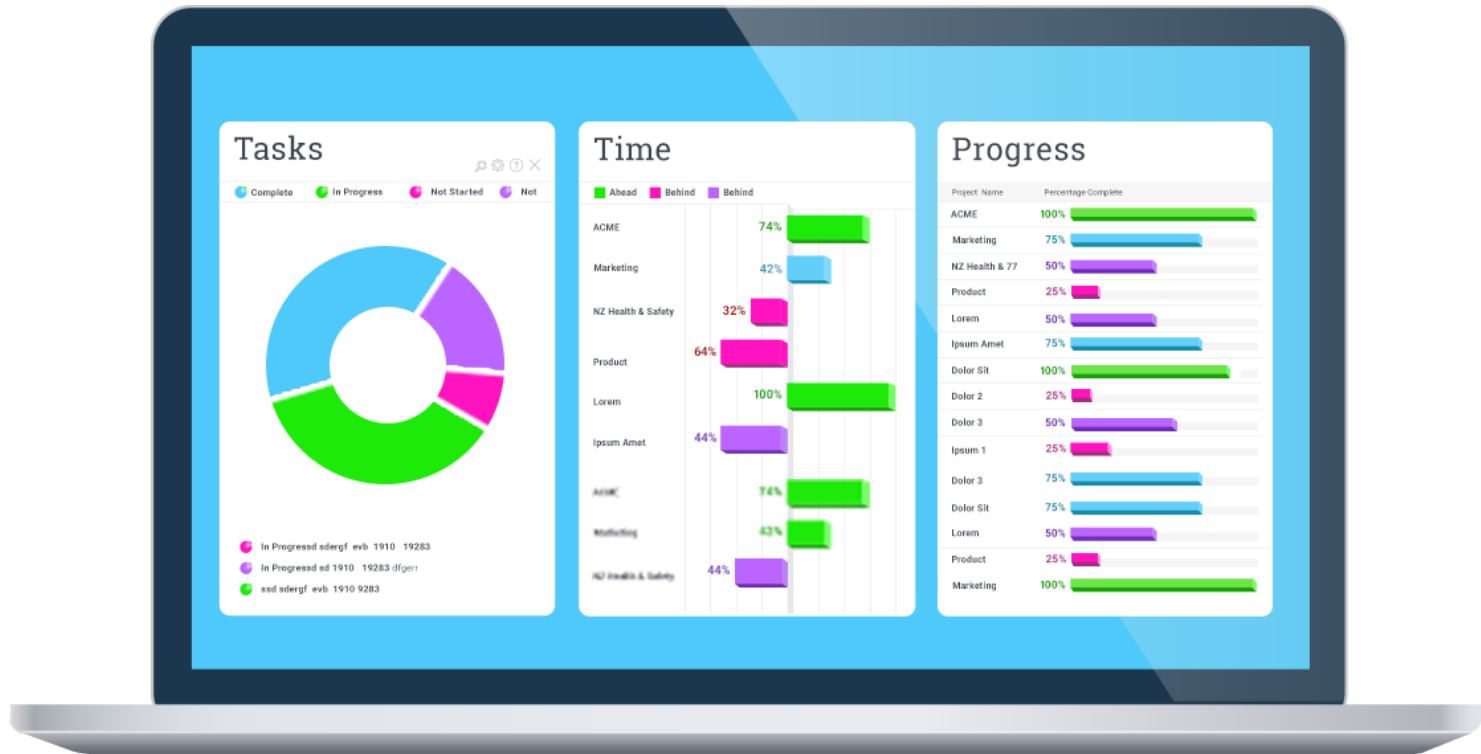


# CSE320

## UNIT 5

### Software Project Management



# Project planning

- Once a project is found to be feasible, software project managers undertake project planning.
- Project planning is undertaken and completed even before any development activity starts.



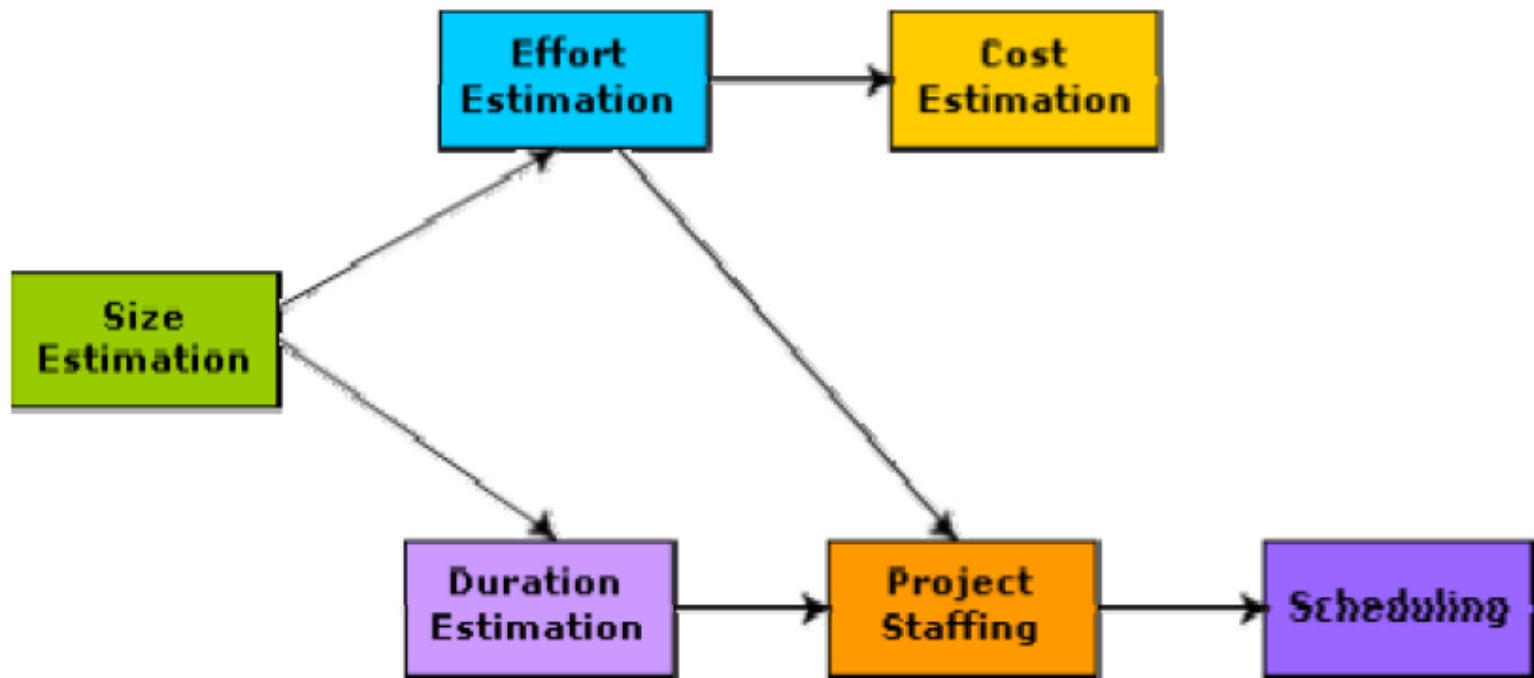
# Essential activities in Project Planning:

## 1) Estimating the following attributes of the project:

- **Project size:** What will be problem complexity in terms of the effort and time required to develop the product?
  - **Cost:** How much is it going to cost to develop the project?
  - **Duration:** How long is it going to take to complete development?
  - **Effort:** How much effort would be required?
- The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.

- 2) Staff organization and staffing plans
- 3) Scheduling manpower and other resources
- 4) Risk identification, analysis, and abatement planning
- 5) Miscellaneous plans such as quality assurance plan, configuration management plan, etc.

# Precedence ordering among project planning activities



# Metrics for software project size estimation

- Currently two metrics are popularly being used widely to estimate size:
- Lines of code (LOC) and
- Function point (FP).
- The usage of each of these metrics in project size estimation has its own advantages and disadvantages

# LOC Approach

- LOC is the simplest among all metrics available to estimate project size.
- This metric is very popular because it is the simplest to use.
- Using this metric, the project size is estimated by counting the number of source instructions in the developed program.
- Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.

