Software Project Management

Organization of this Lecture:

- Introduction to Project Planning
- Software Cost Estimation
  - Cost Estimation Models
  - Software Size Metrics
  - Empirical Estimation
  - Heuristic Estimation
  - COCOMO
- Staffing Level Estimation
- Effect of Schedule Compression on Cost
- Summary

Introduction

- Many software projects fail:
  - due to faulty project management practices:
    - It is important to learn different aspects of software project management.

- Goal of software project management:
  - enable a group of engineers to work efficiently towards successful completion of a software project.

Responsibility of project managers

- Project proposal writing,
- Project cost estimation,
- Scheduling,
- Project staffing,
- Project monitoring and control,
- Software configuration management,
- Risk management,
- Managerial report writing and presentations, etc.

Introduction

- A project manager's activities are varied.
  - can be broadly classified into:
    - project planning,
    - project monitoring and control activities.

- Once a project is found to be feasible,
  - project managers undertake project planning.

Project Planning Activities

- Project planning consists of the following essential activities:

- 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?

Project Planning Activities

- The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.
  - **Scheduling** manpower and other resources
  - **Staff organization** and staffing plans
  - **Risk identification**, analysis, and moderation planning
  - **Miscellaneous plans** such as quality assurance plan, configuration

    management plan, etc.

Project planning

- Requires utmost care and attention --- commitments to unrealistic time and resource estimates result in:
  - irritating delays.
  - customer dissatisfaction
  - adverse affect on team morale
    - poor quality work
  - project failure.

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.**

- In the sliding window technique, starting with an initial plan, the project is **planned more accurately in successive development stages.**

- At the start of a project, project managers have incomplete knowledge about the details of the project.

- Their information base gradually improves as the project progresses through different phases.

- After the completion of every phase, the project managers can plan each subsequent phase more accurately and with increasing levels of confidence.

Sliding Window Planning

- Involves project planning over several stages:
  - protects managers from making big commitments too early.
  - More information becomes available as project progresses.
    - Facilitates accurate planning

Organization of SPMP Document

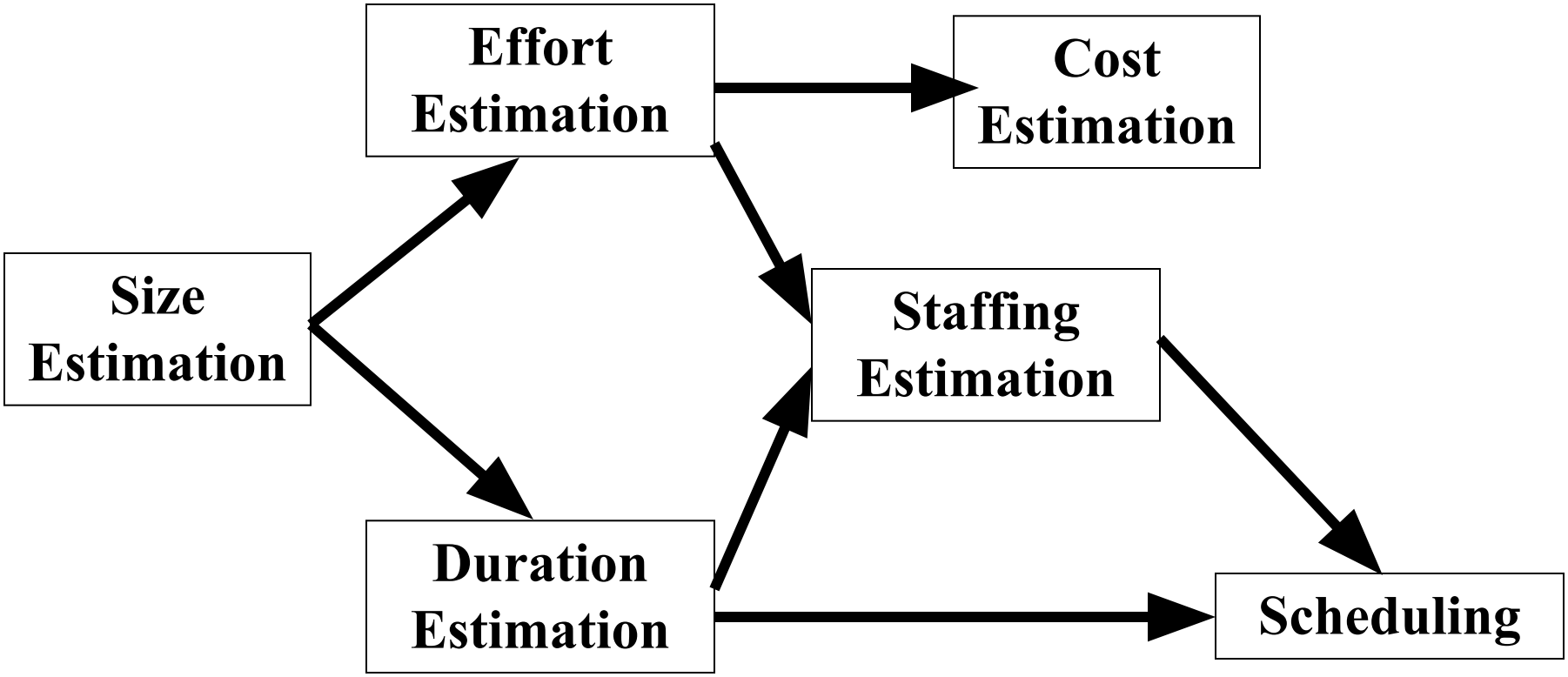
- After planning is complete:
  - Document the plans in a **Software Project Management Plan(SPMP) document.**
    - Introduction (Objectives, Major Functions, Performance Issues, Management and Technical Constraints)
    - Project Estimates (Historical Data, Estimation Techniques, Effort, Cost, and Project Duration Estimates)
    - Project Resources Plan (People, Hardware and Software, Special Resources)
    - Schedules (Work Breakdown Structure, Task Network, Gantt Chart Representation, PERT Chart Representation)
    - Risk Management Plan (Risk Analysis, Risk Identification, Risk Estimation, Abatement Procedures)
    - Project Tracking and Control Plan
    - Miscellaneous Plans(Process Tailoring, Quality Assurance)

Software Project Size Estimation

- The project size is a measure of the problem complexity in terms of the **effort and time** required to develop the product.

- Two metrics widely used to estimate size:
  - lines of code (LOC)
  - function point (FP).

Software Project Size Estimation



Software Cost Estimation

- Three main approaches to estimation:
  - Empirical
  - Heuristic
  - Analytical

Software Project Size/Cost Estimation

- Empirical techniques:
  - an educated guess based on past experience.
- Heuristic techniques:
  - assume that the characteristics to be estimated can be expressed in terms of some mathematical expression.
- Analytical techniques:
  - derive the required results starting from certain simple assumptions.

Software Size Metrics

- LOC (Lines of Code):
  - Simplest among all metrics available to estimate project size.
  - Using this metric, the project size is estimated by counting the number of source instructions in the developed program.
  - 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.
- To estimate project managers usually divide the problem into modules, submodules until the sizes of the different leaf-level modules can be approximately predicted.
- 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.

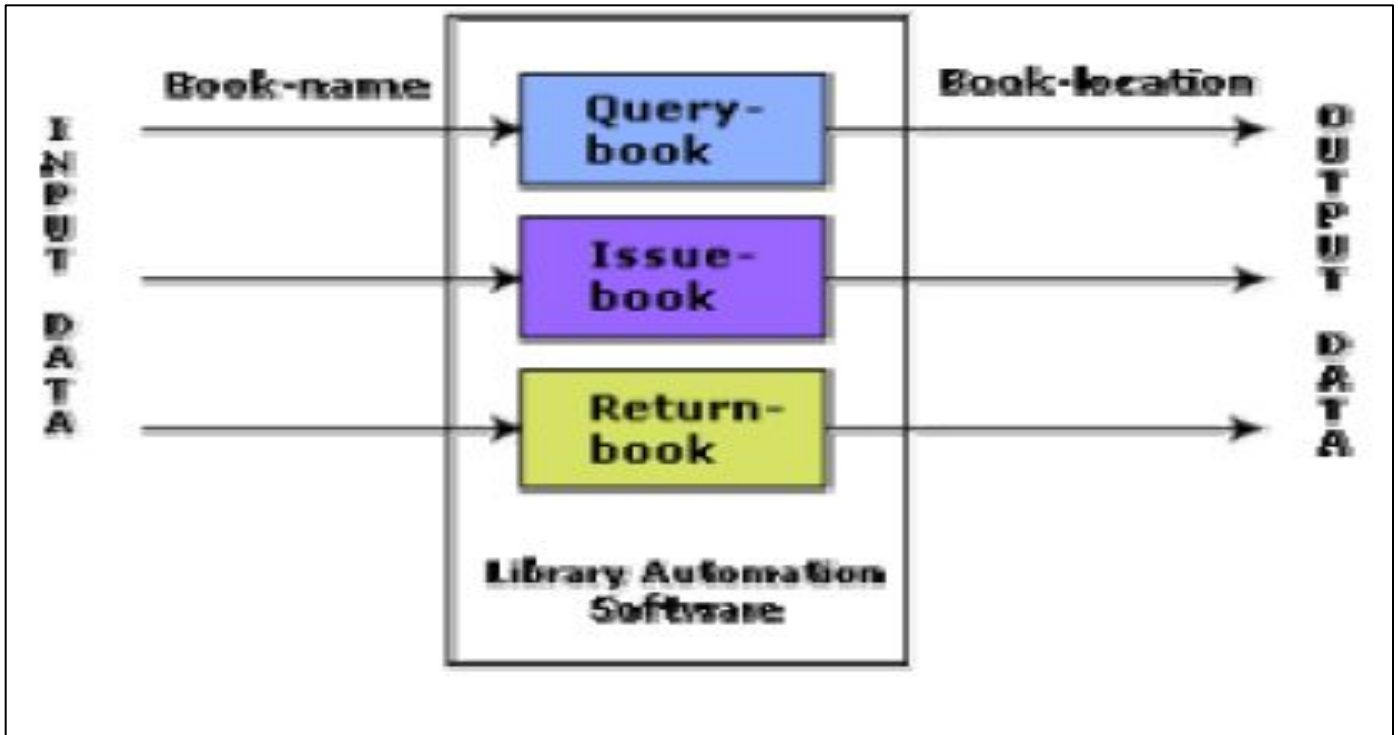
Disadvantages of Using LOC

- Size can vary with coding style.
- Focuses on coding activity alone.
- Correlates poorly with quality and efficiency of code.
- Penalizes higher level programming languages, code reuse, etc.
- Difficult to estimate LOC from problem description.
  - So not useful for project planning

Function Point Metric

- Overcomes some of the shortcomings of the LOC metric
- Estimate the size of a software product directly from the problem specification.
- Size of a software product is directly dependent on the number of different functions or features it supports.

- For example, the issue book feature of a Library Automation Software takes the name of the book as input and displays its location and the number of copies available.
- Thus, a computation of the number of input and the output data values to a system gives some indication of the number of functions supported by the system.
- Albrecht postulated that in addition to the number of basic functions that a software performs, the size is also dependent on the number of files and the number of interfaces.



FP calculations

- Calculate UFP (Unadjusted FP)

1) Refine Parameters
2) Refine UFP based on complexity

Step1: Calculate UFP

Determine the number of components (EI, EO, EQ, ILF, and ELF)

1. **EI – The number of external inputs.** These are elementary processes in which derived data passes across the boundary from outside to inside. In an example library database system, enter an existing patron's library card number.

2. **EO – The number of external output**. These are elementary processes in which derived data passes across the boundary from inside to outside. In an example library database system, display a list of books checked out to a patron.

3. **EQ – The number of external queries.** These are elementary processes with both input and output components that result in data retrieval from one or more internal logical files and external interface files. In an example library database system, determine what books are currently checked out to a patron.

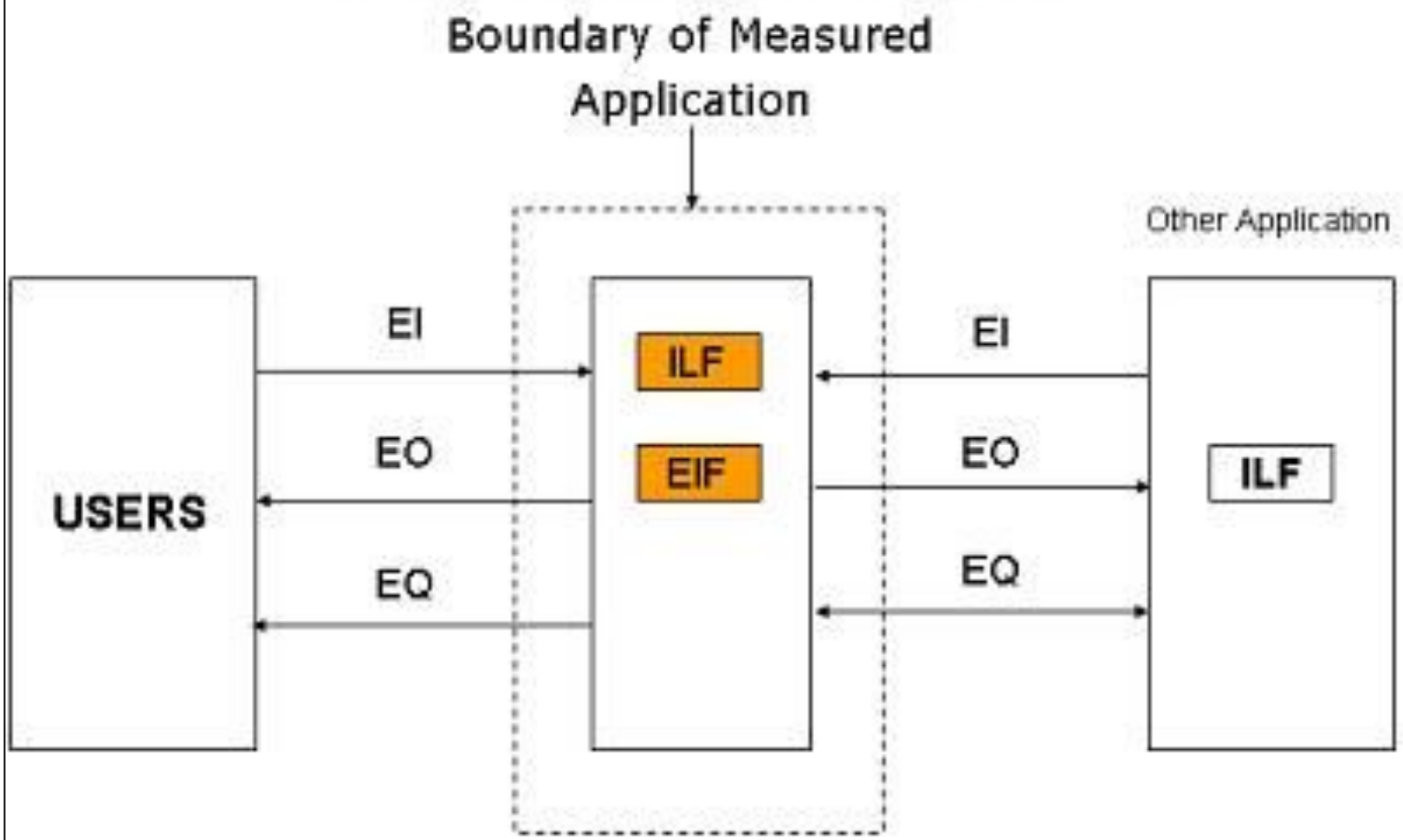
4. **ILF – The number of internal log files**. These are user identifiable groups of logically related data that resides entirely within the applications boundary that are maintained through external inputs. In an example library database system, the file of books in the library.

