Software development challenges

https://www.geeksforgeeks.org/challenges-of-software-developers/

Human Side of software development

https://community.nasscom.in/index.php/communities/it-services/human-side-software-development-behind-screen#:~:text=%E2%80%9CThe%20Human%20Side%20of%20Software,of%20creating%20and%20maintaining%20software.

Software development methodologies

https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/

https://www.uptech.team/blog/software-development-methodologies

Software Process Models

https://www.educative.io/blog/software-process-model-types

Requirements Engineering process.

https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/

Agile Requirements engineering

https://www.javatpoint.com/software-engineering-agile-model

Waterfall model

https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

Iterative Waterfall model

https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/

V-model

https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

Evolutionary Model

https://www.geeksforgeeks.org/software-engineering-evolutionary-model/

https://artoftesting.com/evolutionary-model-in-software-engineering

Prototyping model

https://www.geeksforgeeks.org/software-engineering-prototyping-model/

Spiral Model

https://www.geeksforgeeks.org/software-engineering-spiral-model/

Scrum Model

https://www.geeksforgeeks.org/scrum-software-development/

https://www.geeksforgeeks.org/scrum-methodology-in-software-engineering/

https://www.techtarget.com/searchsoftwarequality/definition/Scrum

XP Model

https://www.geeksforgeeks.org/software-engineering-extreme-programming-xp/

Classification of Requirements

https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/?ref=lbp

How to write a good SRS document

https://www.geeksforgeeks.org/how-to-write-a-good-srs-for-your-project/?ref=lbp

Sequence diagrams

https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/

Class Diagrams

https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

https://www.javatpoint.com/uml-class-diagram

Activity Diagrams

https://www.javatpoint.com/uml-activity-diagram

State Chart Diagrams

https://www.javatpoint.com/uml-state-machine-diagram

Data Flow Diagrams

https://www.javatpoint.com/software-engineering-data-flow-diagrams

https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/

Deployment diagrams

https://www.lucidchart.com/pages/uml-deployment-diagram

https://www.javatpoint.com/uml-deployment-diagram

Software Quality and testing

https://www.geeksforgeeks.org/software-engineering-software-quality/

https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/

https://www.geeksforgeeks.org/software-engineering-software-quality-framework/

https://www.geeksforgeeks.org/software-measurement-and-metrics/

https://www.geeksforgeeks.org/measuring-software-quality-using-quality-metrics/

https://www.geeksforgeeks.org/product-metrics-in-software-engineering/

https://www.geeksforgeeks.org/formal-technical-review-ftr-in-software-engineering/

https://www.geeksforgeeks.org/mccalls-quality-model/

https://www.geeksforgeeks.org/metrics-for-the-design-model-of-the-product/

https://www.1000sourcecodes.com/2012/05/software-engineering-metrics-for_161.html

https://www.geeksforgeeks.org/software-testing-metrics-its-types-and-example/

Software design and development

https://www.geeksforgeeks.org/software-engineering-architectural-design/

https://viblo.asia/p/pattern-based-design-V3m5W0eEKO7

https://www.geeksforgeeks.org/test-driven-development-tdd/

https://www.atlassian.com/devops#:~:text=DevOps%20is%20a%20set%20of,and%20collaboration%2C%20and%20technology%20automation.

https://www.atlassian.com/continuous-delivery/continuous-integration

https://www.geeksforgeeks.org/what-is-ci-cd/

SPM(Software Project Management) with real life examples

SPM (Software Project Management)

- Software project management is an art and science of planning and leading software projects.
- Main goal is to enable a group of developers to work effectively towards successful completion of project.
- Project manager is an administrative leader of the team.
- Various factors make this job very complex (e.g. Changeability, Complexity, Uniqueness, Possibility of multiple solutions etc.)

Job responsibilities of Project Manage

- Planning
- Organizing
- Staffing
- Directing
- Monitoring
- Controlling
- Innovating
- Representing

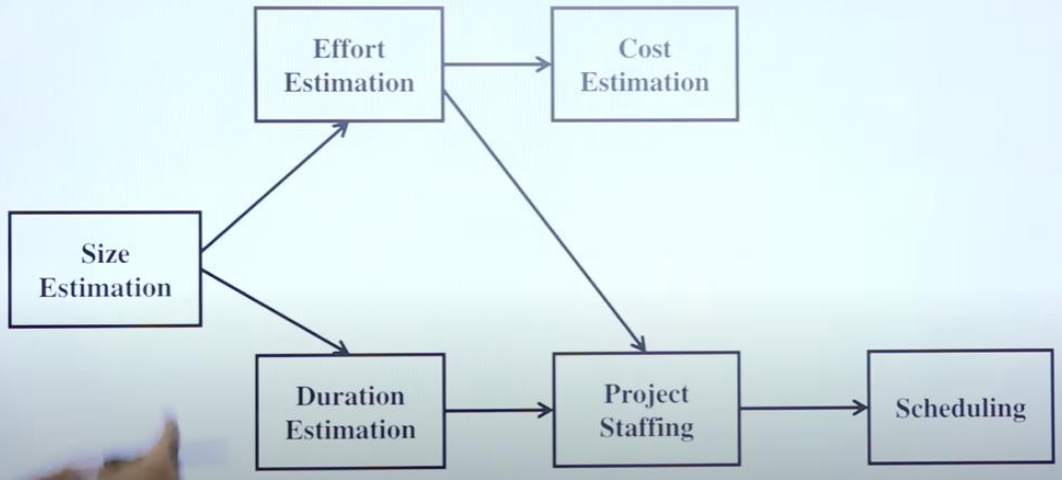
Skills required for Project manager

- Managerial skills
- Technical skills
- Problem solving skills
- Coping skills
- Conceptual skills
- Leadership skills
- Communication skills

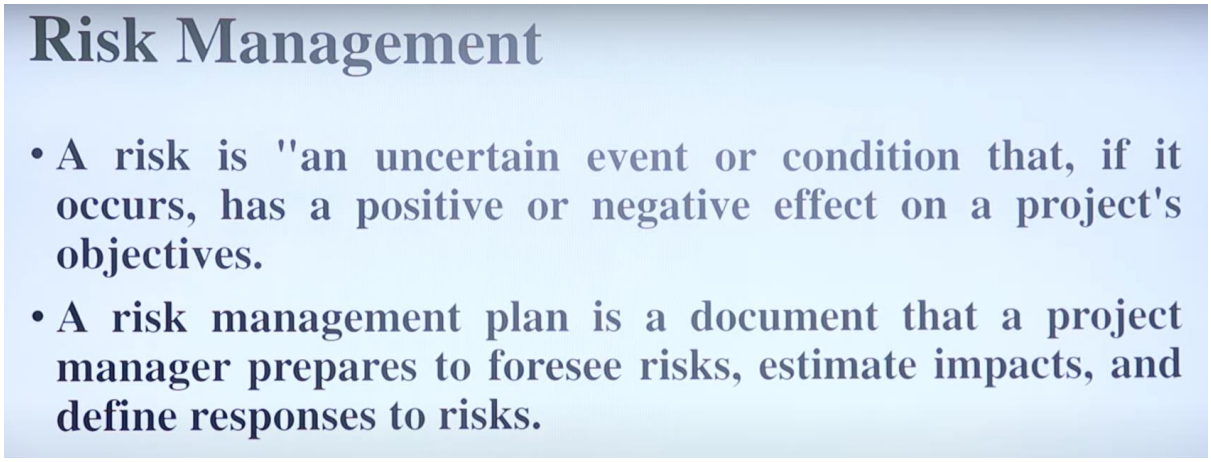
Project Planning

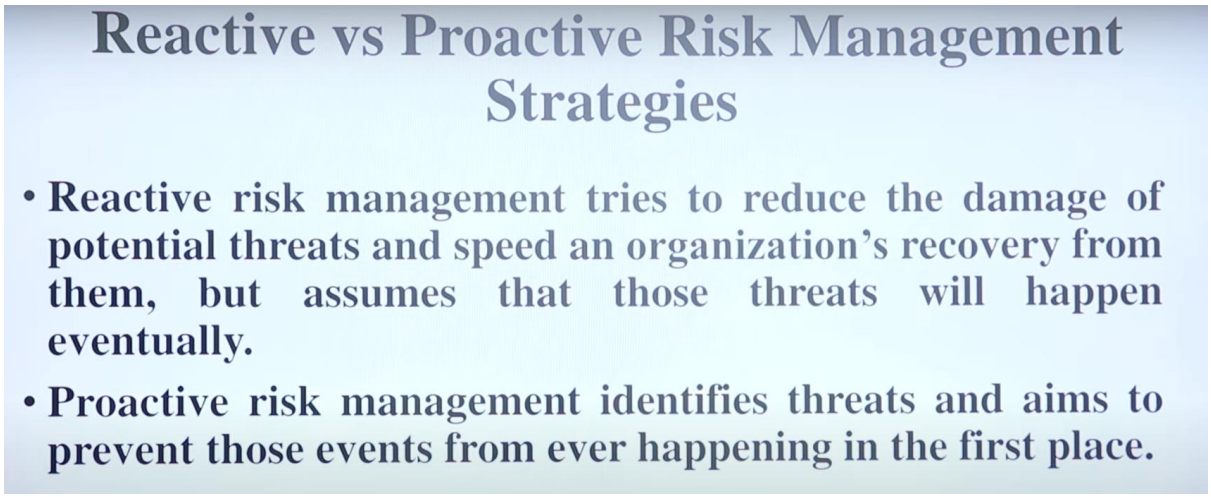
- Estimation(Cost, Duration, Effort)
- Staffing(Staff organization, Staff plans)
- Scheduling manpower & other resources
- Risk Management
- Miscellaneous Plans (Quality assurance plan, configuration & Installation plans)

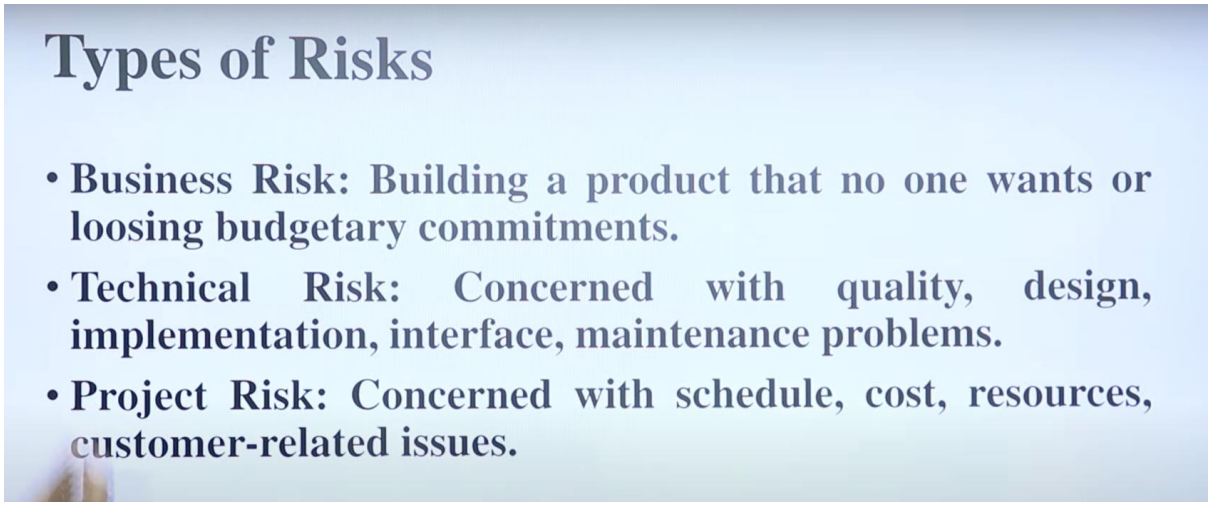
Precedence Ordering

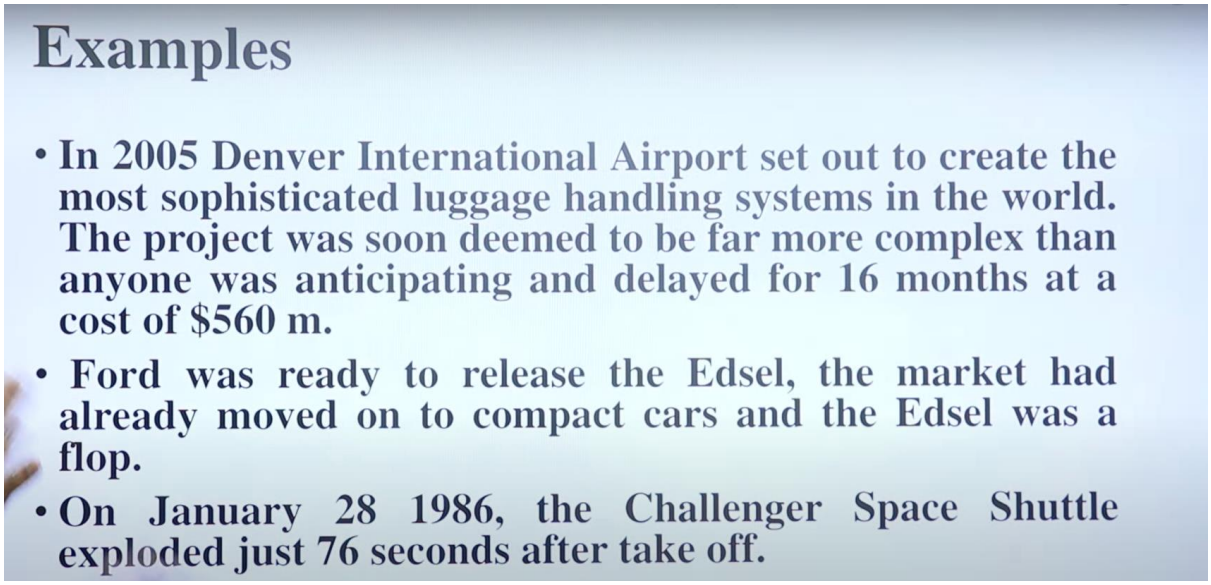


Risk Identification | Reactive vs Proactive Risk Management |Types of Risks with real life examples