# LOC

- Accurate estimation of the LOC count at the beginning of a project is very difficult.
- In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into submodules and so on, until the sizes of the different leaf-level modules can be approximately predicted.
- To be able to do this, past experience in developing similar products is helpful.
- By using the estimation of the lowest level modules, project managers arrive at the total size estimation.



# Drawbacks (LOC)

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

## WEEK 3: Design

## 7) WEEK 4: Design

16) WEEK 5: Design

5) WEEK 6: Dev

WEEK 7: DEV

TRD  
Final

Hocking  
&  
Shapiro  
Recd

### Hazing & Shagging Setup

## Finalize API Goals

Infrastructure Recs:  
- Laravel + Backbone

PLEASE DO NOT ERASE

Initialize Database

Non-normalized Database

Create  
Fixtures  
for  
testing

← Create Fixtures for testing

Admin  
Dashboard  
BIE frame  
work

Database  
API  
<Continuous>

← Engineering

B/E Sorting  
(for All  
Dashboard  
Views)  
w/ population

B/E Filter  
(for all virus)

B/E  
cmd for  
all DB  
Tables

Exporting

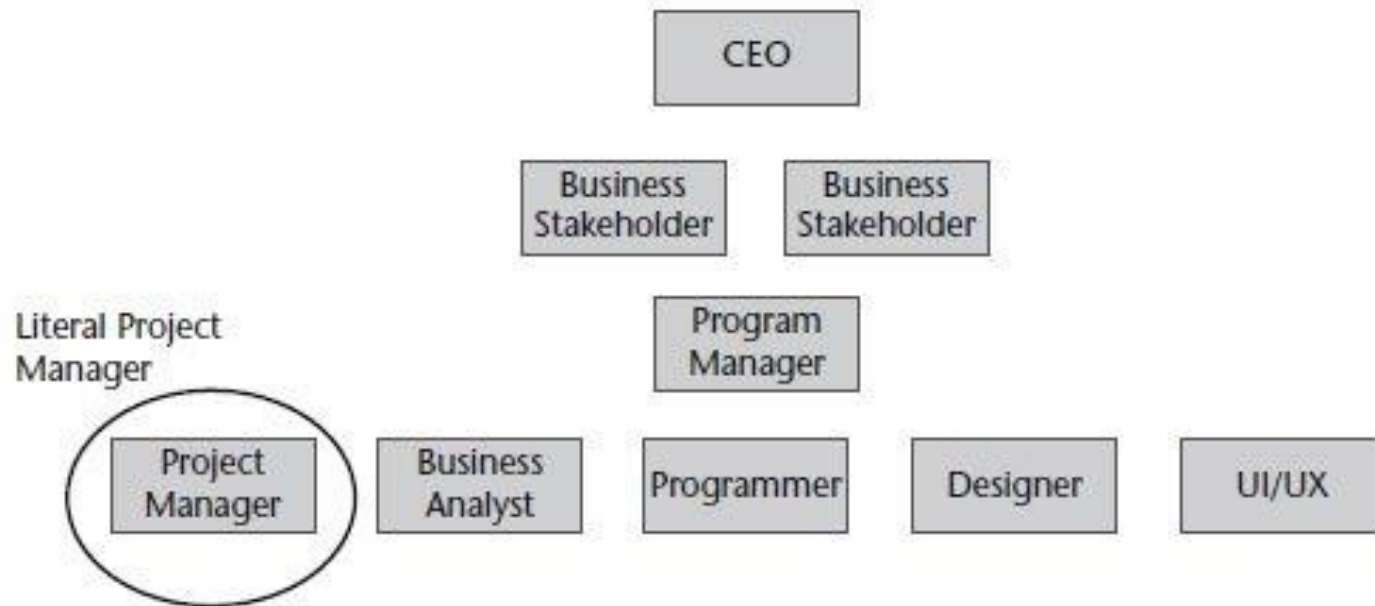
Cron  
Automated  
(and)  
email  
B/E

## ADMIN

API

Clinic on 1/19 Surgery

Submit



# Problem in Project Planning

- Commitment to **unrealistic time and resource estimates** result in **schedule slippage**.
- Schedule delays can cause customer dissatisfaction and adversely affect team morale.
- However, project planning is a very challenging activity. Especially for large projects, it is very much difficult to make accurate plans.
- A part of this difficulty is due to the fact that the proper parameters, scope of the project, project staff, etc. may change during the span of the project.

# Solution: Sliding Window Planning

- Planning a project over a number of stages protects managers from making big commitments too early.
- This technique of staggered planning is known as Sliding Window Planning.
- Starting with an initial plan, the project is planned more accurately in successive development stages.
- After the completion of every phase, the project managers can plan each subsequent phase more accurately and with increasing levels of confidence.

# Responsibilities of a software project manager

- Overall responsibility of **steering a project to success**.
- The job responsibility of a project manager ranges from invisible activities like **building up team morale** to highly visible customer presentations.
- Activities can be broadly classified **into project planning, and project monitoring and control activities**.
- It includes:
  - project proposal writing,
  - project cost estimation,
  - scheduling, project staffing,
  - software process tailoring, project monitoring and control,
  - software configuration management,
  - risk management,
  - interfacing with clients,
  - managerial report writing and presentations, etc.

# Skills necessary for software project management

- Theoretical knowledge of different project management techniques like **cost estimation, risk management, configuration management.**
- Good **qualitative judgment and decision taking** capabilities.
- Good communication skills and the ability get work done.





# Software Project Management Plan (SPMP)

## **1. Introduction**

- (a) Objectives
- (b) Major Functions
- (c) Performance Issues
- (d) Management and Technical Constraints

## **2. Project Estimates**

- (a) Historical Data Used
- (b) Estimation Techniques Used
- (c) Effort, Resource, Cost, and Project Duration Estimates

## **3. Schedule**

- (a) Work Breakdown Structure
- (b) Task Network Representation
- (c) Gantt Chart Representation
- (d) PERT Chart Representation

# Software Project Management Plan (SPMP)

## **4. Project Resources**

- (a) People
- (b) Hardware and Software
- (c) Special Resources

## **5. Staff Organization**

- (a) Team Structure
- (b) Management Reporting

## **6. Risk Management Plan**

- (a) Risk Analysis
- (b) Risk Identification
- (c) Risk Estimation
- (d) Risk Abatement Procedures

## **7. Project Tracking and Control Plan**

## **8. Miscellaneous Plans**

- (a) Process Tailoring
- (b) Quality Assurance Plan
- (c) Configuration Management Plan

# SPMP EXAMPLE 2

Hyper Plan v0.7.0 : C:\Users\andyb\Documents\HyperPlan\project plan.hp

File Edit View Help

Open Add Card Add Property Delete Zoom fit Zoom in Zoom out Magnify Swap X/Y Full screen

Cards Table

**Edit Card Properties**

Name: task 24  
Person: Claire  
Status: doing  
Priority: medium  
Duration: 3  
Due: 1/3/2015  
Notes:

**Layout/Color Cards**

Arrange columns (X) by: Due  
Arrange rows (Y) by: Person  
Set card color by: Due  
Set highlight color by: <Select>  
Show total for: <Select>  
Set row/column color: To background

Show Card Properties  
Filter Cards  
Stored Views  
Appearance

	12/2014	1/2015	2/2015	3/2015	
<b>Andy</b>	task 0 task 4 task 6 task 14 task 15 task 25 task 30 task 32	task 45	task 8 task 39		task 16 task 42 task 43
<b>Claire</b>	task 3 task 7 task 9 task 11 task 18 task 19 task 20 task 21 task 23 task 26 task 33 task 36 task 37 task 40	task 28 task 41 task 47 task 48 task 49	task 31	task 17 task 24	task 22 task 35
<b>John</b>	task 1 task 2 task 5 task 10 task 12 task 13 task 44 task 46	task 27 task 38			task 29 task 34

# Project Estimation techniques

- Empirical estimation techniques
- Heuristic techniques
- Analytical estimation techniques

# Empirical Estimation Techniques

- Based on making an **educated guess** of the **project parameters**.
- **Prior experience** with development of similar products is helpful.
- Although empirical estimation techniques are based on **common sense**, different activities involved in estimation have been formalized over the years.