5. **ELF – The number of external log files**. These are user identifiable groups of logically related data that are used for reference purposes only, and which reside entirely outside the system. In an example library database system, the file that contains transactions in the library's billing system.

Proposed by Albrecht in early 80's:

FP=4 * #inputs + 5 * #Outputs + 4 * #inquiries + 10 * #files + 10 * #interfaces

Function Point Model



Some more clarity on factors

- **Number of inputs:** Each data item input by the user is counted.
  - Data inputs and inquiries are different. Inquiries are user commands such as **print-account-balance**.
  - Individual data items input by the user are not considered in the calculation of the number of inputs, but a group of related inputs are considered as a single input.
  - Example, while entering the data concerning an employee to an employee pay roll software; the data items name, age, address, phone number, etc. are together considered as a single input.

- **Number of outputs:** The outputs considered refer to reports printed, screen outputs, error messages etc.
- **Number of inquiries:** user commands which require specific action by the system.
- **Number of files:** Each logical file is counted.
- **Number of interfaces:** Used to exchange information with other systems. Examples of such interfaces are data files on tapes, disks, communication links with other systems etc.

FP=4 * #inputs + 5 * #Outputs + 4 * #inquiries + 10 * #files + 10 * #interfaces

Measurement parameter	Count	Weighting Factor(Average)	Total
Number of user inputs	50	4	=200
Number of user outputs	40	5	=200
Number of user enquiries	35	4	=140
Number of user files	6	10	=60
Number of external interfaces	4	7	=28

$(200 + 200 + 140 + 60 + 28) = 628$

step2 : Refine parameters

Complexity level of each parameter is graded as simple, average , complex. The weight for the different parameter can be compute based on the following table-

Type	Simple	Average	complex
Input	3	4	6
Output	4	5	7
Inquiry	3	4	6
Number of files	7	10	15
Number of interfaces	5	7	10

step3 :Refine UFP based on complexity

- Then technical complexity factor(TCF)for the project is computed and the TCF is multiply with UFP to get FP.
- There are 14 parameter that can be influence the development effort valued form 0(no influence)-6(strong influence). The resulting numbers are summed to get degree of influence(DI). TCF=(.65+.01*DI)
- DI vary from 0-84 and TCF vary from .65-1.35.
- Finally FP is given as a product of UFP and TCF.
- FP=UFP*TCF.

General System Characteristic		Brief Description
1.	Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2.	Distributed data processing	How are distributed data and processing functions handled?
3.	Performance	Was response time or throughput required by the user?
4.	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
5.	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
6.	On-Line data entry	What percentage of the information is entered On-Line?
7.	End-user efficiency	Was the application designed for end-user efficiency?
8.	On-Line update	How many ILF's are updated by On-Line transaction?
9.	Complex processing	Does the application have extensive logical or mathematical processing?
10.	Reusability	Was the application developed to meet one or many user's needs?
11.	Installation ease	How difficult is conversion and installation?
12.	Operational ease	How effective and/or automated are start-up, back-up, and recovery procedures?
13.	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14.	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

Calculate FP problem

Q: 1

Function point = `FP = UPF x VAF`

`UFP` = Sum of all the complexities i.e. the 5 parameters provided in the question,
`VAF` = Value added Factor i.e. `0.65 + (0.01 * TDI)`,
`TDI` = Total Degree of Influence of the 14 General System Characteristics.

Measurement parameter	Count	Weighting Factor(Average)	Total
Number of user inputs	50	4	=200
Number of user outputs	40	5	=200
Number of user enquiries	35	4	=140
Number of user files	6	10	=60
Number of external interfaces	4	7	=28

Thus function points can be calculated as:

```
= (200 + 200 + 140 + 60 + 28) x (0.65 + (.01 x (14 x 3))
= 628 x (0.65 + 0.42)
= 628 x (1.07)
= 672
```

Calculate FP problem

Q : 2 Determine the function point measure of the size of the following supermarket software. A supermarket needs to develop the following software to encourage regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique *customer number* (CN) by the computer. Based on the generated CN, a clerk manually prepares a customer identity card after getting the market manager's signature on it. A customer can present his customer identity card to the check out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded ₹10,000. The entries against the CN are reset on the last day of every year after the prize winners' lists are generated. Assume that various project characteristics determining the complexity of software development to be average.

UFP = (2*4) +(3*5) + (1*4)+(2*10) +(0*10)

- **EI - INPUT -**
  1. Registration form fill up.
  2. Purchase
- **EO - Output**
  1. Successful registration process, customer data cerated
  2. Update purchase record
  3. Entries against CN are reset at the last of the year
- **EQ - Queries (only fetch, no updates)**
  1. Get the annual report of sales against each customer
- **Internal files (Data)**
  1. Customer details
  2. Daily purhase details.
- **External Interfaces** - None

Refined

UFP = (2*4) + {(2*5) +(1*4)} + (1*4)+(2*10) +(0*10) = 46

- Technical Complexity Factor (TCF)
- Take avg = 3
- TCF = 0.65+ 0.01*DI
- TCF = 0.65 + 0.01 *(14*3) = 1.07
- FP = TCF * UFP

  = 46 * 1.07

  = 49.22

Function Point Metric (CONT.)

- Suffers from a major drawback:
  - the size of a function is considered to be independent of its complexity.
- Extend function point metric:
  - Feature Point metric:
  - considers an extra parameter:
    - **Algorithm Complexity:** Greater is the effort required to develop it and therefore its size should be larger compared to simpler functions.

Function Point Metric (CONT.)

- Proponents claim:
  - FP is language independent.
  - Size can be easily derived from problem description
- Opponents claim:
  - it is subjective --- Different people can come up with different estimates for the same problem.

Software Cost Estimation

- Three main approaches to estimation:
  - Empirical
  - Heuristic
  - Analytical

Empirical Size Estimation Techniques

- Based on making an educated guess of the project parameters. Prior experience with development of similar products is helpful.
- Two popular empirical estimation techniques are:

1)Expert judgment technique and

2) Delphi cost estimation.

- Expert Judgement: 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. Suffers from individual bias.
  - Experts divide a software product into component units:
    - e.g. GUI, database module, data communication module, billing module, etc.
  - Add up the guesses for each of the components.

Delphi Estimation:

- Delphi Estimation:
  - Overcomes some of the problems of expert judgement.

- Team of Experts and a coordinator.
- Experts carry out estimation independently:
  - mention the unusual characteristic of the product which has influenced his estimation.
  - coordinator notes down any extraordinary rationale:
    - circulates among experts.
- Experts re-estimate.
- Experts never meet each other to discuss their viewpoints as many estimators may easily get Influenced.

Heuristic Estimation Techniques

Single Variable Model:

- Models provide 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, staff etc.

Estimated Parameter=c1*ed1

e= characteristic which already have been calculated.

c1 and d1 are constants- calculated from past projects.

Multivariable Model:

- Assumes that the parameter to be estimated depends on more than one characteristic.
- **Estimated Resources=c1*e1d1+c2*e1d1+----**
- e1 and e2 are the basic independent characteristics of the software already estimated.
- c1, c2, d1, d2, are constants.
- Multivariable Estimation Models are expected to give more accurate estimate compared to the Single Variable Models.

COCOMO Model

- COCOMO (COnstructive COst MOdel) is a Constructive Cost Estimation Model proposed by Boehm.

- COCOMO Product classes correspond to:
  - application, utility and system programs respectively.
    - Data processing and scientific programs are considered to be application programs.
    - Compilers, linkers, editors, etc., are utility programs.
    - Operating systems and real-time system programs, etc. are system programs.

Elaboration of Product classes

- Divides software product developments into 3 categories:
- Organic:
  - Relatively small groups
    - working to develop well-understood applications.
- Semidetached:
  - Project team consists of a mixture of experienced and inexperienced staff.
- Embedded:
  - The software is strongly coupled to complex hardware, or real-time systems.

Cocomo product categories

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

COCOMO Model (CONT.)

- For each of the three product categories:
  - From size estimation (in KLOC), Boehm provides equations to predict:
    - project duration in months
    - effort in programmer-months

- Boehm obtained these equations:
  - examined historical data collected from a large number of actual projects.

COCOMO Model (CONT.)

- Software cost estimation is done through three stages:
  - Basic COCOMO,
  - Intermediate COCOMO,
  - Complete COCOMO.

Useful terms

Person months : *Person month is a measurement unit for effort in software engineering. 1 person month means effort put by a person in one month. But 100 person does not mean, work effort put by 100 person in one month or 1 person in 100 months.*

- **Effort:** Amount of labor that will be required to complete a task. It is measured in **person-months units**.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, **months**.
- **Productivity** : Size (KLOC) / Effort (PM) (KLOC per person-month)
- **Avg Staffing :** Effort (PM) / Schedule (M)

  Unit : Persons

Basic COCOMO Model

- It take the form:

Effort(E) = a_b * (KLOC)$^{b}{}_{b}$ (in Person-months)

DevelopmentTime(D) = c_b * (E) $^{d}{}_{b}$ (in month)

Tdev

Average staff size(SS) = E/D (in Person)

Productivity(P) = KLOC / E (in KLOC/Person-month)

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

Development Effort Estimation

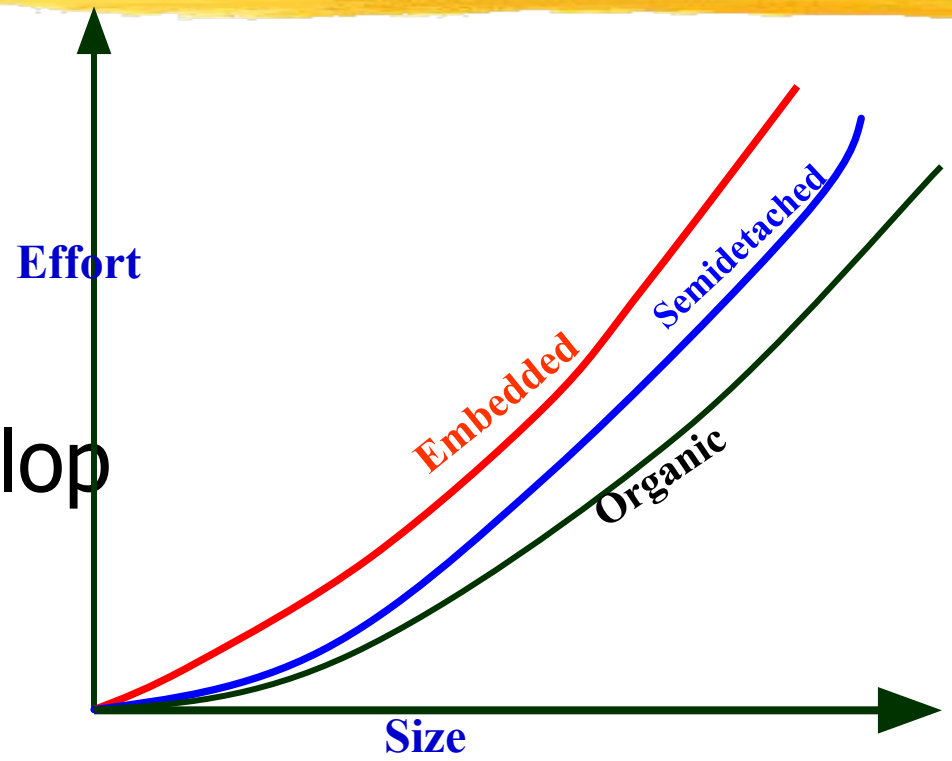
- Organic :
  - Effort = 2.4 (KLOC)1.05 PM
- Semi-detached:
  - Effort = 3.0(KLOC)1.12 PM
- Embedded:
  - Effort = 3.6 (KLOC)1.20PM

Development Time Estimation

- Organic:
  - Tdev = 2.5 (Effort)0.38 Months
- Semi-detached:
  - Tdev = 2.5 (Effort)0.35 Months
- Embedded:
  - Tdev = 2.5 (Effort)0.32 Months

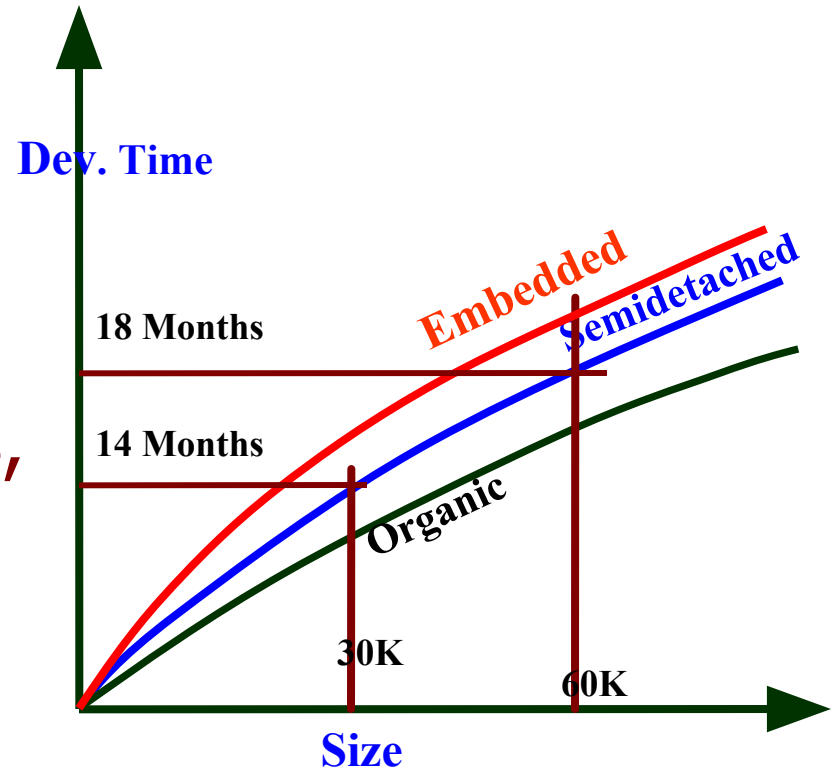
Basic COCOMO Model (CONT.)

- Effort is somewhat super-linear(slop e >1) in problem size.



Basic COCOMO Model (CONT.)

- Development time
  - sublinear(slope<1) function of product size.
- When product size increases two times,
  - development time does not double.
- Time taken:
  - almost same for all the three product categories.



Basic COCOMO Model (CONT.)

- Development time does not increase linearly with product size:
  - For larger products more parallel activities can be identified:
    - can be carried out simultaneously by a number of engineers.

Basic COCOMO Model (CONT.)

- Development time is roughly the same for all the three categories of products:
  - For example, a 60 KLOC program can be developed in approximately 18 months
    - regardless of whether it is of organic, semi-detached, or embedded type.
  - There is more scope for parallel activities for system and application programs,
    - than utility programs.

Basic COCOMO Example

Topic :- Constructive Cost Model (COCOMO)

Example :

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:

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

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

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**

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

$$D = 2.5 (4772.81)^{0.32} = 37.59 \text{ M}$$

Basic COCOMO Model: Example

- We have determined our project fits the characteristics of **Semi-Detached** mode
- We estimate our project will have **32,000** Delivered Source Instructions. Using the formulas, we can estimate:
- **Effort** = $3.0*(32)^{1.12}$ = 146 man-months
- **Schedule** = $2.5*(146)^{0.35}$ = 14 months
- **Productivity** = 32,000 DSI / 146 MM
  = 219 DSI/MM
- **Average Staffing** = 146 MM / 14 months
  = 10 FSP

Basic cocomo: Example

- A project size of 200KLOC is to be developed.S/W development team has average experience on similar type of projects.The project schedule is not very tight.Calculate the effort and development time of the project.

Ans: 200 KLOC implies semi-detached mode.

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

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

Avg. staff size(SS) = E/D

= 1133.12/29.3=38.67 Persons.

Productivity (P) = KLOC/E

= 200/1133.12=0.1765 KLOC/PM.