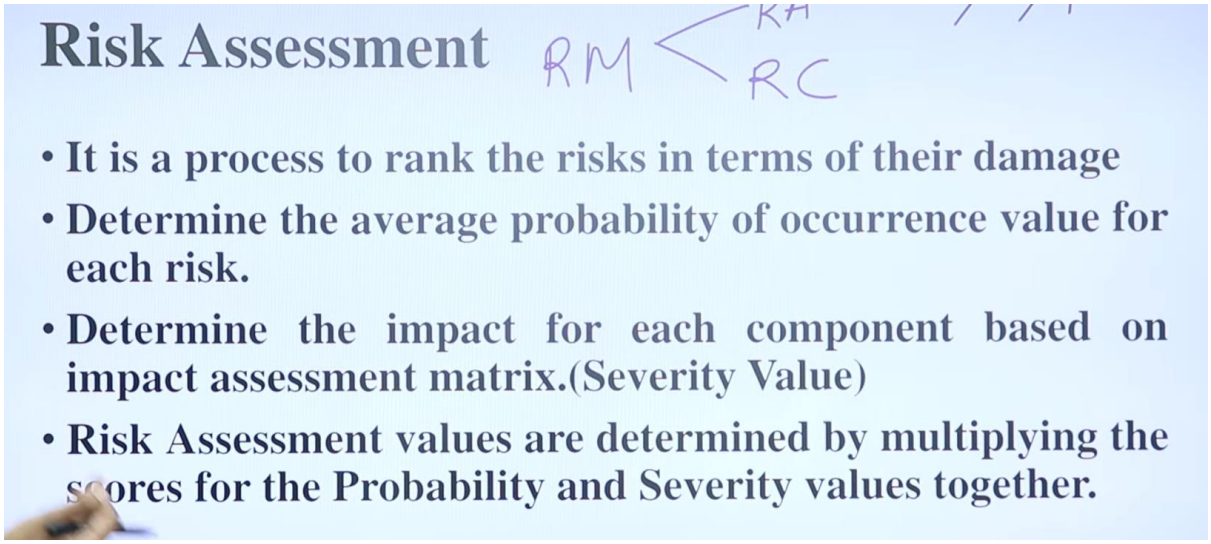


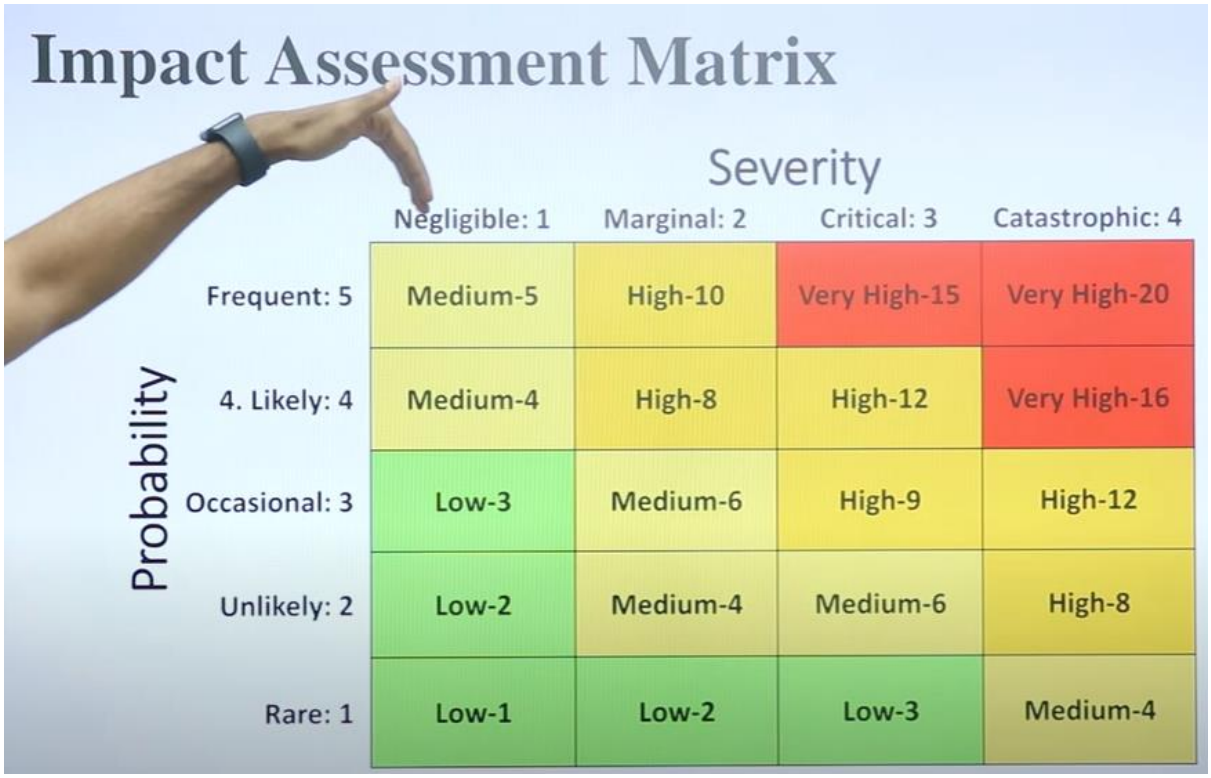




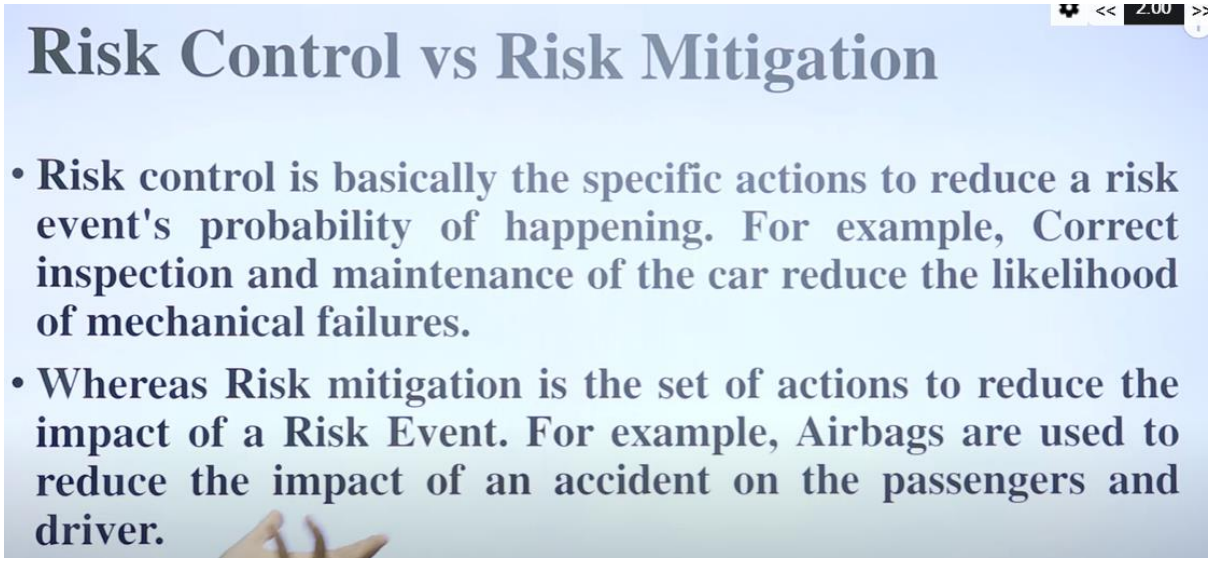


Risk Assessment with examples | Risk Management | Software Engineering

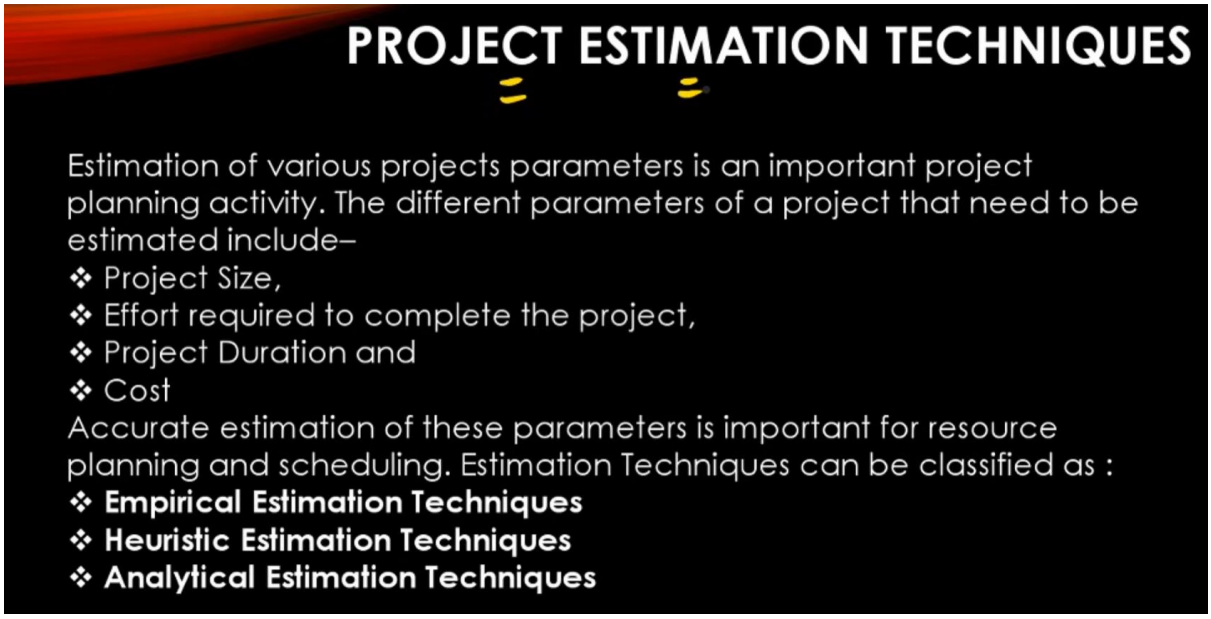


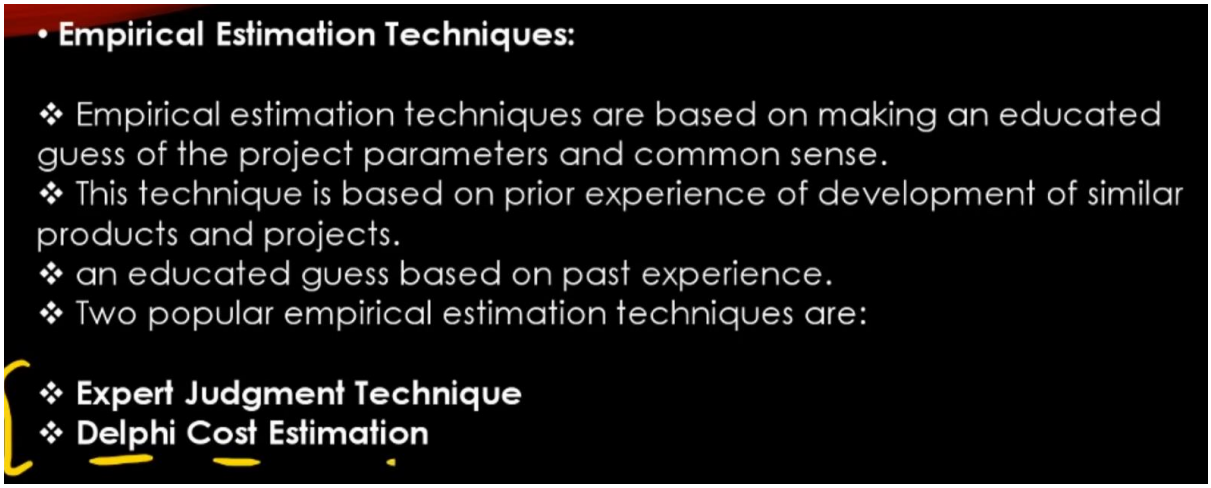


Risk Control vs Risk Mitigation with example



Basic COCOMO & Intermediate COCOMO with Numerical





• **Expert Judgment Technique –**

❖ In this an expert makes an educated guess of the problem size after analyzing the problem thoroughly.

❖ The expert estimates the cost of the different components of the system: e.g. GUI, database module, communication module, billing module, etc.

❖ Combines them to arrive at the overall estimate.

• **Delphi Cost Estimation-**

❖ Is carried out by a team comprising of a group of experts and a coordinator.

❖ 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.

• **Heuristic Techniques:**

❖ In Heuristic Techniques the relationship that exist among the different project parameters can modeled using suitable mathematical expressions.

❖ Once the independent parameters are known, the dependent parameters can be easily determined by substituting the values of the independent parameters in the corresponding mathematical expressions .

❖assume that the characteristics to be estimated can be expressed in terms of
some mathematical expression.

❖Can be classified as Single variable and multivariable models.

• **Single Variable Models:**

$$\text{Estimated Parameter} = c1^{2*} e^{d1}$$

❖ In the above expression, e is the characteristic of the software which has already been estimated (independent variable).

❖ Estimated Parameter is the dependent parameter to be estimated.

❖ The dependent parameter to be estimated could be effort, project duration, staff size, etc.

❖ c1 and d1are constants and are usually determined using data collected from historical data.

❖ The basic COCOMO model is an example of single variable cost estimation model.

•**A Multivariable Cost Estimation** model takes the following form:

Estimated Resource = $c1*e1^{d1} + c2*e2^{d2} + \ldots$

❖Where e1, e2, ... are the basic (independent) characteristics of the software already estimated

❖ c1, c2, d1, d2, ... are constants.

❖The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.

COCOMO(CONSTRUCTIVE COST MODEL)

- Was first proposed by Dr. Barry Boehm in 1981.
- Is a heuristic estimation technique- this technique assumes that relationship among different parameters can be modeled using some mathematical expression.
- This approach implies that size is primary factor for cost, other factors have lesser effect.
- "constructive" implies that the complexity.
- COCOMO prescribes a three stage process for project estimation.
- An initial estimate is obtained, and over next two stages the initial estimate is refined to arrive at a more accurate estimate.
- projects used in this model have following attributes :-
  1. ranging in size from 2,000 to 100,000 lines of code
  2. programming languages ranging from assembly to PL/I.
  3. These projects were based on the waterfall model of software

- projects used in this model have following attributes :-
  1. ranging in size from 2,000 to 100,000 lines of code
  2. programming languages ranging from assembly to PL/I.
  3. These projects were based on the waterfall model of software development.
- Boehm stated that any software development project can be classified into three categories :-

1. Organic:

❖ If the project deals with developing a well understood application program.

❖ The size of development is reasonably small and experienced.

❖ The team members are experienced in developing similar kind of projects.

2. Semidetached:

- If the development team consists of a combination of both experienced and inexperienced staff.
- Team members have limited experience about some aspects but are totally unfamiliar with some aspects of the system being developed.
- Mixed Experience.

3. Embedded:

- If the software being developed is strongly coupled to complex hardware.
- Software projects that must be developed within a set of tight software, hardware and operational constraints.

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.	Significan	Tight	Complex Hardware/ customer Interfaces required

PERSON MONTH(PM)

- The effort estimation is expressed in units of person-months (PM).
- 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 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. As requirement of staff varies time to time in the development so there is not constant no of people is there to work. It is calculated from the graph(calculate area under time axis) between no of people working and time(in month).

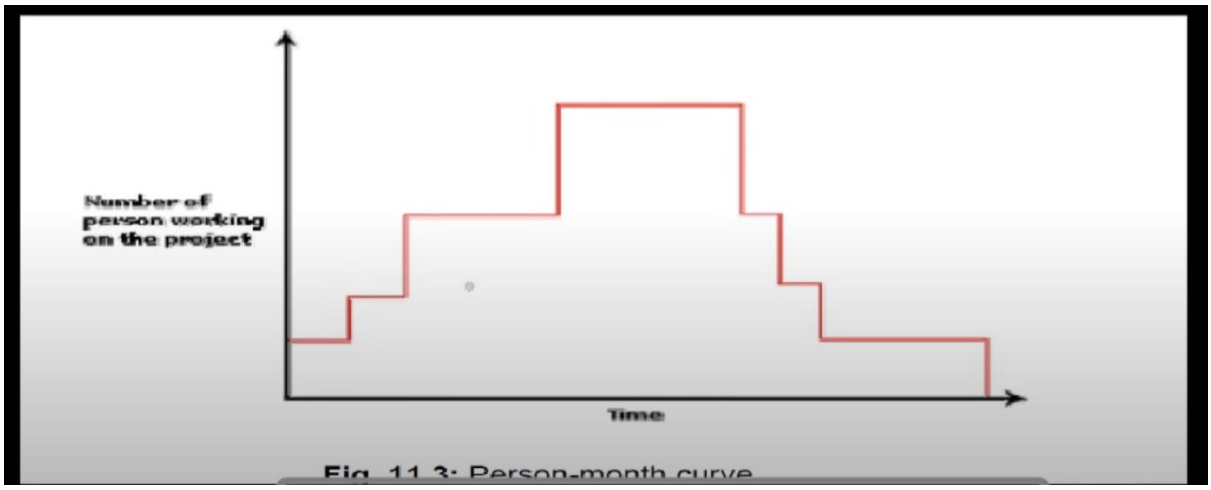


Fig. 11.3: Person-month curve

1. BASIC COCOMO MODEL

- Basic COCOMO model computes software development effort, time and cost as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC, KLOC).

$$EFFORT = a_1 \times (KLOC)^{a2} \ PM$$
$$T_{dev} = b_1 \times (Effort)^{b2} \ Months$$

Where

- KLOC is the estimated number of delivered lines (expressed in thousands) of code for project, estimated size of software product.
- The coefficients a_1, a_2, b_1 and b_2 are constants for each category of software products.
- T_{dev} is the estimated time to develop the software, in months.
- Effort is the total efforts required to develop the software product, expressed in Person Months (PM)

ESTIMATION OF DEVELOPMENT TIME

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

Software Projects	a_1	a_2	b_1	b_2
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

- Example: consider a software project using semi-detached mode with 30,000 lines of code . We will obtain estimation for this project as follows:

(1)Effort estimation E= a1(KLOC)Exp(a2)person-months

E=3.0(30)Exp(1.12) where lines of code=30000=30 KLOC

E=135 person-month

(2) Duration estimation D=b1(E)Exp(b2)months =2.5(135)Exp(0.35)

D=14 months

(3)Person estimation N=E/D

=135/14 N=10 persons approx.

CALCULATING EFFORT AND PRODUCTIVITY

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

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

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

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

Merits-

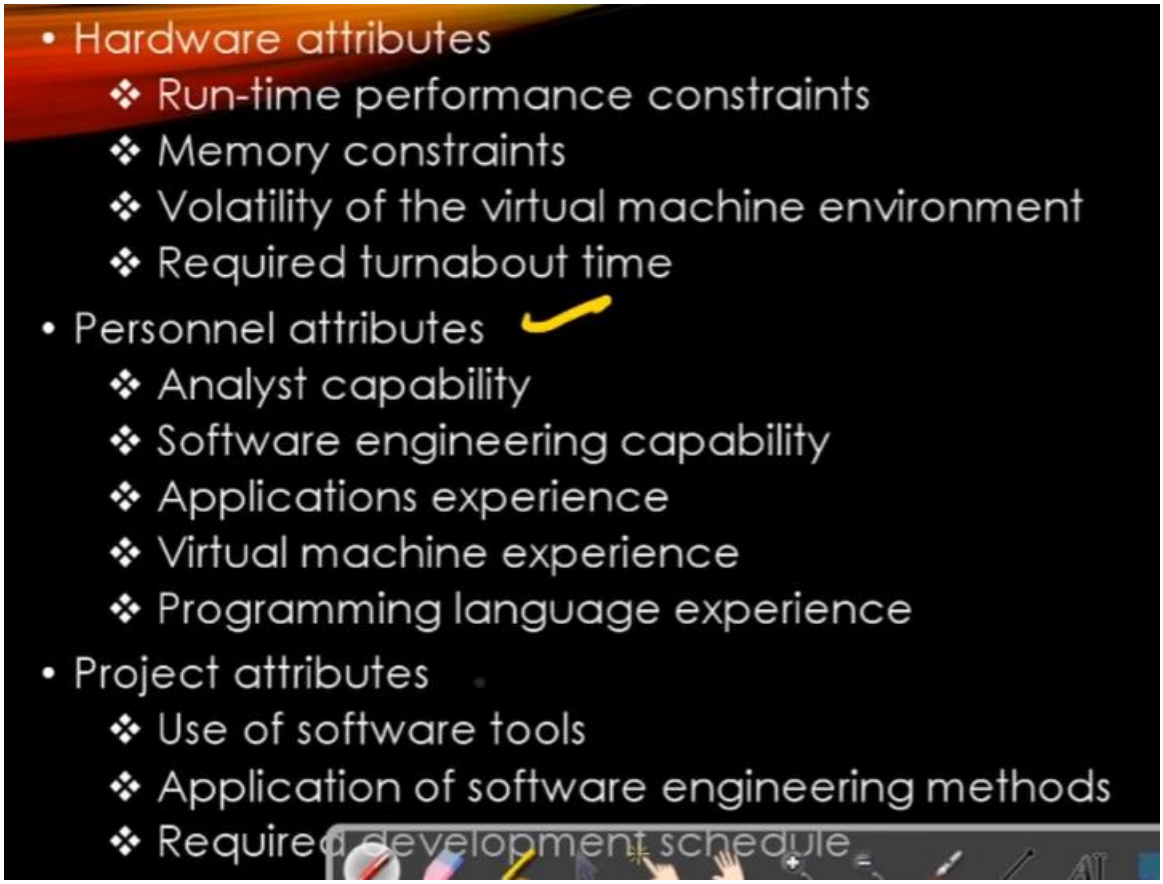
- Basic COCOMO is good for quick , rough and early estimate of software costs.

Demerits-

- It does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.
- The accuracy of this model is limited because it does not consider certain factors for cost estimation of software.

INTERMEDIATE COCOMO MODEL

- *Intermediate* COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes.
- The Intermediate COCOMO model refines the initial estimate obtained from the basic COCOMO by scaling the estimate up or down based on attributes of software development.
- This model uses a set of 15 cost drivers, these cost drivers are multiplied with the initial cost and effort estimates to scale the estimates up and down.
- This extension considers a set of "cost drivers", each with a number of subsidiary attributes:
- Product attributes
  - ❖ Required software reliability
  - ❖ Size of application database
  - ❖ Complexity of the product



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		0.07	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	

- Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value).
- formula now takes the form:

$$E=a_1 (KLOC)^{(b1)} \times (EAF)$$

Where

E : Effort applied in terms of person- months

KLOC : Kilo lines of code for the project

EAF : It is the effort adjustment factor

- The value of a1 and b1 for various class of software projects:

Software Projects	a_1	b_1
Organic	3.2	1.05
Semi – Detached	3.0	1.12
Embedded	2.8	1.20

Example:

Consider a project having 30,000 lines of code which in an embedded software with critical area hence reliability is high.The estimation can be

$$E=a_1 (KLOC)^{a2} \times (EAF)$$

As reliability is high EAF=1.15(product attribute)

a_i=2.8

b_i=1.20. for embedded software

$E=2.8(30)^{1.20} \times 1.15$

=191 person month

$D=b_1 (E)^{b2}$

$=2.5(191)^{0.32}$

=13 months approximately

N=E/D =191/13

N=15 persons approx.

Merits:

- This model can be applied to almost to entire software product for easy and rough cost estimation during early stage.
- It can be applied at the software product component level for obtaining more accurate cost estimation.

Demerits:

- The effort multipliers are not dependent on phases.
- A product with many components is difficult to estimate.

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.
  - ❖ So, complete COCOMO was proposed to overcome these limitations of basic and intermediate COCOMO.

COMPLETE COCOMO MODEL

- Incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.
- The complete COCOMO model considers the differences in characteristics of all the subsystems and estimates the effort and development time as sum of the estimates for the individual sub systems.
- Uses different effort multipliers for each cost driver attribute. These **Phase Sensitive** effort multipliers are each to determine the amount of effort required to complete each phase. In complete COCOMO, the whole software is divided in different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort
- The effort is calculated as function of program size and a set of cost drivers given according to each phase of software life cycle.
- A complete project schedule is never static.