Two popular empirical estimation techniques are:

- 1) Expert judgment technique
- 2) Delphi cost estimation

# Expert Judgment Technique

- An expert makes an educated guess of the **problem size** after analyzing the problem thoroughly.
- Estimates the **cost** of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate.
- However, this technique is subject to human errors and individual bias.
- For example, he may be conversant with the database and user interface parts but may not be very knowledgeable about the computer communication part.
- A more refined form of expert judgment is the estimation made by group of experts.

# Delphi cost estimation

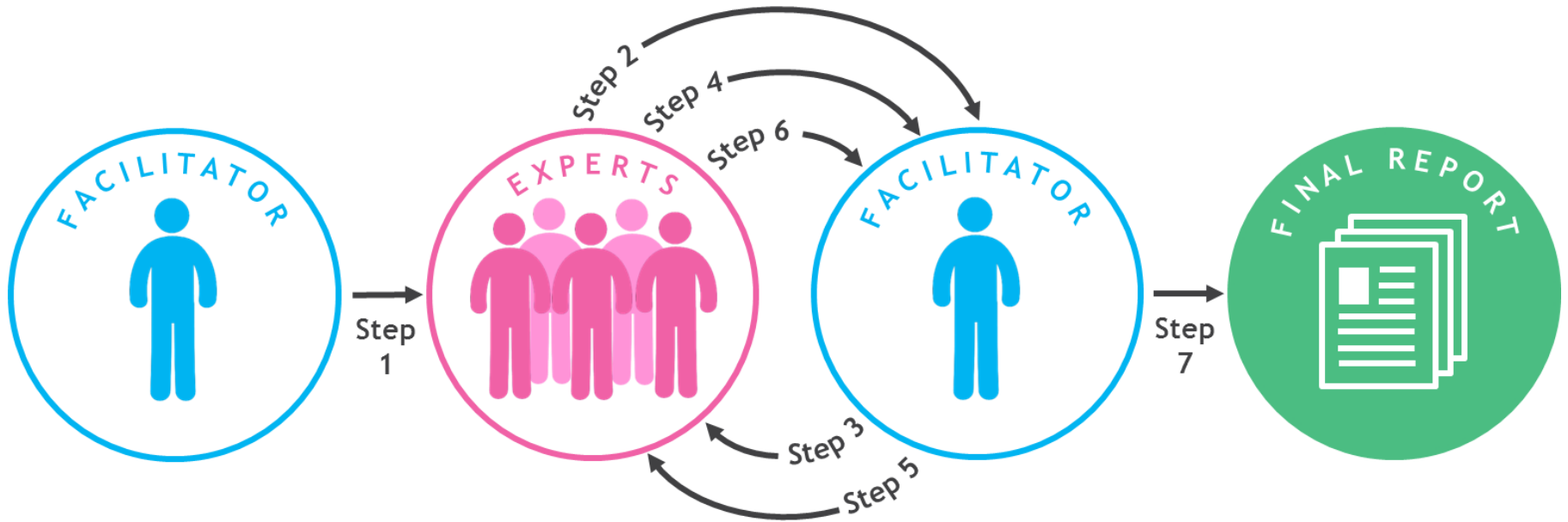
- It overcomes some of the shortcomings of the expert judgment approach.
- Delphi estimation is carried out by a team comprising of a **group of experts and a coordinator**.
- No discussion among the estimators is allowed during the entire estimation process.



# Delphi cost estimation

- The coordinator provides each estimator with a copy of the SRS document and a form for recording his cost estimate.
- Estimators complete their **individual estimates** anonymously and submit to the coordinator.
- The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators.
- Based on this summary, the **estimators re-estimate**.
- This process is iterated for several rounds.





Facilitator seeks individual assessments from a pool of experts.



Experts respond to the request, receive feedback and revise their responses.



Facilitator compiles the responses and sends a revised set of questions to each expert. Several cycles of feedback may be needed.



Facilitator produces report on experts' responses, noting key outliers.

# Heuristic Techniques

- It assumes that the **relationships among the different project parameters** can be modelled using **suitable mathematical expressions**.
- Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression.
- Two classes:
  - 1) single variable model and
  - 2) multi variable model

# Single variable estimation models

- It provides a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size.

$$\text{Estimated Parameter} = c_1 * e^{d_1}$$

- $e$  is the characteristic of the software which has already been estimated (independent variable)
- $c_1$  and  $d_1$  are constants determined using data collected from past projects (historical data).

# Multivariable cost estimation model

$$\text{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \dots$$

- Where  $e_1, e_2, \dots$  are the basic (independent) characteristics of the software already estimated, and  $c_1, c_2, d_1, d_2, \dots$  are constants.
- Multivariable estimation models are expected to give more accurate estimates.

# Introduction

- COCOMO is one of the most widely used software estimation models in the world
- It was developed by Barry Boehm in 1981
- COCOMO predicts the effort and schedule for a software product development based on inputs relating to the **size** of the software and a number of **cost drivers** that affect productivity

# **Types Of COCOMO**

```
graph TD; A[Types Of COCOMO] --> B[Basic Model]; A --> C[Intermediate Model]; A --> D[Detailed COCOMO Model];
```

**Basic Model**

**Intermediate  
Model**

**Detailed  
COCOMO Model**

# COnstructive COst Model (COCOMO)

- Combines statistical figures, mathematical equations, and expert judgement
- Widely used
- Three levels based on the level of details taken into account
  - – **Basic**: development effort is estimated as a function of program size in KLOC
  - – **Intermediate** : cost drivers are considered
  - – **Detailed**: cost drivers impact on each step of the development is considered

# COCOMO Models

- COCOMO has three different models that reflect the **complexity**:
  - the Basic Model
  - the Intermediate Model
  - and the Detailed Model



# Project Types/Modes

- – **Organic projects**: organization has a lot of experience doing such projects; requirements are less stringent (small and straightforward projects)
- – **Semi-detached projects**: medium-size, more complex (e.g., an operating system, or a compiler project)
- – **Embedded system projects**: stringent requirements, fairly complex, organization has little or no experience in that area

# Comparison of three COCOMO modes

Mode	Project Size	Nature of Project	Innovation	Deadline of the Project	Development Environment
Organic	Typically 2 – 50 KLOC	Small Size Projects, experienced developers.	Little	Not tight	Familiar And In house
Semi-Detached	Typically 50 – 300 KLOC	Medium size project, average previous experience on similar projects.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large projects, complex interfaces, very little previous experience.	Significant	Tight	Complex Hardware / Customer interfaces required

# Project Types

## **Organic:**

Routine project

- Well understood domain
- Team works well and efficiently together
- Project expected to run smoothly
- Typically a smaller system and smaller team

## **Embedded:**

- Difficulties expected
- Project that is hard (control software for a nuclear plant, or spacecraft)
- Team has little experience in domain
- New or inexperienced team
- Tend to be large projects with lots of constraints

# person-months (PM).

- The effort estimation is expressed in units of person-months (PM).
- Person month is a measurement unit for effort in software engineering.
- 
- An effort of 100 PM does not imply that 100 persons should work for 1 month nor does it imply that 1 person should be employed for 100 months, but it denotes the area under the person-month curve
- It is the area under the person-month plot.

