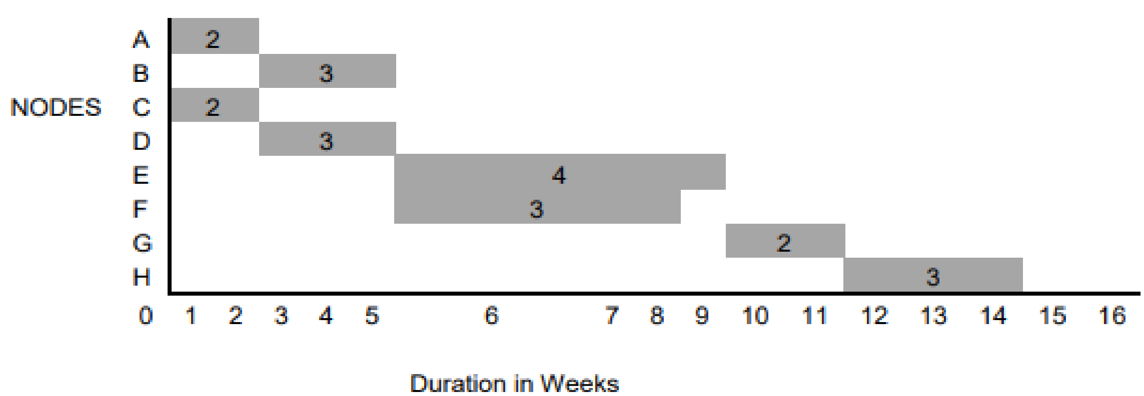
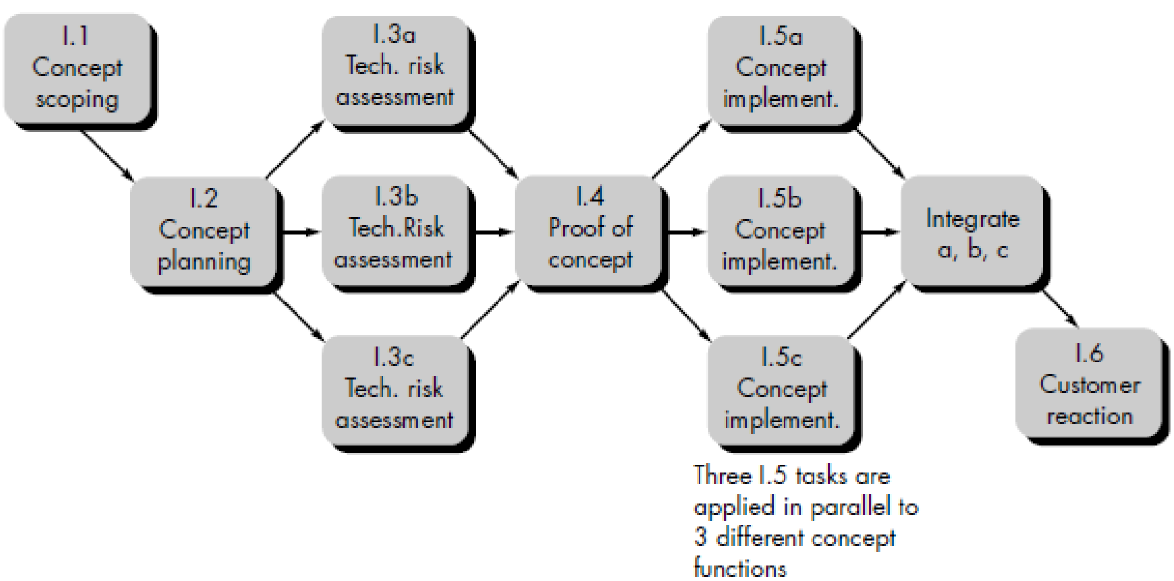


Fig. Gantt Chart

Note → ① For a project more than 1 critical path is possible.
② Activities on critical path are called critical activities.

Work-Breakdown-Structure (WBS)

- is a deliverable-oriented decomposition of a project in to smaller components.
- WBS element may be a product, data, service or any combination of it.
- WBS also provides the necessary framework for detailed cost estimating and control along with providing guidance for schedule development & control
- WBS is hierarchical and incremental decomposition of the project
- Well designed WBS makes it easy to assign each project activity to one and only one terminal element of the WBS.



Earned Value Analysis (EVA):

Earned Value Analysis: **One approach <u>to measuring progress in a software project</u>** is to calculate how much has been accomplished. This is called earned value analysis.

It is basically the percentage of the estimated time that has been completed. Additional measures can be calculated. Although this is based on estimated effort, it could be based on any quantity that can be estimated and is related.

Basic measures

- **Budgeted Cost of Work (BCW) :** The estimated effort for each work task.
- **Budgeted Cost of Work Scheduled (BCWS) :** The sum of the estimated effort for each work task that was scheduled to be completed by the specified time.
- **Budget at Completion (BAC) :** The total of BCWS and thus the estimate of the total effort for the project.
- **Planned Value (PV) :** The percentage of the total estimated effort that is assigned to a particular work task; PV=BCW/BAC.
- **Budgeted Cost of Work Performed (BCWP) :** The sum of the estimated efforts for the work tasks that have been completed by the specified time.
- **Actual Cost of Work Performed (ACWP) :** The sum of the actual efforts for the work tasks that have been completed.

Progress indicators

- Earned Value(EV) = BCWP/BAC
  = The sum of the PVs for all completed work tasks
  = PC : Percent complete
- Schedule Performance Index(SPI) = BCWP/BCWS
- Schedule Variance(SV) = BCWP – BCWS
- Cost Performance Index(CPI) = BCWP/ACWP
- Cost Variance(CV) = BCWP – ACWP

Earned value analysis is approach to measuring progress in a software project is to calculate how much has been accomplished.

Example: Using the following job log, calculate all of the basic measures and the progress indicators. Is the project on schedule? Assume **that it is currently 5/1/2001**

Work Task	Estimated Effort (programmer-days)	Actual Effort So Far (programmer-days)	Estimated Completion Date	Actual Date Completed
1	50	70	1/15/2001	2/1/2001
2	35	20	2/15/2001	2/15/2001
3	20	40	2/25/2001	3/1/2001
4	40	40	4/15/2001	4/1/2001
5	60	10	6/1/2001	
6	80	20	7/1/2001	

Solution:

The BCWS is 50 + 35 + 20 + 40 = 145 programmer-days.

The BAC is 50 + 35 + 20 + 40 + 60 + 80 = 285 programmer-days.

The planned values (PVs) for the work tasks are (BCW/BAC)

2) 50/285=17.5 %
3) 35/285=12.3 %
4) 20/285=7.01%
5) 40/285=14.03 %
6) 60/285=21.05 %
7) 80/285=28.1 %

The earned value = 17.5 percent + 12.3 percent + 7.01 percent + 14.03 percent
= 50.7 percent.

The BCWP for 5/1/2001 is the same as BCWS in this example because the scheduled work has been completed.

Thus SPI = BCWP/BCWS= 145/145 = 1.

The schedule variance =BCWP-BCWS=145 – 145 = 0.

The cost performance index = BCWP/ACWP **(where ACWP=70+20+40+40=170)**
=145/170
=0.853 percent.

This indicates that the actual effort is greater than the estimated effort.

The cost variance is 145 – 170 = – 25.

This also indicates that more effort has been required than was estimated.
The project appears to be on schedule but is costing more than was planned.