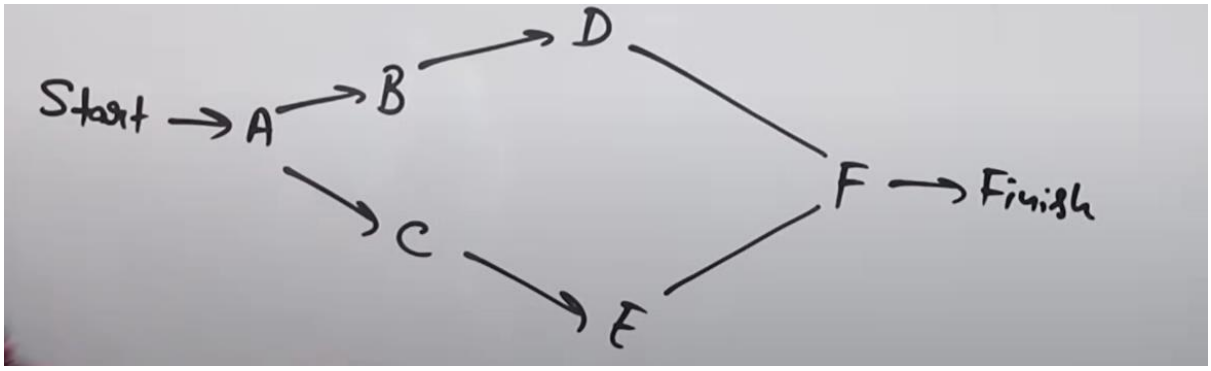
- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

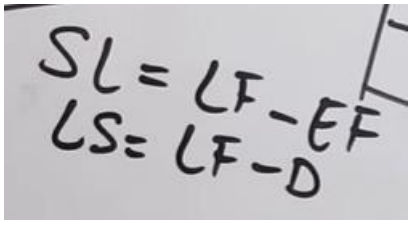
- **Example of complete COCOMO Model-**
- A Management Information System (MIS) for an organization having offices at several places across the country:
  - Database part (semi-detached)
  - Graphical User Interface (GUI) part (organic)
  - Communication part (embedded)
- Costs of the components are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

Ques. Do CPM (Critical Path Method) analysis and identify the critical path of project.

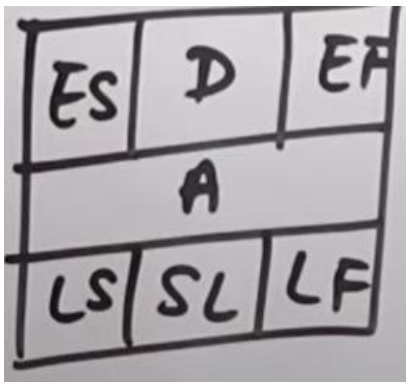
Activity	Predecessor	Duration (in months)
A	–	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D, E	4

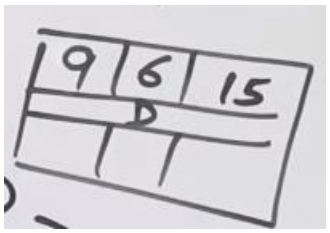
a) A → C → E → F
b) A → B → D → F
c) A → B → E → F
d) A → C → D → F

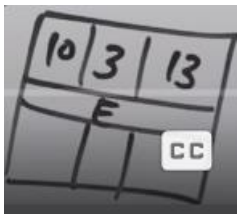




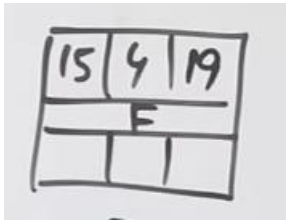
For each node in forward pass



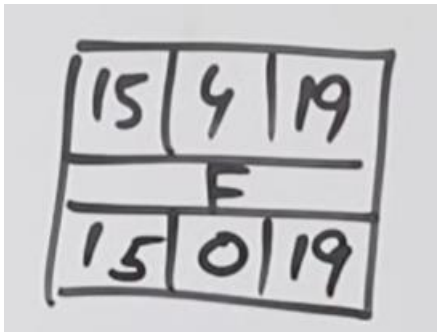




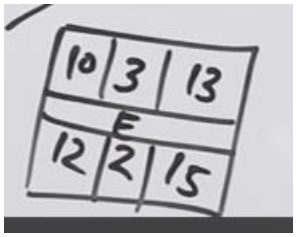
In forward pass we take the maximum of D and E

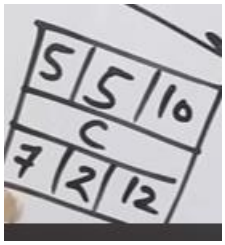


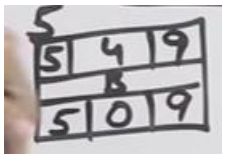
For each node in backward pass



In backward pass latest finish of D and E depends on latest start of F.







In backward pass we take the minimum of 5 and 7

a) A → C → E → F

~~b)~~ A → B → D → F

xc) A → B → E → F

xd) A → C → D → F

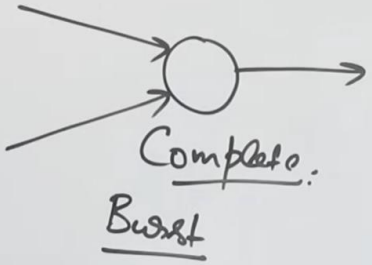
Q.) In PERT/CPM, the Merge event represents ____ of two or more events.

a) Splitting

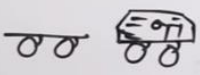
~~b)~~ Completion

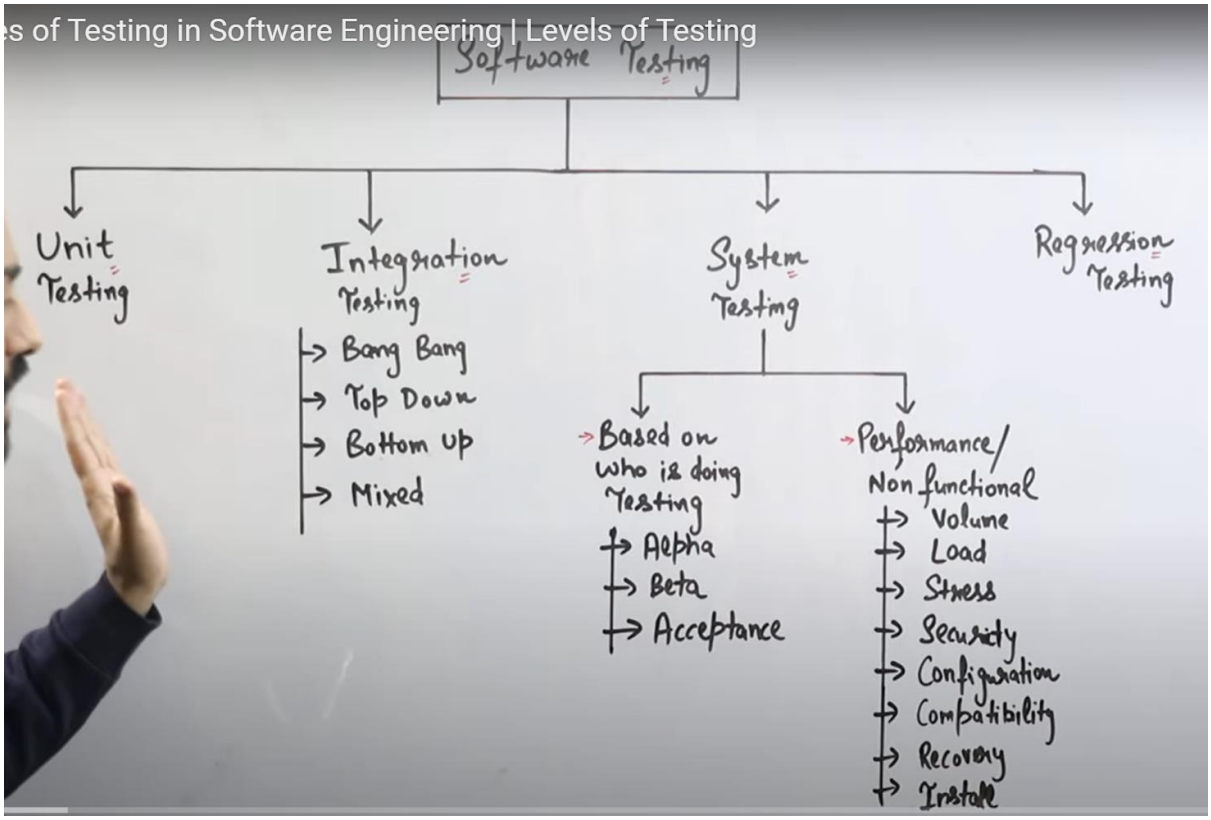
c) beginning

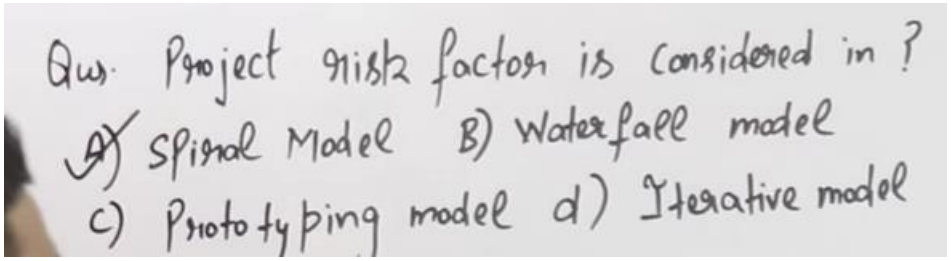
d) Joining

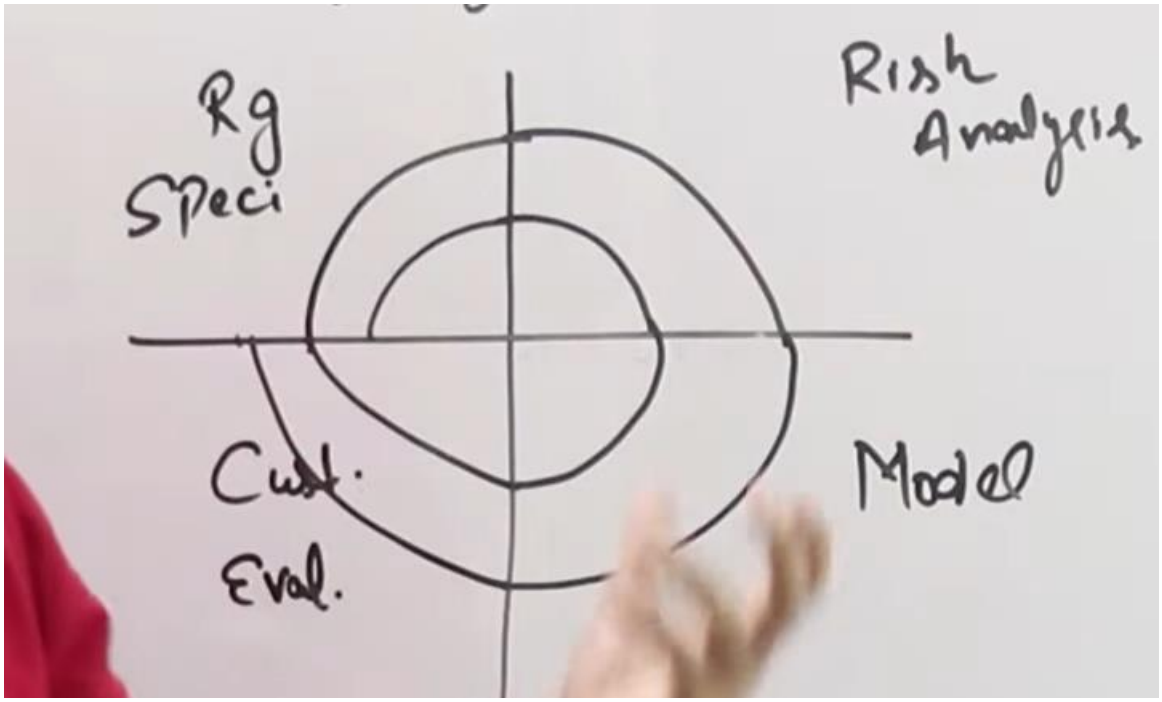


"Verification"	"Validation"
1) Are you building it Right ?	1) Have you build the Right thing ?
2) Check wheather an artifact Conforms its previous artifact.	2) Check the final Product against specification.
3) Done by Developer	3) Done by Tester
4) Concered with phase Containment of errors.	4) Aim is to make final product error free.
5)* Methods involves Review, Inspection, Unit testing and integration testing.	5) Involves System Testing.
6) Static and Dynamic Activities	6) Only Dynamic




s of Testing in Software Engineering | Levels of Testing
Software Testing
Unit Testing
Integration Testing
Bang Bang
Top Down
Bottom up
Mixed
System Testing
Based on who is doing Testing
Alpha
Beta
Acceptance
Performance/ Non functional
Volume
Load
Stress
Security
Configuration
Compatibility
Recovery
Install
Regression Testing


Qus. Project risk factor is considered in ?
A) Spiral Model
B) Waterfall model
C) Prototyping model
d) Iterative model


Rg
Speci
Risk
Analysis
Cust.
Eval.
Model

Qus. The no. of function points of a proposed system is calculated as 500. Suppose the system is planned to develop in JavA and LOC/FP ratio of Java is 50. Estimate the effort (E) required to Complete the project using the effort formula of basic COCOMO given below

$E = a(KLOC)^b$ Assume $a = 2.5$

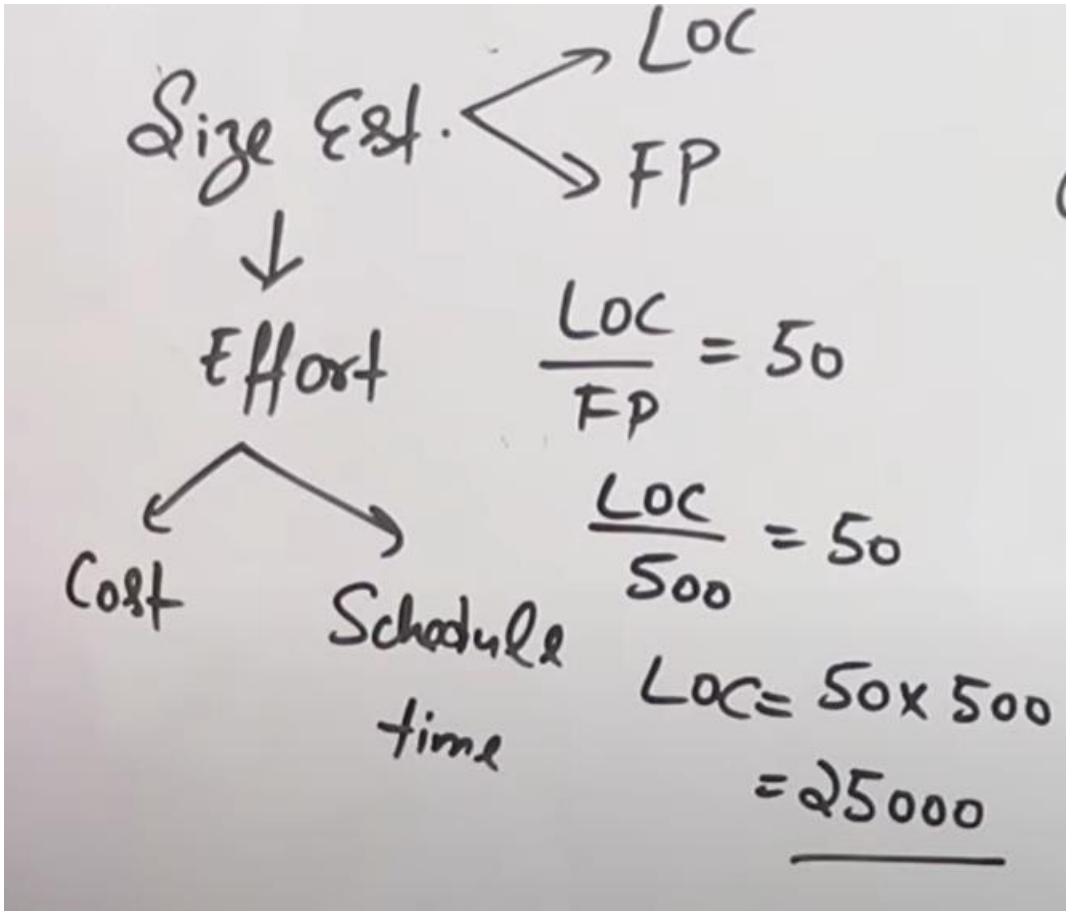
$b = 1.0$

A) 25 Person months

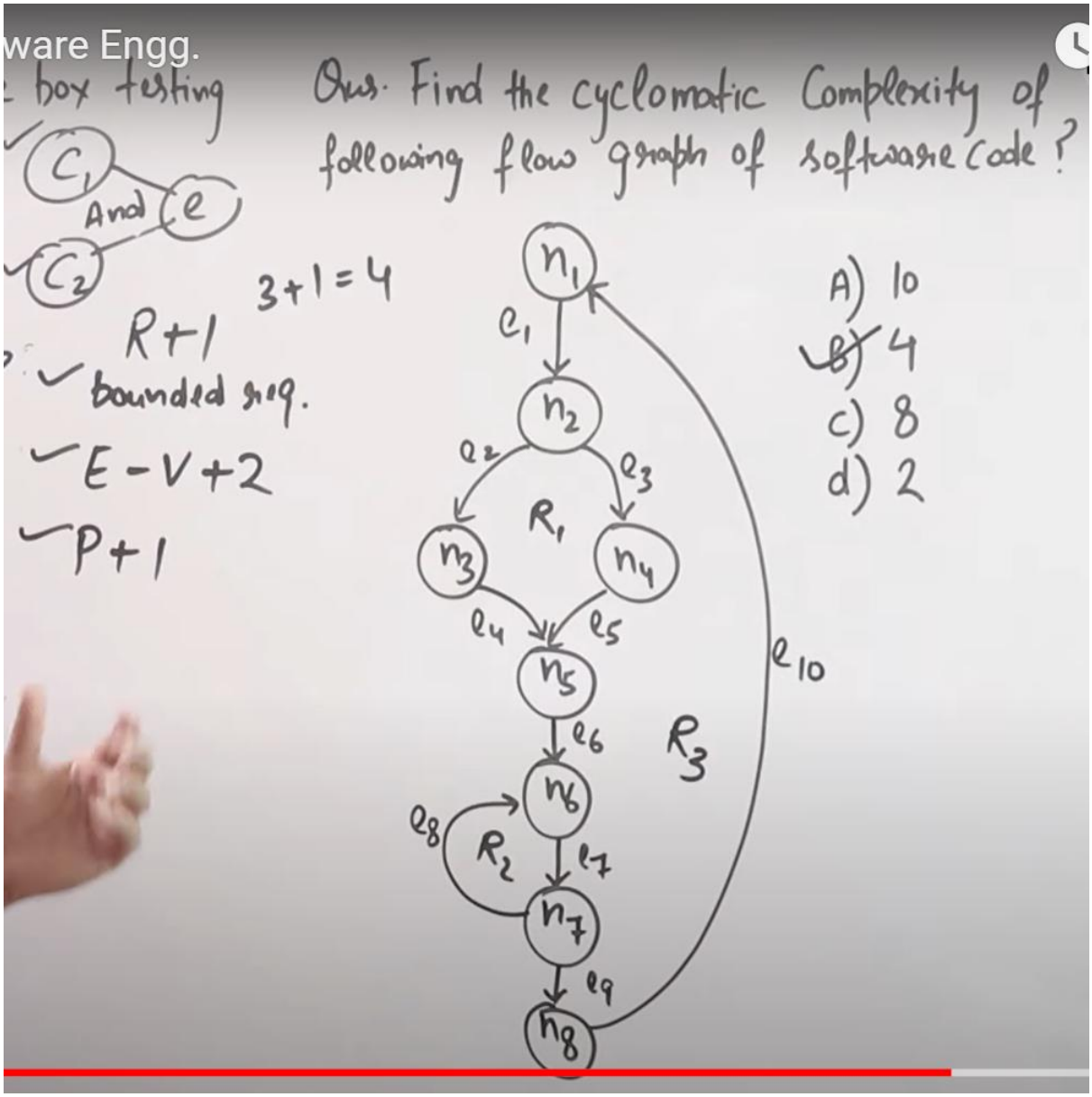
B) 75 pm

C) 62.5 pm

d) 72.5 pm


Size Est.
LOC
FP
Effort
Cost
Schedule time
LOC / FP = 50
LOC / 500 = 50
LOC = 50 x 500
= 25000