Example

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

- Effort = 2.4*(32)1.05 = 91 PM
- Nominal development time = 2.5*(91)0.38 = 14 months
- Cost required to develop the product = **14 x 15,000**
  = **Rs. 210,000/-**

Intermediate COCOMO

- Basic COCOMO model assumes
  - effort and development time depend on product size alone.
- However, several parameters affect effort and development time:
  - Reliability requirements
  - Availability of CASE tools and modern facilities to the developers
  - Size of data to be handled

Intermediate COCOMO

- For accurate estimation,
  - the effect of all relevant parameters must be considered:
  - Intermediate COCOMO model recognizes this fact:
    - refines the initial estimate obtained by the basic COCOMO by using a set of 15 cost drivers (multipliers).

Classification of Cost Drivers and their attributes:

(i) Product attributes –

- Required software reliability extent
- Size of the application database
- The complexity of the product

(ii) Hardware attributes –

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

(iii) Personnel attributes –

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

(iv) Project attributes –

- Use of software tools
- Application of software engineering methods
- Required development schedule

Intermediate COCOMO
(CONT.)

- If modern programming practices are used,
  - initial estimates are scaled downwards.
- If there are stringent reliability requirements on the product :
  - initial estimate is scaled upwards.

Project Characteristics Table

Cost adjustments for computing the EAF (Effort Adjustment Factor)

	v.low	low	nominal	high	v.high	ex.high
product attributes						
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
computer attributes						
execution time constraints			1.00	1.11	1.30	1.66
main storage constraints			1.00	1.06	1.21	1.56
virtual machine volitility	0.87	1.00	1.15	1.30		
computer turnaround time		0.87	1.00	1.07	1.15	
personnel attributes						
analyst capability	1.46	1.19	1.00	0.86	0.71	
applications experience	1.29	1.13	1.00	0.91	0.82	
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		
project attributes						
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	

Intermediate COCOMO (CONT.)

- Cost driver classes:
  - Product: characteristics of the product that include inherent complexity of the product, reliability requirements of the product, etc.
  - Computer: Execution time, storage requirements, etc.
  - Personnel: Experience of personnel, programming capability, analysis capability, etc.
  - Development Environment: Sophistication of the tools used for software development.

Intermediate COCOMO (CONT.)

- **Effort Adjustment Factor**
- The Effort Adjustment Factor in the effort equation is simply the product of the effort multipliers corresponding to each of the cost drivers for your project.
- For example, if your project is rated Very High for Complexity (effort multiplier of 1.34), and Low for Language & Tools Experience (effort multiplier of 1.09), and all of the other cost drivers are rated to be Nominal (effort multiplier of 1.00), the EAF is the product of 1.34 and 1.09.
- **Effort Adjustment Factor = EAF = 1.30 * 1.10 = 1.43**

The Calculation

- Multiply all 15 Cost Drivers to get **Effort Adjustment Factor(EAF)**
- **E(Effort) = a_b(KLOC)^{b_b} * EAF** (in Person-Month)
- **D(Development Time) = c_b(E)^{d_b}** (in month)
- **SS (Avg Staff Size) = E/D** (in persons)
- **P (Productivity) = KLOC/E** (in KLOC/Person-month)

Project	a_b	b_b	c_b	d_b
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

Intermediate Model: An Example

- Project A is to be a **32,000 DSI semi-detached** software. It is in a mission critical area, so the **reliability** is high (RELY=high=1.15). Then we can estimate:
- **Effort** = $1.15*3.0*(32)^{1.12}$ = 167 man-months
- **Schedule** = $2.5*(167)^{0.35}$ = 15 months
- **Productivity** = 32,000 DSI/167 MM
  = 192 DSI/MM
- **Average Staffing** = 167 MM/15 months
  = 11 FSP

DSI = Delivered Source Instructions/LOC

Shortcoming of basic and intermediate COCOMO models

- Both models:
  - consider a software product as a single homogeneous entity:
  - However, most large systems are made up of several smaller sub-systems.
    - Some sub-systems may be considered as organic type, some may be considered embedded, etc.
    - for some the reliability requirements may be high, and so on.

Complete COCOMO

- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

- **Example:** A Management Information System (MIS) for an organization having offices at several places across the country:
  - Database part (semi-detached)
  - Graphical User Interface (GUI) part (organic)
  - Communication part (embedded)
- Costs of the components are estimated separately:
  - summed up to give the overall cost of the system.

Analytical Estimation Techniques

- Analytical estimation techniques derive the required results starting with basic assumptions regarding the project.

- Analytical techniques do have scientific basis.

- Halstead's software science is an example of an analytical technique.
  - Can be used to derive some interesting results starting with a few simple assumptions.
  - Useful for estimating software maintenance efforts.
  - In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.

Halstead's Software Science

- An analytical technique to estimate:
  - size,
  - development effort,
  - development time.

- Halstead used a few primitive program parameters
  - number of operators and operands (unique, total)
- Derived expressions for:
  - over all program length,
  - potential minimum volume
  - actual volume,
  - language level,
  - effort, and
  - development time.

Staffing Level Estimation

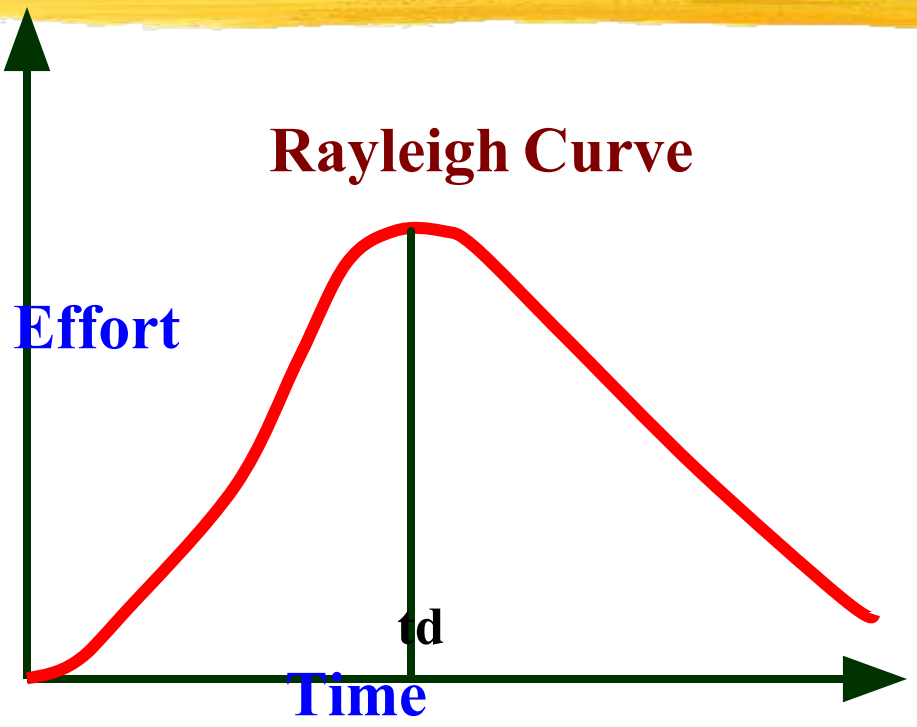
- Once the effort required to develop a software has been determined, it is necessary to determine the staffing requirement for the project.

- Number of personnel required during any development project:
  - not constant.

- Norden in 1958 analyzed many R&D projects, and observed:
  - Rayleigh curve represents the number of full-time personnel required at any time.

Rayleigh Curve

- Rayleigh curve is specified by two parameters:
  - td the time at which the curve reaches its maximum
  - K the total area under the curve.
- L=f(K, td)



Rayleigh Curve

- Very small number of engineers are needed at the beginning of a project
  - carry out planning and specification.
- As the project progresses:
  - more detailed work is required,
  - number of engineers slowly increases and reaches a peak.
- Putnam observed that:
  - the time at which the Rayleigh curve reaches its maximum value
    - corresponds to system testing and product release.
  - After system testing,
    - the number of project staff falls till product installation and delivery.

Project Scheduling

- It involves deciding which tasks would be taken up when. In order to schedule the project activities, a software project manager needs to do the following task:
  - Identify all the tasks needed to complete the project.
  - **Break down** large tasks into small activities.
  - **Determine the dependency** among different activities.
  - Establish the most likely **estimates for the time durations** necessary to complete the activities.
  - Plan the **starting and ending dates** for various activities.
  - Determine the critical path. **A critical path is the chain of activities that determines the duration of the project.**

Work breakdown structure

- Decompose a given task set recursively into small activities.
- 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.

Activity networks and critical path method

- Activity network shows the different activities making up a project, their estimated durations, and interdependencies.

- Each activity is represented by a rectangular node and the duration of the activity is shown alongside each task.

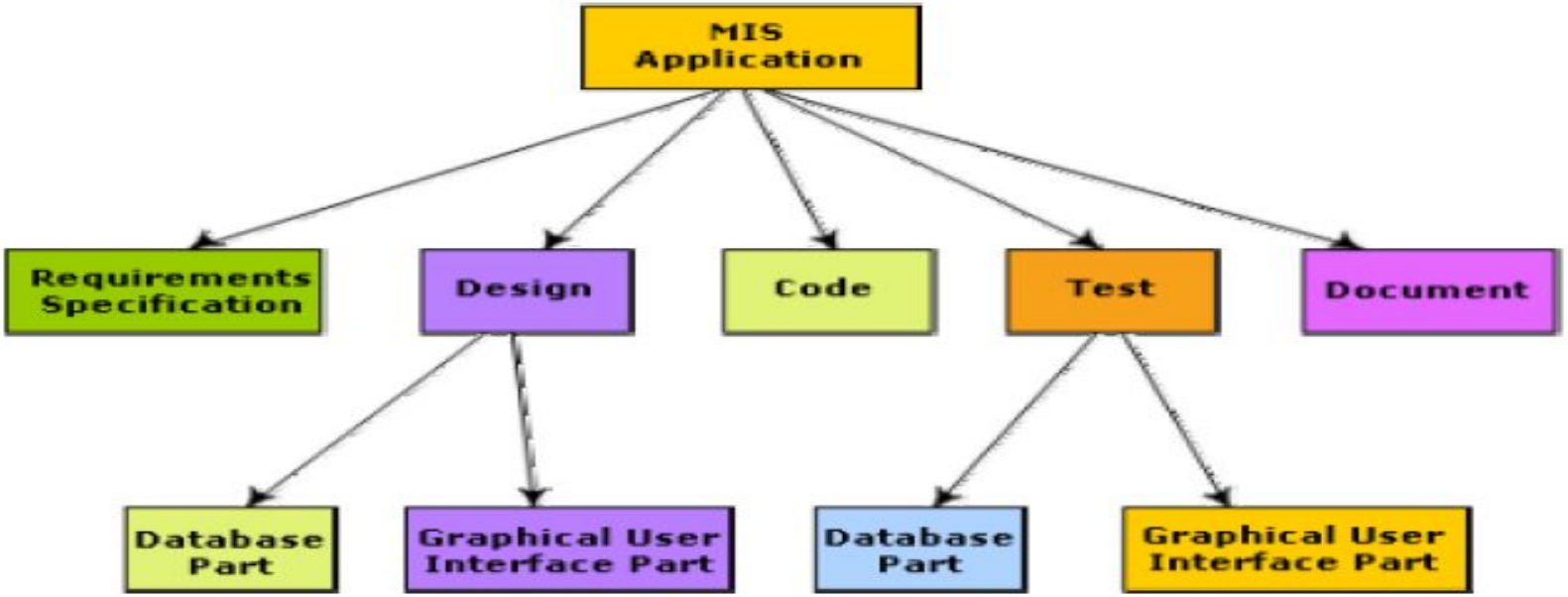
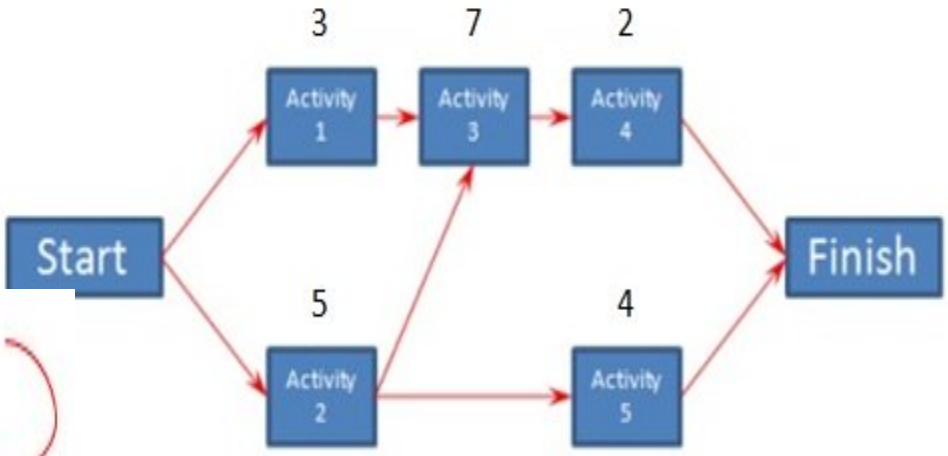


Fig. 11.7: Work breakdown structure of an MIS problem

Critical Path Method (CPM)

- The **Critical Path Method** (CPM) can help you keep your projects on track. *Critical path schedules*...
- Identify the activities that must be completed on time in order to complete the whole project on time.
- Show you which tasks can be delayed and for how long without impacting the overall project schedule.
- Calculate the minimum amount of time it will take to complete the project.
- The CPM has four key elements...
  - Critical Path Analysis
  - Float Determination
  - Early Start & Early Finish Calculation
  - Late Start & Late Finish Calculation