# person-months (PM)

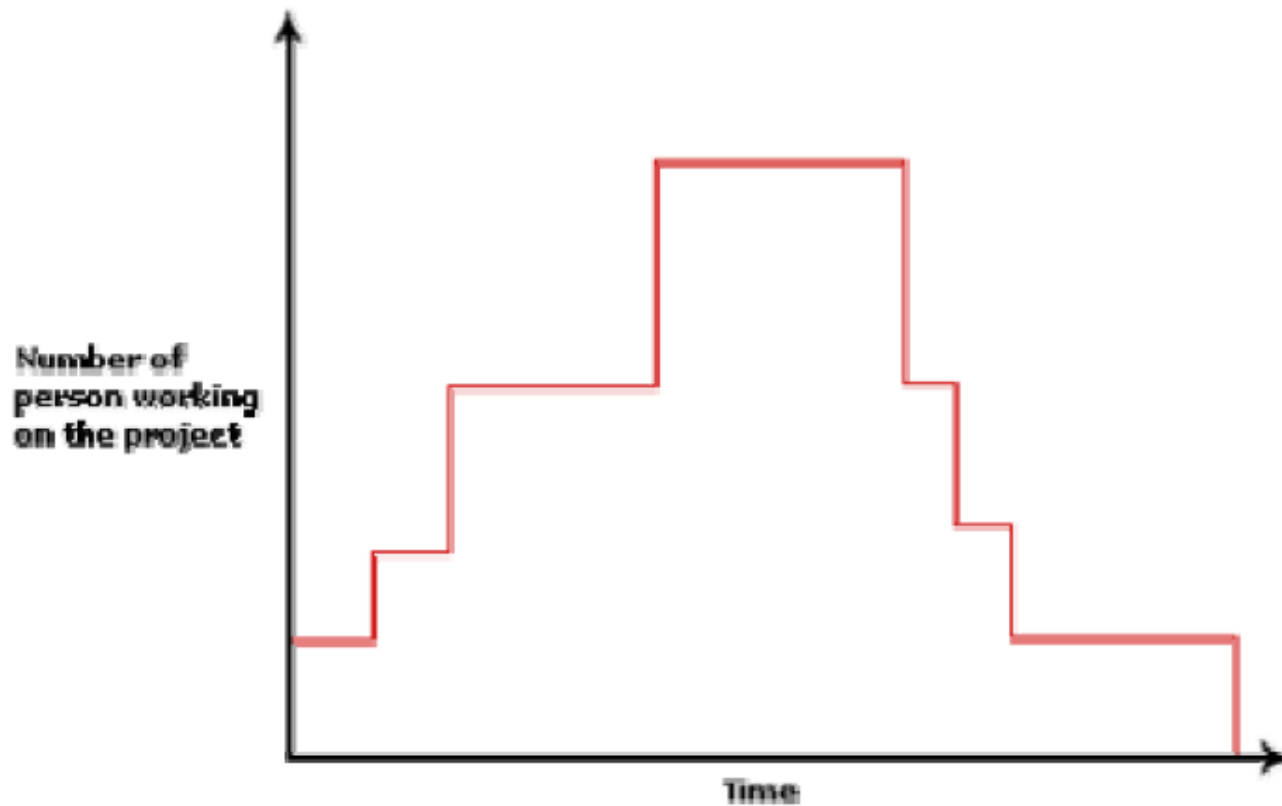


Fig. 11.3: Person-month curve

## Basic COCOMO Co- efficient

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

The Basic COCOMO equations take the form:

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

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

$$SS = E/D \text{ persons}$$

$$P = KLOC/E$$

E = effort

D = Deployment time

SS = staff size

P = productivity

$a_b, b_b, c_b, d_b$  = Coefficients

# Estimation of development effort

Organic	:	$\text{Effort} = 2.4(KLOC)^{1.05}$	PM
Semi-detached	:	$\text{Effort} = 3.0(KLOC)^{1.12}$	PM
Embedded	:	$\text{Effort} = 3.6(KLOC)^{1.20}$	PM



# Estimation of development time

Organic	:	$T_{dev} = 2.5(Effort)^{0.38}$	Months
Semi-detached	:	$T_{dev} = 2.5(Effort)^{0.35}$	Months
Embedded	:	$T_{dev} = 2.5(Effort)^{0.32}$	Months

# Example:

- Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code.
- Assume that the average cost of software be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

From the basic COCOMO estimation formula for organic software:

$$\text{Effort} = 2.4 \times (32)^{1.05} = 91 \text{ PM}$$

$$\text{Nominal development time} = 2.5 \times (91)^{0.38} = 14 \text{ months}$$

$$\begin{aligned} \text{Cost required to develop the product} &= 14 \times 15,000 \\ &= \text{Rs. } 210,000/- \end{aligned}$$

- 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 equations take the form:

$$E = a * (KLOC)^b$$

$$D = c * (E)^d$$

Estimated size of the project = 400 KLOC

➤ **1. Organic Mode**

- $E = 2.4 (400)^{1.05} = 1295.31 \text{ PM}$
- $D = 2.5 (1295.31)^{0.38} = 38.07 \text{ M}$

➤ **2. Semi detached Mode**

- $E = 3.0 (400)^{1.12} = 2462.79 \text{ PM}$
- $D = 2.5 (2462.79)^{0.35} = 38.45 \text{ M}$

➤ **3. Embedded Mode**

- $E = 3.6 (400)^{1.20} = 4772.81 \text{ PM}$
- $D = 2.5 (4772.81)^{0.32} = 37.59 \text{ M}$

# Example 3

- A large chemical products company, is planning to develop a new computer program to keep track of raw materials. It will be developed by an in-house team of programmers and analysts who have been developing similar programs for several years. An initial study has determined that the size of the program will be roughly 25,000 delivered source instructions.
- **Calculate efforts, productivity and time for development.**

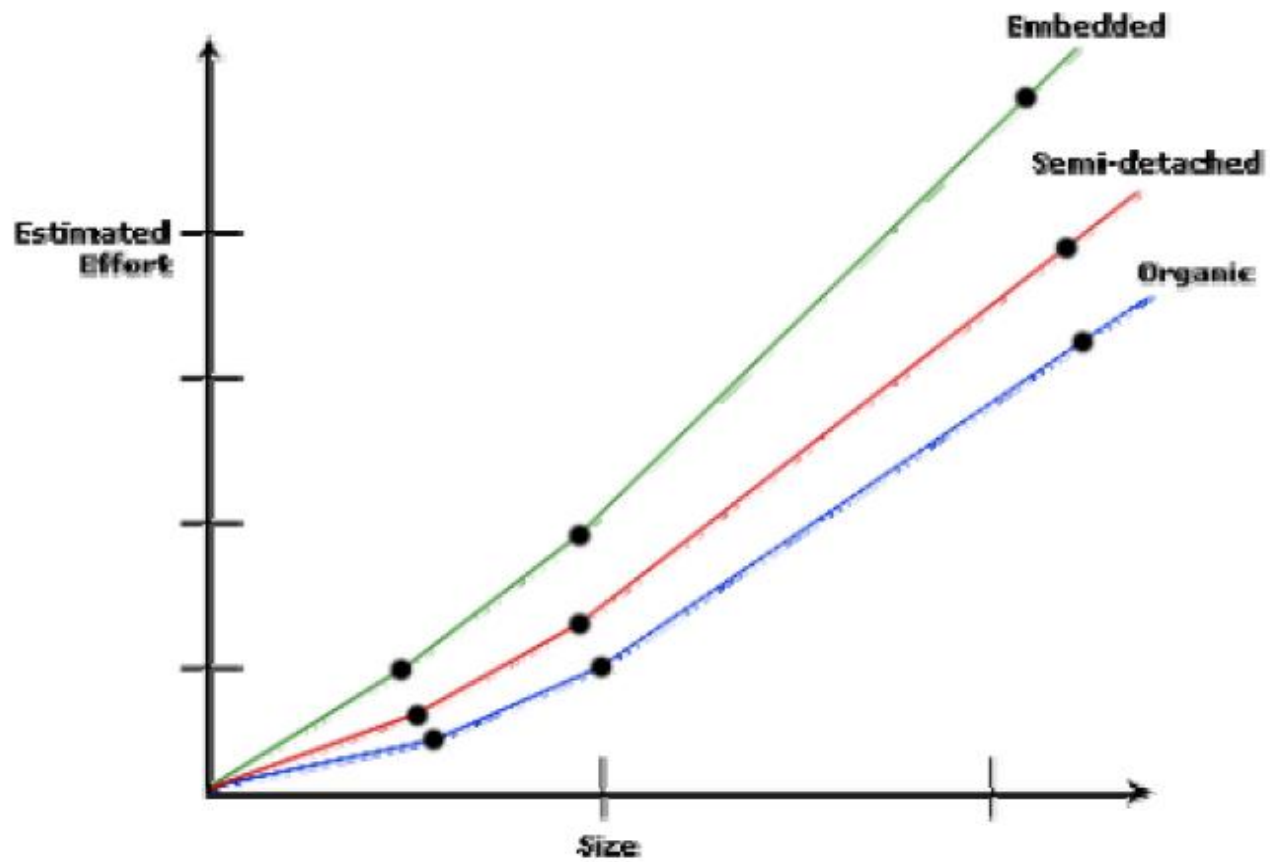
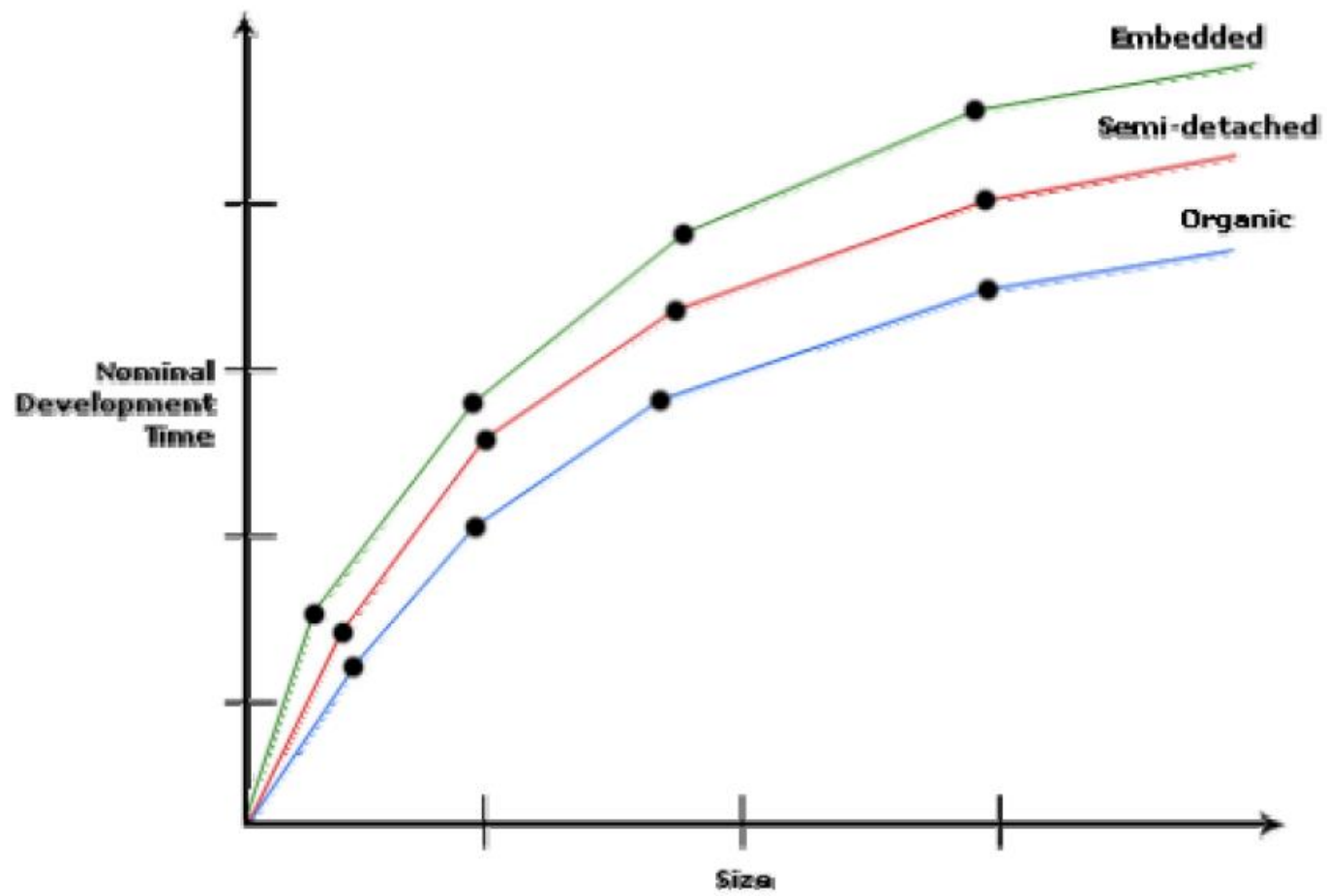


Fig. 11.4: Effort versus product size





# Intermediate COCOMO model

- The basic COCOMO model assumes that effort and development time are **functions of the product size** alone.
- However, a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time.
- Therefore, in order to obtain **an accurate estimation** of the effort and project duration, the **effect of all relevant parameters** must be taken into account.

# Intermediate COCOMO model

$$E = a (KLOC)^b \times C$$

New

■ Where

- E is the effort
- a and b are constants (as before)
- KLOC is thousands of lines of code
- C is the effort adjustment factor

# Intermediate COCOMO TABLE

The Intermediate COCOMO equations take the form:

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

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

$$\text{SS} = E/D \text{ persons}$$

$$P = \text{KLOC}/E$$

EAF = Effort Adjustment factor

E = effort

D = Deployment time

SS = staff size

P = productivity

$a_i, b_i, c_i, d_i$  = Coefficients

## Co- efficients for Intermediate COCOMO

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

# 15 Cost Drivers

- Boehm requires the project manager to **rate these 15 different parameters** for a particular project on a scale of one to three.
- Then, depending on these ratings, he suggests **appropriate cost driver values** which should be multiplied with the initial estimate obtained using the basic COCOMO.

# Classification of Cost Drivers

- Product:** The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc.
- Computer:** Characteristics of the computer that are considered include the execution speed required, storage space required etc.
- Personnel:** The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.
- Development Environment:** Development environment attributes capture the development facilities available to the developers. An important parameter that is considered is the sophistication of the automation (CASE) tools used for software development.

- In the Intermediate model Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment. Cost drivers are used to adjust the nominal cost of a project to the actual project environment, hence increasing the accuracy of the estimate.
- **The cost drivers are grouped into 4 categories:-**
- Product attributes
  - Required software reliability (RELY)
  - Databasesize(DATA)
  - Product complexity (CPLX)
- Computer attributes
  - Execution time constraint (TIME)
  - Main store constraint (STOR)
  - Virtual machine volatility (VIRT)
  - Computer turnaround time (TURN)

3. Personnel attributes
  - a. Analyst capability (ACAP)
  - b. Application experience (AEXP)
  - c. Programmer capability (PCAP)
  - d. Virtual machine experience (VEXP)
  - e. Programming Language experience (LEXP)
4. Project attributes
  - a. Morden programming practices (MODP)
  - b. Use of software tool (TOOL)
  - c. Required development schedule (SCED)

Each cost driver is rated for a given project environment. The rating uses a scale very low, low, nominal, high, very high, extra high which describes to what extent the cost driver applies to the project being estimated.



# Multiplier Values For Effort Calculations

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
<b>Product attributes</b>						
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
<b>Computer attributes</b>						
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	-



# Multiplier Values For Effort Calculations

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
<b>Personnel attributes</b>						
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	-	-
<b>Project attributes</b>						
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	-

## Example :

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 : with very high application experience and very little experience in the programming language being used or developers of very low application experience 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

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

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

**Case 1:** Developers are with very high application experience and very little experience in the programming language being used.

$$\text{EAF} = 0.82 * 1.14 = 0.9348$$

$$E = 2.8(400)^{1.20} * 0.9348 = 3470 \text{ PM}$$

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

**Case 2:** developers of very low application experience but a lot of experience with the programming language.

$$\text{EAF} = 1.29 * 0.95 = 1.22$$

$$E = 2.8 (400)^{1.20} * 1.22 = 4528 \text{ PM}$$

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

Case 2 requires more effort and time. Hence, low quality application experience but a lot of programming language experience could not match with the very high application experience and very little programming language experience.


# Project Characteristics Table

Cost adjustments for computing the EAF (Effort Adjustment Factor)

	v. low	low	nominal	high	v. high	ex. high
<b>product attributes</b>						
required software						
reliability	0.75	0.88	1.00	1.15	1.40	
database size		0.94	1.00	1.08	1.16	
product complexity	0.70	0.85	1.00	1.15	1.30	1.65
<b>computer attributes</b>						
execution time						
constraints			1.00	1.11	1.30	1.66
main storage constraints			1.00	1.06	1.21	1.56
virtual machine						
volatility	0.87	1.00	1.15	1.30		
computer turnaround time		0.87	1.00	1.07	1.15	
<b>personnel attributes</b>						
analyst capability	1.46	1.19	1.00	0.86	0.71	
applications experience	1.29	1.13	1.00	0.91	0.82	
programmer 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		
<b>project attributes</b>						
use of modern						
programming practices	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 development						
schedule	1.23	1.08	1.00	1.04	1.10	

**TABLE 8-6** Cost Driver Ratings: Microprocessor Communications Software

Cost Driver	Situation	Rating	Effort Multiplier
RELY	Local use of system. No serious recovery problems	Nominal	1.00
DATA	20,000 bytes	Low	0.94
CPLX	Communications processing	Very high	1.30
TIME	Will use 70% of available time	High	1.11
STOR	45K of 64K store (70%)	High	1.06
VIRT	Based on commercial microprocessor hardware	Nominal	1.00
TURN	Two-hour average turnaround time	Nominal	1.00
ACAP	Good senior analysts	High	0.86
AEXP*	Three years	Nominal	1.00
PCAP	Good senior programmers	High	0.86
VEXP	Six months	Low	1.10
LEXP	Twelve months	Nominal	1.00
MODP	Most techniques in use over one year	High	0.91
TOOL	At basic minicomputer tool level	Low	1.10
SCED	Nine months	Nominal	1.00
Effort adjustment factor (product of effort multipliers)			1.17

**1.17**

# Complete 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 subsystems may have widely different characteristics.
- For example, some subsystems may be considered as organic type, some semidetached, and some embedded.

# 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.**

# Staffing level estimation

- Once the effort required to develop a software has been determined, **it is necessary to determine the staffing requirement for the project.**
- Norden and Putnam studied the staffing patterns of several R & D projects.

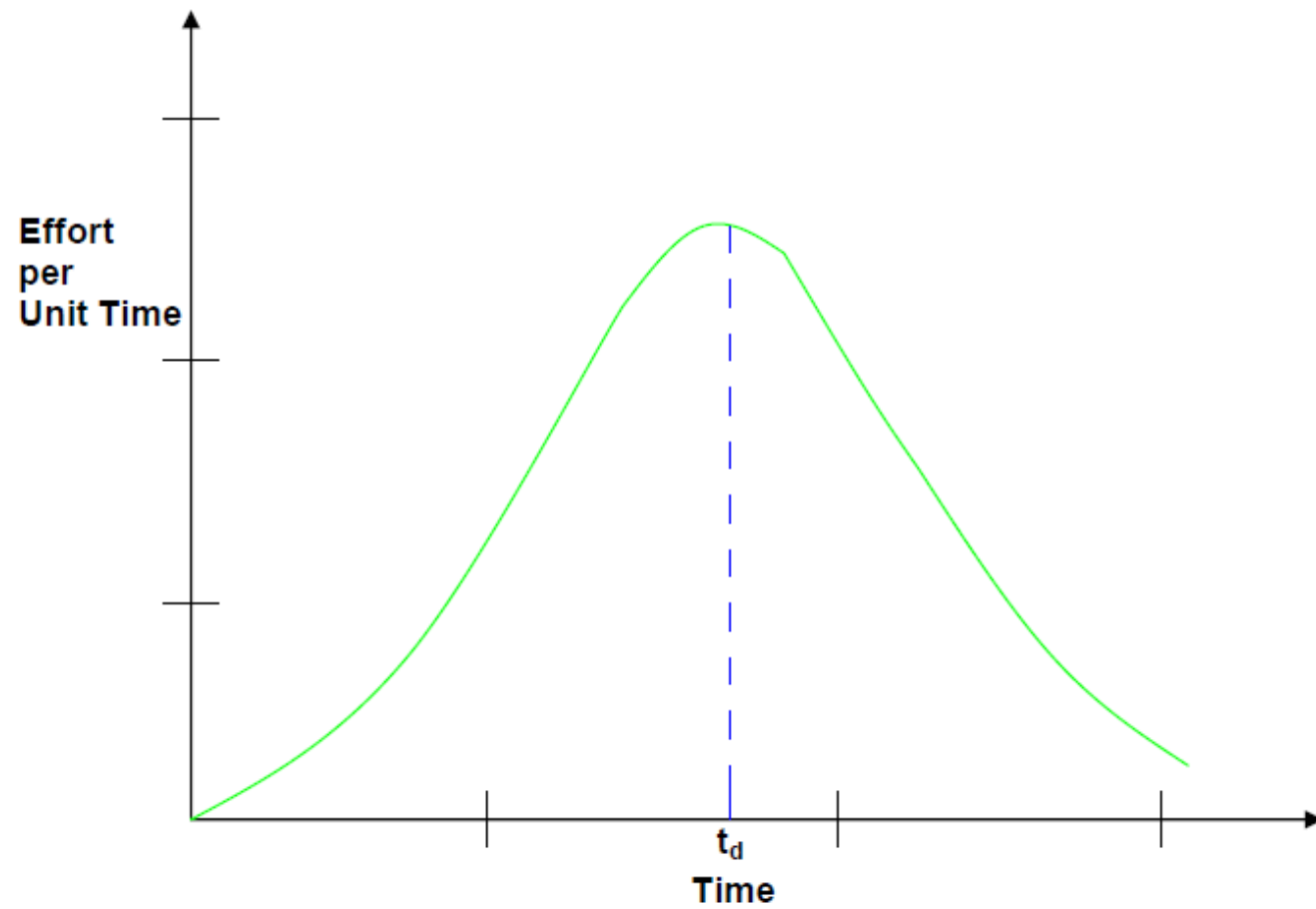
# Norden's Work

- He found that the staffing pattern can be approximated by the **Rayleigh distribution curve**.
- Norden represented the Rayleigh curve by the following equation:

$$E = K/t_d^2 * t * e^{-t^2 / 2 t_d^2}$$

- E is the effort required at time t. E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project
- K is the area under the curve, and
- $t_d$  is the time at which the curve attains its maximum value.





**Fig. 11.6:** Rayleigh curve

# Function Point Analysis

- What is Function Point Analysis (FPA)?
  - It is designed to estimate and measure the time, and thereby the cost, of developing new software applications and maintaining existing software applications.
  - It is also useful in comparing and highlighting opportunities for productivity improvements in software development.
  - It was developed by A.J. Albrecht of the IBM Corporation in the early 1980s.
  - The main other approach used for measuring the size, and therefore the time required, of software project is lines of code (LOC) – which has a number of inherent problems.

# How is Function Point Analysis done?

- Working from the project design specifications, the following system functions are measured (counted):
  1. Inputs
  2. Outputs
  3. Inquiries
  4. Files (Internal logical files)
  5. Interfaces (External)

# Project scheduling

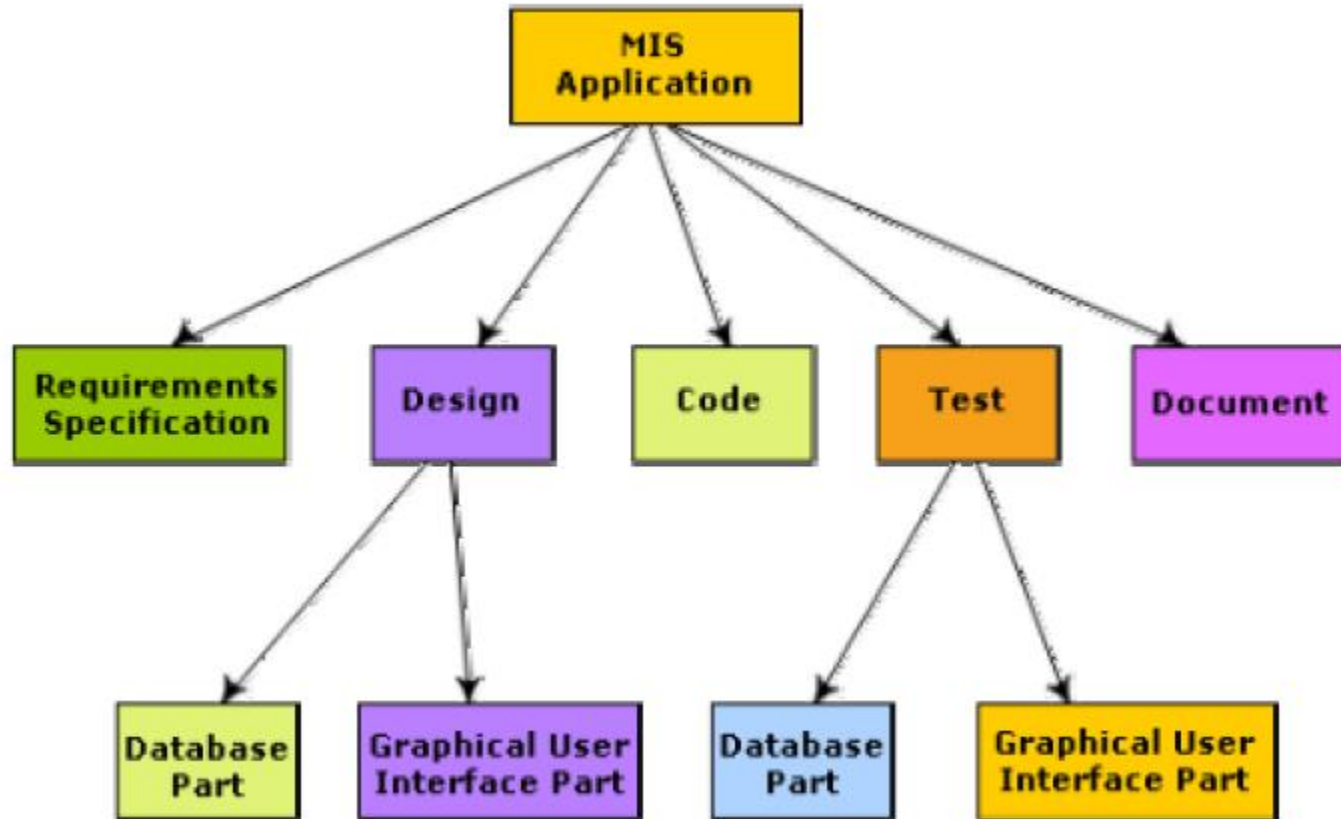
**It involves deciding which tasks would be taken up when. Activities undertaken:**

1. Identify all the tasks needed to complete the project.
2. Break down large tasks into small activities.
3. Determine the dependency among different activities.
4. Establish the most likely estimates for the time durations necessary to complete the activities.
5. Allocate resources to activities.
6. Plan the starting and ending dates for various activities.
7. Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

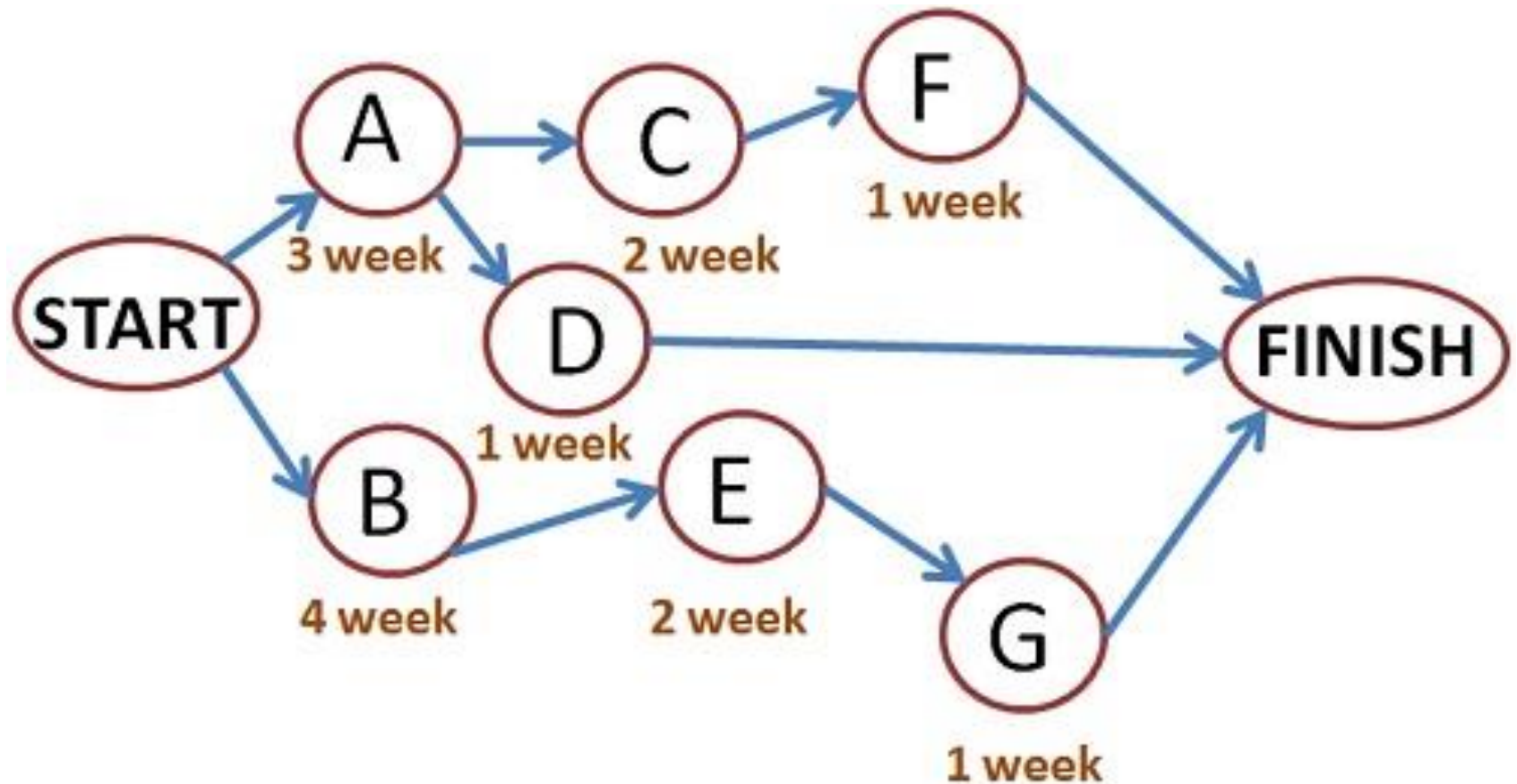
# Work breakdown structure

- Work Breakdown Structure (WBS) is used to decompose a given task set recursively into small activities.
- The root of the tree is labelled by the **problem name**.
- Each node of the tree is broken down into smaller activities that are made the children of the node.
- Each activity is recursively decomposed into smaller sub-activities until at the **leaf level**, the activities requires approximately **two weeks to develop**.

# Work breakdown structure of MIS problem



# Making ACTIVITY CHARTS and GRAPHS

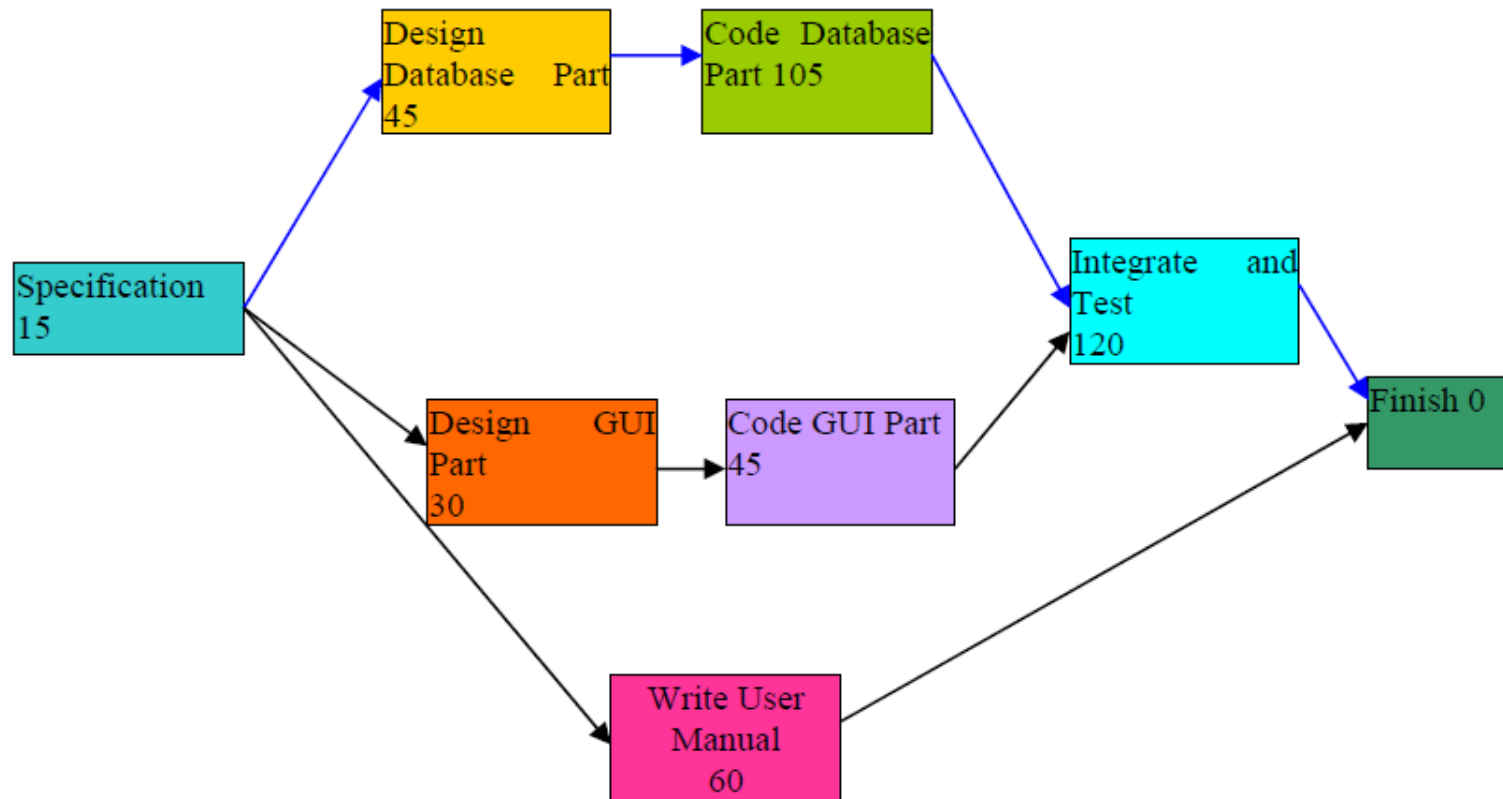


# Activity networks and critical path method

- WBS representation of a project is transformed into an activity network by representing activities identified in WBS along with their interdependencies.
- An activity network shows the different activities making up a project, their estimated durations, and interdependencies.



# Activity network representation of the MIS problem



# Critical Path Method (CPM)

- The minimum time (MT) to complete the project is the maximum of all paths from start to finish.
- The earliest start (ES) time of a task is the maximum of all paths from the start to the task.
- The latest start time is the difference between MT and the maximum of all paths from this task to the finish.
- The earliest finish time (EF) of a task is the sum of the earliest start time of the task and the duration of the task.
- The latest finish (LF) time of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.
- The slack time (ST) is  $LS - EF$  and equivalently can be written as **LF – EF**.

# Slack Time

- The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project.
- The slack time indicates the “flexibility” in starting and completion of tasks.
- A critical task is one with a zero slack time. A path from the start node to the finish node containing only critical tasks is called a **critical path**.

# Parameters for different tasks of MIS Problem

Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210

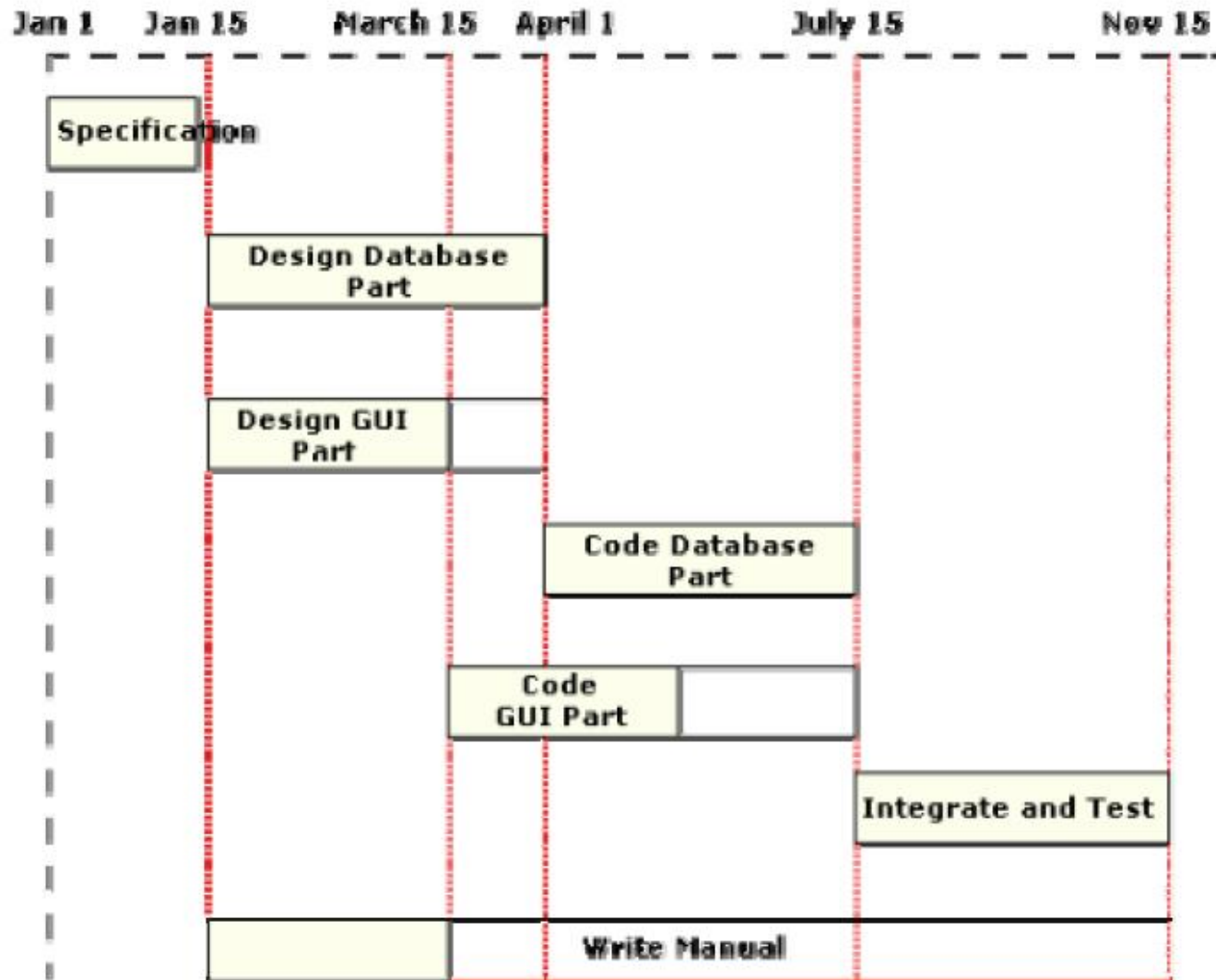
# Gantt chart

- Used to allocate resources to activities (resource planning).

The resources allocated to activities include staff, hardware, and software.

- A Gantt chart is a special type of bar chart where each bar represents an activity.
  - The bars are drawn along a time line.
  - The length of each bar is proportional to the duration of time planned for the corresponding activity.

# Gantt chart for MIS problem



# PERT chart

- PERT (Project Evaluation and Review Technique) charts consist of a network of boxes and arrows.
- The boxes represent activities and the arrows represent task dependencies.
- The boxes of PERT charts are usually annotated with the pessimistic, likely, and optimistic estimates for every task.

- Gantt chart representation of a project schedule is helpful in planning the utilization of resources, while PERT chart is useful for monitoring the timely progress of activities.
- Also, it is easier to identify parallel activities in a project using a PERT chart.



# PERT chart representation of the MIS problem