$$= 2.5(25)^{1.0}$$
$$= 62.5$$

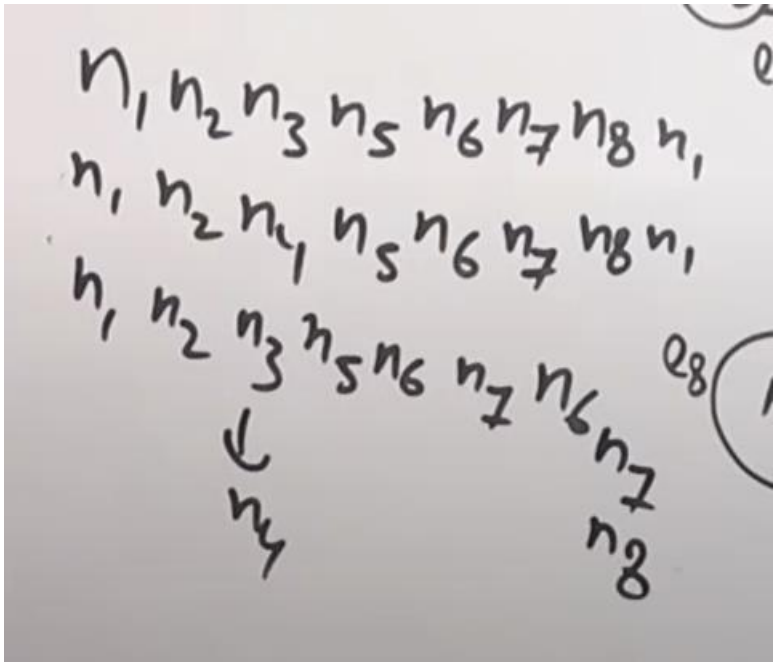


No. of independent paths = cyclomatic complexity

R = Bounded Regions

E = Edges, V = Vertices

P = Predicate nodes (nodes from where two vertices come out or conditional nodes) for e.g. n2



Introduction

Design phase transforms SRS document:

into a form easily implementable in some programming language.

A module consists of:

1. several functions
2. associated data structures.

Modularity

Modularity is a fundamental attributes of any good design.

Decomposition of a problem cleanly into modules:

1. Modules are almost independent of each other
2. divide and conquer principle.

Modularity

If modules are independent:

- modules can be understood separately.
- reduces the complexity greatly.

To understand why this is so,

- remember that it is very difficult to break a bunch of sticks but very easy to break the sticks individually.

In technical terms, modules should display:

1. high cohesion
2. low coupling.

Cohesion and Coupling

Cohesion is a measure of:

- functional strength of a module.

- A cohesive module performs a single task or function.

Coupling between two modules:

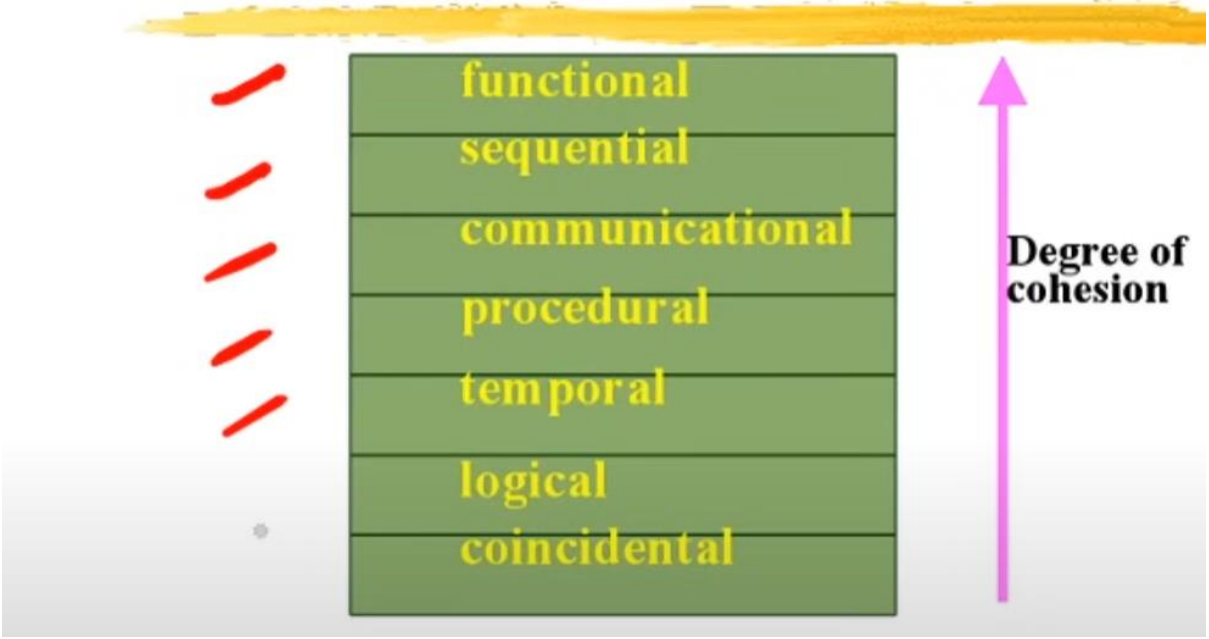
- a measure of the degree of interdependence or interaction between the two modules.

A module having high cohesion and low coupling:

functionally independent of other modules:

A functionally independent module has minimal interaction with other modules.

Classification of Cohesiveness



Coincidental cohesion

The module performs a set of tasks:
which relate to each other very loosely, if at all.

- the module contains a random collection of functions.
- **cohesion** measures the strength of relationship between pieces of functionality within a given module.

- It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

Logical cohesion

All elements of the module perform similar operations:
e.g. error handling, data input, data output, etc.

When logically categorized elements are put together into a module, it is called logical cohesion.

Temporal cohesion

The module contains tasks that are related by the fact that all the tasks must be executed in the same time span.

When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

Procedural cohesion

If the set of functions of the module all part of a procedure (algorithm) in which certain sequence of steps have to be carried out in a certain order for achieving an objective,

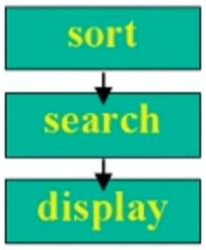
e.g. the algorithm for decoding a message.

When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

Sequential cohesion

If the Elements of a module forms different parts of a sequence, output from one element of the sequence is input to the next.

Example:



When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.

Functional cohesion

If the Different elements of a module cooperate to achieve a single function.

It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

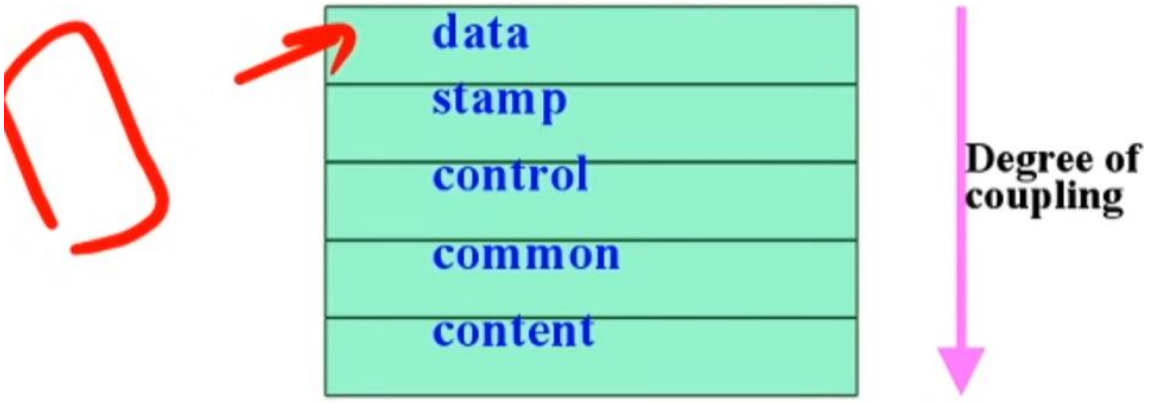
Coupling

Coupling indicates:

- how closely two modules interact or how interdependent they are.

- The degree of coupling between two modules depends on their interface complexity.

Classes of coupling



Data coupling

Two modules are data coupled,

if they communicate by an elementary data item that is passed as a parameter between the two, eg an integer, a floa, character etc.

Content coupling

Content coupling exists between two modules:

if they share code,

e.g, branching from one module into another module.

The degree of coupling increases from data coupling to content coupling.

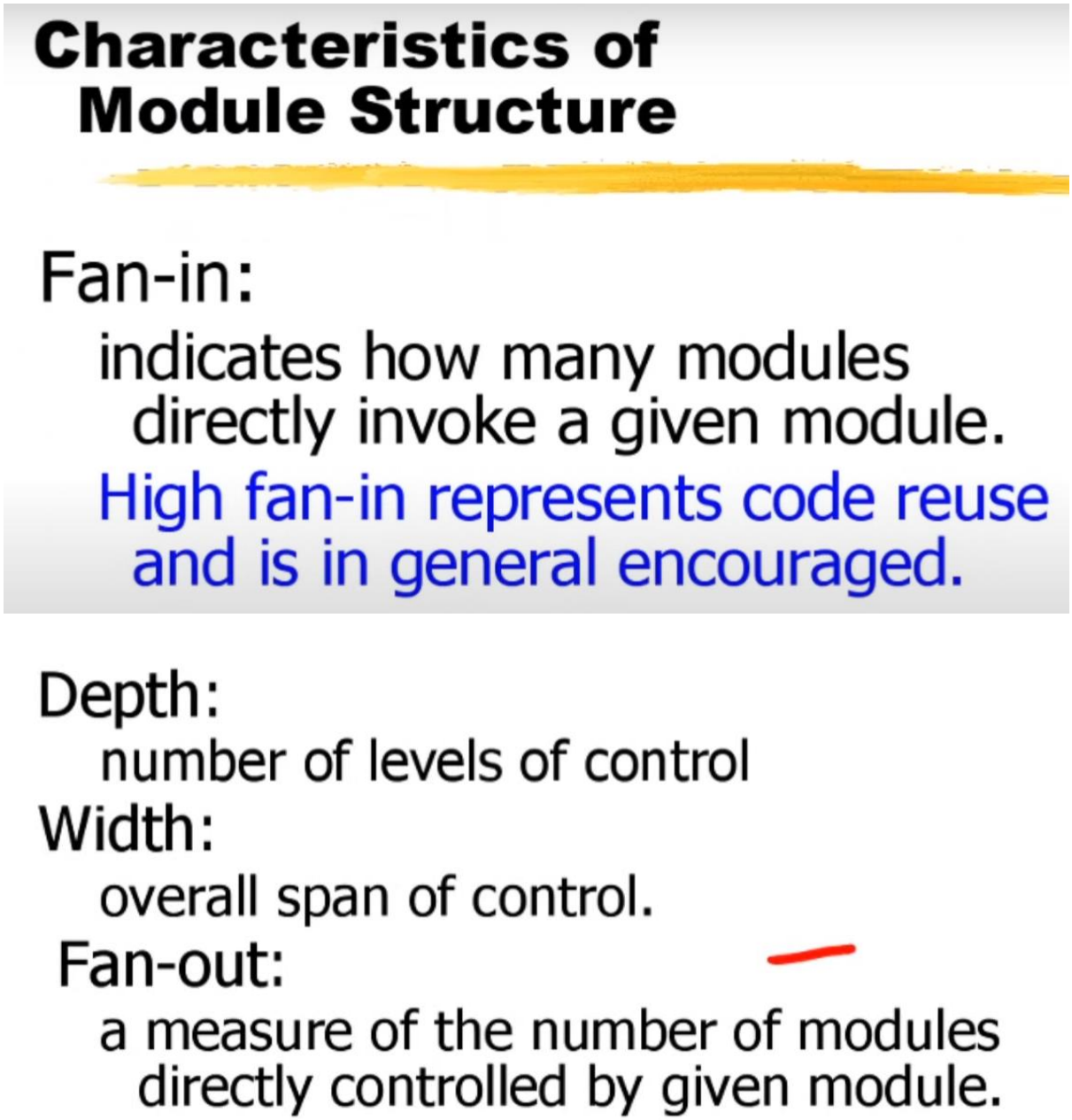
Stamp coupling

Two modules are stamp coupled,

if they communicate via a composite data item: (Data Structure) eg: linked list

- such as a record in PASCAL

or a structure in C.

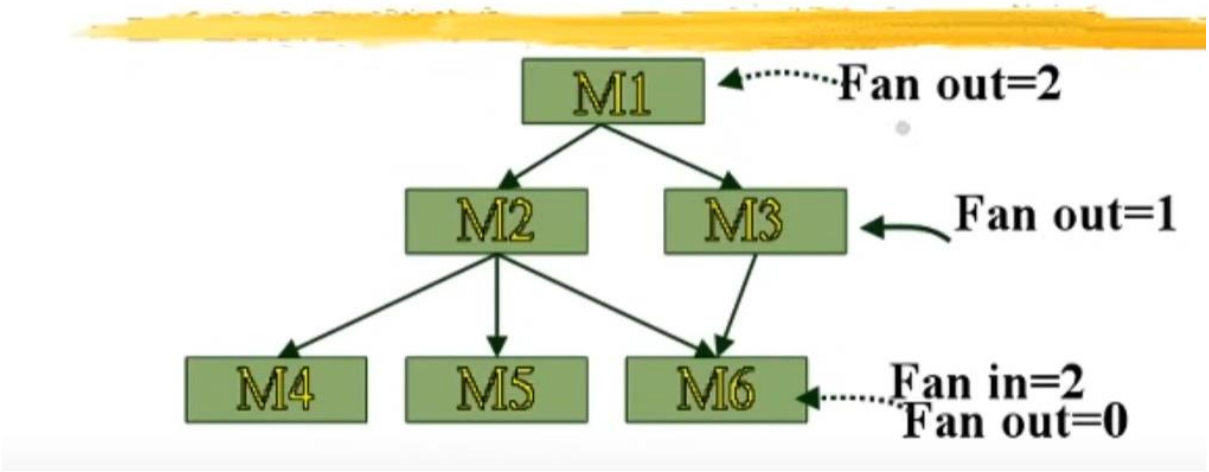


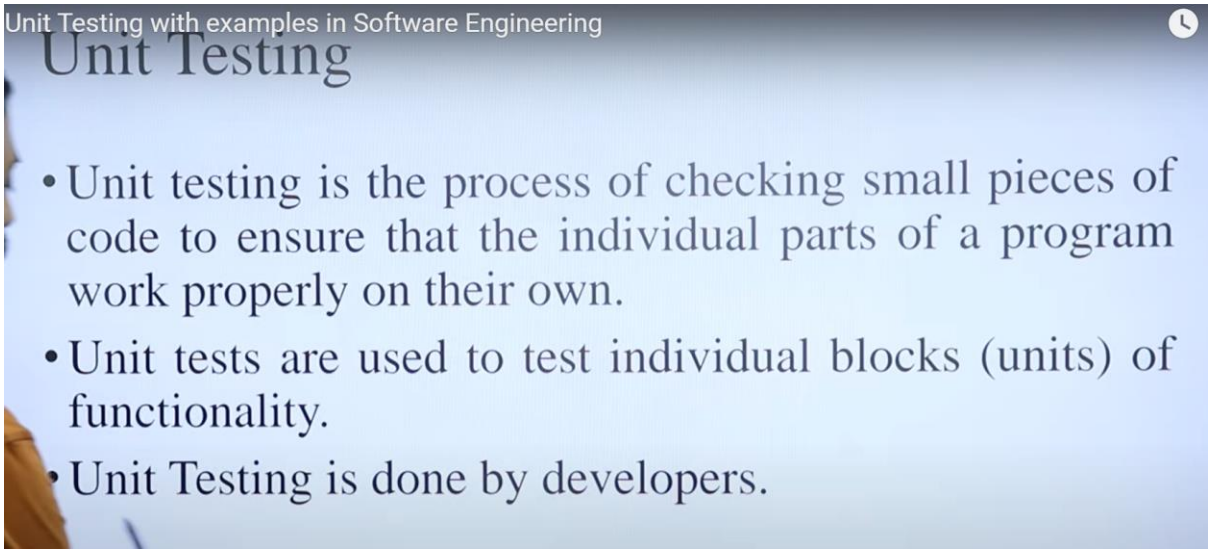
Fan in should be more and Fan out should be less.