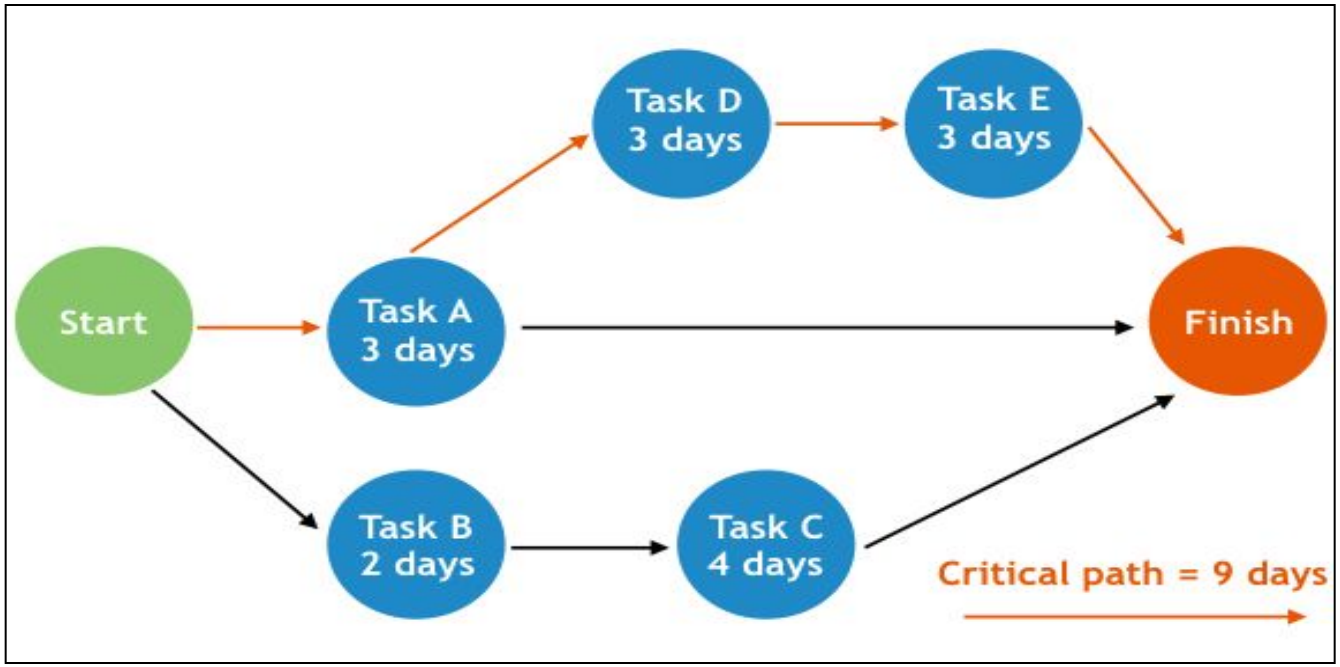


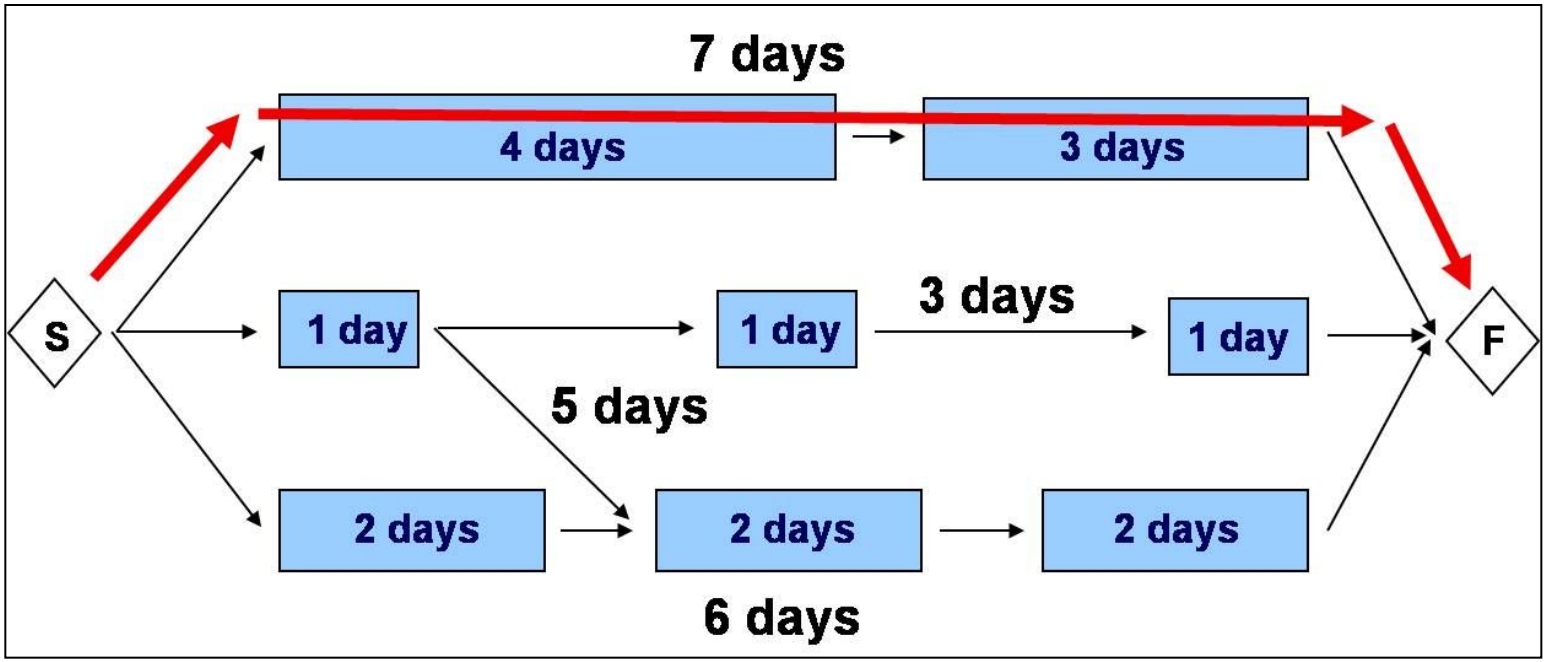
Critical Path

Start → Activity 1 → Activity 3 → Activity 4 → Finish	3 + 7 + 2 = 12
Start → Activity 2 → Activity 3 → Activity 4 → Finish	5 + 7 + 2 = 14
Start → Activity 2 → Activity 5 → Finish	5 + 4 = 9

Critical path calculation

The critical path is the sequence of deoendent activities with the longest duration. A delay in any of these activities will result in a delay for the whole project.




7 days
4 days
3 days
S
1 day
1 day
3 days
1 day
F
5 days
2 days
2 days
2 days
6 days

Float Determination/Slack Time

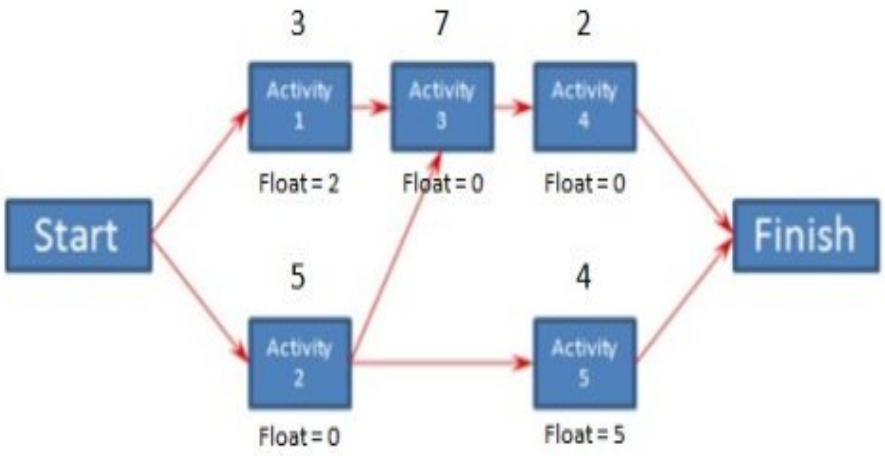
- **Float (slack time) is the amount of time an activity can slip before it causes your project to be delayed.**

- Using the critical path diagram from the previous section, Activities 2, 3, and 4 are on the critical path so they have a float of zero.

- The next longest path is Activities 1, 3, and 4. Since Activities 3 and 4 are also on the critical path, their float will remain as zero.

- For any remaining activities, in this case Activity 1, the float will be the duration of the critical path minus the duration of this path. 14 - 12 = 2. So Activity 1 has a float of 2.

- **Next longest path is Activities 2 and 5. Activity 2 is on the critical path so it will have a float of zero. Activity 5 has a float of 14 - 9, which is 5.**



Minimum time (MT): It is the minimum time required to complete the project. It i computed by determining the maximum of all paths from start to finish.

Earliest start (ES): It is the time of a task is the maximum of all paths from the start to this task. The ES for a task is the ES of the previous task plus the duration of the preceding task.

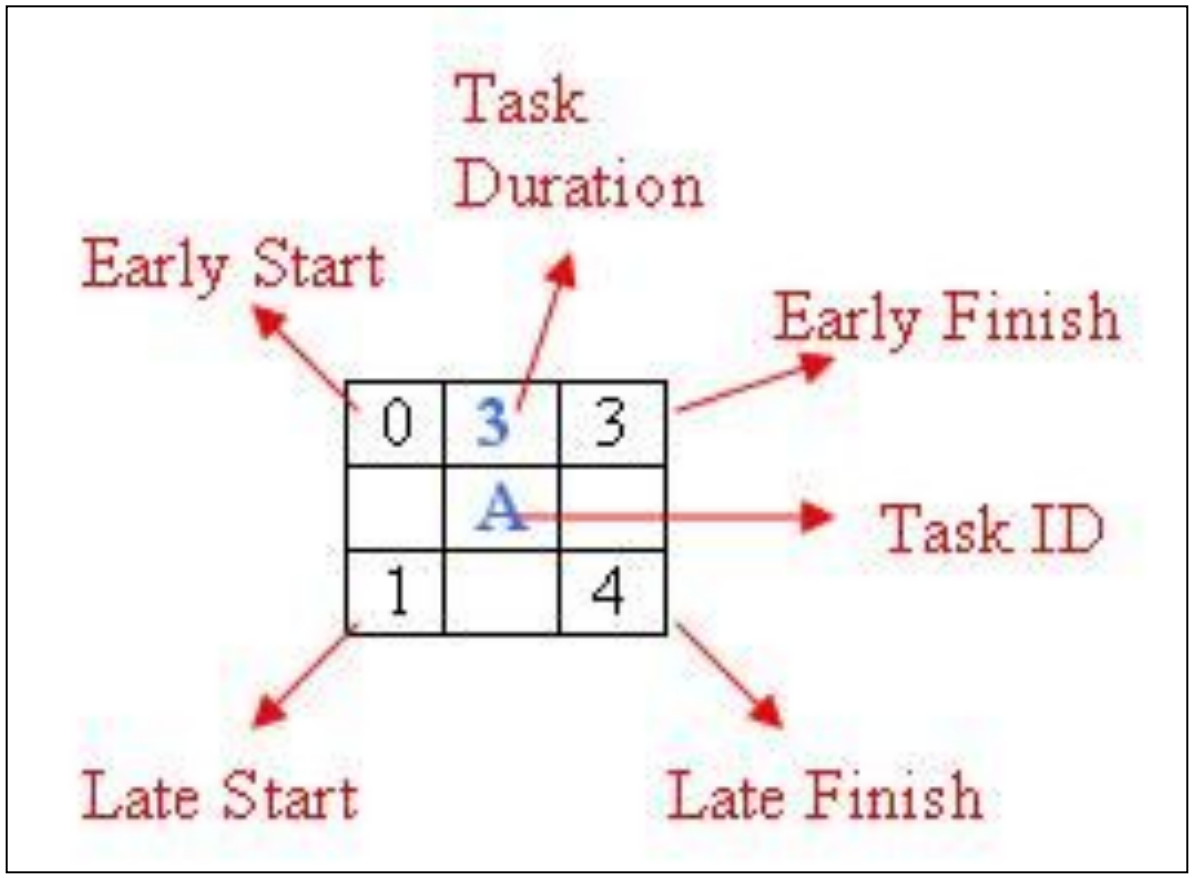
Latest start time (LST): It is the difference between MT and the maximum of all paths from this task to the finish. The LST can be computed by subtracting the duration of the subsequent task from the LST of the subsequent task.

Earliest finish time (EF): The EF for a task is the sum of the earliest start time of the task and the duration of the task.

Latest finish (LF): LF indicates the latest time by which a task can finish without affecting the final completion time of the project. A task completing beyond its LF would cause project delay. LF of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.

Slack time (ST): 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. ST for a task is LS-ES and can equivalently be written as LF-EF.

ES, EF, LS, LF



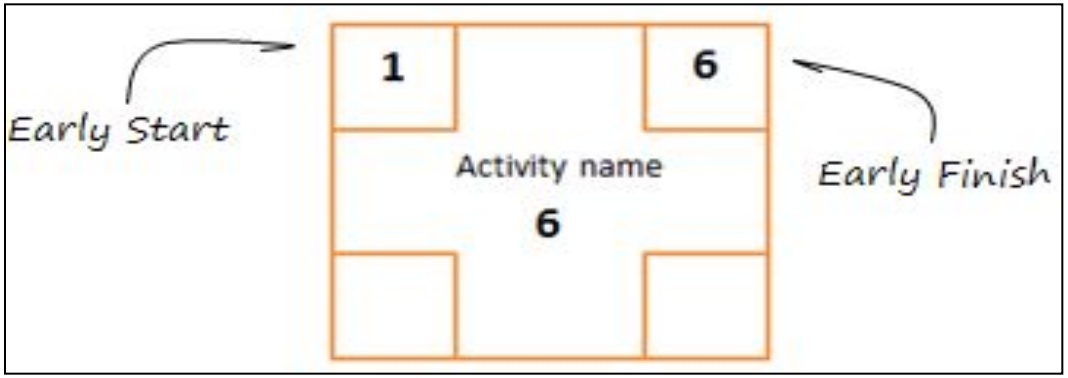
ES = Early Start
EF = Early Finish
LS = Late Start
LF = Late Finish

Early Start and Early Finish

- Indicates the earliest time an activity on a network path can start and earliest it can finish.
- If you decide to start an activity on its early start (assuming previous activities on that network path are completed on their early finishes), that activity can finish on its early finish (if it does not slip).
- And when the last activity on a network path is completed by its early finish, you have all the resources of those activities at your disposal to deploy on other high risk activities.

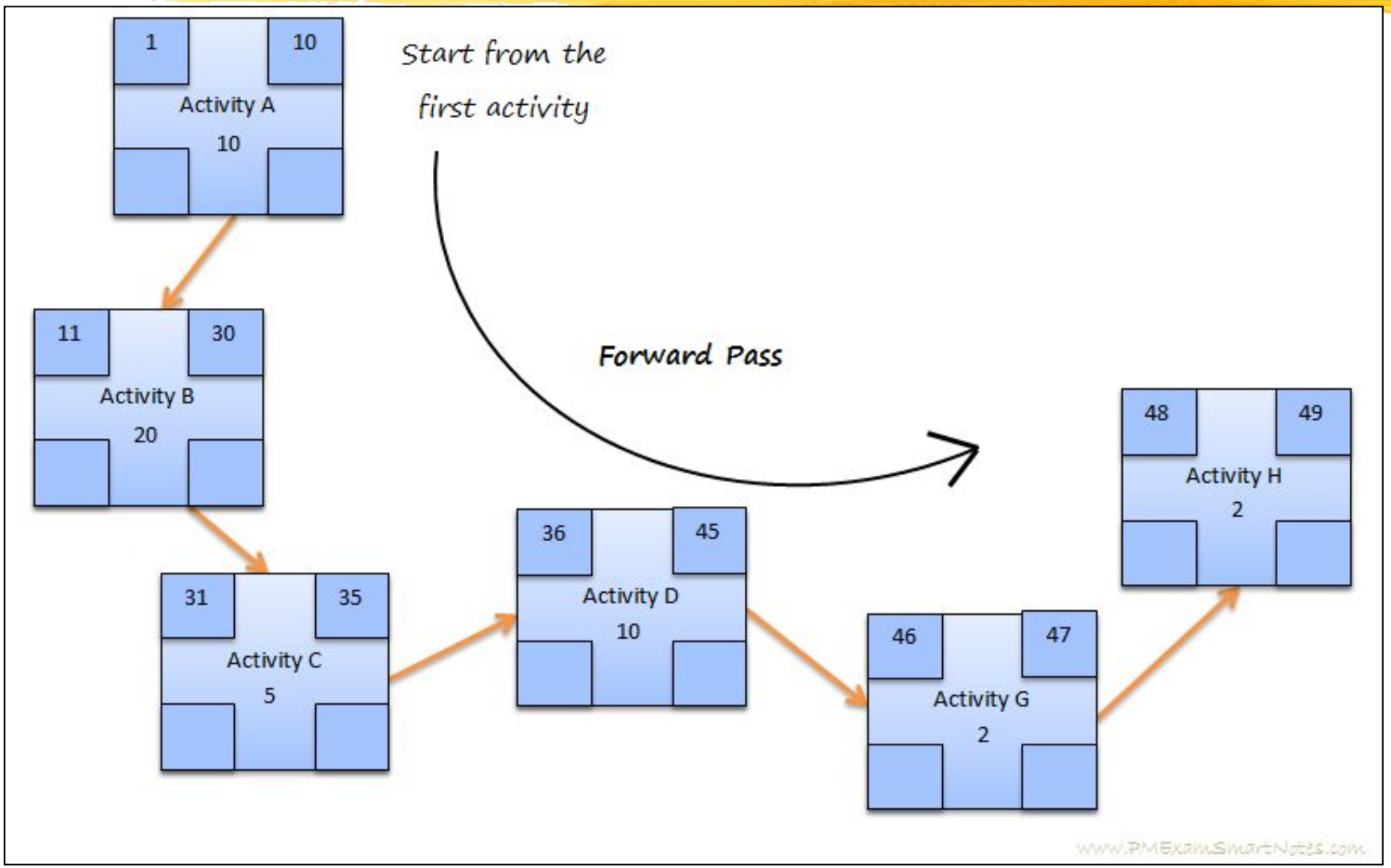
Calculate ES and EF

- **Step-1: Select the critical path**. **Early start of first activity on critical path is always 1.** Write it at the top left corner of that activity box (see the image below).
- **Step 2: Add its activity duration to this early start number and reduce it by one**. Write the resulting number on the top right corner of activity box.
- **Step 3**: **Take the subsequent number of this early finish and write as early start for next activity.** Continue this till you reach the end of critical path.
- **Step 4: Select the network path with second highest total duration**, and calculate early starts and finishes. If you find an activity with early start and finish already written do not overwrite them. Do the same for remaining network paths.