In degree = Fan in

Out degree = Fan out

Module Structure



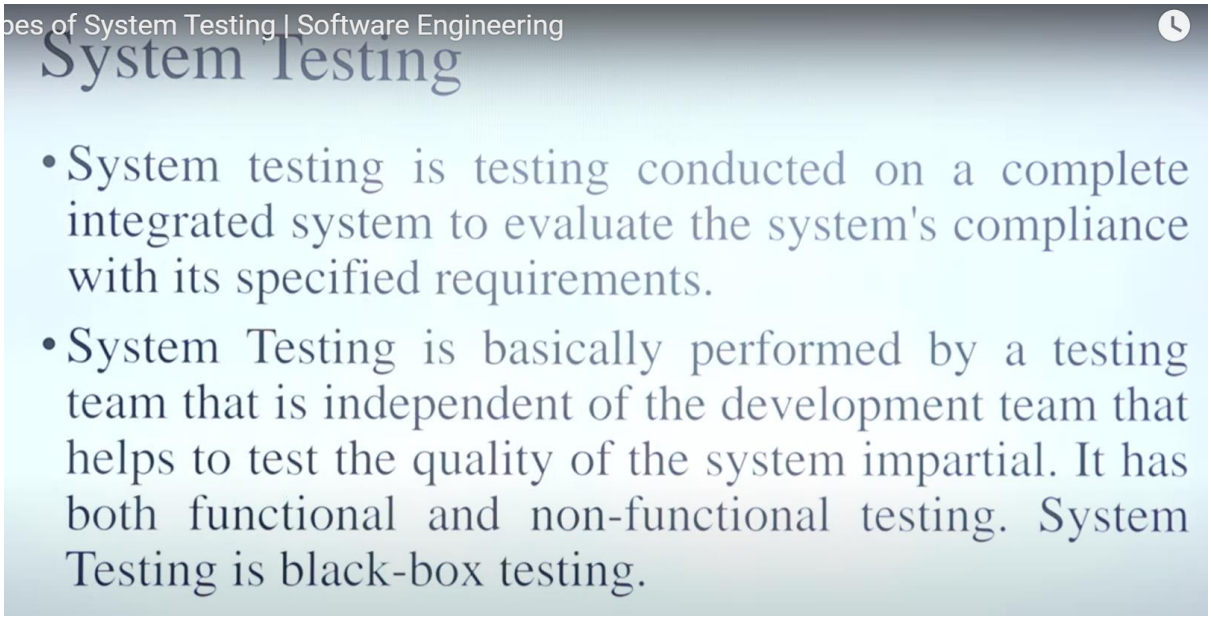


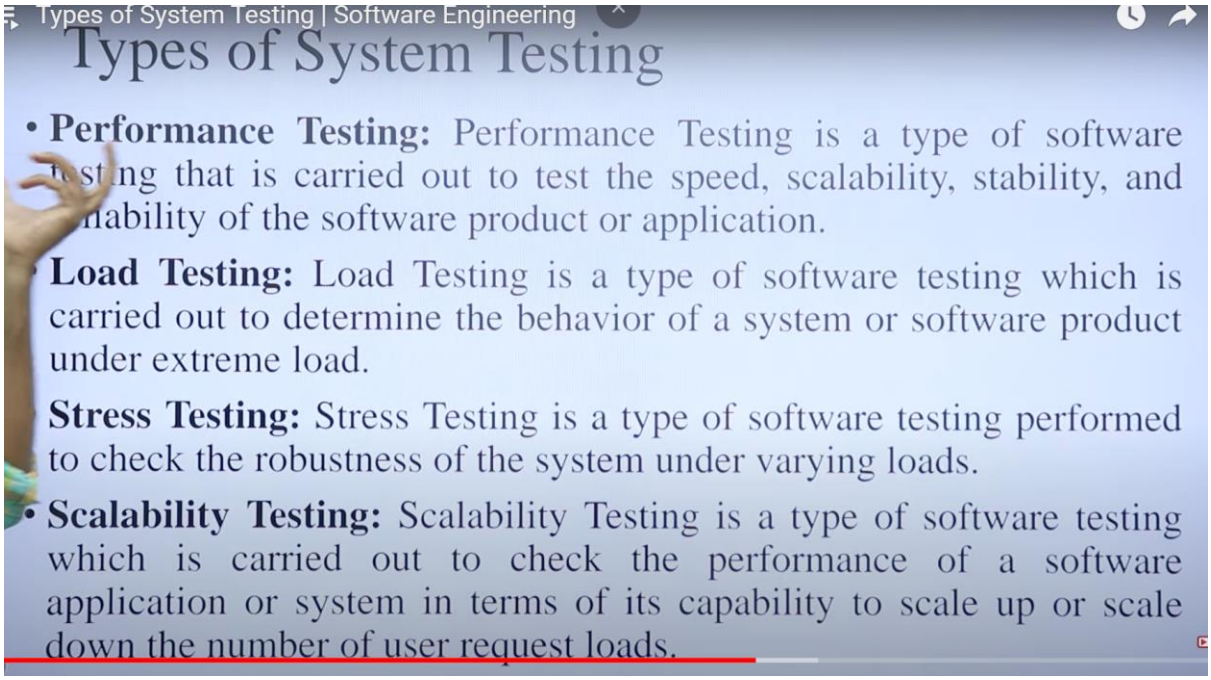
Integration Testing

- Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.
- It occurs after unit testing and before system testing.
- Types of Integration Testing
    1) Big-bang
    2) Mixed (Sandwich)
    3) Top-down
    4) Bottom-up.

System Testing

- System Testing is a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- System Testing is a black-box testing.
- System testing categories based on: Who is Doing the Testing?
- System testing categories based on: Functional/Non-Functional Requirements?





https://www.javatpoint.com/white-box-testing

White-box testing

- White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of software testing that tests internal structures or workings of an application.
- White-box testing can be applied at the unit, integration and system levels of the software testing process.
- Here are some of the top white box testing tools to use: Veracode, CppUnit, Nunit, RCUNIT etc.

White-box test design techniques

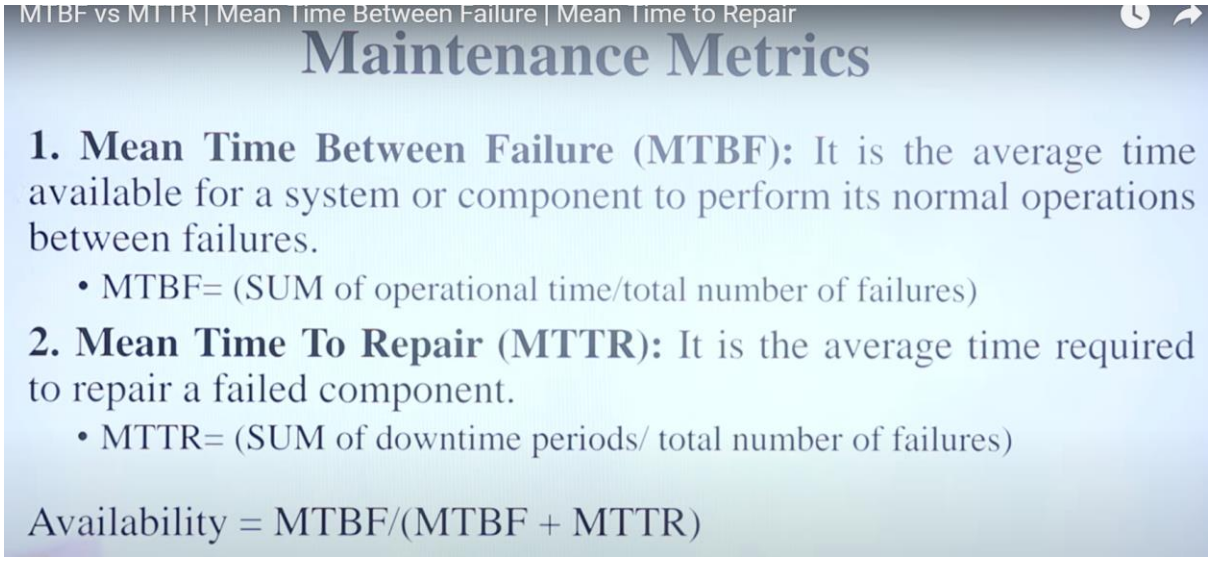
- Control flow testing
- Data flow testing
- Branch testing
- Statement coverage
- Decision coverage
- Path testing

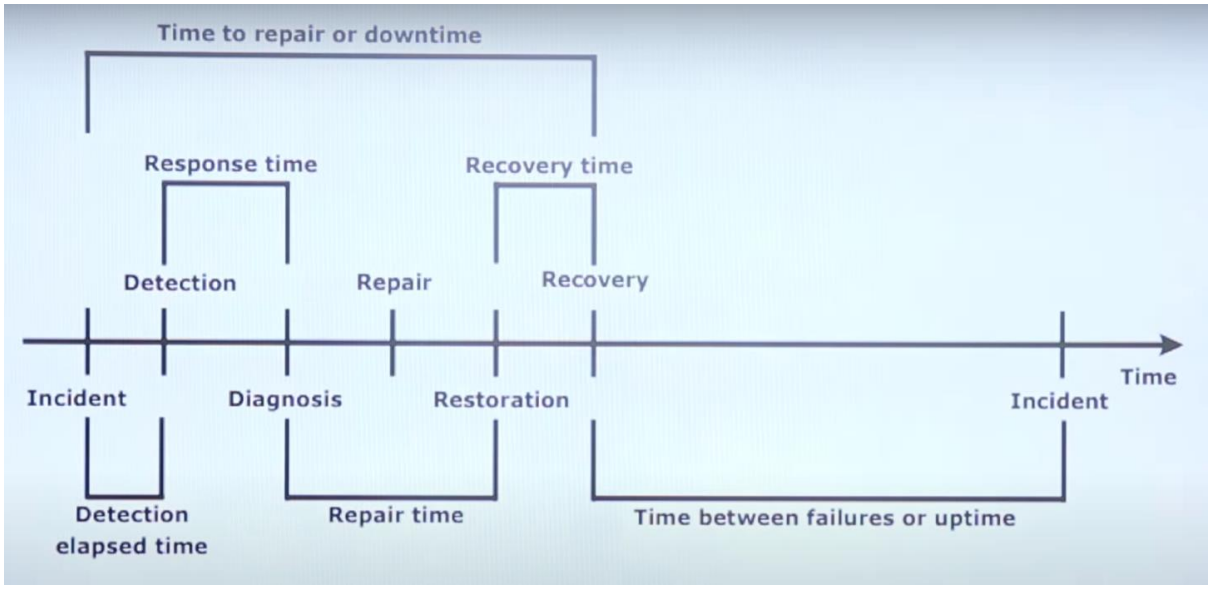
White-box testing	Black box testing
The developers can perform white box testing.	The test engineers perform the black box testing.
what the software is supposed to do, also aware of how it does it.	what the software is supposed to do but is not aware of how it does it.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.
Test design techniques:Control flow testing, Data flow testing, Branch testing, Statement coverage, Decision coverage, Path testing.	Test design techniques: Decision table testing, All-pairs testing, Equivalence partitioning, Boundary value analysis, Cause–effect graph
Can be applied mainly at unit testing level but ... in integration, system level also.	Can be applied virtually to every level of software testing: unit, integration, system and acceptance

https://www.geeksforgeeks.org/statement-coverage-testing/

https://www.guru99.com/code-coverage.html

https://www.geeksforgeeks.org/software-testing-boundary-value-analysis/





Q. A machine that runs for 24 hours. During that time, it failed twice, and each time it took an hour to get it back up and running. Find the MTBF and MTTR.

$2 \times 1 = 2$

$MTBF = \frac{22}{2} = 11$

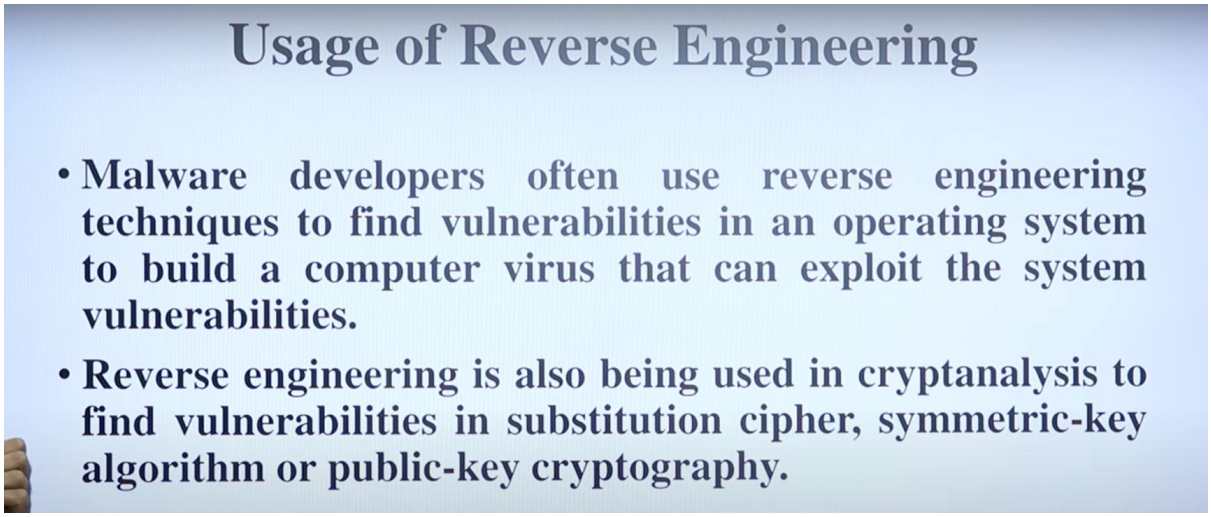
$MTTR = \frac{2}{2} = 1$

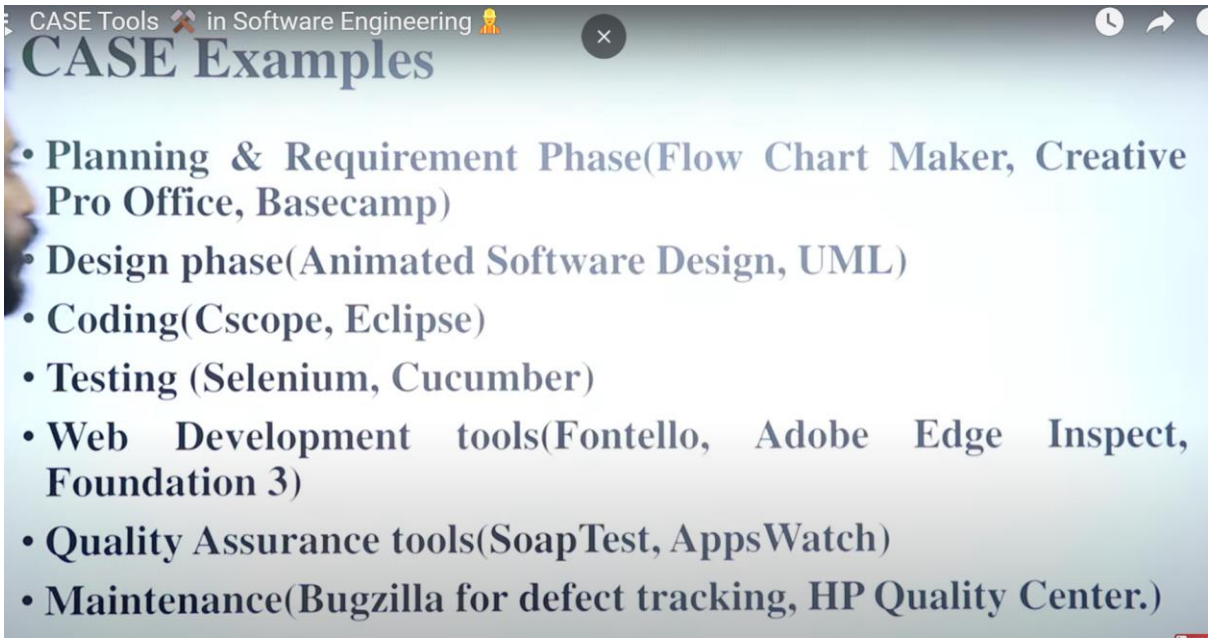
$A = \frac{11}{11+1} = 11/12 = 91.6\%$

Reverse Engineering/ Backward Engineering

- **Software reverse engineering can help to improve the understanding of the underlying source code for the maintenance and improvement of the software.**
- **In some cases the goal of the reverse engineering process can simply be a redocumentation of legacy systems.**

Code → Module Specification → Design → Requirement Specifications





https://www.geeksforgeeks.org/performance-testing-software-testing/

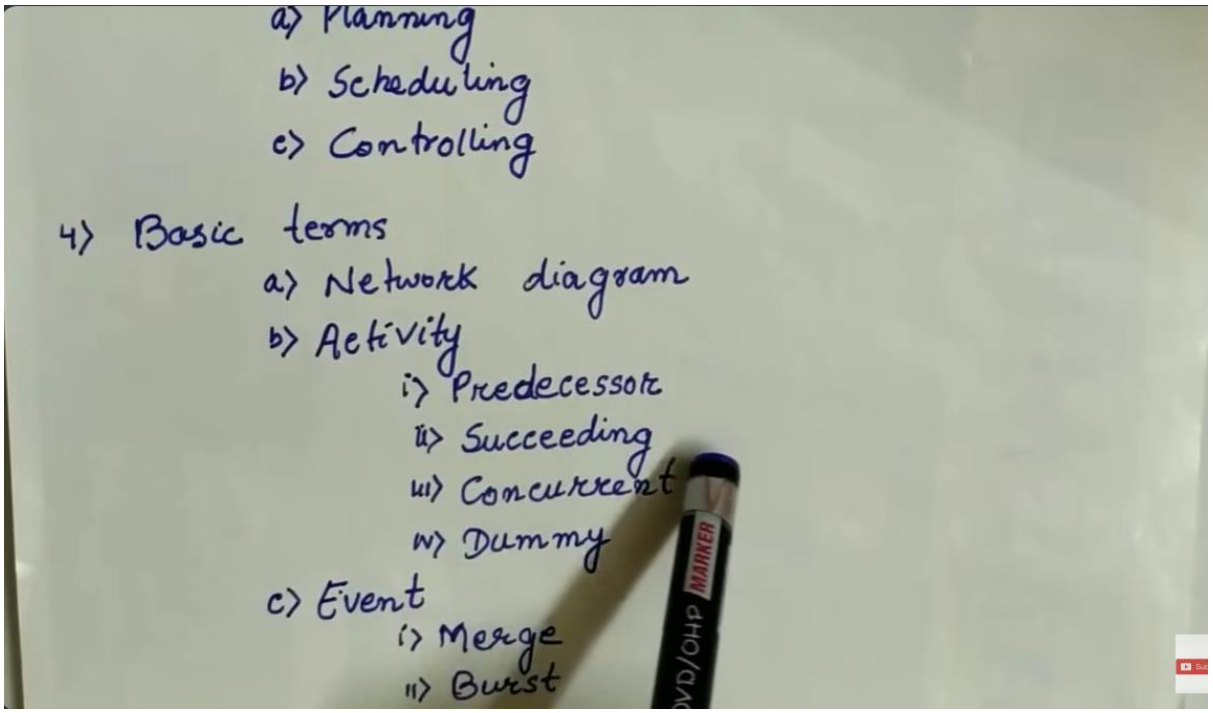
https://www.geeksforgeeks.org/software-engineering-regression-testing/

https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/

Network Diagram and related topics

Topics

1) Introduction of Network Scheduling
2) Project
3) Phases of project
   a) Planning
   b) Scheduling
   c) Controlling
4) Basic terms
   a) Network diagram
   b) Activity
      i) Predecessor



iv) Dummy

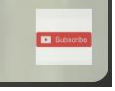
c) Event
- i) Merge
- ii) Burst

5) Common Errors
- a) looping (cycling)
- b) Dangling
- c) Redundancy

6) Rules of Network construction

7) Numbering the events

) Construction of Network



Network Scheduling

8) Time Analysis
- a) forward Pass computation
- b) Backward Pass computation
- c) Determinations of floats and slack times
  - i) Total float
  - ii) free float
  - iii) Independent float
  - iv) Critical activity
  - v) Critical path.

Path Method (CPM)

9) Critical Path Method (CPM)

10) Programme Evaluation An Review technique (PERT)

1) **Introduction** : It is a technique used for planning and scheduling large projects, in the field of construction, maintanance, fabrication and purchasing of computer systems etc.

2) **Projects** : A project is defined as combination of interrelated activities, all of which must be executed in a certain order for its completion.

3) **Phases of project Management** :

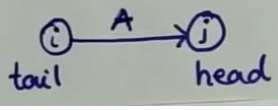
a) **Planning** :
- Divide the project into distinct activities
- Estimate time requirement for activities
- Establish precedence relationship
- Contruct arrow diagram.

b) **Scheduling** :
- Determine the start and end time for activity
- Determine the critical path on which the activity require attention.
- Determine the slack and float for non-critical activity
- Determine the slack and float for non-critical path.

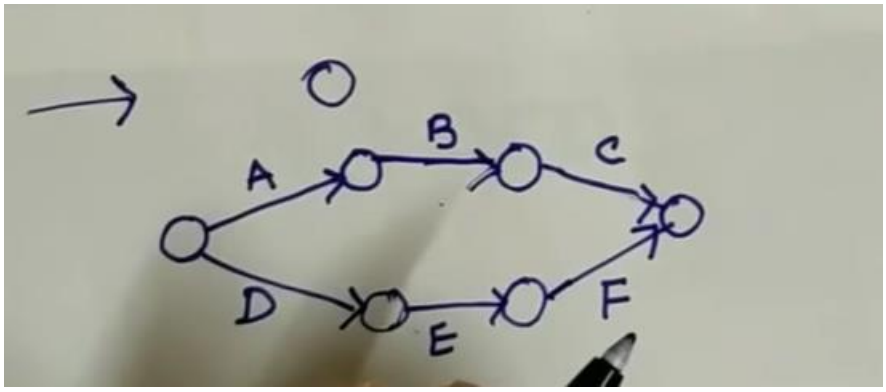
c) **Controlling** :
- Making the periodical progress report
- Reviewing the progress
- Analyzing the status of project

a) **Network diagram** : It is the graphical representation of logically and sequentially connected arrows and node, representing activities and events in a project.

b) **Activity** : It represent some action and is a time consuming effort neccessary to complete a particular part of over all project.



A = activity.



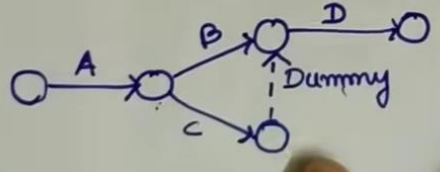
i) **Preceeding activity** : Activity that must be accomplished before a given event can occur.

ii) **Succeeding activity** : Activity that cannot be accomplis until an event has occured.

iii) **Concurrent activity** : Activity takeing place at the same time or in the same location.

time or in the same location.

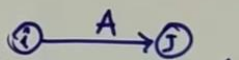
b) Dummy activity : Activity which neither consume time nor resources but are used simply to represent a connection or a link between the events are known as dummies.
It is shown in network by a dotted lines.

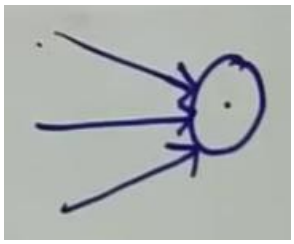


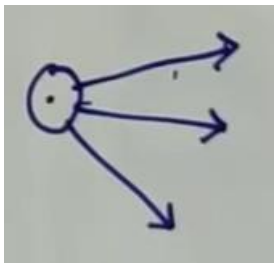
c) Event : Beginning and end points of an activity are called events or nodes.

i) Merge event.

ii) Burst event.

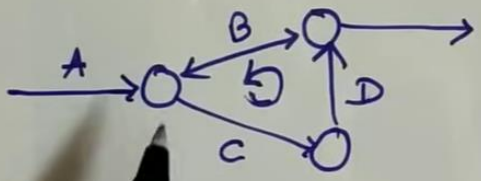



Merge

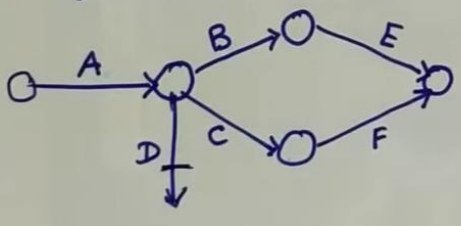

Burst

b) Common Errors :

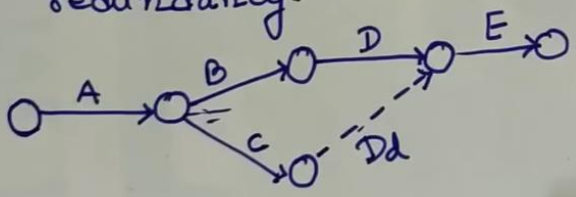
a) Looping (cycling) : Drawing an endless loop in a network is known as error of looping.

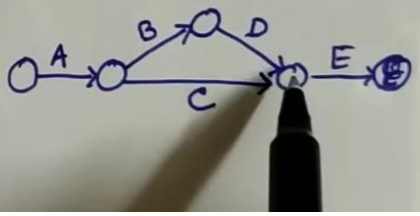


b) Dangling : To disconnect an activity before the completion of all activities, called error of dangling



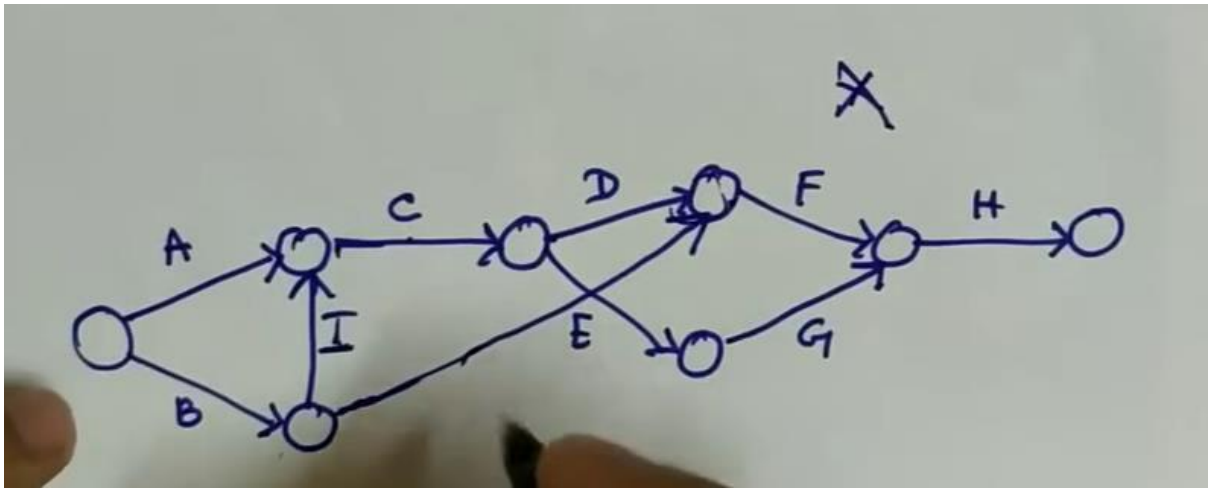
c) Redundancy : If a dummy activity is the only activity emanating from event and can be eliminated called redundancy.

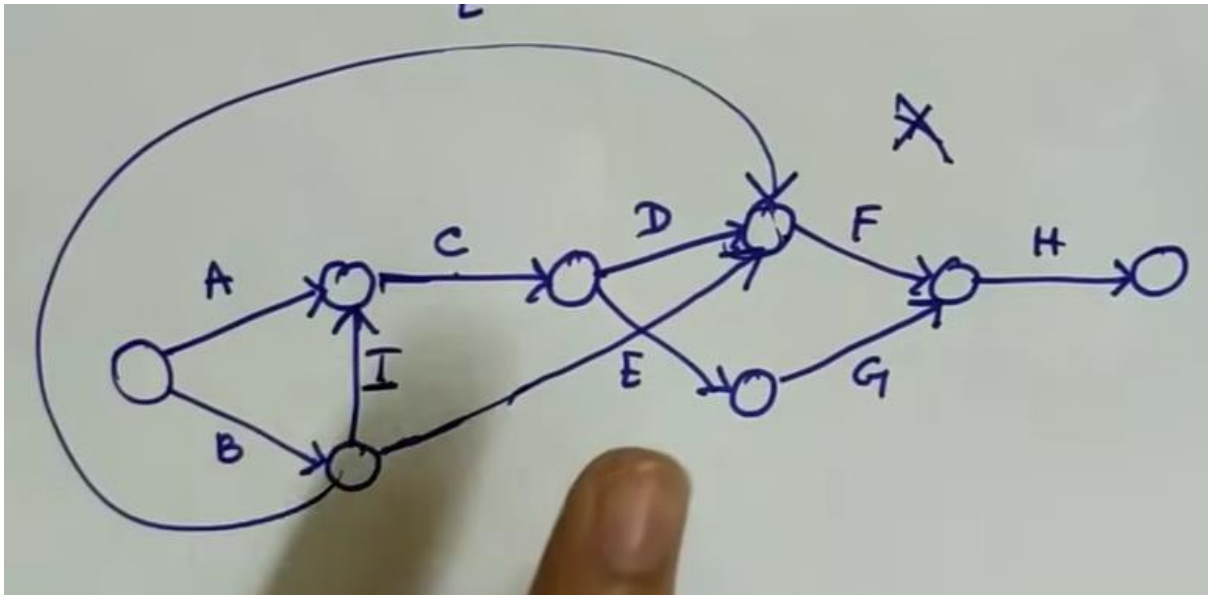


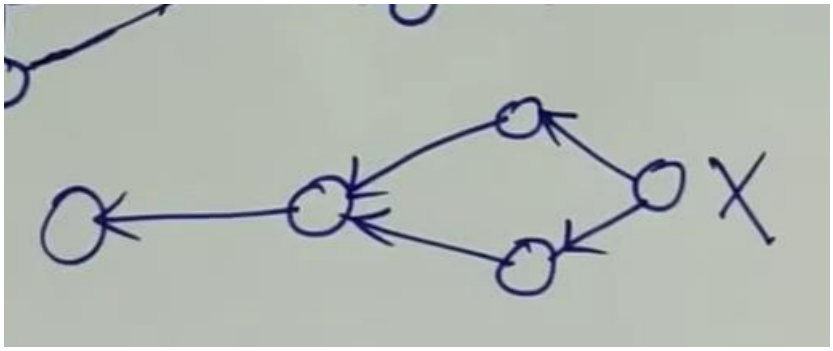


5) Rules of Network construction :

- Try to avoid the arrow that cross each other
- Use straight arrow
- No event can occur until every activity preceeding it has been completed.
- An event can't occur twice.
- Dummies should be introduced only, if it is extremely neccessary.
- Network has only one entry point called start event and one paint of emergence called end event.
- Use arrow left to right. Avoid mixing two directions.







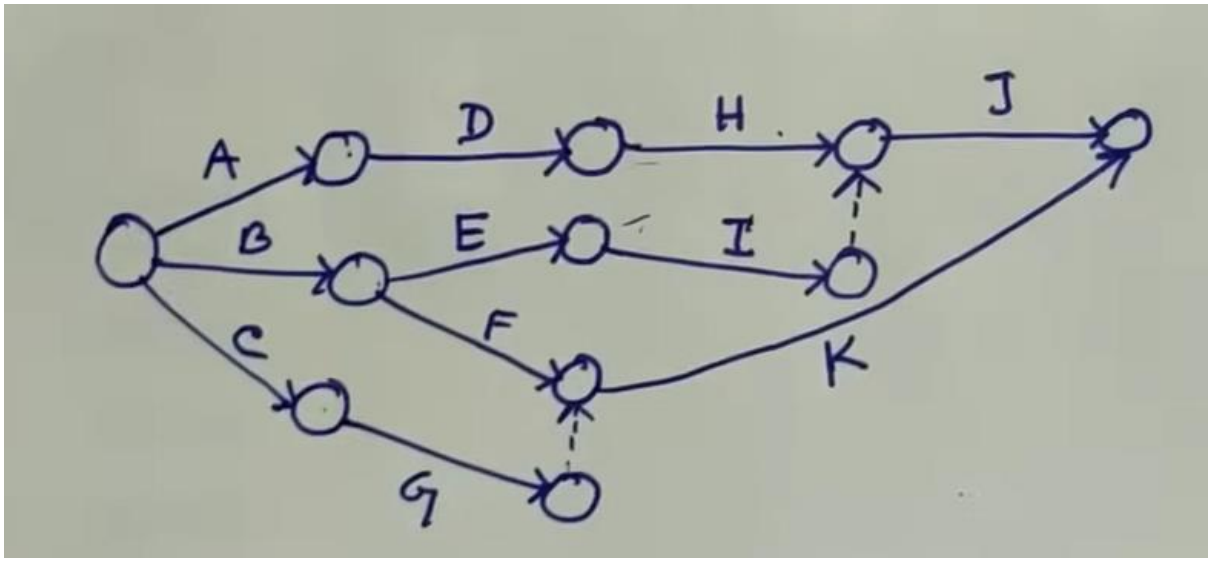
7) Numbering the Events :

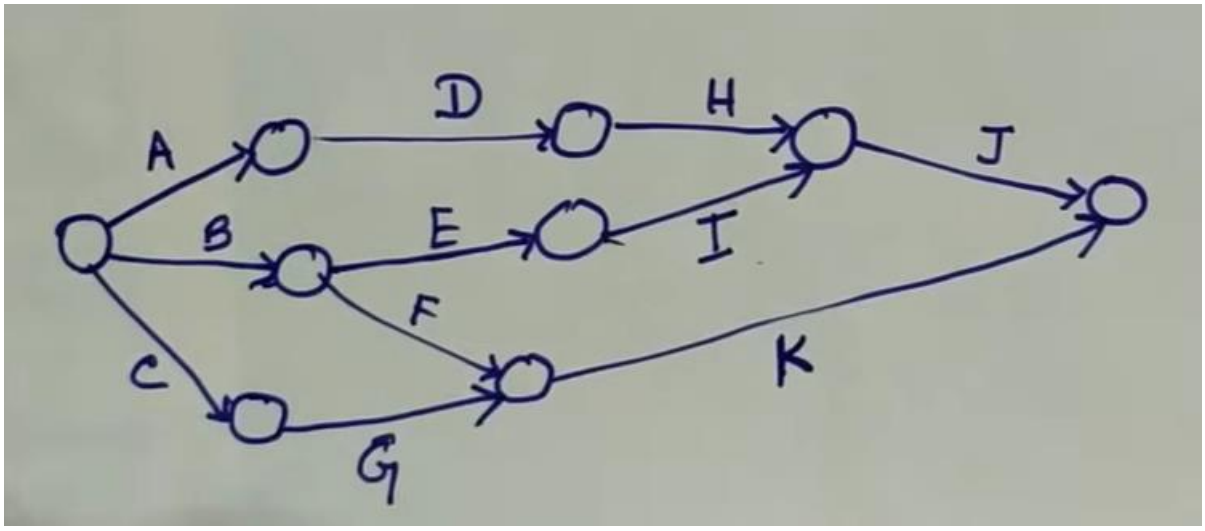
- Number must be unique
- Number should be carried out on a sequential basis, from left to right
- Intial event must be numbered as 1.
- Number all new start events 2, 3 and so on.

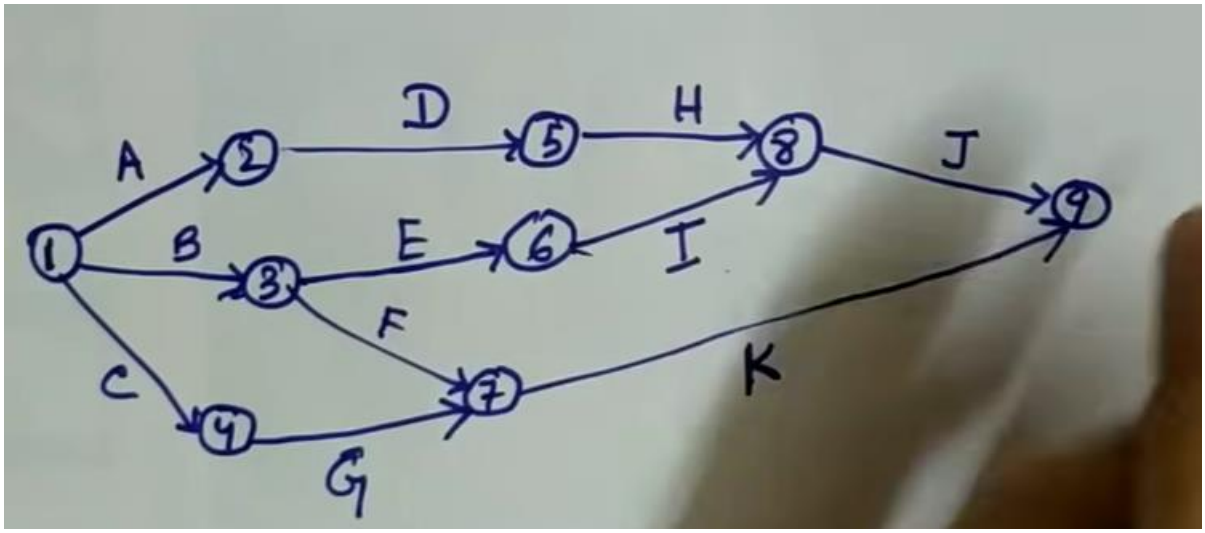
Contruction of Network

Q: Contruct a nework for the project whose activities and precedence relationships are as given below:

Activity	A	B	C	D	E	F	G	H	I	J	K
Predecessor	-	-	-	A	B	B	C	D	E	H, I	F, G







Network Scheduling.

Time Analysis :

Representations :

T_{ij} = Estimate completion time of activity (i,j).

ES_{ij} = Earliest starting time of activity (i,j)

EF_{ij} = Earliest finishing time of activity (i,j)

LS_{ij} = Latest starting of activity (i,j).