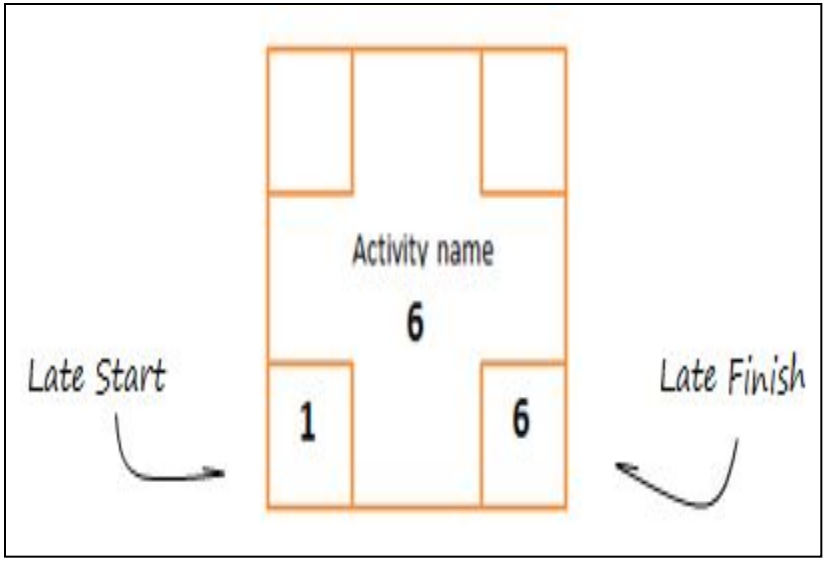
ES and EF

Early start number is written at the top left corner of activity box, and early finish on the top right corner.



Late start(LS) and Late finish(LF)

- **Indicates the latest time an activity on a network path can start and latest it can finish.**
- Knowing how late the last activity on the network path can start and still finish within the time to not impact critical path, will let you decide how much of flexibility you want to exercise on its schedule.
- However, once the last activity on the network path starts on its late start day it should not slip, else it will impact project completion date.



Calculate LS and LF

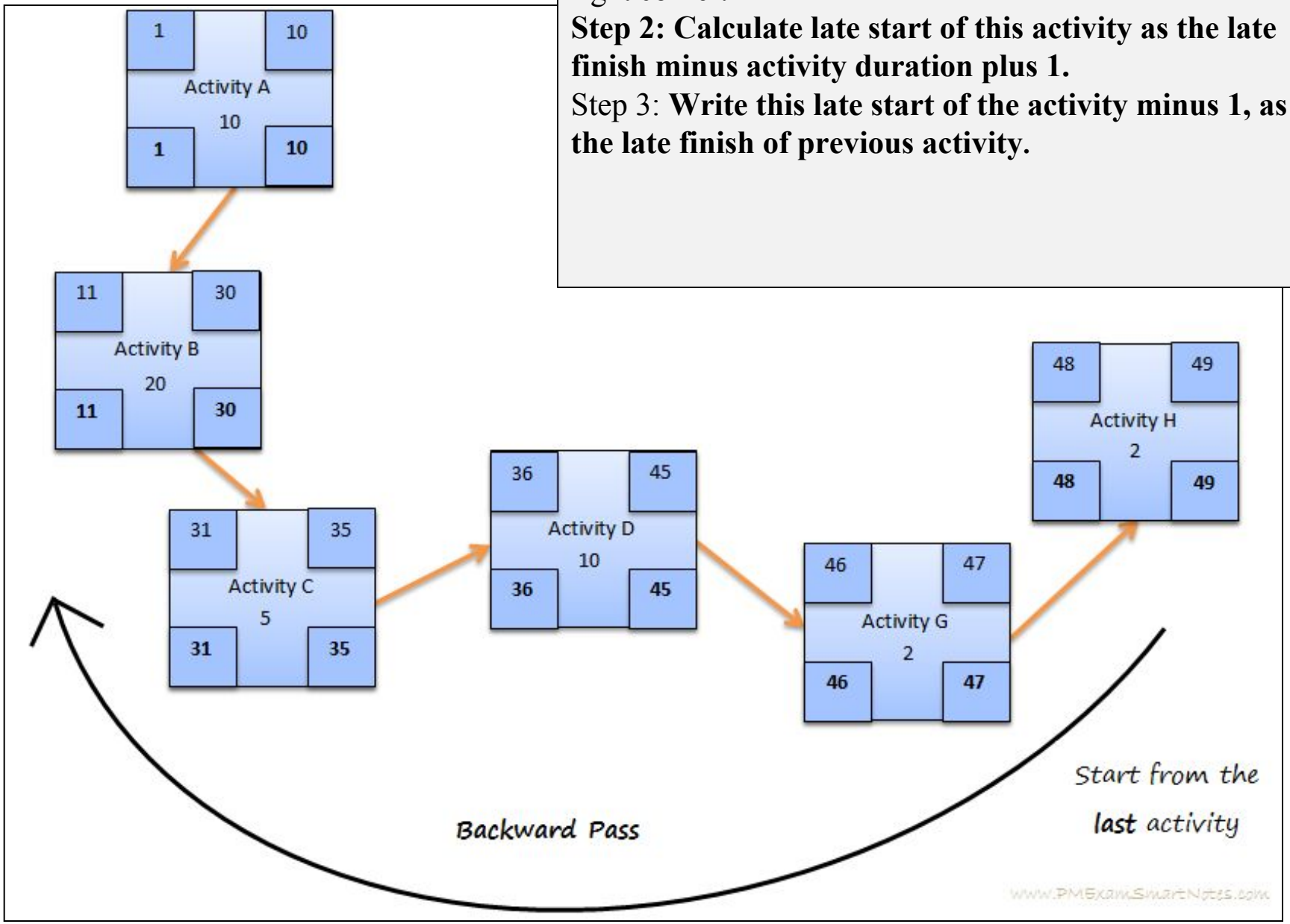
- Remember!: Start with the critical path, beginning at the last activity's late finish.
- Step 1: **Late finish of last activity on the critical path is same as its early finish.** Write this number at the bottom right corner.
- Step 2: **Calculate late start of this activity as the late finish minus activity duration plus 1.** This calculation has the same reason – start and finish are both included in the duration. Write this number at the bottom left corner.
- Step 3: **Write this late start of the activity minus 1, as the late finish of previous activity.** Continue this way all way till you reach the late start of first activity on the critical path.
- Step 4: **Select the network path with second highest total duration, and write late starts and finishes beginning at the last activity of that path. Do the same for remaining network paths.**

LS and LF

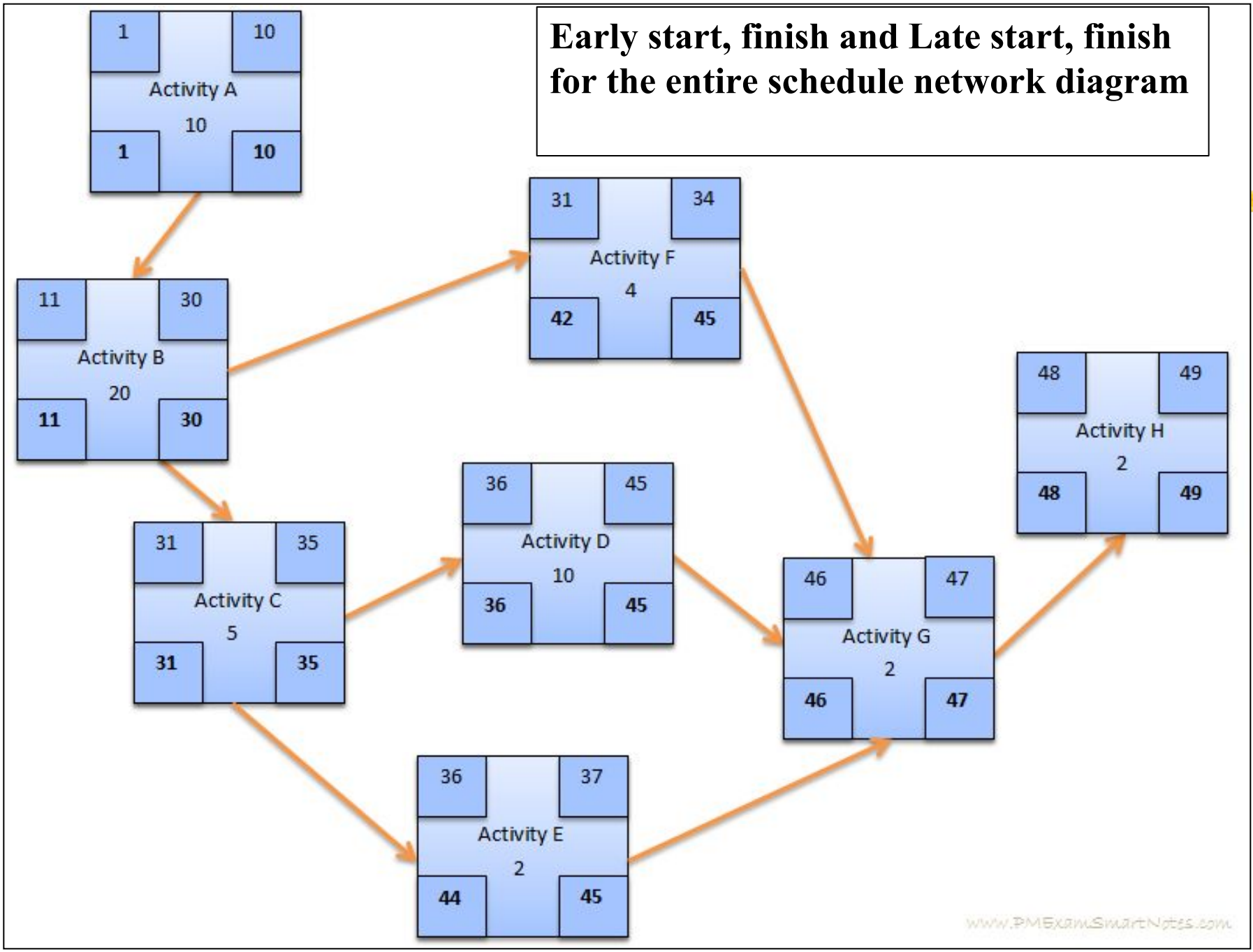
Step 1: **Late finish of last activity on the critical path is same as its early finish.** Write this number at the bottom right corner.

Step 2: Calculate late start of this activity as the late finish minus activity duration plus 1.

Step 3: **Write this late start of the activity minus 1, as the late finish of previous activity.**



Early start, finish and Late start, finish for the entire schedule network diagram

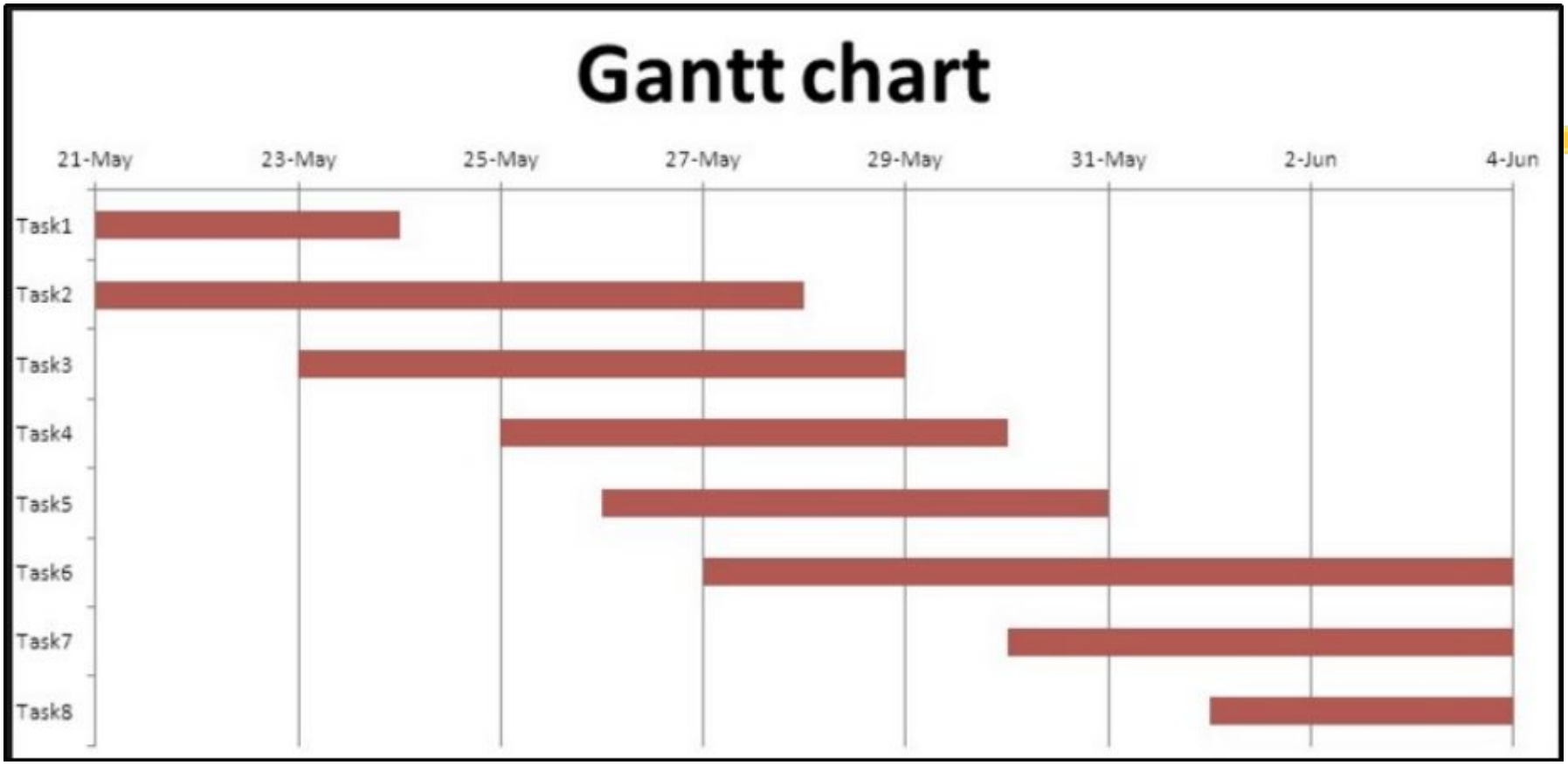


Gantt chart

- Named after its developer Henry Grantt. Its a bar chart.
- Gantt charts are mainly used to **allocate resources** to activities.
- The resources allocated to activities include staff, hardware, and software. a special type of bar chart where each bar represents an activity.
- The **bars are drawn along a time line**. **Each bar is an activity.**
- The **length of each bar is proportional to the duration** of time planned for the corresponding activity.