LF_{ij} = Latest finishing time of activity (i,j)

a) Forward Pass Computation :

- Zero be the starting time for the project.
- $(EF)_{ij} = (ES)_{ij} + t_{ij}$
- $E_j = \max_i (E_j + t_{ij})$

b) Backward Pass computation :

- For ending event assume, $E = L$
- $(LF_{ij}) = (LF)_{ij} - t_{ij}$

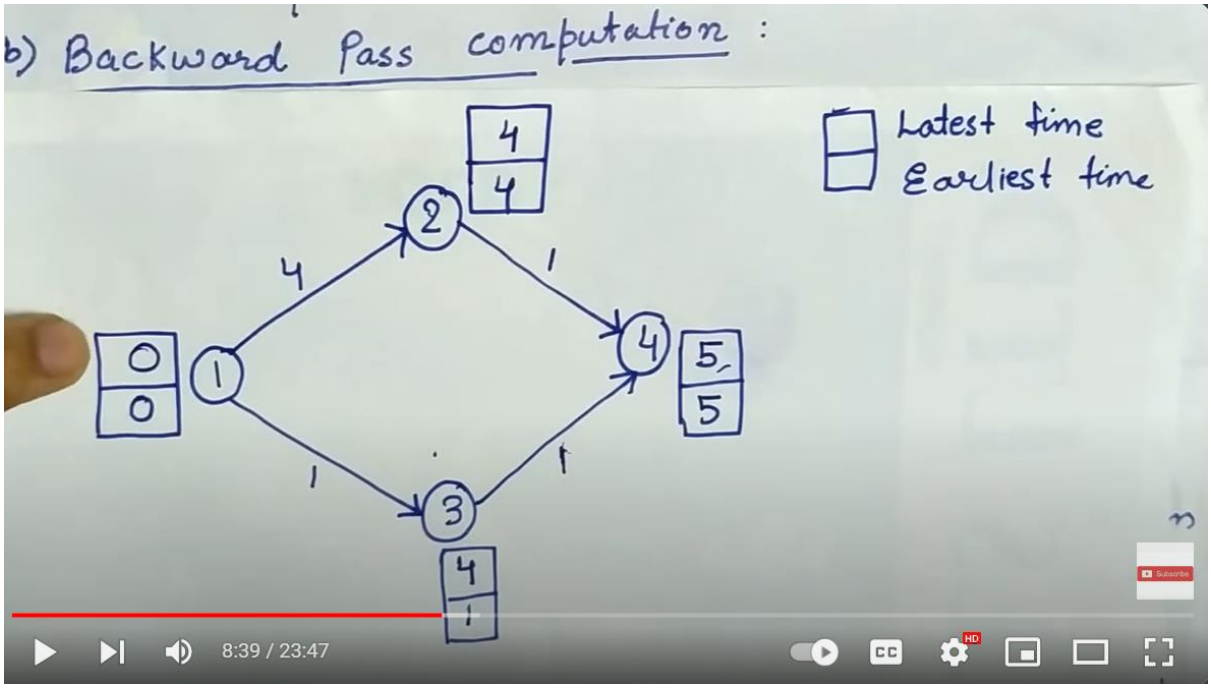
$= L_j - t_{ij}$

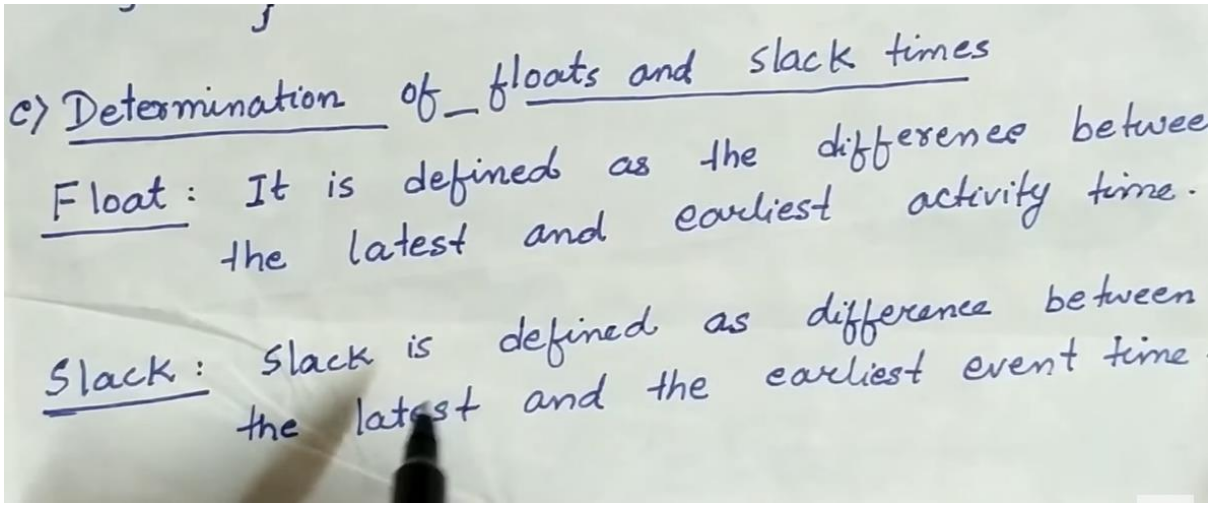
b) Backward Pass computation :

- For ending event assume, $E = L$
- $(LF_{ij}) = (LF)_{ij} - t_{ij}$

$= L_j - t_{ij}$

- $L_j = \min_j (L_j - t_{ij})$


b) Backward Pass computation :
Latest time
Earliest time
4
4
2
4
1
0
0
1
5
5
1
1
3
4
1
8:39 / 23:47


c) Determination of floats and slack times
Float : It is defined as the difference betwee
the latest and earliest activity time.
Slack : Slack is defined as difference between
the latest and the earliest event time

Total float :

$$(TF)_{ij} = (LS)_{ij} - (ES)_{ij}$$

or, $$(TF)_{ij} = (L_j - E_i) - t_{ij}$$

where, E_i = Earliest time for tail event
L_j = Latest time for head event
t_{ij} = Normal time for activity (i,j)

Activity	Normal time	Earliest		Latest	
		Start (ES)	Finishing	Start	Finish
	t_{ij}	ES (E_i)	EF = $E_i + t_{ij}$	LS = $L_j - t_{ij}$	LF (L_j)
1-2	4	0	4	0	4
1-3	1	0	1	3	4
2-4	1	4	5	4	5
3-4	1	1	2	4	5

Total Float (TF)	Free float (FF)	Independent float (IF)	E_j	L_i
LS - ES	$E_j - E_i - t_{ij}$	$E_j - L_i - t_{ij}$		
0	0	0	4	0
3	0	0	1	0
0	0	0	5	4
3	3	0	5	4

Free float:

$FF_{ij} = (E_j - E_i) - t_{ij}$

FF_{ij} = Total float − head event slack.

Head event slack $= L_j - E_j$

Independent float:

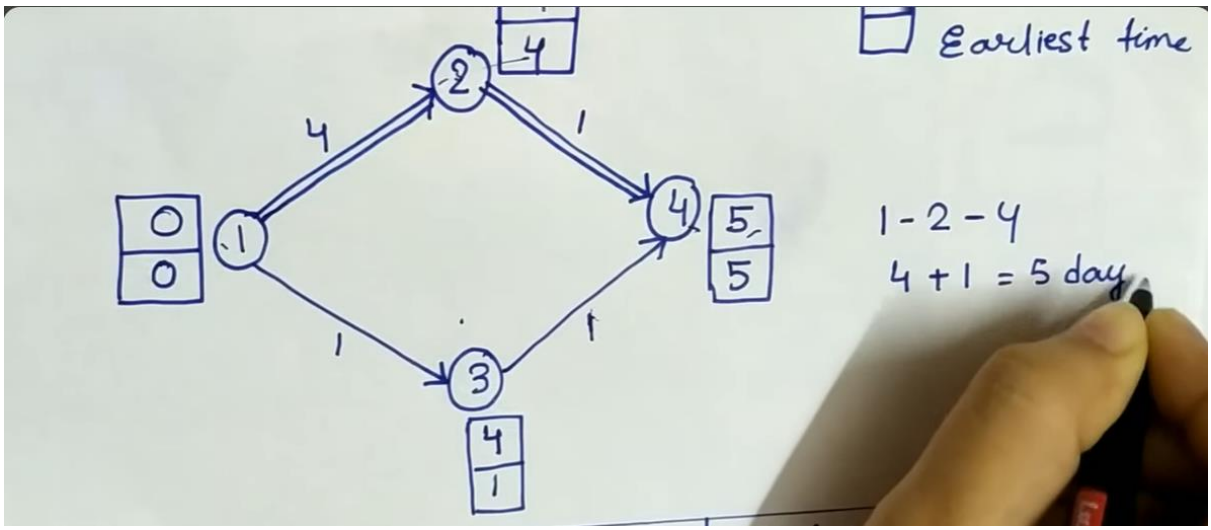
$IF_{ij} = (E_j - L_i) - t_{ij}$

$(IF)_{ij}$ = Free float − Tail event slack

Tail event slack $= L_i - E_i$

Critical Activity: An activity is said to be critical, if the total float TF_{ij} for any activity (i, j) is zero.

Critical path is shown in network diagram with double lines



Critical path conditions :

i) $ES_i = LF_i$

ii) $ES_j = LF_j$

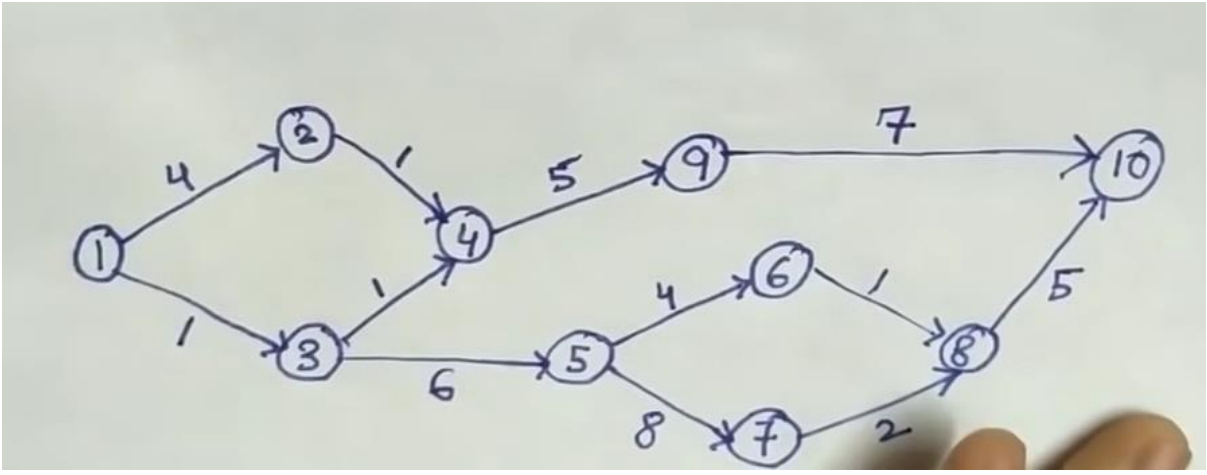
iii) $ES_j - ES_i = LF_j - LF_i = t_{ij}$

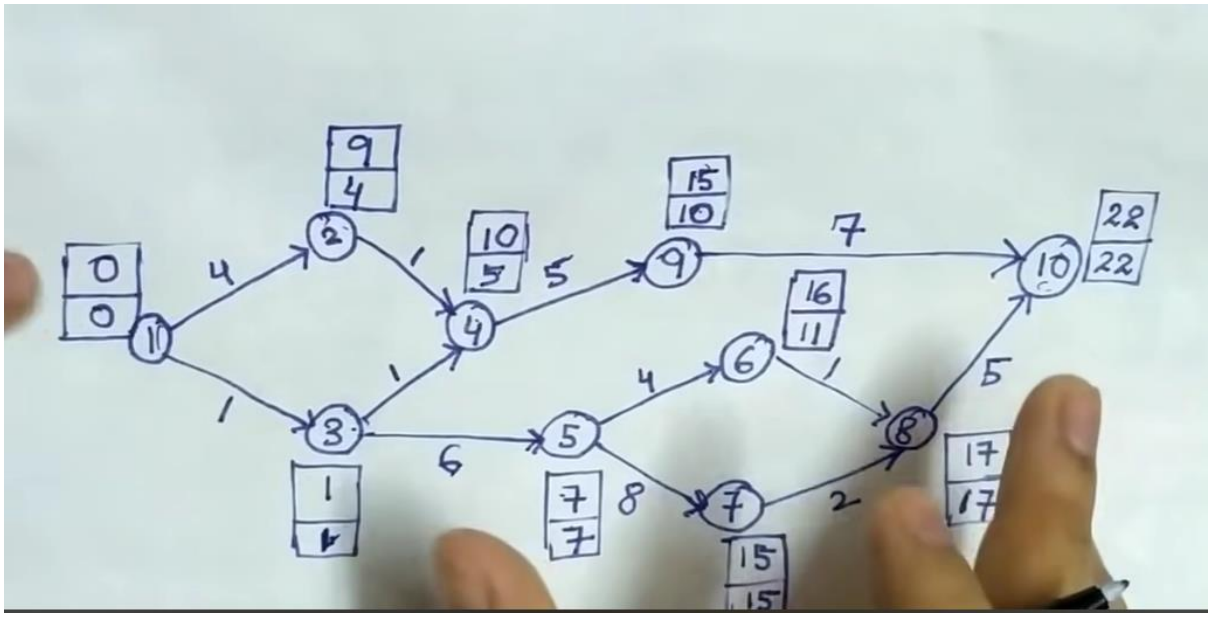
Critical Path Method (CPM)

Q: A project schedule has the following characteristics

Activity	1-2	1-3	2-4	3-4	3-5	4-9	5-6	5-7	6-8	7-8	8-10	9-10
Time (days)	4	1	1	1	6	5	4	8	1	2	5	7

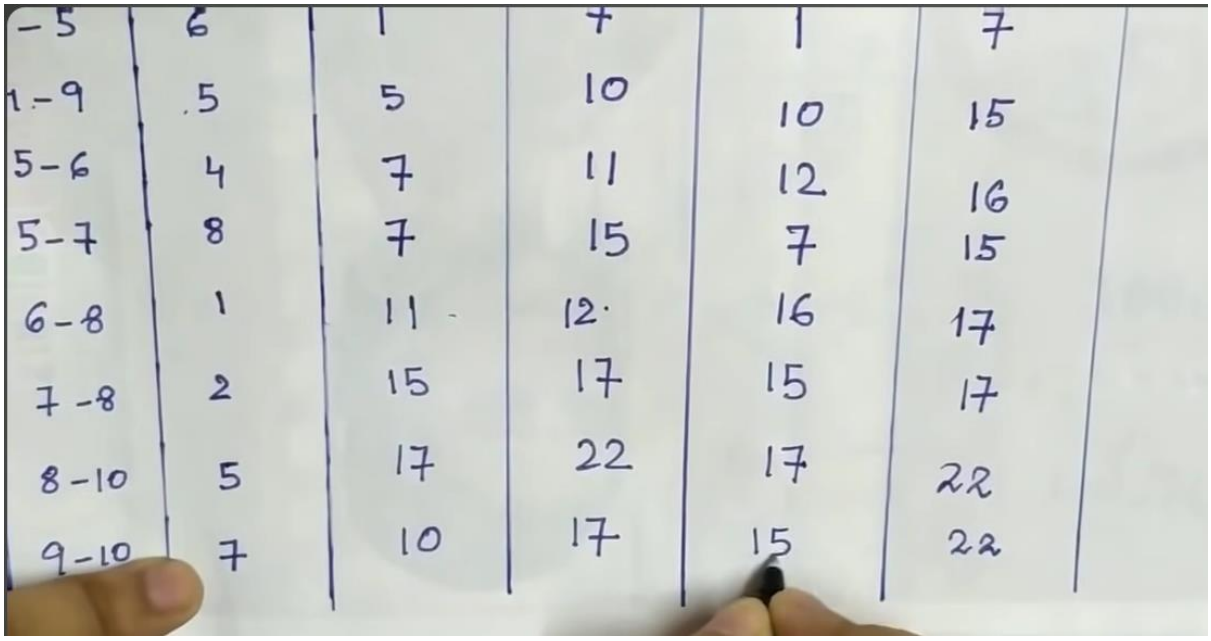
1. Construct Network diagram.
2. Compute the earliest event time and Latest event time
3. Determine the critical path and total project duration.
4. Compute total and free float for each activity.



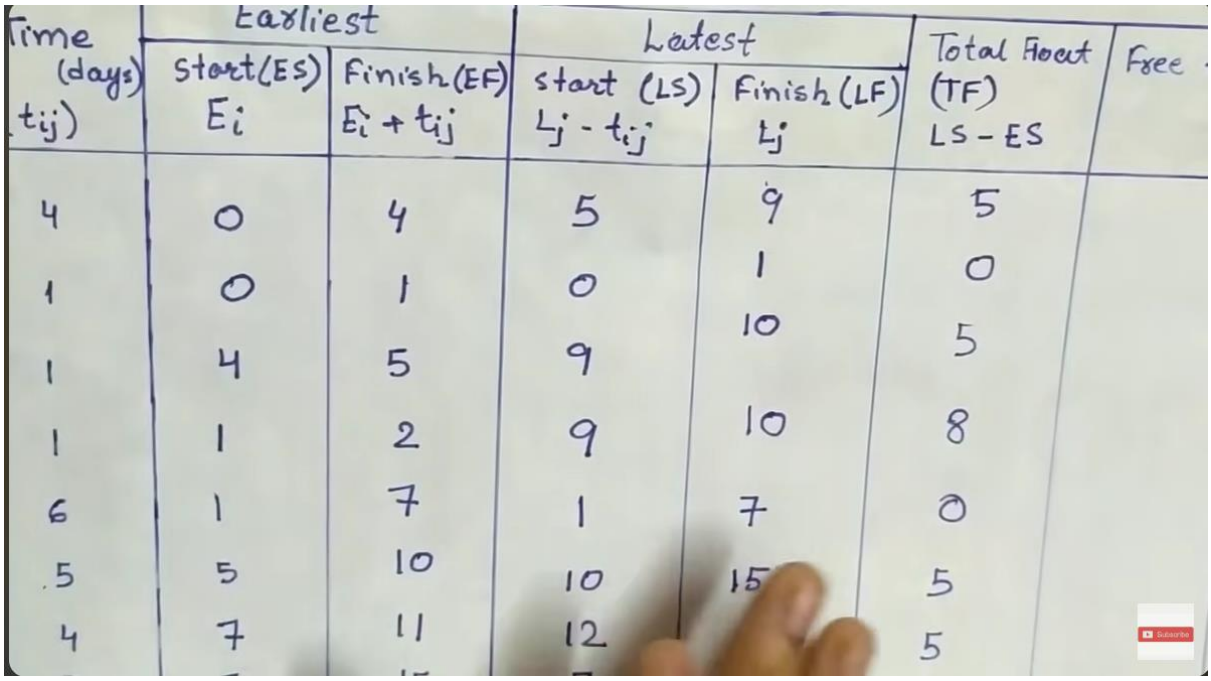


ollowing tables gives the information.

Activity	Time (days) (t_{ij})	Earliest		Latest		Total Float
		Start (ES) E_i	Finish (EF) $E_i + t_{ij}$	Start (LS) $L_j - t_{ij}$	Finish (LF) L_j	
1 - 2	4	0	4	5	9	
1 - 3	1	0	1	0	1	
2 - 4	1	4	5	9	10	
3 - 4	1	1	2	9	10	
3 - 5	6	1	7	1	7	
4 - 9	5	5	10	10	15	

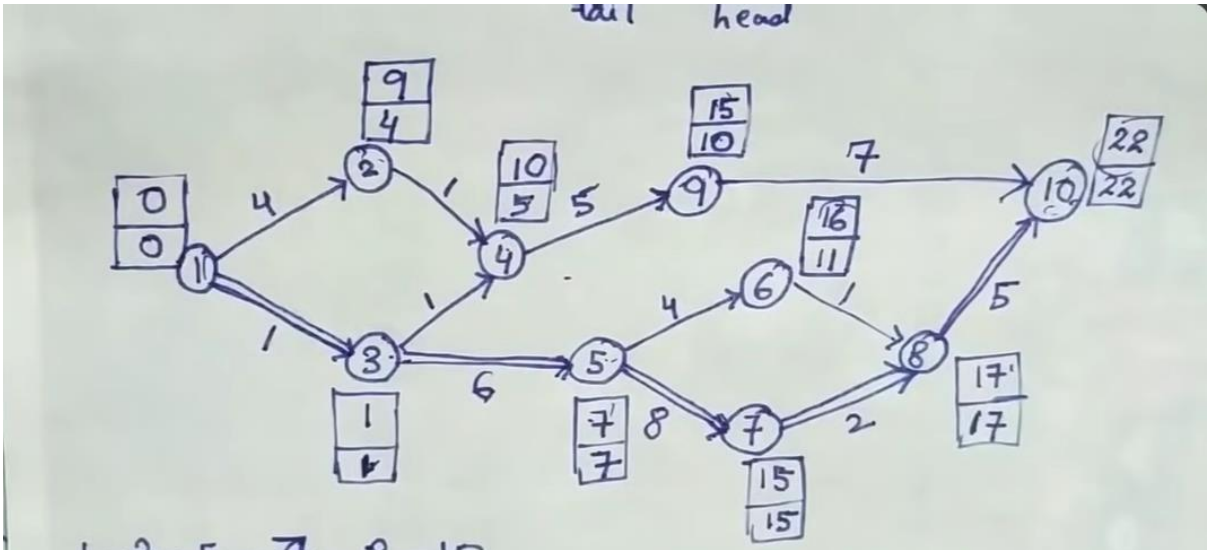


1-5	6	1	7	1	7
1-9	5	5	10	10	15
5-6	4	7	11	12	16
5-7	8	7	15	7	15
6-8	1	11	12	16	17
7-8	2	15	17	15	17
8-10	5	17	22	17	22
9-10	7	10	17	15	22



Time (days) t_{ij}	Earliest		Latest		Total Float (TF) LS - ES	Free
	Start (ES) E_i	Finish (EF) $E_i + t_{ij}$	start (LS) $L_j - t_{ij}$	Finish (LF) L_j		
4	0	4	5	9	5	
1	0	1	0	1	0	
1	4	5	9	10	5	
1	1	2	9	10	8	
6	1	7	1	7	0	
5	5	10	10	15	5	
4	7	11	12		5	

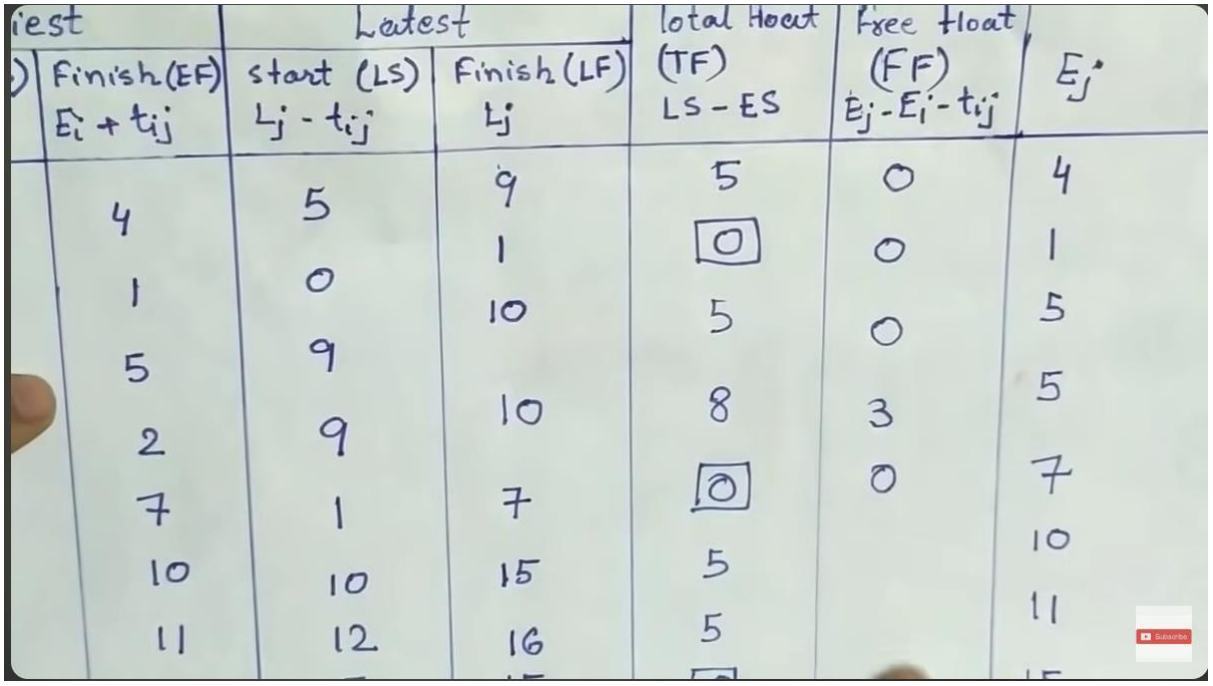
	1	2	9	10	8
	1	7	1	7	0
5	5	10	10	15	5
4	7	11	12	16	5
8	7	15	7	15	0
1	11	12	16	17	5
2	15	17	15	17	0
5	17	22	17	22	0
7	10	17	15	22	5



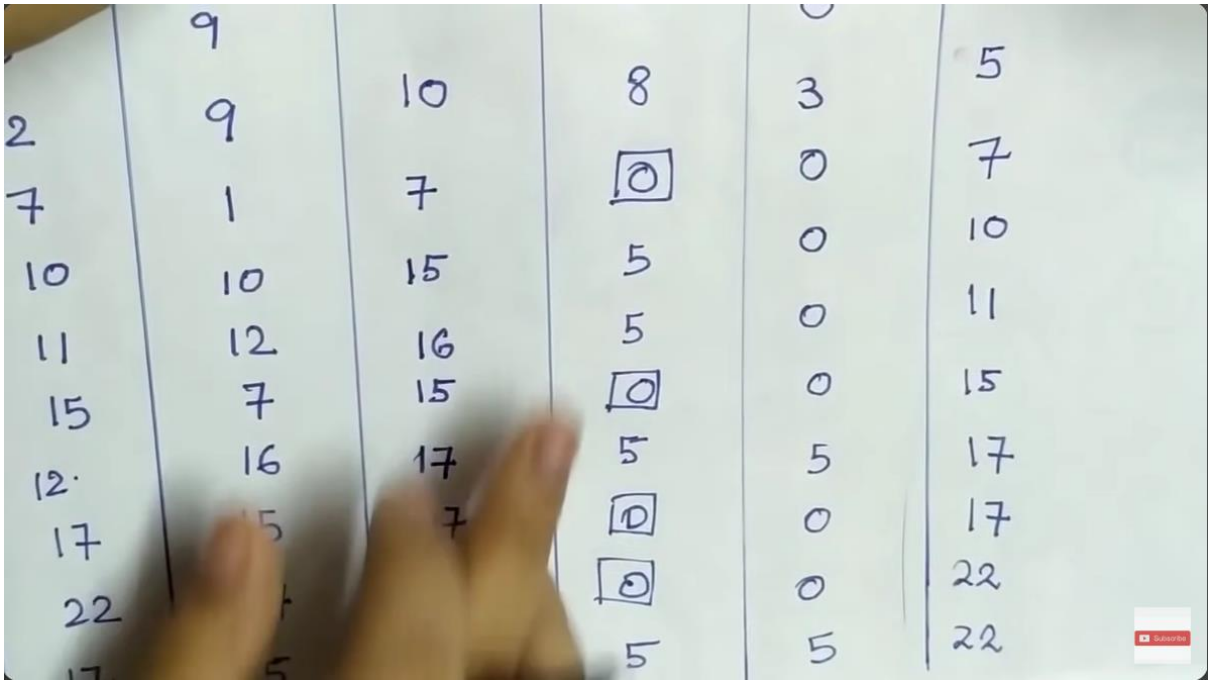
1-3-5-7-8-10

1 + 6 + 8 + 2 + 5 = 22

Total project duration = 22 days.



iest		Latest		Total Float (TF) LS − ES	Free float (FF) $E_j - E_i - t_{ij}$	E_j
)	Finish (EF) $E_i + t_{ij}$	start (LS) $L_j - t_{ij}$	Finish (LF) L_j			
	4	5	9	5	0	4
	1	0	1	0	0	1
	5	9	10	5	0	5
	2	9	10	8	3	5
	7	1	7	0	0	7
	10	10	15	5		10
	11	12	16	5		11



	9		8	3	5	
2	9	10	0	0	7	
7	1	7	5	0	10	
10	10	15	5	0	11	
11	12	16	0	0	15	
15	7	15	5	5	17	
12	16	17	0	0	17	
17	5	7	0	0	22	
22			5	5	22	

Activity	Time (days) (t_{ij})	Earliest		Latest		Total Float (TF)	Free Float (FF)	E_j
		start (ES) E_i	Finish (EF) $E_i + t_{ij}$	start (LS) $L_j - t_{ij}$	Finish (LF) L_j	LS − ES	$E_j - E_i - t_{ij}$	
1 - 2	4	0	4	5	9	5	0	4
(1 - 3)	1	0	1	0	1	[0]	0	1
2 - 4	1	4	5	9	10	5	0	5
3 - 4	1	1	2	9	10	8	3	5
(3 - 5)	6	1	7	1	7	[0]	0	7
4 - 9	5	5	10	10	15	5	0	10
5 - 6	4	7	11	12	16	5	0	11
(5 - 7)	8	7	15	7	15	[0]	0	15
6 - 8	1	11	12	16	17	5	5	17
(7 - 8)	2	15	17	15	17	[0]	0	17
(8 - 10)	5	17	22	17	22	[0]	0	22
		10	17	15	22	5	5	22

Activity	Time (days) (t_{ij})	Earliest		Latest		Total Float (TF)	Free Float (FF)	E_j
		start (ES) E_i	Finish (EF) $E_i + t_{ij}$	start (LS) $L_j - t_{ij}$	Finish (LF) L_j	LS − ES	$E_j - E_i - t_{ij}$	
1 - 2	4	0	4	5	9	5	0	4
(1 - 3)	1	0	1	0	1	[0]	0	1
2 - 4	1	4	5	9	10	5	0	5
3 - 4	1	1	2	9	10	8	3	5
(3 - 5)	6	1	7	1	7	[0]	0	7
4 - 9	5	5	10	10	15	5	0	10
5 - 6	4	7	11	12	16	5	0	11
(5 - 7)	8	7	15	7	15	[0]	0	15
6 - 8	1	11	12	16	17	5	5	17
(7 - 8)	2	15	17	15	17	[0]	0	17
(8 - 10)	5	17	22	17	22	[0]	0	22
		10	17	15	22	5	5	22

Programme Evaluation And Review Technique (PERT)

Q: The following table shows the jobs of a network along with their time estimates.

Activity	Estimated duration (weeks)		
	optimistic (t_o)	Most likely (t_m)	Pessimistic (t_p)
1 - 2	1	7	13
1 - 6	2	5	14
2 - 3	2	14	26
2 - 4	2	5	8
3 - 5	7	10	19
4 - 5	5		17

Activity	Estimated duration (weeks)		
	optimistic (t_o)	Most likely (t_m)	Pessimistic (t_p)
1 - 2	1	7	13
1 - 6	2	5	14
2 - 3	2	14	26
2 - 4	2	5	8
3 - 5	7	10	19
4 - 5	5	5	17
6 - 7	5	8	29
5 - 8	3	3	9
7 - 8	8	17	32

You are required to:

1) Draw the project network
2) Find the expected duration and variance of each activity.
3) Calculate the earliest and Latest occurance for each event
4) Calculate expected project length.
5) Calculate the variance and standard deviations of project length.
6) Find the probability of the project completing in 40 days.

PERT is a probabilistic method, where the activity times are represented by a probability distribution. This distribution of activity times is based on three different time estimates made for each activity, which are as follows :

(i) Optimistic time estimate.
(ii) Most likely time estimate.
(iii) Pessimistic time estimate.

For these three estimate, we have to calculate the expected time of an activity.

$$t_e = \frac{t_o + 4t_m + t_p}{6}$$

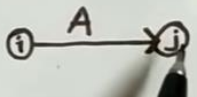
(i) Optimistic time estimate. (t_o or a)
(ii) Most likely time estimate. (t_m or m)
(iii) Pessimistic time estimate. (t_p or b)

For these three estimate, we have to calculate the expected time of an activity.

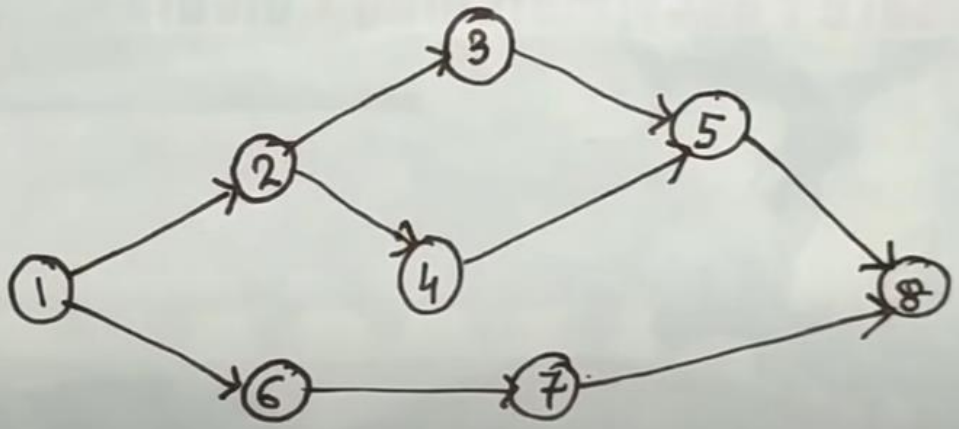
$$t_e = \frac{t_o + 4t_m + t_p}{6}$$

Variance of an activity is given by,

$$\sigma^2 = \left(\frac{t_p - t_o}{6}\right)^2$$



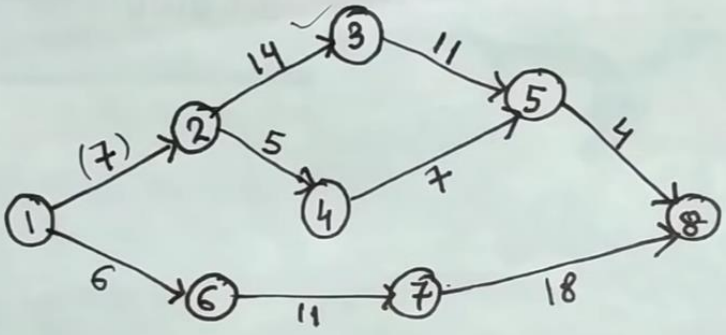
Network diagram,



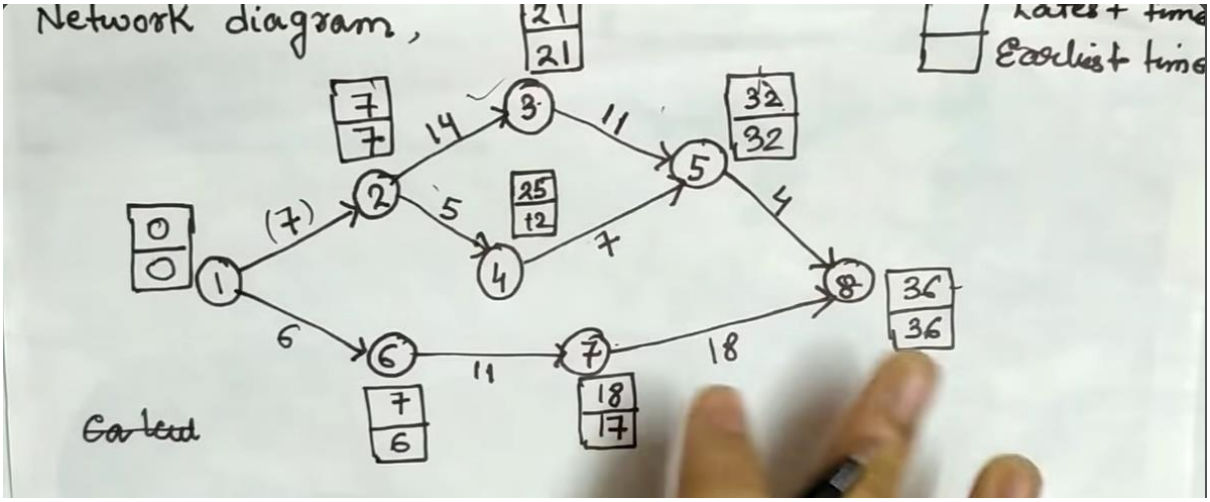
Activity	Estimated duration			$t_e = \frac{t_o + 4t_m + t_p}{6}$	$\sigma^2 = \left(\frac{t_p - t_o}{6}\right)^2$
	t_o	t_m	t_p		
1 - 2	1	7	13	$\frac{1+4\times7+13}{6} = 7$	$\left(\frac{13-1}{6}\right)^2 = 4$
1 - 6	2	5	14	$\frac{2+4\times5+14}{6} = 6$	$\left(\frac{14-2}{6}\right)^2 = 4$
2 - 3	2	14	26	$\frac{2+4\times14+26}{6} = 14$	$\left(\frac{26-2}{6}\right)^2 = 16$
2 - 4	2	5	8	5	1
3 - 5	7	10	19	11	4
4 - 5	5	5	17	7	4
6 - 7	5	8	29	11	16
5 - 8	3	3	9	4	1
7 - 8	8	17	32	18	16

Network diagram,

Latest t
Earliest t



Network diagram,



	t_o	t_m	t_p	$\frac{t_p - t_o}{6}$	$\left(\frac{t_p - t_o}{6}\right)^2$
(1-2)	1	7	13	$\frac{1+4\times7+13}{6} = 7$	$\left(\frac{13-1}{6}\right)^2 = 4$
1-6	2	5	14	$\frac{2+4\times5+14}{6} = 6$	$\left(\frac{14-2}{6}\right)^2 = 4$
(2-3)	2	14	26	$\frac{2+4\times14+26}{6} = 14$	$\left(\frac{26-2}{6}\right)^2 = 16$
2-4	2	5	8	5	1
(3-5)	7	10	19	11	4
4-5	5	5	17	7	4
6-7	5	8	29	11	16
(5-8)	3	3	9	4	1
7-8	8	17	32	18	16

Expected project duration = 7+14+11+4 = 36 weeks

7-8	8	17	32	18	

Expected project duration = 7+14+11+4 = 36 weeks

Critical path 1-2-3-5-8

project length variance, $\sigma^2 = 4+16+4+1 = 25$

project length standard deviation, $\sigma = 5$

Calculate the standard normal variable

$Z = \frac{T_s - T_e}{\sigma}$, where T_s is the schedule time to complete the project.

T_e = Normal expected project length

σ = Expected standard deviation of the project length

project length standard

The probability that the project will be completed in 40 days is given by $P(z \leq D)$

$$D = \frac{T_s - T_e}{\sigma} = \frac{40 - 36}{5} = 0.8$$

$P(Z \leq 0.8) = 0.7881$ this is calculated for normal deviation table

PERT	CPM
i) Stands for programme evaluation and review technique	i) Stands for critical path method.
ii) Event Oriented	ii) Activity Oriented
iii) Associated with probabilistic activity	iii) Associated with deterministic activity.

iv) Based on three time estimate namely • Optimistic • Pessmistic • Most likely	iv) Based on single time to complete
esource such as Labour, equiptments, material etc are limited	v) No limitation of resources.
vi) Mainly used for research & development project.	vi) Mainly used for contruction project

• Pessmistic • Most likely	
v) Resource such as Labour, equiptments, material etc are limited	v) No limitation of resources.
vi) Mainly used for research & development project.	vi) Mainly used for contruction project
vii) It is a technique of planning and control of time.	vii) It is a method to control costs and time.

viii) PERT technique is suited for a high precision time estimate.	viii) CPM is appropriate for a reasonable time estimate

estimate	estimate.
ix) PERT is used where the nature of the job is non-repetitive.	x) CPM involves the job of repetitive nature.
x) Crashing concept is not applicable to PERT	xi) Crashing is a compression technique applied to CPM, to shorten the project duration, along with least additional cost.