Use:

- Planning and
- resource utilization.


Gantt chart
21-May
23-May
25-May
27-May
29-May
31-May
2-Jun
4-Jun
Task1
Task2
Task3
Task4
Task5
Task6
Task7
Task8

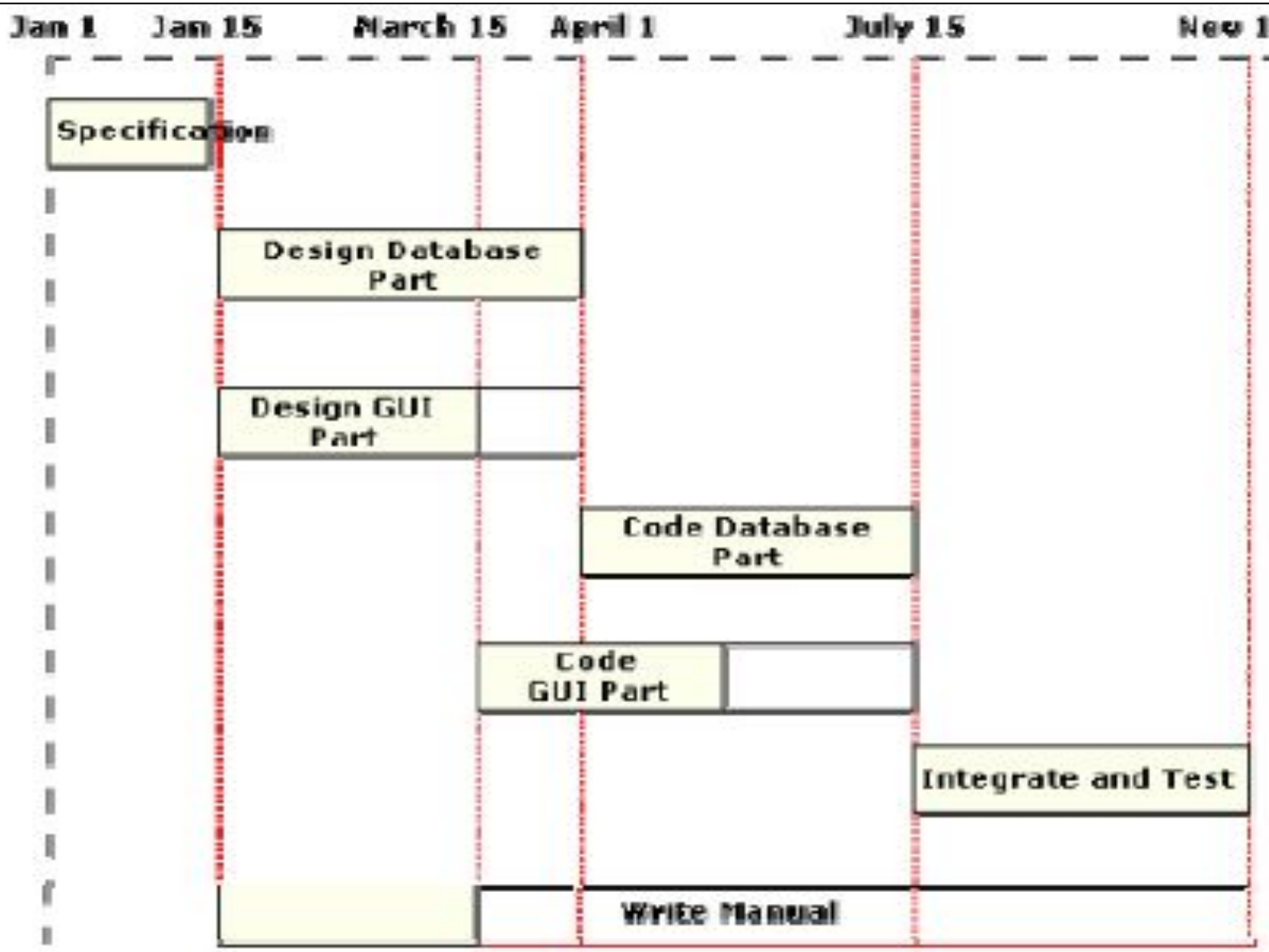
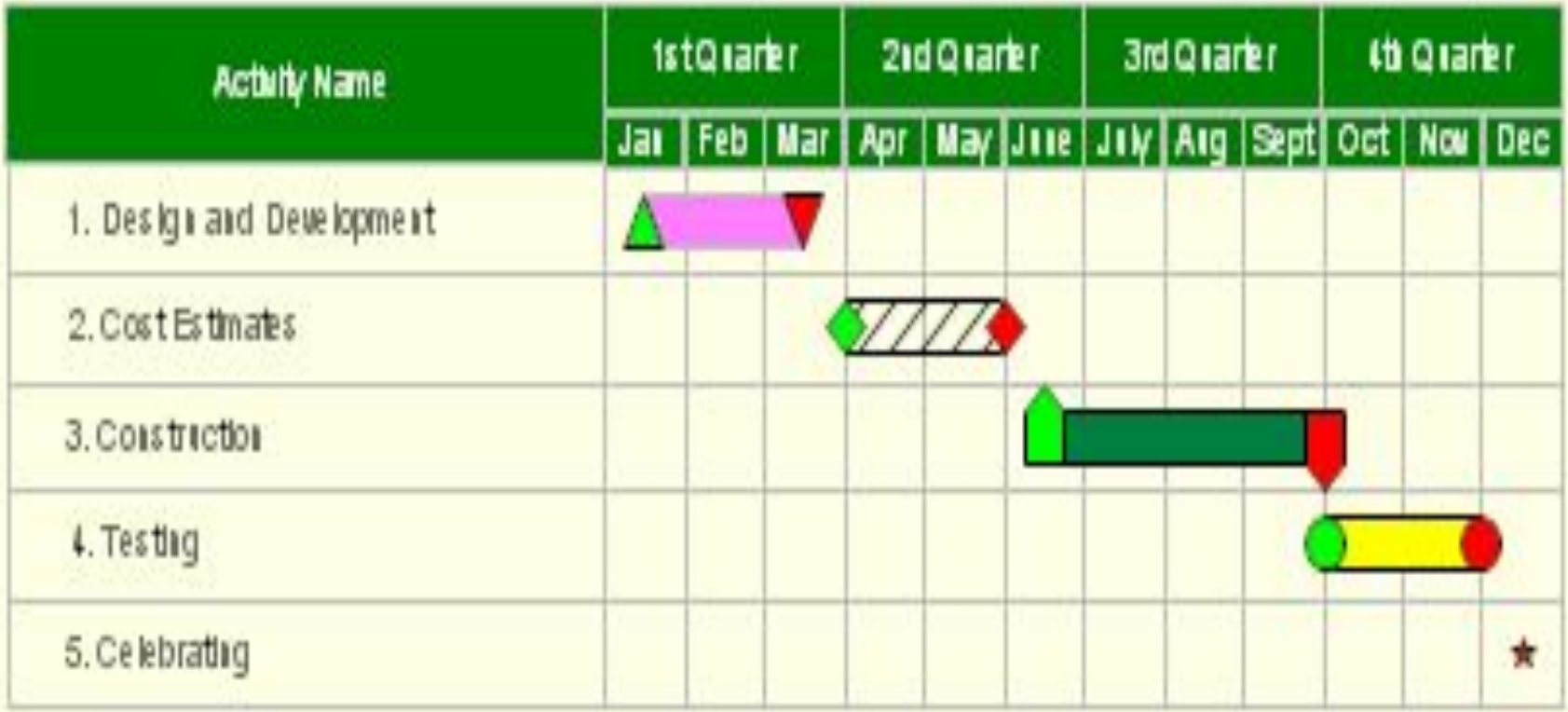


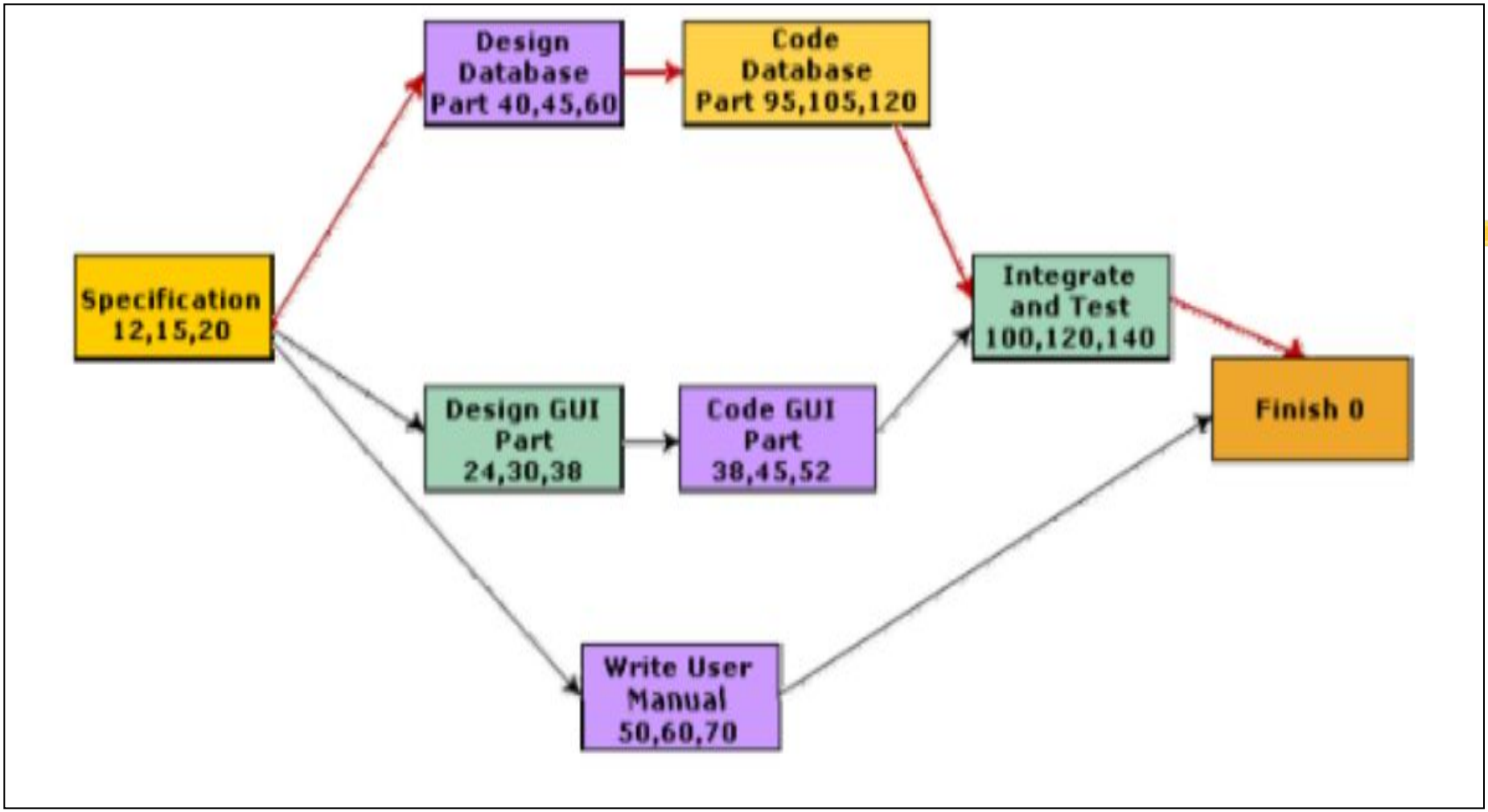
Fig. 11.9: Gantt chart representation of the MIS problem

Gantt Chart - Project Development



PERT chart

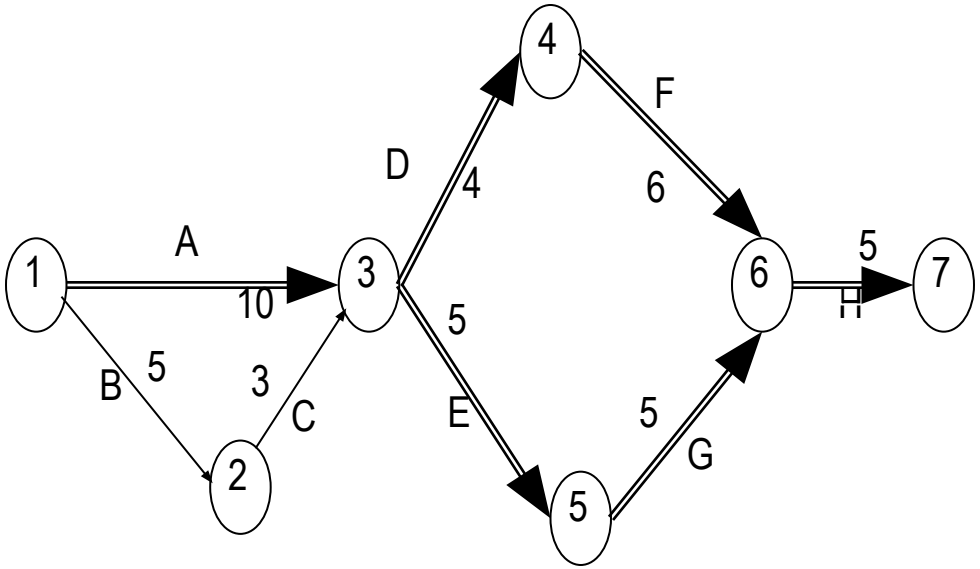
- **PERT (Project Evaluation and Review Technique**) charts consist of a network of boxes and arrows.
- The boxes represent activities and the arrows represent task dependencies. PERT chart represents the statistical variations in the project estimates assuming a normal distribution.
- Thus, in a PERT chart instead of making a single estimate for each task, 1. **pessimistic, worst case (W)**
- **2. most likely estimates (M),**
- **3. optimistic estimates (O) are made.**
- There can be may critical paths, depending on various permutations of the estimates of each task.
- A critical path in a PERT chart is shown by using thicker arrows.



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.

activities tabulated below

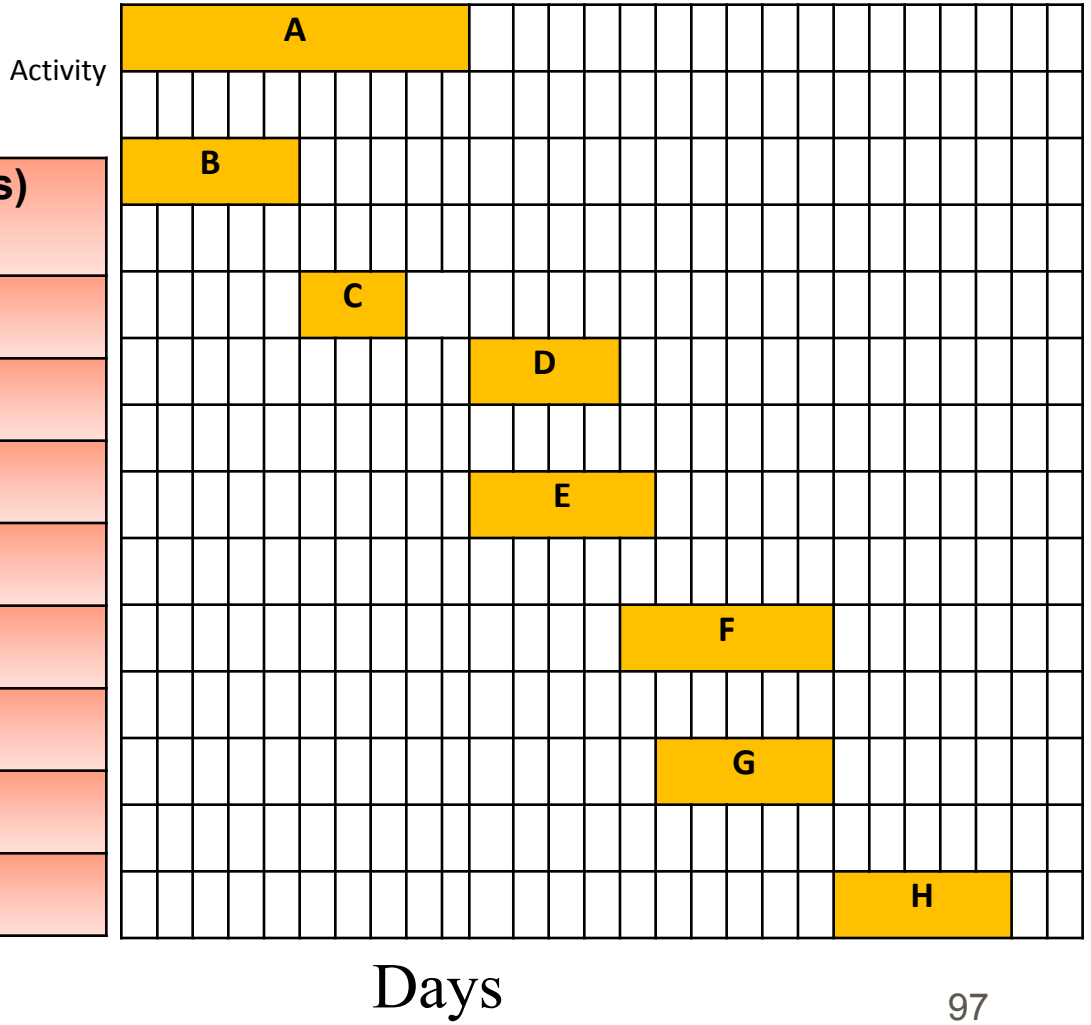
Activity	Predecessor	Duration (days)
A	-	10
B	-	5
C	B	3
D	A, C	4
E	A, C	5
F	D	6
G	E	5
H	F, G	5



Gantt Chart

- Critical path-1 A→D→F→H B→C Slack time 2 days
- Critical Path-2 A→E→G→H
- Project Duration = 25 days
- GANTT CHART

Activity	Predecessor	Duration (days)
A	-	10
B	-	5
C	B	3
D	A, C	4
E	A, C	5
F	D	6
G	E	5
H	F, G	5



Risk management

Project risks: risks concern varies forms of

- budgetary,
- schedule,
- personnel,
- resource, and
- customer-related problems. An important project risk is schedule slippage.

- **Technical risks:** risks concern
  - potential design,
  - implementation,
  - interfacing,
  - testing, and
  - maintenance problems. Most technical risks occur due to the development team's insufficient knowledge about the project.
- **Business risks:** risks include risks of
  - building an excellent product that no one wants,
  - losing budgetary or personnel commitments, etc.

Risk assessment:

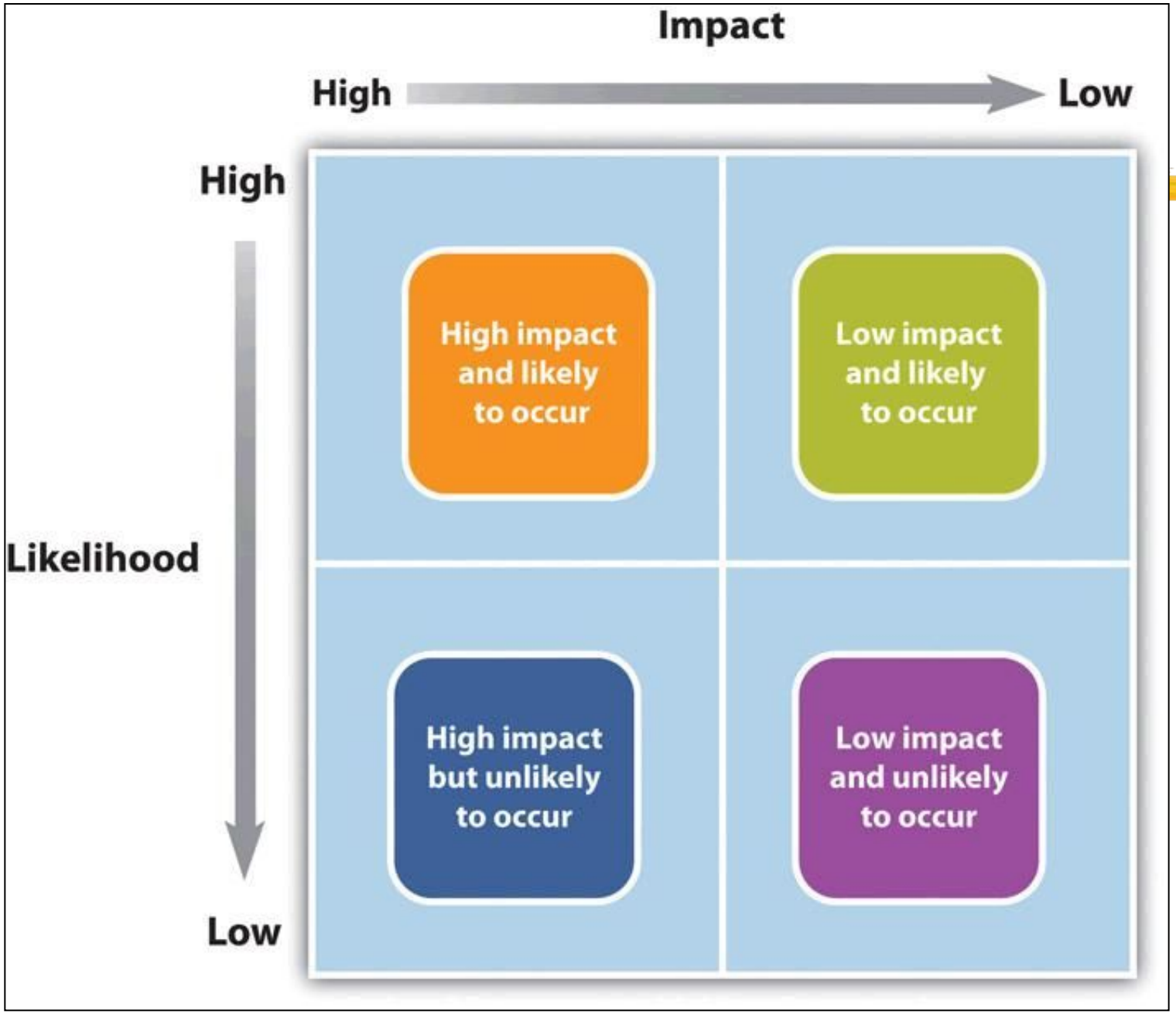
- to rank the risks in terms of their damage causing potential. **p = r * s** where

  p is the priority with which the risk must be handled,

  r is the probability of the risk becoming true,

  s is the severity of damage caused due to the risk becoming true.

	Severity			
Probablity	Catrastrophic - 4	Critical - 3	Marginal - 2	Negligible - 1
Frequent - 5	High - 20	High - 15	High - 10	Medium - 5
Probable - 4	High - 16	High - 12	Serious - 8	Medium - 4
Occasional - 3	High - 12	Serious - 9	Medium - 6	Low - 3
Remote - 2	Serious - 8	Medium - 6	Medium - 4	Low -2
Improbable - 1	Medium - 4	Low - 3	Low - 2	Low -1


Impact
High
Low
High
Likelihood
Low
High impact and likely to occur
Low impact and likely to occur
High impact but unlikely to occur
Low impact and unlikely to occur

Risk containment

- Risks of a project are assessed, plans must be made to contain the most damaging and the most likely risks.
- Different risks require different containment procedures. Three main strategies to plan for risk containment:
  - **Avoid the risk:** discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to the engineers to avoid the risk of manpower turnover, etc.
  - **Transfer the risk:** Involves getting the risky component developed by a third party, buying insurance cover, etc.
  - **Risk reduction:** Planning ways to contain the damage due to a risk. For example, if there is risk that some key personnel might leave, new recruitment may be planned.

Risk leverage

- Different strategies must be considered for handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk. More formally,

  risk leverage = (risk exposure before reduction – risk exposure after reduction) / (cost of reduction)

Software Configuration Management (SCM)

- Configuration of a SW product
- Release, version and revision of SW product
- Necessity of SCM
- Principal activities of SCM
- Configuration identification.
- Configuration control.
- Configuration management tools.

✔*Figuring out what you have and recording it somewhere*
✔*Controlling who can make changes*
✔*Keeping a record of the changes you've made*

Software Configuration Management

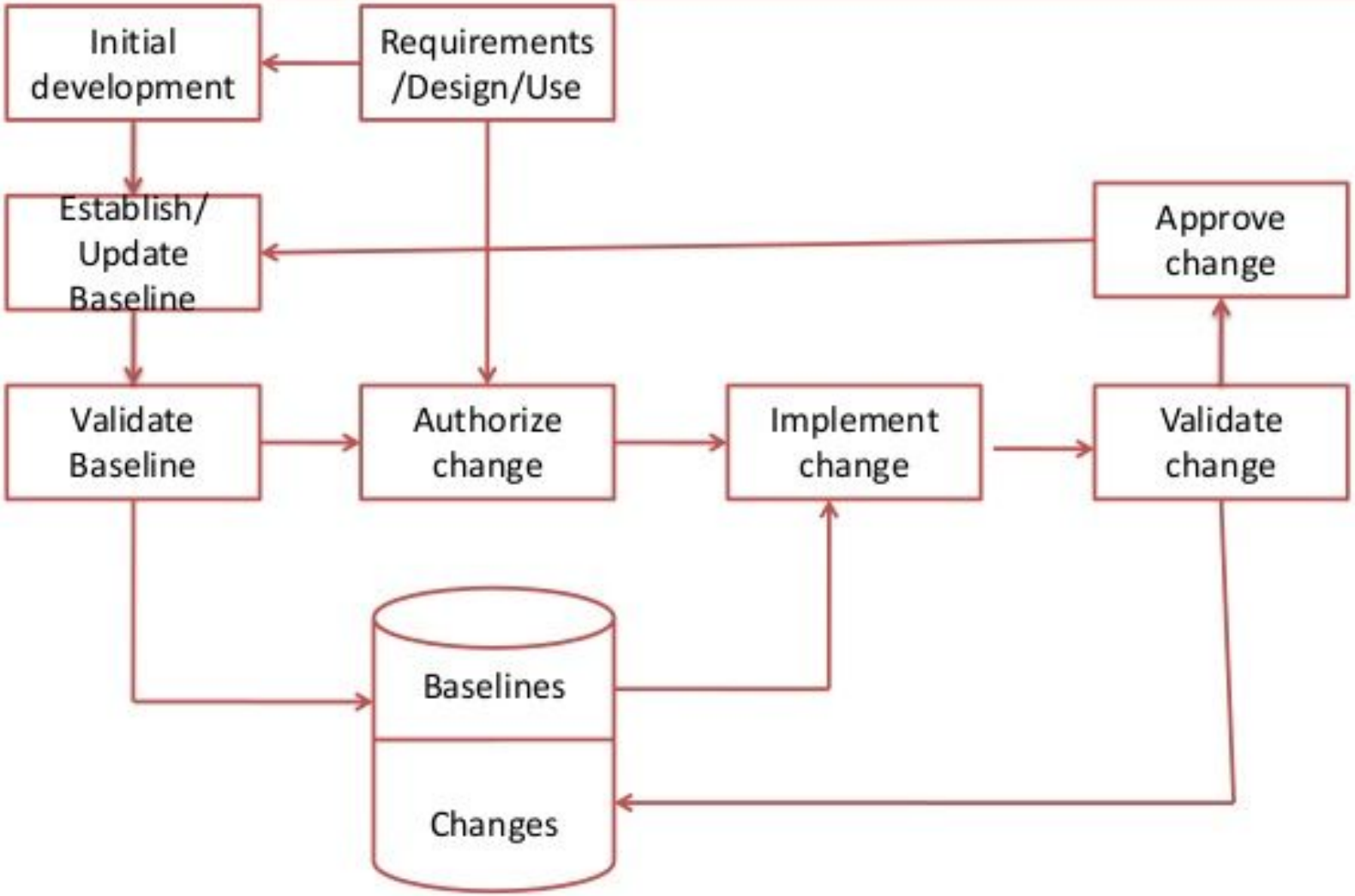
- The deliverables of a SW product consist of a number of objects, e.g.
  - source code,
  - design document,
  - SRS document,
  - test document(plan and script),
  - user's manual, etc.
- These objects are modified by many software engineers through out development cycle.
- The state of each object changes as bugs are detected and fixed during development .

- The configuration of the software is the state of all project deliverables at any point of time.
- SCM deals with effectively tracking and controlling the configuration of a software during its life cycle.

Release vs. Version vs. Revision of SW

- A new **version** results from a significant change in functionality, technology, or the platform
  - Example: one version of a SW might be Unix-based, another Windows based.
- **revision** refers to minor bug fix .
- A **release** results from bug fix, minor enhancements to the functionality, usability
- A new release of SW is an improved system replacing the old one.
- Systems are described as Version m, Release n; or simple m.n.

Configuration Management Overview



Necessity of SCM

- To control access to deliverable objects with a view to avoiding the following problems
  - **Inconsistency problem when the objects are replicated.**
  - **Problems associated with concurrent access.**
  - **Providing a stable development environment.**
  - **System accounting and status information.**
  - **Handling variants.** If a bug is found in one of the variants, it has to be fixed in all variants

SCM activities

- **Configuration identification** :
  - deciding which objects (configuration items) are to be kept track of.
- **Configuration control :**
  - ensuring that changes to a system happen without ambiguity.

- **Baseline**
  - When an effective SCM is in place, the manager freezes the objects to form a base line.
  - A baseline is the status of all the objects under configuration control. When any of the objects under configuration control is changed , a new baseline is formed.

Configuration item identification

- **Categories of objects:**
  - Controlled objects: are under Configuration Control (CC) . Formal procedures followed to change them.
  - Precontrolled objects: are not yet under CC, but will eventually be under CC.
  - Uncontrolled objects: are not subjected to CC
  - Controllable objects include both **controlled and precontrolled objects**; examples:

  SRS document,

  Design documents,

  Source code ,

  Test cases

  The SCM plan is written during the project planning phase and it lists all controlled objects.

- **Configuration Control (CC):**

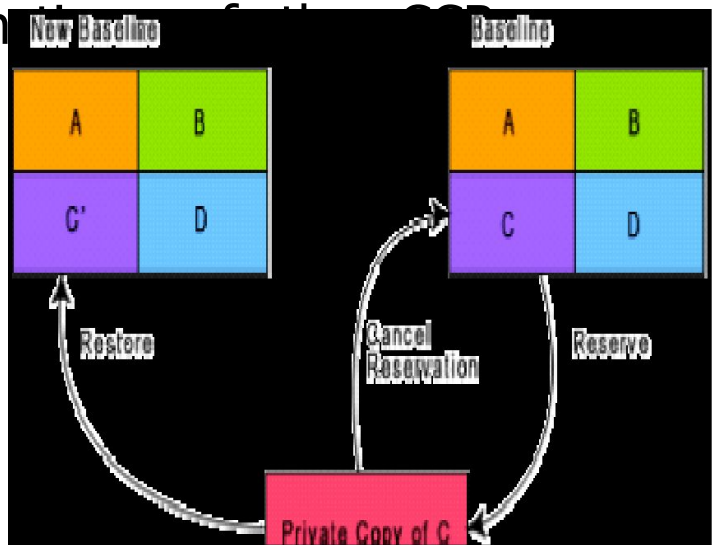
- process of managing changes to controlled objects. Allows authorized changes to the controlled objects and prevents unauthorized changes .
  - In order to change a controlled object such as a module, a developer can get a private copy of the module by a reserve operation .
  - Configuration management tools allow only one person to reserve a module at a time. Once an object is reserved, it does not allow any one else to reserve this module until the reserved module is restored .

Changing the Baseline

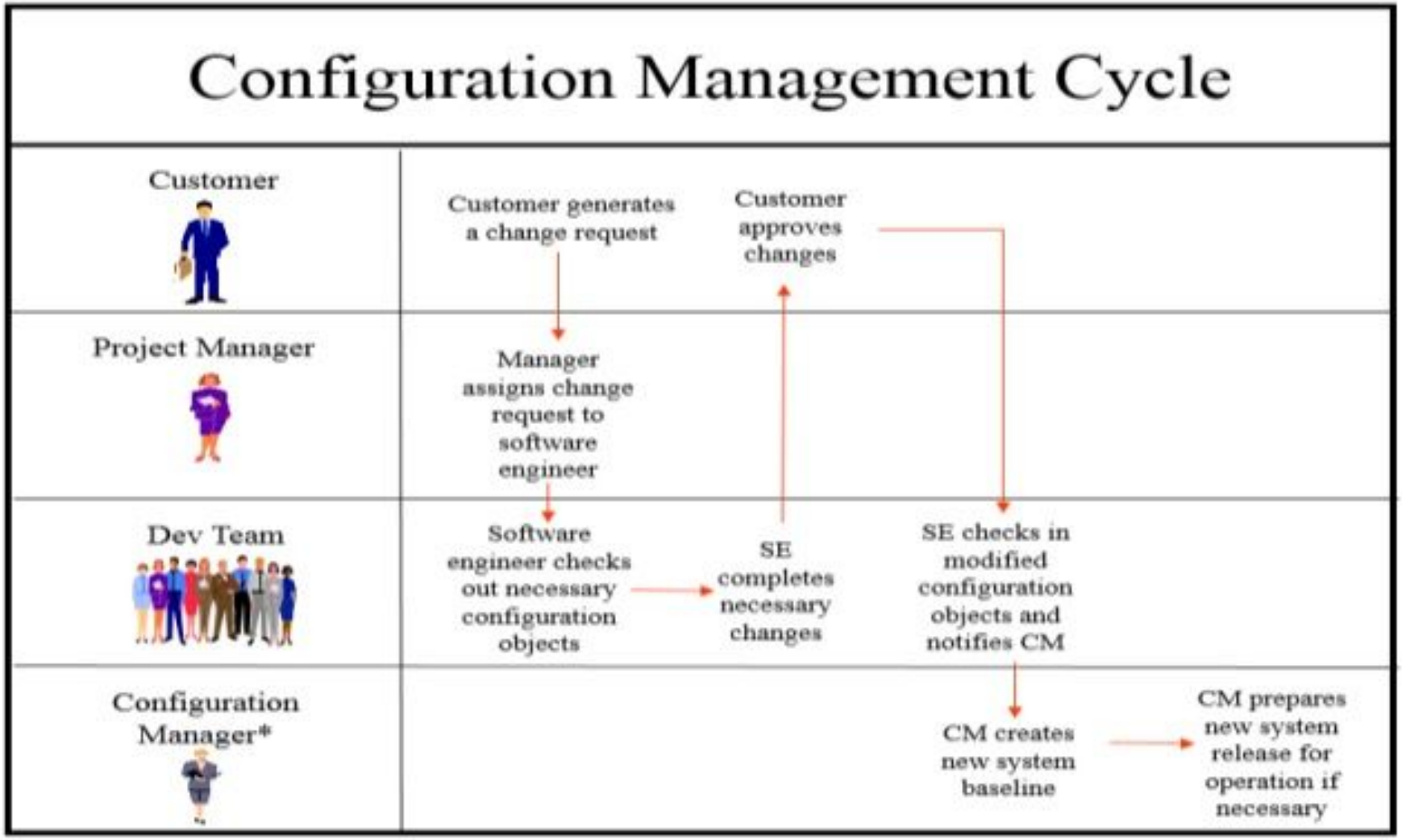
- When one needs to change an object under configuration control, he is provided with a copy of the base line item.
- The requester makes changes to his private copy.
- After modifications, updated item replaces the old item and a new base line gets formed .

Reserve and restore operation in configuration control

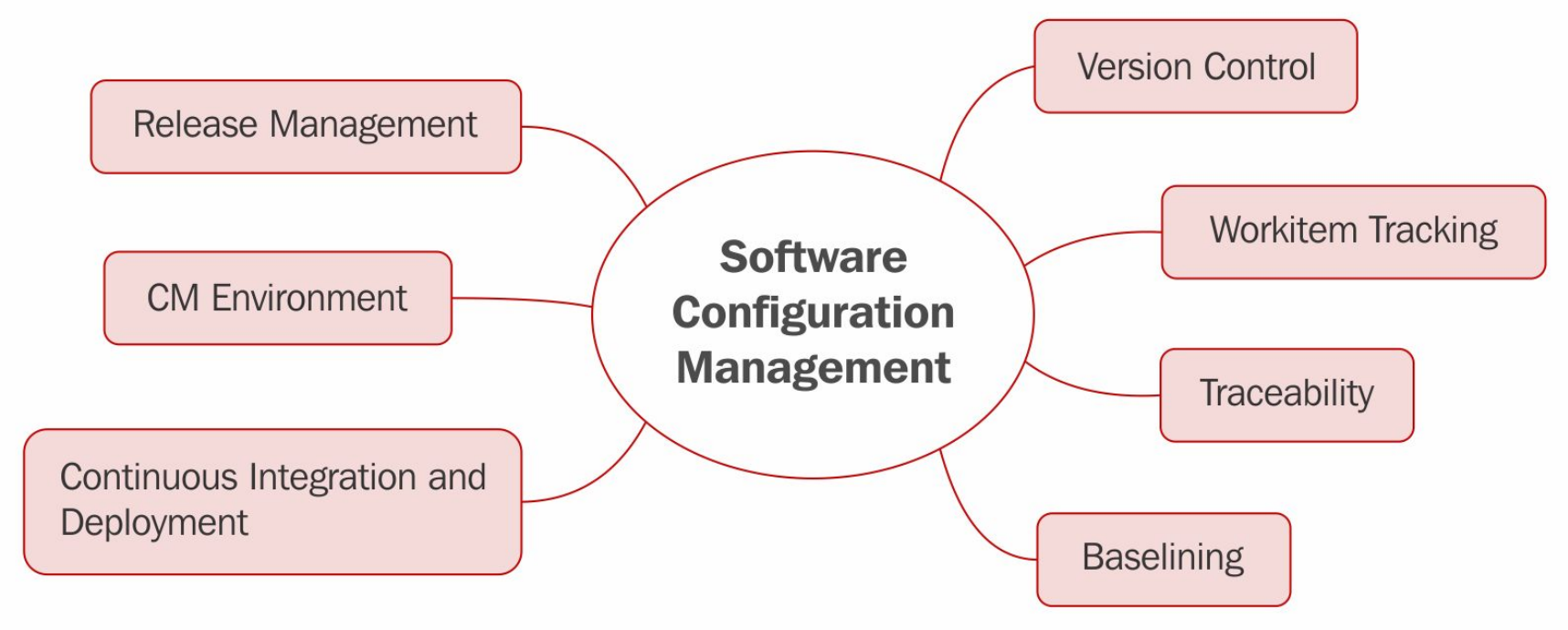
- obtains a private copy of the module through a reserve operation.
- carries out all changes on this private copy.
- restoring the changed module to the baseline requires the permission of a change control board(CCB).
- Except for very large projects, the fun discharged by the project manager himself.



Software Configuration Auditing



summary

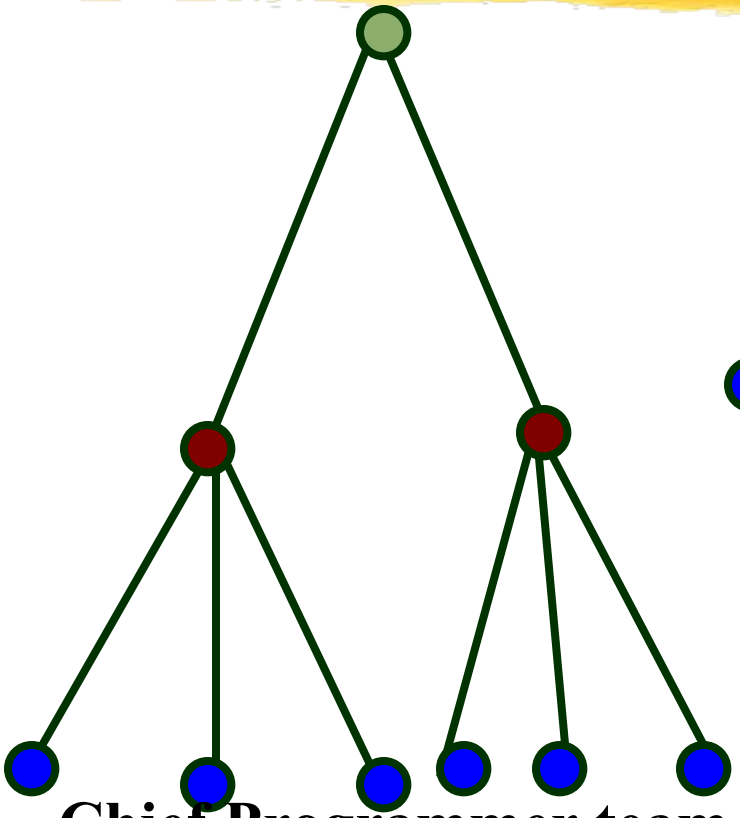

Release Management
CM Environment
Continuous Integration and Deployment
Software Configuration Management
Version Control
Workitem Tracking
Traceability
Baselining

Organization Structure

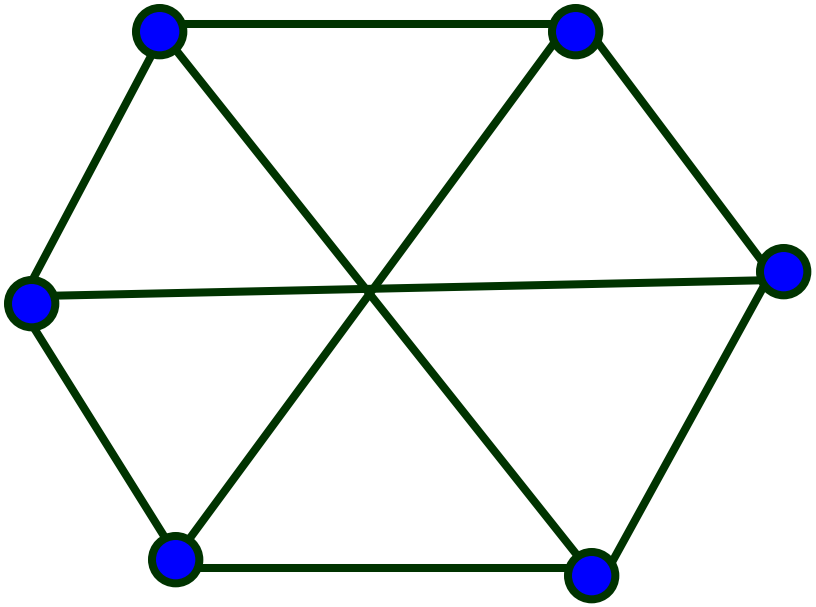
- Functional Organization:
  - Engineers are organized into functional groups, e.g.
    - specification, design, coding, testing, maintenance, etc.
  - Engineers from functional groups get assigned to different projects

- Problems of different complexities and sizes require different team structures:
  - Chief-programmer team
  - Democratic team
  - Mixed organization

Team Organization

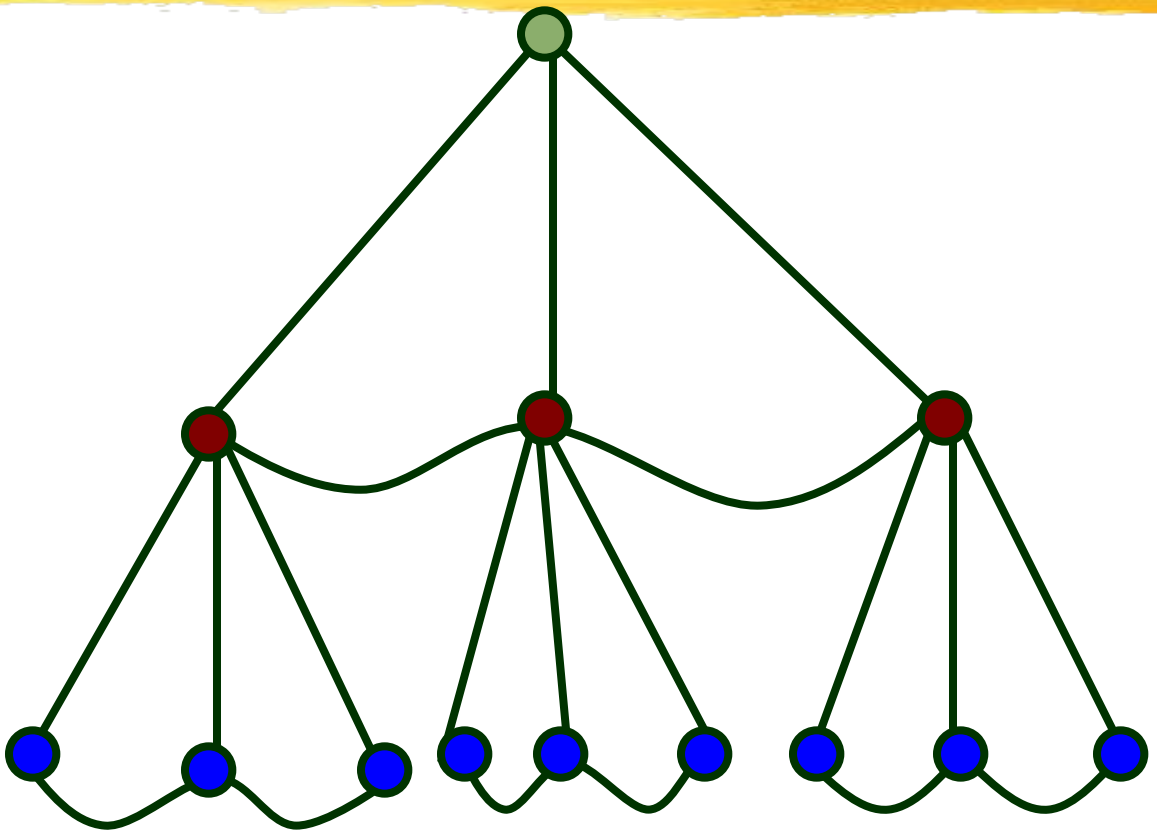


Chief Programmer team



Democratic Team

Mixed team organization



Chief Programmer Team

- A senior engineer provides technical leadership:
  - partitions the task among the team members.
  - verifies and integrates the products developed by the members.

- Works well when
  - the task is well understood
    - also within the intellectual grasp of a single individual,
  - importance of early completion outweighs other factors
    - team morale, personal development, etc.

- Chief programmer team is subject to single point failure:
  - too much responsibility and authority is assigned to the chief programmer.

Democratic Teams

- Suitable for:
  - small projects requiring less than five or six engineers
  - research-oriented projects
- A manager provides administrative leadership:
  - at different times different members of the group provide technical leadership.
- Democratic organization provides
  - higher morale and job satisfaction to the engineers
  - therefore leads to less employee turnover.

- Disadvantage:
  - team members may waste a lot time arguing about trivial points:
    - absence of any authority in the team.

Mixed Control Team Organization

- Draws upon ideas from both:
  - democratic organization and
  - chief-programmer team organization.
- Communication is limited
  - to a small group that is most likely to benefit from it.
- Suitable for large organizations.

Mixed Control Team: