Software components : A program is a subset of software and it becomes software only if proper documentation and operating procedure manual are prepared.

Documentation mainly consists descriptions, program instructions pertaining to design coding testing and preparation of software.

User Manual - Provides information about what is the software how to work with it how to install it and how to control all the activities of the software.

Types of software - (i) System software.
(ii) Application software.
(iii) Engineering/Scientific software. used to solve complex numerical problems.
(iv) Special purpose software
(v) Embedded Software - provide Limited feature and functionally. (Microwave oven)
(vi) Web/Mobile Apps. - soft^n that runs on a web or mobile browser.
(vii) Legacy Software. very old and traditional software.
(viii) Real time software. Monitor, control and analyse real word events (weather app) in real time.

Why we learn software Engineering.
In software Engineering we understand that there is an urgent need to use a proper strategy, process and development so that we can produce.

Software characteristic : (i) Correctness - the ability of soft^n to perform their intended task effectively and meet the user requirement.
(ii) Usability : The ease with user can learn and navigate the software.
(iii) Reliability : Consistency in producing accurate result and maintaining performance overtime.

(iv) Efficiency : the optimal use of system resources.

(v) Maintainability : The ease of modifying and fixing the software.

(vi) Portability : The ability of the software to operate in different ~~machines~~ platform.

(vii) Scalability : Software's capability

(viii) Security :

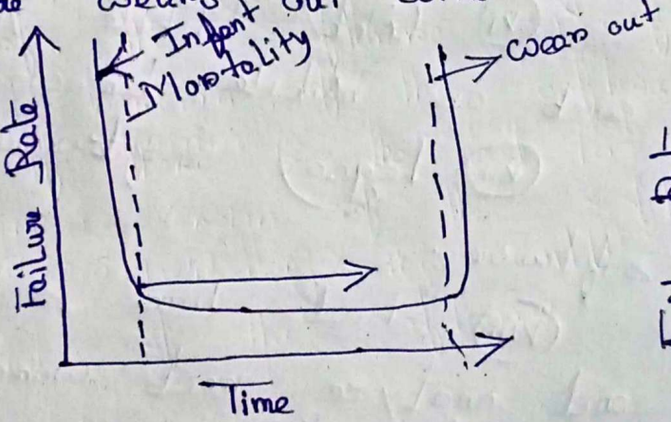
(ix) Modularity : The degree to which software components are organized into separate manageable unit.

(x) Reusability : The potential of the software components to be used in other applica" or content reducing development time and cost.

(xi) Testability : how easily the software ca be tested to ensure it meets the requirements.

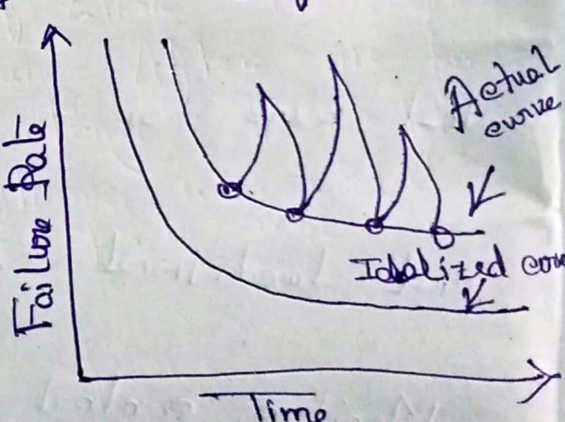
18/7/24

Hardware wears out with time.



Bathtub Curve

Software curve



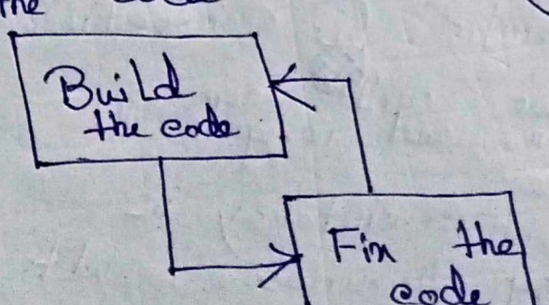
IF doesn't wear out it deteriorates.

Build and Fix Model

it is also called exploratory development cycle.

This is two-phase model.

(i) Writing the code (ii) Fixing it.



In build and fix model we are concentrating on error correction not prevention.

Drawbacks: Build and Fix Model prove to be insufficient for developing large and complex program.

(ii) It is very difficult to write cost-effective and correct programs using this method.

(iii) It is very difficult to understand and maintain the program written by the other.

Major Problems in Software Development

(i) Inadequate requirement gathering.
(ii) Lack of communication.
(iii) Poor Project Management.
(iv) Lack of Risk Management.
(v) Lack of Resource.
(vi) Selection of wrong technology and tools.
(vii) Insufficient time and budget.
(viii) Unrealistic Deadlines.
(ix) Inefficient Resource Allocation.
(x) Lack of skilled personality.
(xi) Persitent to change.
(xii)

Software Process or Methodology:

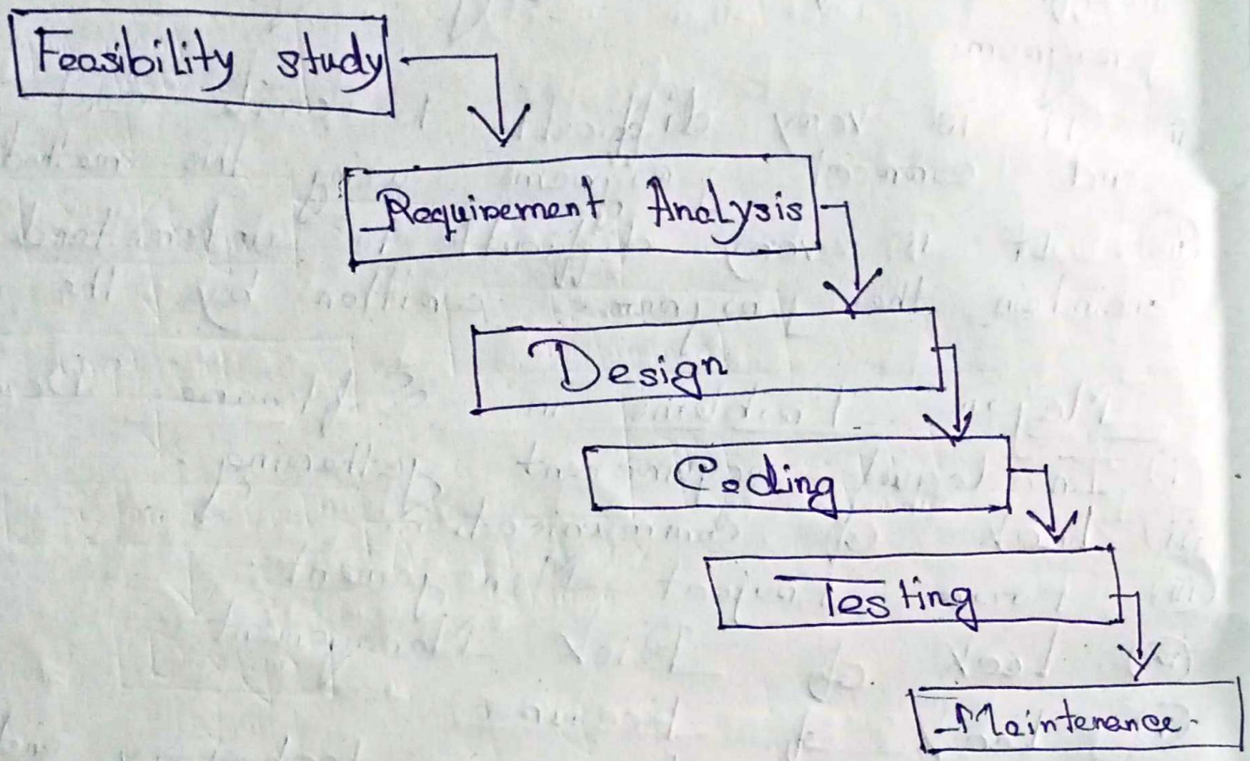
Softn development process is a set of related activities that leads to the production of software from scratch or modifying an existing system.

Software Lifecycle development SDLC.

i) Feasibility study
ii) Requirement Analysis
iii) Designing
iv) Coding
v) Testing
vi) Implementation
vii) Maintenance.

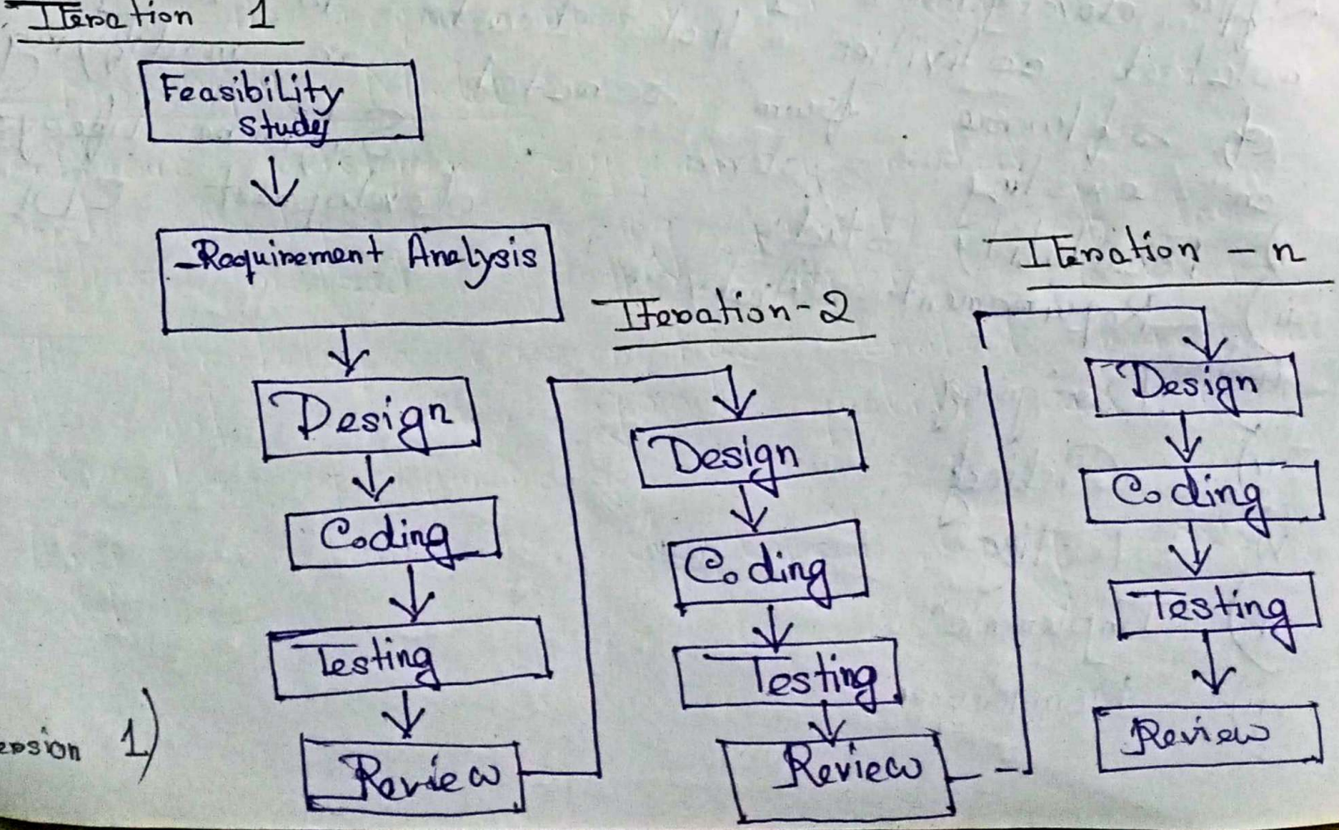
Waterfall Model / Classical Lifecycle Development Model.

W. Royce ⇒ 1970s.



Iterative Model:

In the iterative Model the organization start with some of the software specification and develop the first version of the software. After the first version if there is the need to change the software then a new version of the software is created with a new iteration. This will repeat until the deployment of the software.



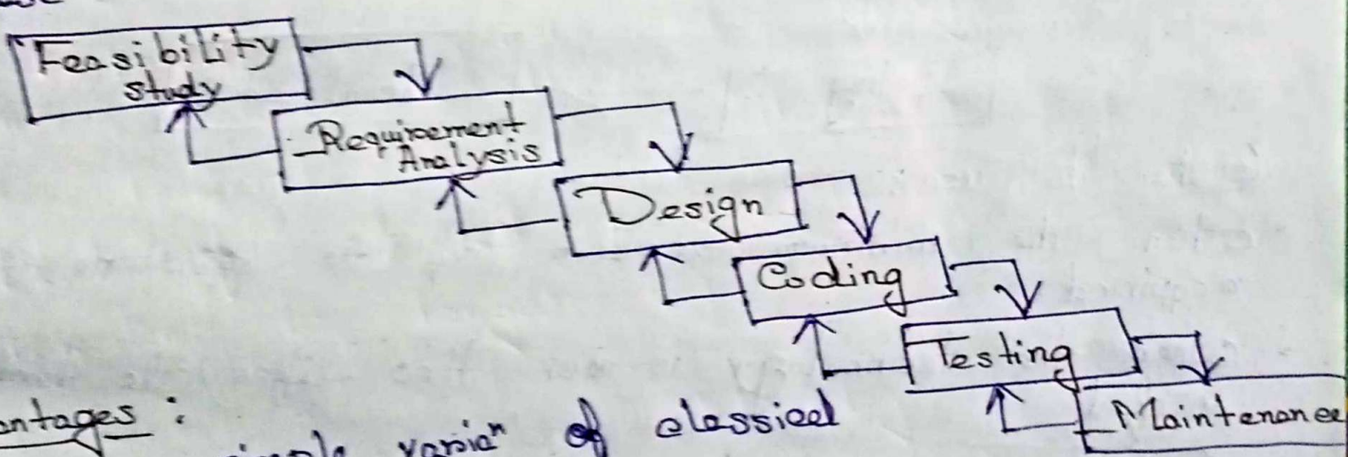
Iterative model is useful when the project ... Major requirements must be defined however, some details can evolve with time.

Advantage: generates working software quickly and early during the software lifecycle.
ii) Easier to test and design during a smaller iteration.
iii) Easier to manage risky pieces of code hence the development cost is ~~some~~ safe.

Disadvantage:
i) It is not suitable for changing requirement.
ii) It is not suitable for smaller project.
iii) More management attention is required.

Iterative Waterfall Model

The iterative waterfall model provides feedback paths from every phase to its preceding phase.

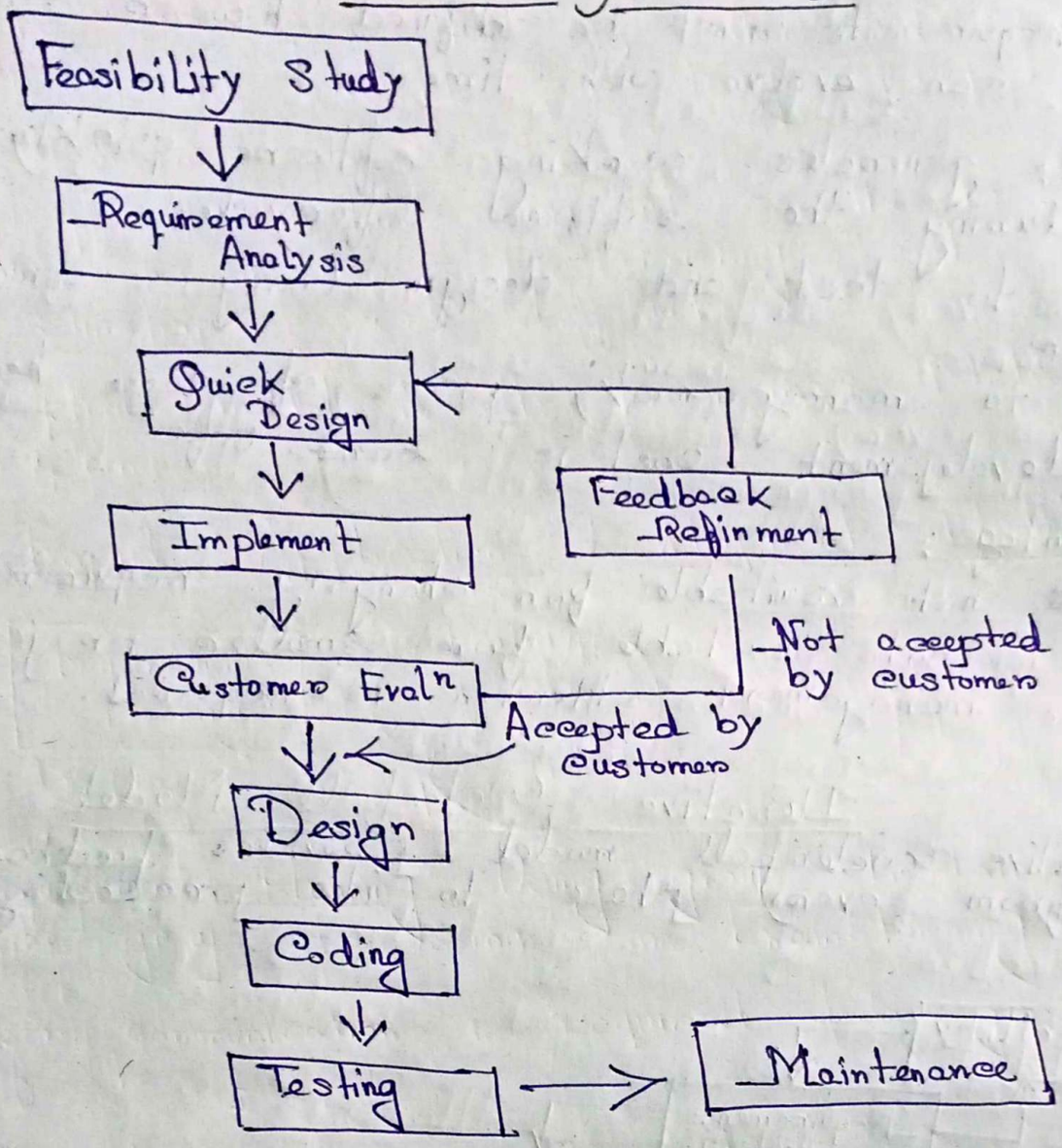


Advantages:
i) It is a simple variation of classical waterfall model.
ii) It includes feedback path so it is easier to manage the errors and additional requirements more easily.

Disadvantages:
i) Incremental delivery is not supported.
ii) Overlapping of phases is not supported.
iii) Risk handling is not supported.

H.W Diff bet^n iterative and evolutionary model.
Diff bet^n waterfall -

Evolutionary Model



When to use:

- when the minimum version of the system is not required.
- when the technology is new this lifecycle model is useful.
- Complex project.
- when the requirements is not clear.

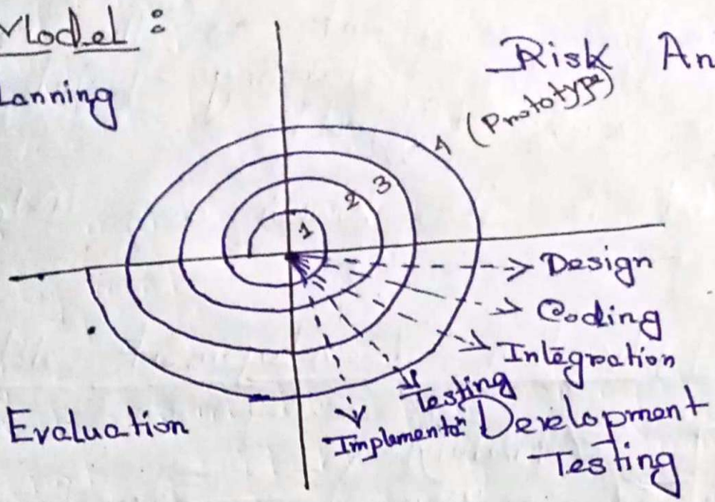
Prototyping Model:

Continuous customer support is needed.

- First develop a working prototype of the software instead of the complete product.
- The prototype version has limited functionality this not a very good quality and it has very Low performance.
- Final system is developed using waterfall model.

- help us to refine the customer requirements.
- help us to gather experience for developing the final system.

Spiral Model:



Planning: Requirements are gathered from the customers and the objectives are identified and these will be done at the start of every phase. Requirements like business requirement specification (BSR and SRS) are gathered in this quadrant. Then alternative solutions possible for the phase are proposed.

Risk Analysis: During the second quadrant all the possible solutions are evaluated to select the best possible soln. Then the risk associated with that solution is identified and the risks are resolved with the best possible strategy. And output of this quadrant is a prototype dealt with that best possible solution.

Development and Testing: During the third quadrant the identified features are developed and verified with testing.

Evaluation: This phase allows the customer to evaluate the output of the project before the project continues to the next spiral.

Customer evaluate the software and provide their feedback and approval.

Advantage: This is the best model for risk analysis and risk handling.

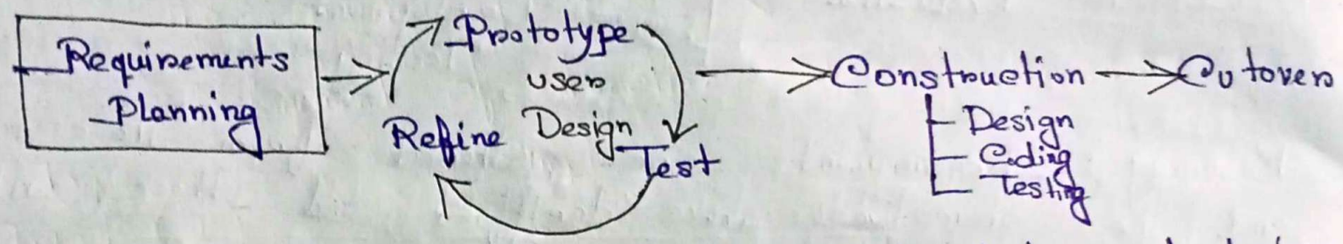
- It is good for large projects.
- Flexibility in requirements
- Customer satisfaction.
- Product is produced early.

Disadvantage:

- This is complex than any other SDLM.
- It is not suitable for large project as it is expensive.
- Difficulty in time manggement. as the no. of phases in unknown at the starting point so time estimation is very difficult.

Spiral model is also known as Meta Model. Explain.

Rapid Application Development Model (RAD):



Requirement Planning: Using group elicitation techniques like group discussion or brainstorming.

User communication is required for good understanding of the project.

User design: A joint team of customers and developer understands and reviews the gathered requirement.

Automated tools may be used.

Construction: Under this construction phase design Actual product is released in this phase and it will be validated with test cases.

Automated tools are used in the coding phase as code generator and screen generator.

Cutover: Install the product in the customer side

- acceptance testing is done by the customer.
- Training of the user.

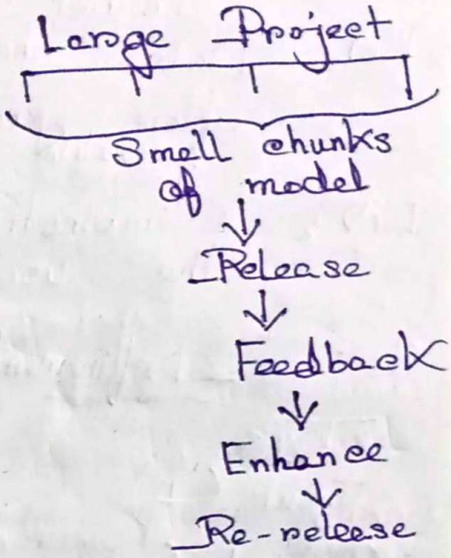
Advantage:

i) Rapid prototype
ii) Quick initial views about the product is gathered.
iii) Powerful development tools. so development time is reduced due to use of automated tools.

iv) User Development: User involvement is easy acceptability by the user.

Disadvantage:
i) Continuous involvement of the customer is not needed every time.
ii) High skilled and specialized developers are needed.

AGILE MODEL:



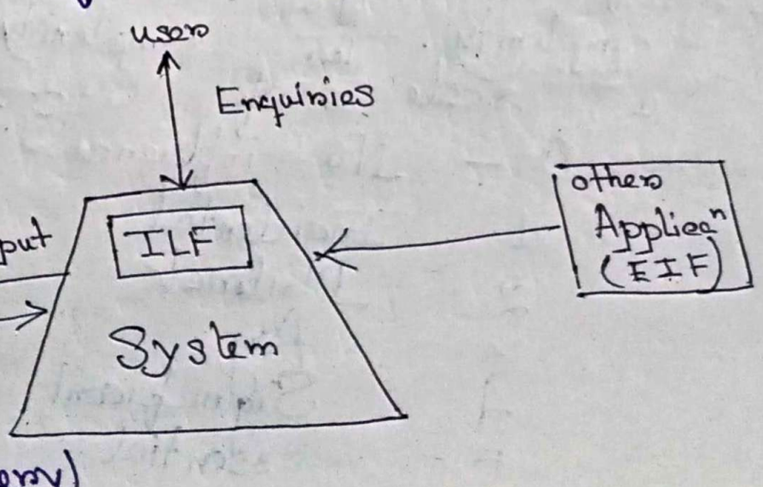
29/7/21

Line of Code - Line of code includes declaration, actual code including logic and computation. Blank Lines and comments are not included in LOC.

Advantage - very easy to count and calculate from the developer's code.

Disadvantage: i) The major of LOC is platform or language dependent to another.

Function Points: It measures functionality from user's point of view. That means what the user receives from the software and request from the software. In simple terms it focuses what is the functionality delivered by the system.



1) EIF (External Interface File)
2) Internal (ILF) logical File
3) EI (External input)
4) EO (External output)
5) EQ (External Query)

ILF: It controls information or logically related data that is present within the system.

EIF: It controls data or other logical information that is required by the system but present in another system. Eg:- Payment gateway interfaces

EI: Data or control information that comes outside from the system. Eg- User registration Product search form, add to cart Form.

EO: Data that goes out of the system after generation. Eg- Confirmation email, invoice generation.

EQ: Combination of input and output resolving the user problems. Eg- Product Availability check

Refinement of Function Point Entities

Type	Simple	Avg	Complex
Input (I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
No of Files (F)	7	10	15
No. of Interfaces	5	7	10

Step 1: Each function point is rank according to their complexities such as low avg high

Step 2: Calculate unadjusted function point or (UFP) by multiplying each function point by its corresponding weight factor.

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} (Z_{ij} * \omega_{ij})$$

Z_{ij} = No. of matrices

Step 3: Calculate the final function Point. Calculated using the complexity adjustment factor using 14 aspects of processing complexity. The complexity is analysed on a scale of 0-5.

0 - No influence.
1 - Incidental.
2 - Moderate.
3 - Avg.
4 - Significant
5 - Essential.

$$[0.65 + 0.01 \times \Sigma F_i]$$

Max value of F_i be 14.

Given the following values computes F varies from 0 - 5 when all complexity adjustment factors and weiting factors are avg.

the function point

user i/p = 50
user o/p = 40
user enquiries = 35
user files = 6
External interface = 4

Avg user i/p = (50 × 4) = 200
o/p = (40 × 5) = 200
enquiries = (35 × 4) = 140
files = (6 × 10) = 60
interface = (4 × 7) = 28
UFP = 628

Value adjustment Factor

VAF = $(0.65 + 0.01 \times (3) \times 14) = 1.07$

(3) ← 0-14

FP = UFP * VAF
= 628 * 1.07
= 671.96

Advantage: Size of the software deliver is measured independent of language and technology.

- Func^n Point can be directly calculated from the requirement analysis case before designing and coding.

COCOMO: Constructive Cost Model.

developed by Barry Bohem. It is a hierarchy of software cost estimation Model. Basic intermediate detail or complete.

Basic COCOMO Model:

It estimate a software in a rough and quick manner. Mostly useful for small to medium project.

Three modes of development are there.

organic
Semi Detached
Embedded

	Organic	Semi Detached	Embedded
size	2-50 KLOC	50-300 KLOC	300 - above KLOC
Team size	Small	Medium	large
Developer Experience	Experienced Developer Needed	~~less familiar~~ Avg experience	Very Little previous experience.
Environment	Familiers	Less familiers	Significant environment changes.
Innovation	Little	Medium	Major
Deadline	Not tight	Moderate	Tight
Example	Pay Roll, Library Management, online shopping etc	Utility Management Software.	Air Traffic Control.

Basic Model Evaluation :

1. Effort = $a(KLOC)^b$ person months.
2. Development time = $c(Effort)^d$ months
3. Avg staff size = $\frac{Effort}{Dev\ Time}$

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Suppose a project was estimated to have 400 KLOC. Calculate the effort, development time, avg staff size and productivity of the project using basic COCOMO Model.

Effort $= a(KLOC)^b$ person/month (PM)

$= 3.6 \times 400^{1.20}$

$= 4772.81$ person/month (PM)

Dev Time $T_{dev} = 2.5 \times (4772.81)^{0.32}$

$= 37.597$ months

Avg staff size $= \frac{4773.7}{38}$ — Productivity $= \frac{KLOC}{E}$

$= 125.$ person

COCOMO Intermediate Model:

Intermediate Model includes a set of 15 additional predictors (cost drivers)

- It also takes development environment into account during cost estimation.

Use of Cost Drivers:

Cost drivers adjust nominal cost of project to actual project environment.

Function Point UAF $(20 + 300 + 92 + 80 + 14) = 506$

$0.65 + 0.01 \times \{4 \times 3 + 6 \times 4\} = 1.01$

0.36

1.01

$FP = 506 \times 1.01 = 404.66$

	v.Low	Low	nominal	high	v.high	ext
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	1.65
product complexity	0.70	0.85	1.00	1.15	1.30	
Computer Attributes						
execution time constraints						
main storage constraints			1.00	1.11	1.30	1.66
virtual machine volatility	0.87	1.00	1.00	1.06	1.21	1.56
computer turnaround time		0.87	1.15	1.30		
			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 pratices	1.24	1.10	1.00	0.91	0.82	
use of software tools	1.24	1.10	1.00	0.91	0.83	
req development schedule	1.23	1.08	1.10	1.04	1.10	

Effort Adjustment Factor (EAF)

Calculated by multiplying all the values that have been obtained after categorizing each cost drivers

$$\text{Effort} \Rightarrow a_i (KLOC)^{b_i} \times EAF$$

$$\text{Development time} = c_i (\text{Effort})^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Q A new project with estimated 400 KLOC has to be developed. Project manager has a choice of hiring a ~~public~~ pools of developers.

i) very high capable with very little experience in programming language or developers of low quality but a lot of programming language. which one will be the better choice.

case-1

$EAF = 0.71 \times 1.14$

$= 0.809$

$\text{Effort} \Rightarrow 2.8 \times (400)^{1.20} \times 0.8094$

$= 3004.64$ PM

$\text{Dev. Time} \Rightarrow 2.5 \times (3004.64)^{0.32}$

$= 32.42$ months

case 2

$EAF = 1.46 \times 0.95$

$= 1.387$

$\text{Effort} = 2.8 \times (400)^{1.20 \times 1.387}$

$= 5149$ PM

$\text{Dev Time} = 2.5 \times (5149)^{0.32}$

$= 38.52$ months

COCOMO 2 or Complete COCOMO :

It calculates the effect of cost drivers on each phase of SDLCM. It uses phase sensitive effort multipliers for each cost drivers. It estimates module - subsystem system hierarchy.

Adjustment Factor (A) is calculated

$$A = 0.4(DD) + 0.3C + 0.3I$$

DD → Design and Development
C = Coding
I = Integration Testing

Size equivalent to $\frac{(S \times A)}{100}$

Mode and code size Life Cycle Phase	Plan & Req value of τ_p	System Design τ_p	Detail Design τ_p	Code & Test τ_p	Integ & Test τ_p
Organic Small S=2	0.06	0.16	0.26	0.42	0.16
Org. Medium S≈32	0.06	0.16	0.24	0.38	.22
S.Det Medium S≈32	0.07	0.17	0.25	0.33	.25
S.Det large S≈128	0.07	0.17	0.24	0.31	.28
Embedded large S≈128	0.08	0.18	0.25	0.26	.31
Embd XL S≈320	0.08	0.18	0.24	0.24	.34

Life Cycle Phase value of τ_p

organic small S=2	0.10	.19	.24	.39	.18
org. Med S≈32	0.12	.19	.21	.34	.26
S.Det Med S≈32	.20	.26	.21	.27	.26
S.Det large S≈128	.22	.27	.19	.25	.29
Embedded large S≈128	.36	.36	.18	.18	.29
Embd XL S≈320	.40	.38	.16	.16	.30

$$Effort_d = \mu_p E$$

$$Dev\ Time = \tau_p D$$

Effort = $3.2 \times (12)^{1.05} \times 1.216$
= 52.87 PM

EAF = $1.15 \times 1.07 \times 1.15 \times 0.86$
= 1.216

S/w Reliability = High
Language exp. = Low
Product Complexity = High
Analyst capability = High

Dev Time = $2.5 (52.9)^{0.38}$ months
= 11.29 months

Effort
Plan & Req = 0.06×52.9 PM
=
Design = 0.16×52.9 PM
=

5/8/21

Coupling: Coupling is the measure of degree of interdependence between modules. Low coupling is highly desire because high coupling leads to more errors ad it makes isolation of errors at the time debugging is very difficult.

When modules store global/share data they make calls to each other. Control the cmt. of information exchange between modules is a way to control the coupling pass only the data not the control information.

Cohesion

The degree to which elements of a module are functionally related to each other. A strongly cohesive module implements functionality focusing on a single feature of the solution or software. High cohesion is always desirable.

Types of Cohesion :

i) Functional Cohesion :

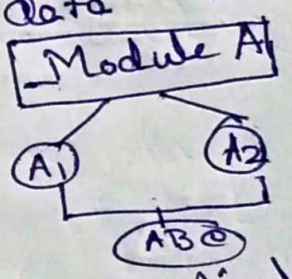
When two operations present within a module perform the same functional task or they are part of the same purpose

ii) Sequential Cohesion :

In a module two operations are such that x's output → y's input } x, y ∈ same module.

iii) Communicational Cohesion :

Elements inside a module that operates on same input data or contribute to same data



iv) Procedural Cohesion

Modules whose instructions do different task but to ensure a particular order in which task are performed, they are put into same module.

A1 ↓
A2 ↓
A3 ↓
An

This type of modules leads to poor maintainace

v) Temporal :

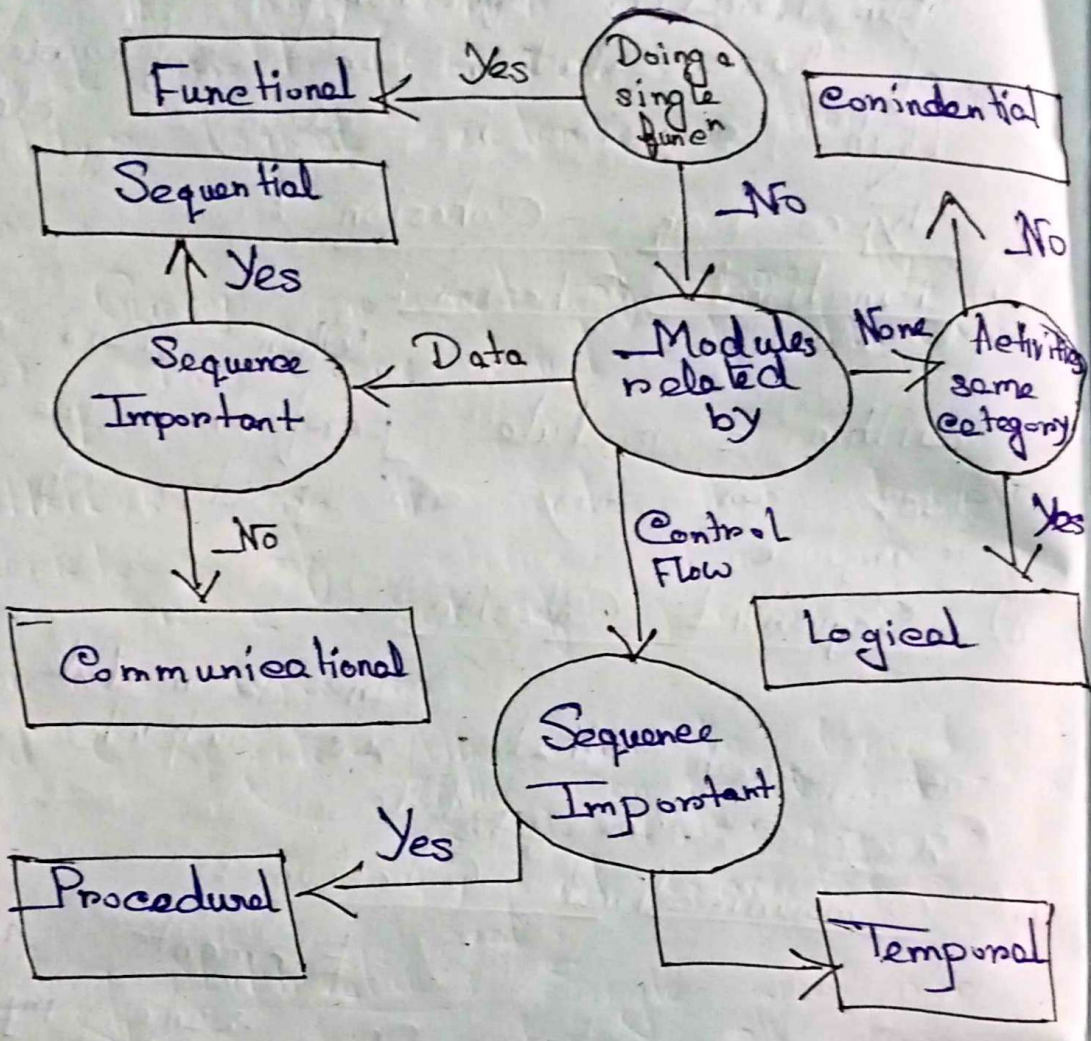
Instruction that must be executed at same time are put together.

vi) Logical Cohesion :

When modules have logically similar instruction or elements

Coincidential :

Instructions are mostly not related to each other. This is not desirable at all.



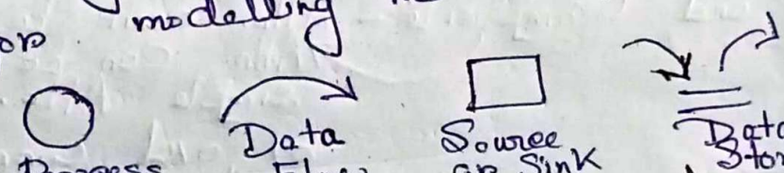
Function Oriented Design

It is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function. That mean the system is designed functional POV.

Types of Function Oriented Design -

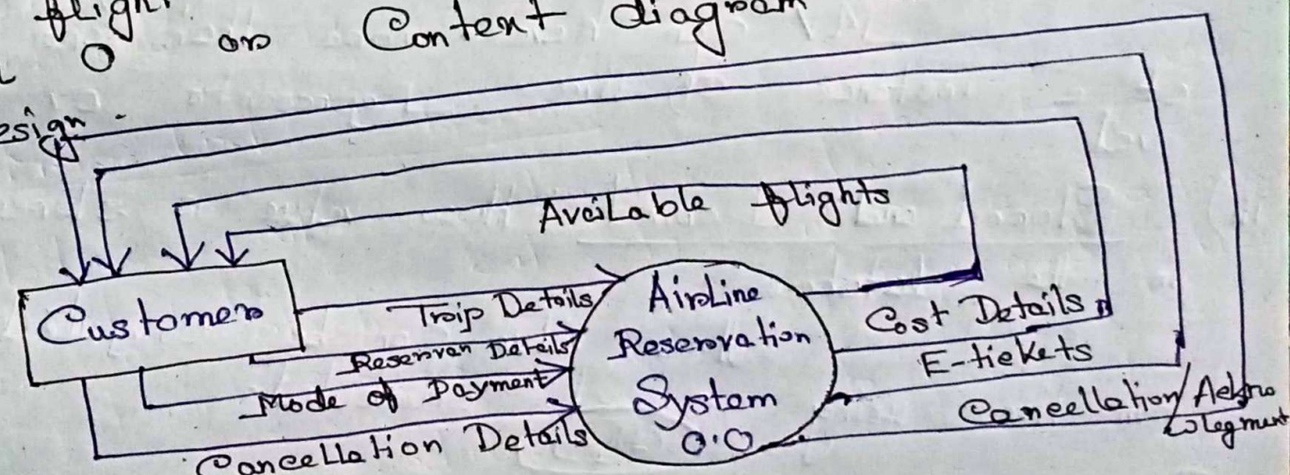
1) DFD (Data Flow Diagram)
2) Data Dictionary
3) Structure Chart
4) Pseudo Code.

DFD: also known as Bubble Chart. DFD shows the flow of data through the system and is also used for modelling the requirements.



Draw a level 0 level 1 and level 2 DFD of flight ticket reservation System.

Level 0 or Context diagram or Fundamental Design.



Entity Relationship Diagram

An entity is a thing or an object in a real world that is distinguishable from others objects best on the values of the attribute is processed

Tangible Entity: The entity which is physically exist in the real world. (Car, Bank Locker)

Intangible Entity: Logical existence is there. (Bank Acc.)

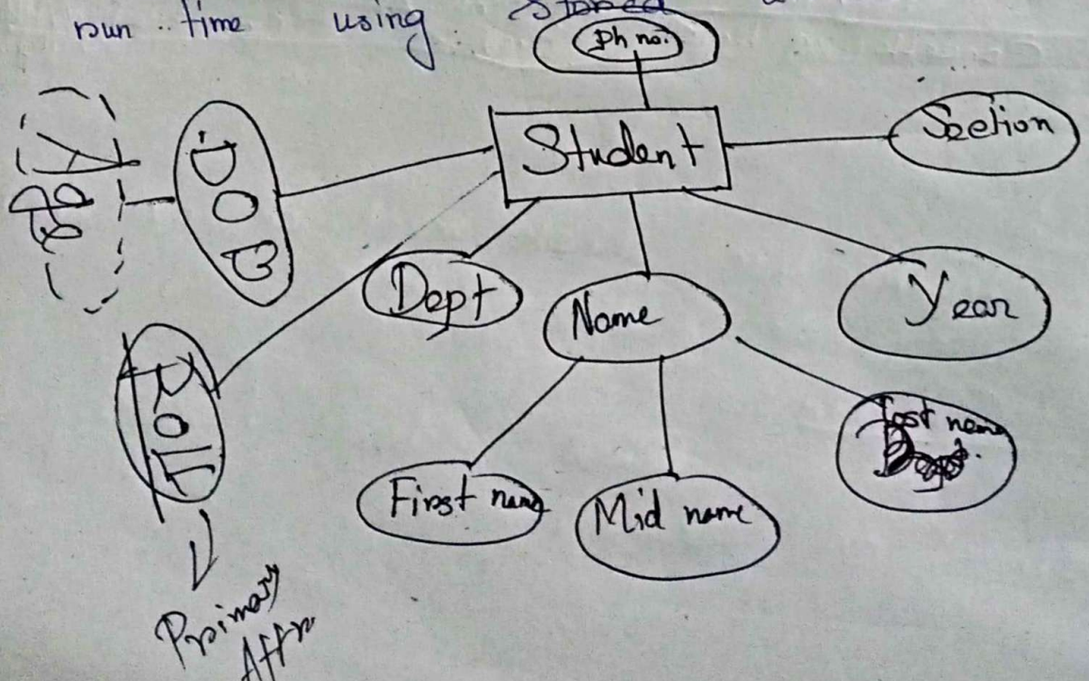
Entity Set: Group or set of same type of entities that share same kind of attributes or entity set.

Attribute: Attributes are the units that describes the characteristics of entities. For each attribute there is a set of permitted values called domains. Types are-

i) Simple and Composite – Attr. can't be derived further
ii) Attr. can be divided in simple Attr. represented by oval connected to another oval.

ii) **Single & Multivalue:** Single attr. can have only one value at a instance of time. MV can have more than one value at a instance of time. Represents by double oval

iii) **Stored and Derived:** How value is stored in the database.
How value can be computed in run time using stored attributed.



Relationship is an association betn two more entities of same or different entity set.

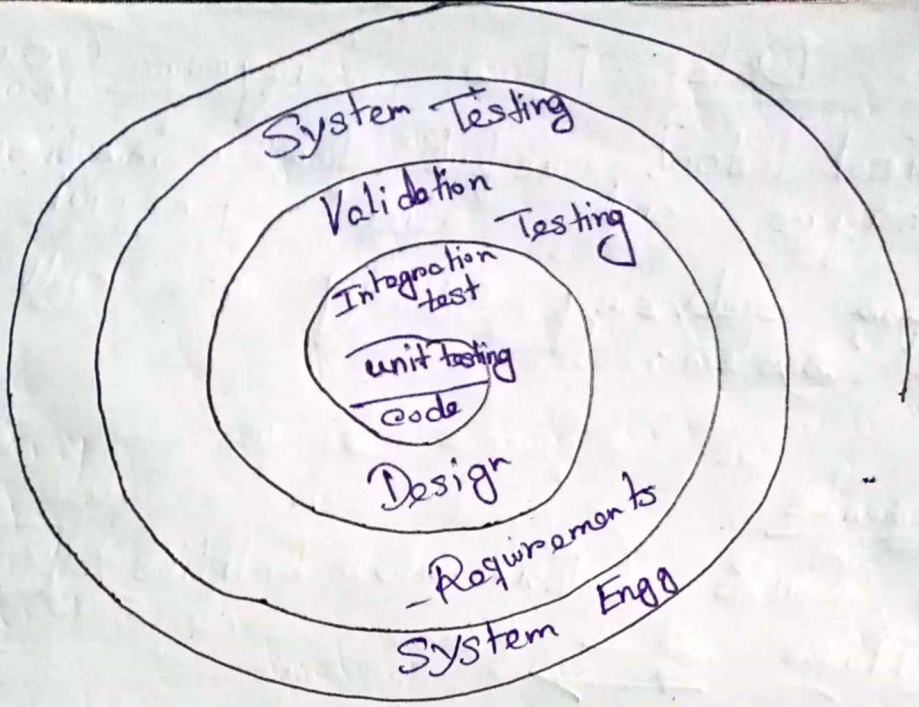
Binary Ternary Quaternary

Degree of the relationship set: number of entity set associated and participated in the relationship set.

Cardinality: Cardinality ratio or mapping cardinality Express the numbers of entity to which other entity can be related via relationship.

Type of Cardinality:

i) one to many
ii) one to one
iii) Many to many



Here we'll check software architecture and

Validation Testing - Requirements establish are validated against the developed software. Here behaviour of the software performance requirements and all kind of validation criteria are matched

System Testing - The software is tested in other system elements as hardware and software.

Test Case Design : A test case is a triplet of [I, S, O] where I is the data input to the system, S is the state of the system at which the data is the input and O is the expected output of the system.

Data Flow Diagrams (Bubble Chart)

- A graphical tool, useful for communicating with users, managers, and other personnel.
- Useful for analyzing existing as well as proposed system.

DFD Elements

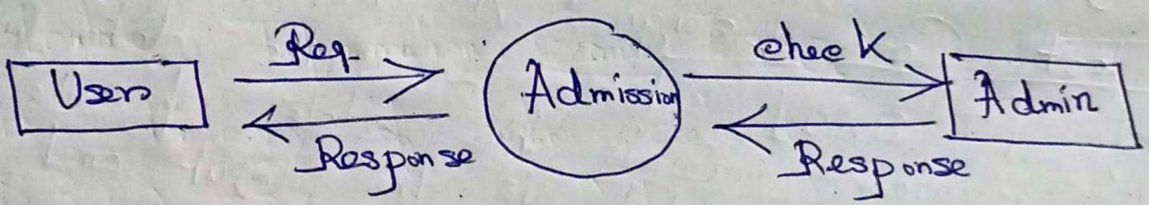
- Source / Sinks (External Entities)
- Data Flows
- Process
- Data Store / Data Base

write Read

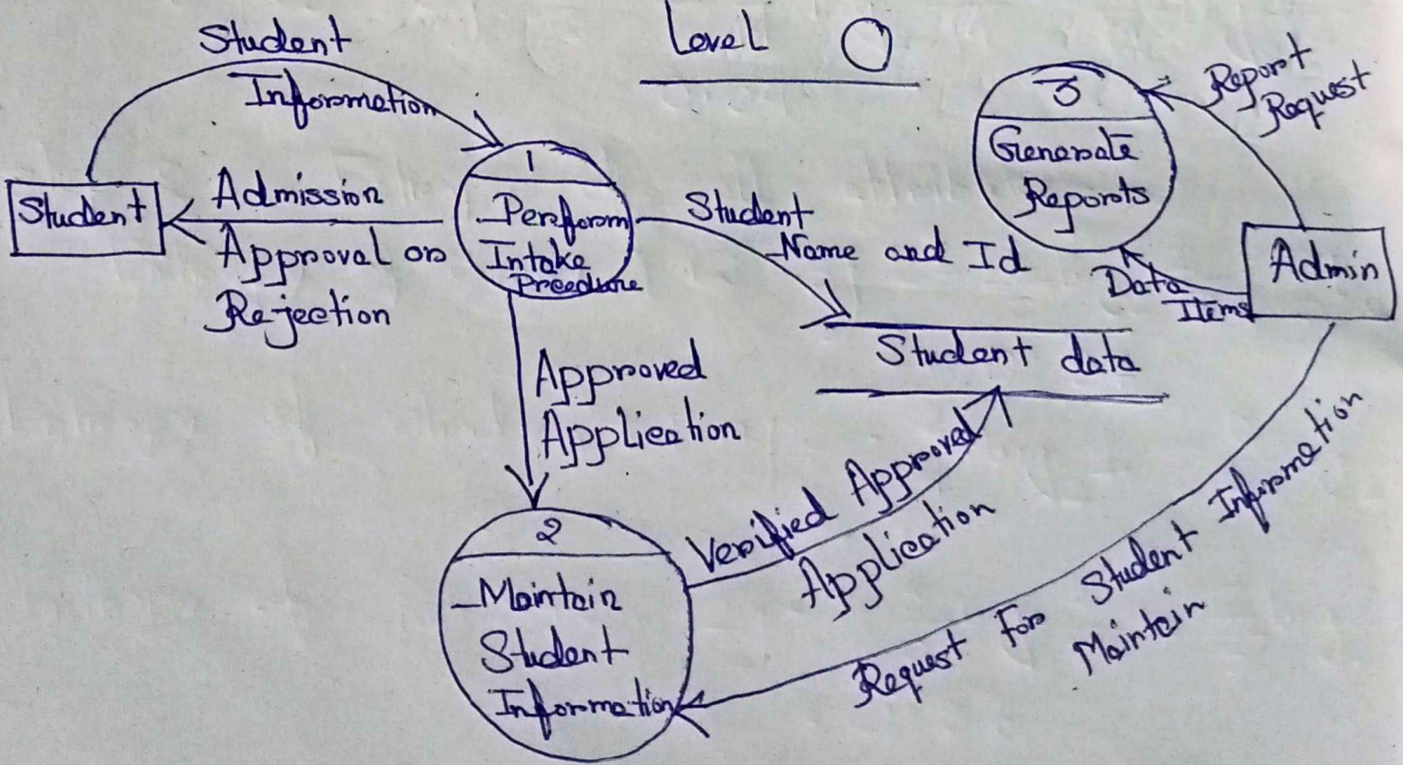
Source - Entity that supplies data to the system.

Sink - Entity that recieves data from system.

DFD for University Admission System



Level 0



Level 1

Software Design Approaches

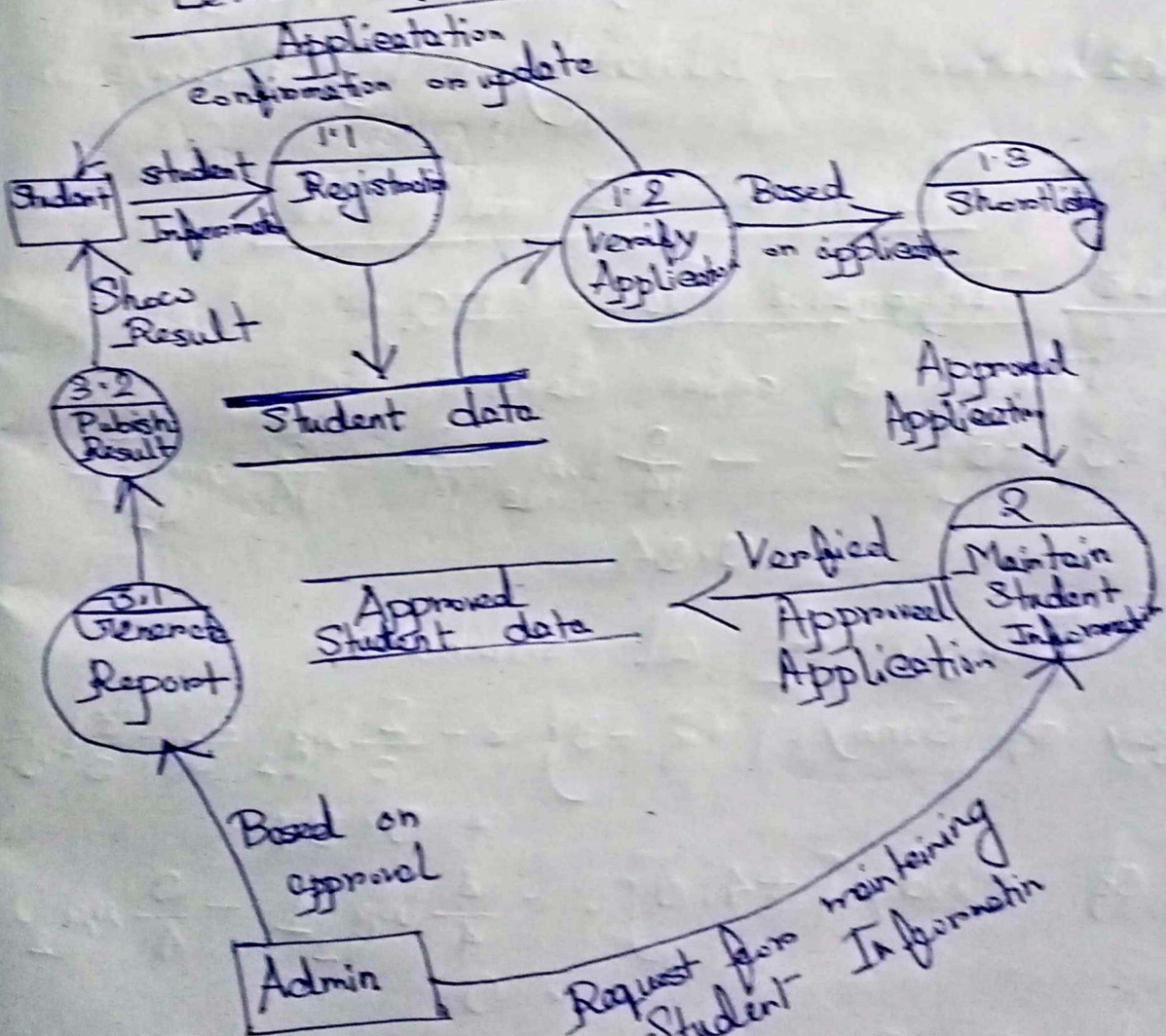
Function Oriented Design

- System is designed from a functional viewpoint
- Top - Bottom decomposition
- DAC approach
- DFD is used.
- Begins by considering use case diagram and scenario.
-

Object Oriented Design

- System is viewed as a collection of objects (i.e. entities)
- Bottom-up approach.
- UML is used.
- Begins by identifying object and classes

Level 2 DFD for University Admission Syste



Software Design: Software design is the process of designing the elements of a software such as the architecture, modules and components, different interfaces of those components and the data that goes into it.

In software design, the development is divided into several sub-activities, which coordinate with each other to achive the main objective of software development.

Problem Partitioning / Decomposition: It refers to break-down a complex system into components or smaller-subsystem.

another system design process are Abstraction, Modularity, Top-Down, Bottom-up

Decision Tree and Table

Calculate Information Gain (IG)

Step-1: Entropy of entire dataset

$$S\{+9,-5\} = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14}$$

$$= 0.94$$

Step-2: Entropy of all attributes.

Entropy of Sunny $\{+2,-3\} = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}$

$= 0.97$.

Entropy of cloudy $\{+4,0\} = -\frac{4}{4}\log_2 1 - \frac{0}{4}\log_2 0$

$= 0$

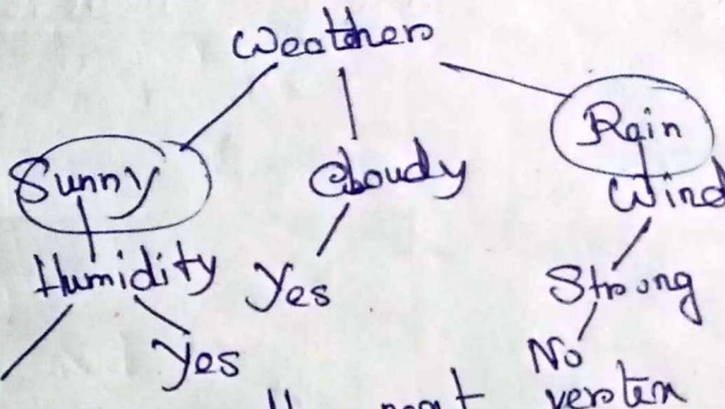
Entropy of Rain $\{+3,-2\} = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}$

$= 0.97$

$$IG = Ent(whole) - \frac{5}{14}Ent(S) - \frac{4}{14}Ent(c) - \frac{5}{14}Ent(R)$$

$= 0.246$

Then Calculate the IG of all columns. Greatest IG will be the root node.



Then for finding the next vertex have to calculate IG for temperature, based on the whole entropy is Sunny.

Step 1 Entropy of Sunny (+2, -3)

$$= -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97$$

Entropy of all Attributes:

Ent of cool {+1, 0} $= -\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1}$

$= 0$

Ent (Mild) {+1, -1} = 1.0

Ent (hot) {+0, -2} = 0

$IG = Ent(Sunny) - \frac{1}{5}Ent(Cool) - \frac{2}{5}Ent(Mild) - \frac{2}{5}Ent(hot)$

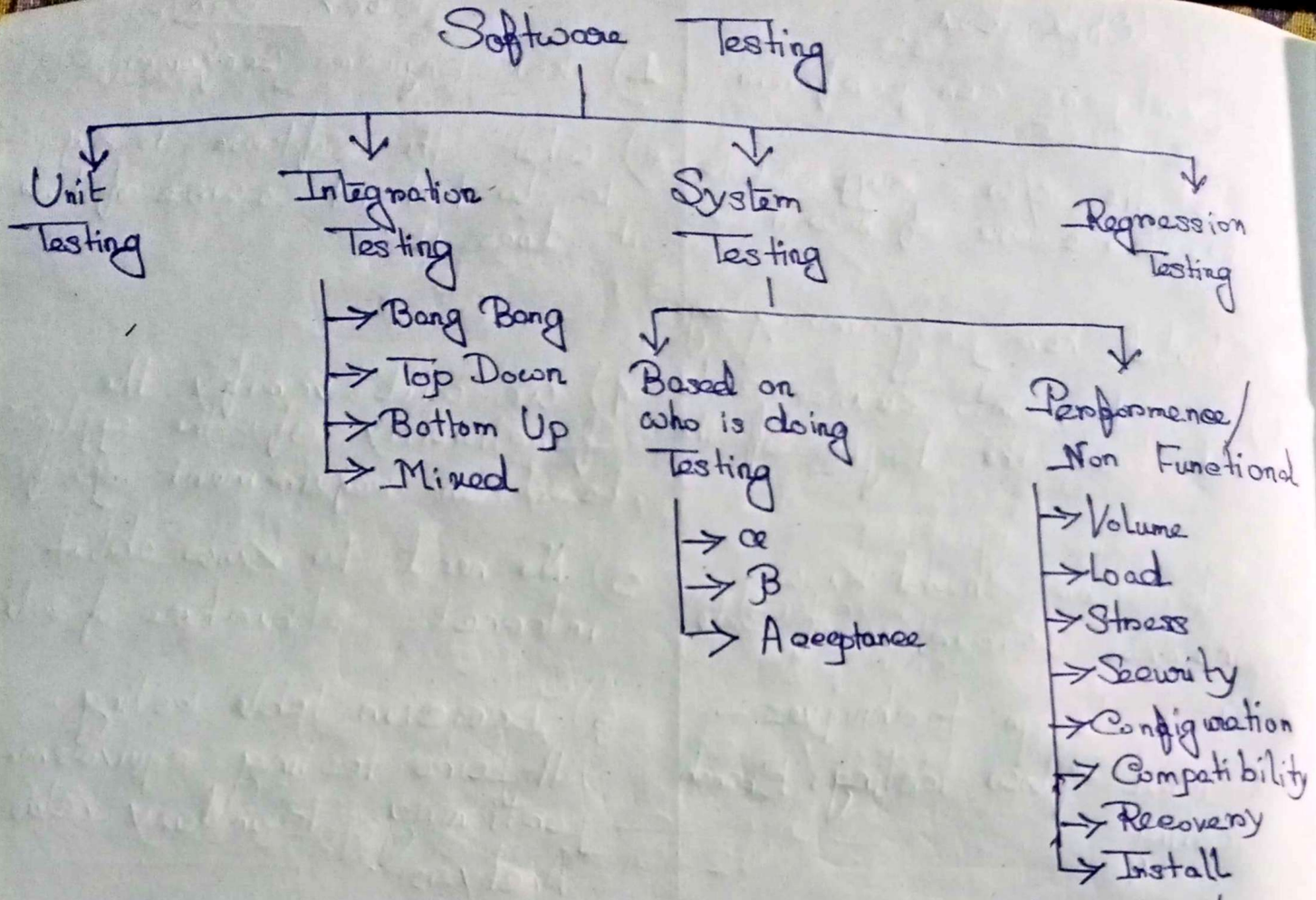
$= 0.57$

IG of (humidity/Sunny) = 0.97

IG of (Wind/Sunny) = 0.019

IG of (Temp/Sunny) = 0.57

White Box	Black Box
1) Developers can perform WBT.	1) Test Engineers perform BBT.
2) What the s/w is supposed to do, also aware of how it does it.	2) what the software is supposed to do but not aware of how it does it.
3) Should have programming lang.	- -
4) we look into source code and test the logic of the	4) we will verify the functionality of the applica based on requirement specifi
5) developer should know internal structure of code.	5) No need to know about internal structure of code
6) Test design techniques - Control-flow testing, Branch testing. . . .	6) Decision table testing, All-pairs testing, equivalence partioning, Boundary Value Analysis.
7) Can be applied mainly in unit testing but now integra system testing also.	7) System testing.



White Box Testing: Also known as Clear box/Glass box/ Transparent box and Structural Testing. It is a method of Software testing that tests internal structures of working of an application. White-box testing can be applied at the unit, integration and system levels of the software testing process.

WB Testing tools are Veracode, CPPUNIT, RCUNIT etc.

Design Techniques

- Control Flow Techniques.
- Data " "
- Branch Technique.
- Statement Coverage.
- Decision coverage
- Path Testing.

Unit Testing

It is the process of taking a module and testing it in isolation from the rest of the software and comparing the actual results with the results define in specification and design. Unit testing is the process in checking small pieces of code to ensure that individual parts of a program work properly by their own. done by developers.

What to test?

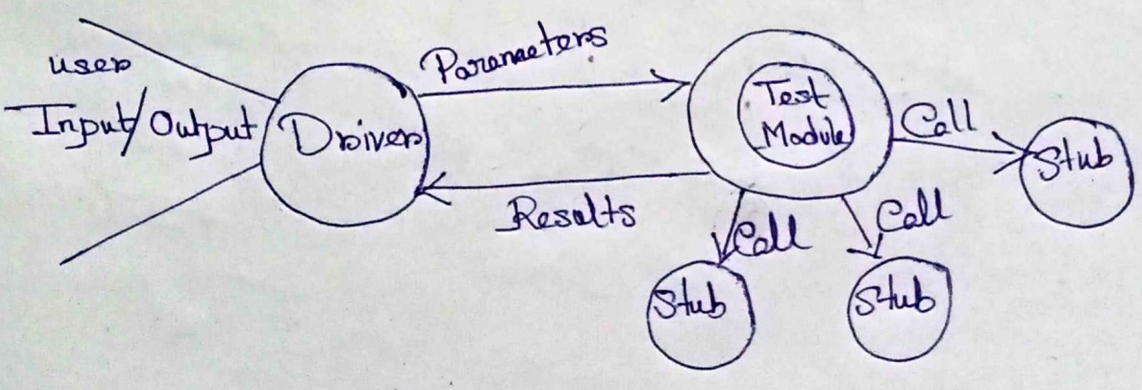
ii) **Local Data Structure** — To ensure that intermediate variables and data structures are initialized and executed properly.

iii) **~~Independent~~ ~~Intermediate~~ / Basic Paths** : Independent or basic path tested to ensure all the statement within the module atleast once during testing

iv) **Boundary Condition** : Output or computations and boundary value is tested.

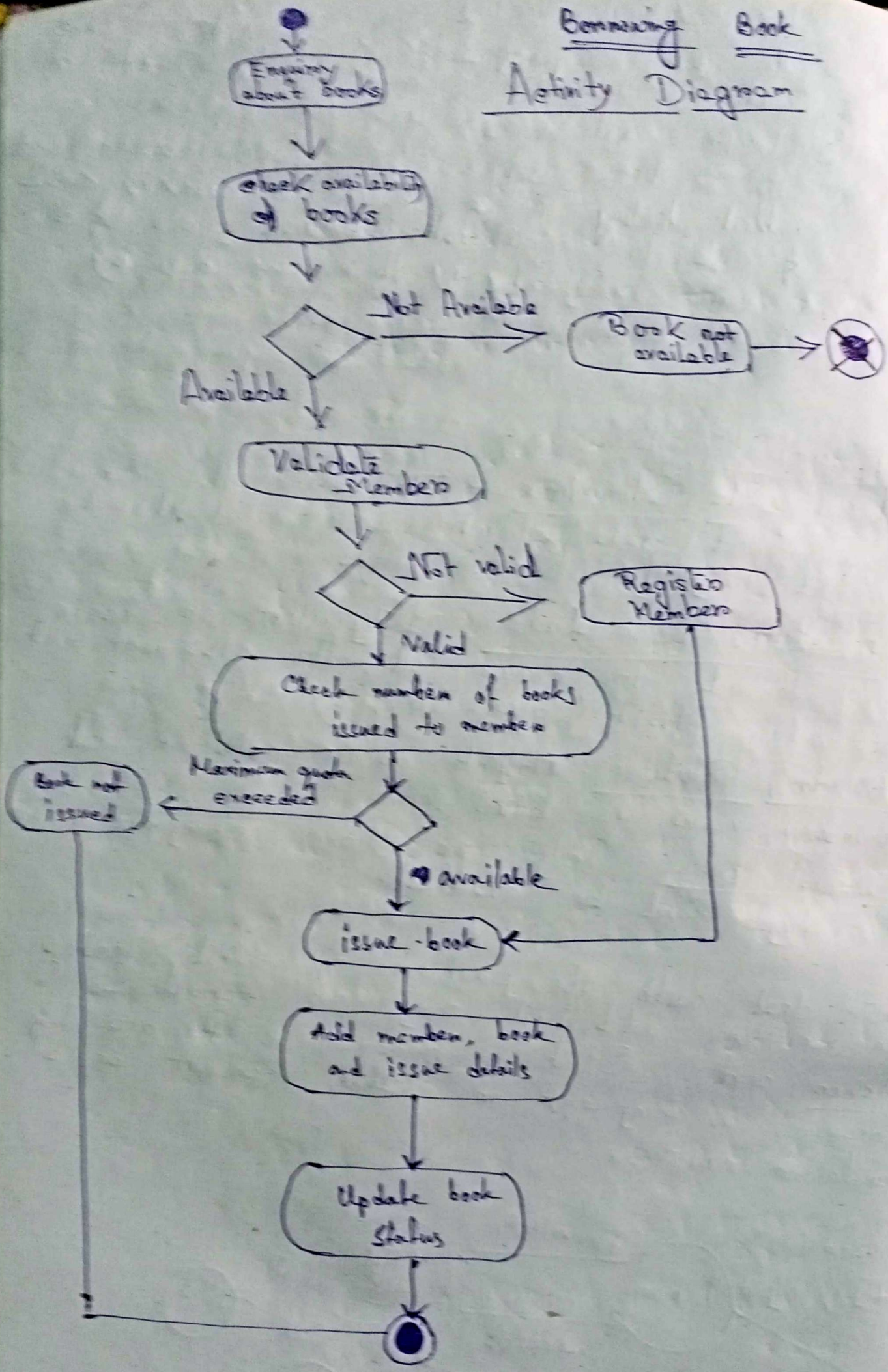
v) **Internal Logic** : Loop, procedures, comparision or data type will be checked.

To deal with the problems of Unit Testing we use two additional components named as **Drivers** and **Stub**. Because in the real life scenario a component or module can not be a standalone unit.

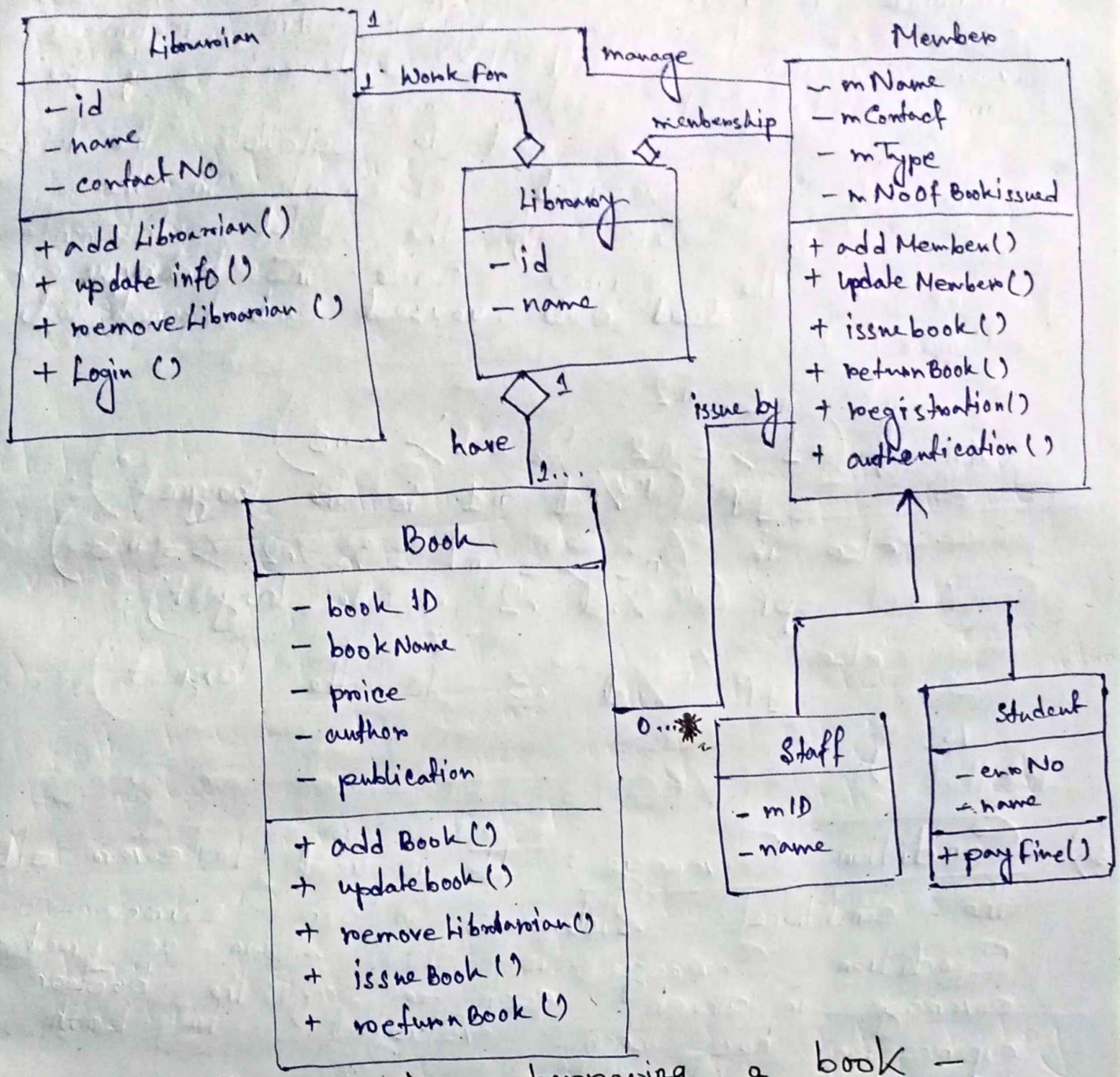


Borrowing Book

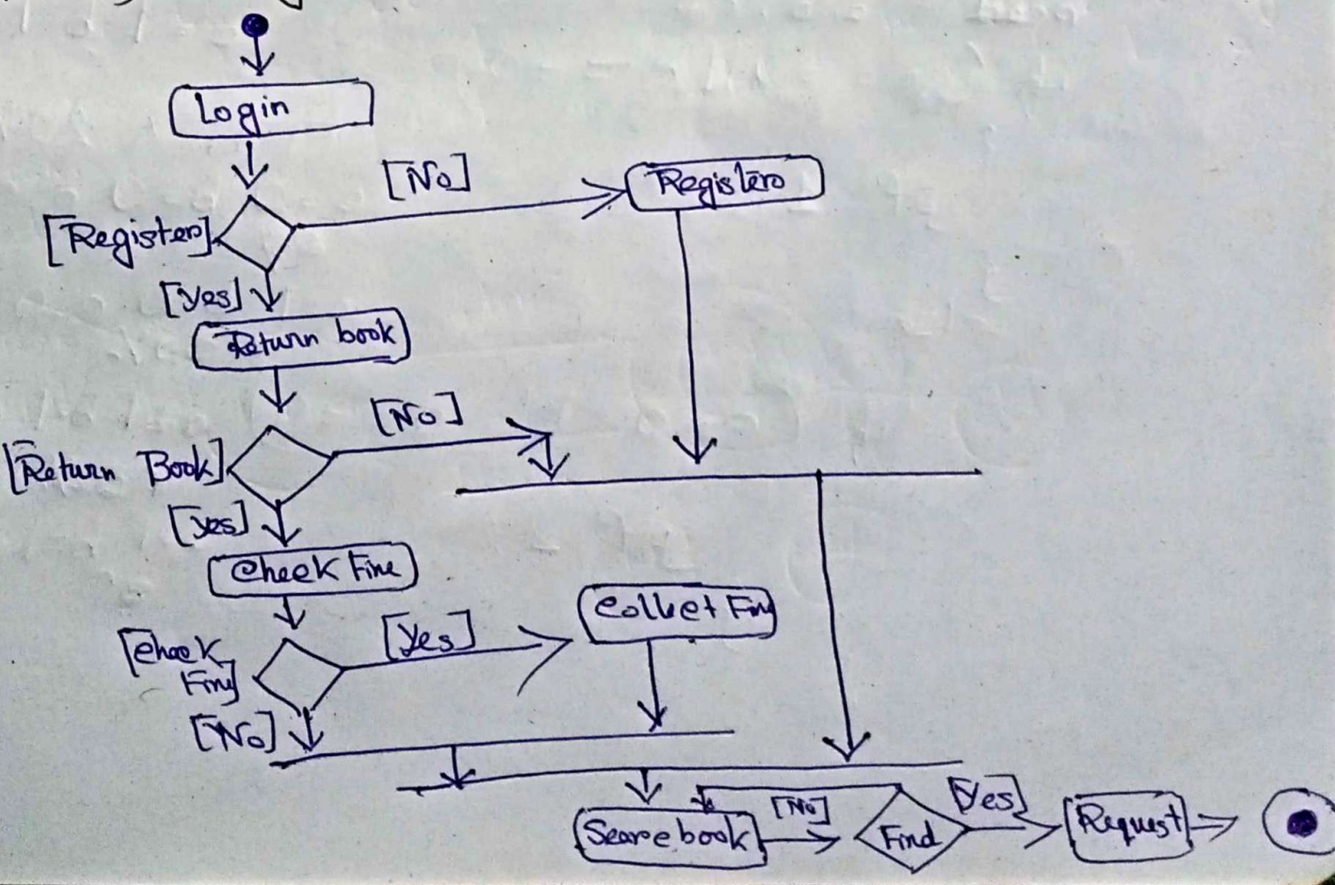
Activity Diagram



Class Diagram



Activity Diagram for borrowing a book -



1) **Statement Coverage**: The statement coverage technique is used to design white box test cases as the name suggest this technique involve execution of all the statements of the source code at least once.

It is used to calculate the total number of executed statement in the source code out of the total statements present in the source code. This technique ignore dead code, unused code, branch code

```
1. main ( )
2. {
3.   int n1, n2;
4.   if (n1 >= n2 && n1 >= n3)
5.     pf ("%d n1 is the largest");
6.   else if (n2 >= n1 && n2 >= n3)
7.     pf ("%d n2 is the largest");
8.   else
9.     pf ("%d n3 is the largest");
10. }
```

2) **Condition Coverage**: It is used to cover all the conditional statements in our source code Condition Coverage is also known as predicate coverage in which each one of the boolean expression have been evaluated to both true and false.

```
read a, b, c, d;
if ( a == 0 || b == 0)
  {
    pf 1;
  }
else
  {if (c==0 && d == 0)
    {
      pf 2;
    }
  }
```

a=1 b=1 c=0d

T || X 25%

a=0 b=0 c=0 d=0

25%

a=1 b=0 c=0 d=0

50%

a=1 b=1 c=0 d=

100%

a=1 b=1 c=1 d=

8

3) • Path Coverage: It requires designing test cases such that all linearly independent paths or basis paths in the program are executed at least once. A linearly independent path can be defined in terms of control flow Graph (CFG).

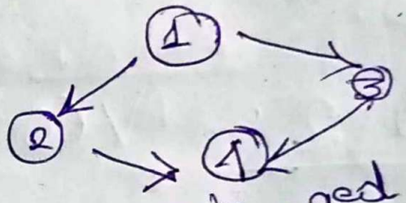
Control Flow Graph: describes the sequence in which different instruction of a program get executed. It describes how the control flows to the program.

Sequential

```
1. a = 5;
2. b = a * 2 - 1;
```

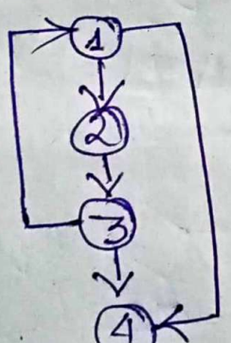
Selection

```
1. if (a > b)
2.     c = 3;
3. else c = 5;
4. c = c * c;
```

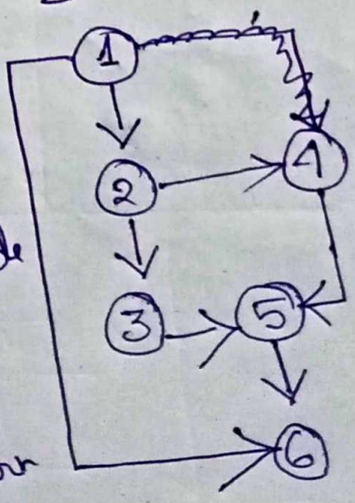


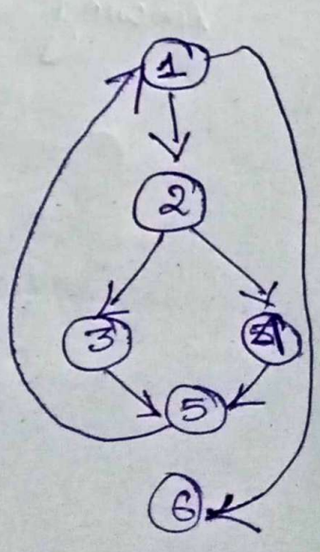
Iteration

```
1. while (a > b) {
2.     b = b - 1;
3.     b = b * a; }
4. c = a + b;
```




```
int compute_gcd (int x, int y)
{
1. while (x ! y) {
2.     if (x > y) then
3.         x = x - y;
4.     else y = y - x
5. }
6. return x;
}
```





1 2 3 5 1 6
1 2 4 5 1 6
1 6

How to find out the path:
A path through a program is any node and edge sequence from the start node to the terminal node of the CFG of a program in the presence of or exit statement.

Linearly independent set of path -

set of Path for a given program LISP
on basic path if each path in the set
introduces at least one new edge that
is not included in any other path in the set.

MC Cabe's Cyclomatic Complexity:

Cyclometric Complexity defines an upperbound
on the numbers of independent path in a
program.

Method 1: Given a CFG G of a program the cyclomatic complexity $V(G)$ can be computed as $V(G) = E - N + 2$

$= 7 - 6 + 2$

$= 3$

Method 2: $V(G) =$ Total no. of non overlapping bounded areas $+ 1$

$2 + 1 = 3$

Method 3: Compute the no. of decision and loops present in the program. If the no. is defined by

$n \cdot V(G) = n + 1$

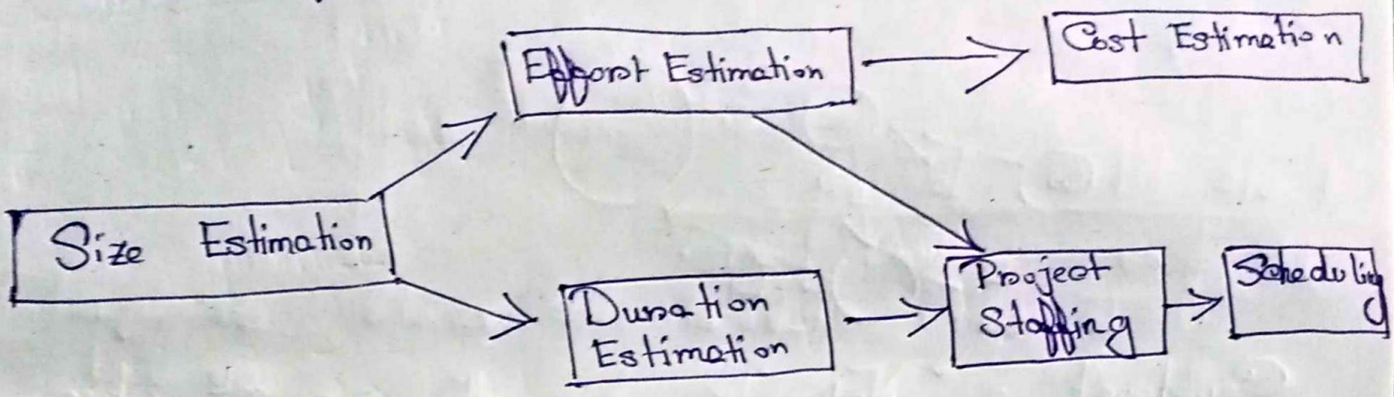
$2 + 1 = 3$

Knowing the no. of

Software Project Management

Software Project Management is an art and Science of planning and leading Software Projects. The main goal is to enable a group of developers to work effectively to ours successful completion of projects.

- Role of Software Project Manager.



Project Ordering

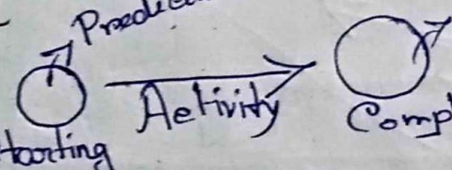
Program Evaluation and Review Technique -

PERT -

Network: A network consists of a set of points and set of lines connecting different pairs of points.

Events: The phases or the steps required during the software project developments represented by circles in the network diagram.

Activity: The Arrows are called as activity.

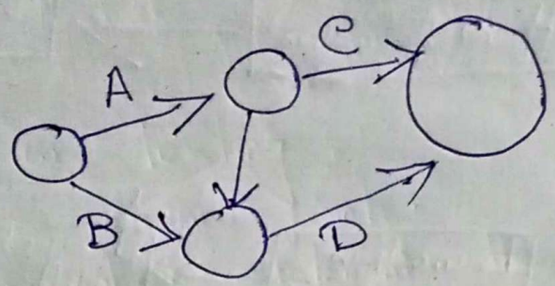


Dummy Activity: An activity which doesn't consume either any resources is known an dummy Activity.

Draw a Network Diagram to express following Relationship:

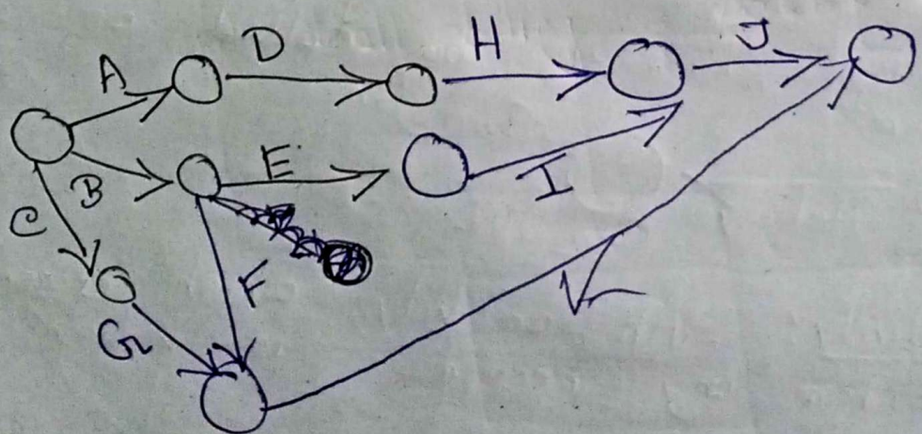
Activity	Immediate Predecessor
A	—
B	—
C	A
D	A, B

48
25



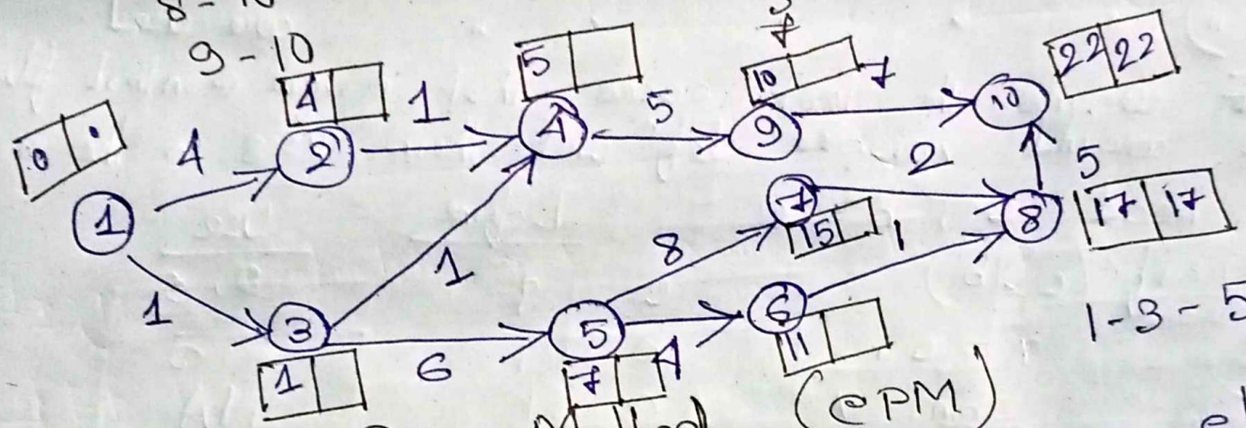
Construct a Network for the project whose activity and Precedence Relationship are given below.

Activity	Precedence
A	—
B	—
C	—
D	A
E	B
F	B
G	C
H	D
I	E
J	H, I
K	F, G



Construct a Network Diagram for the project whose Activity, Precedence, Time is given.

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

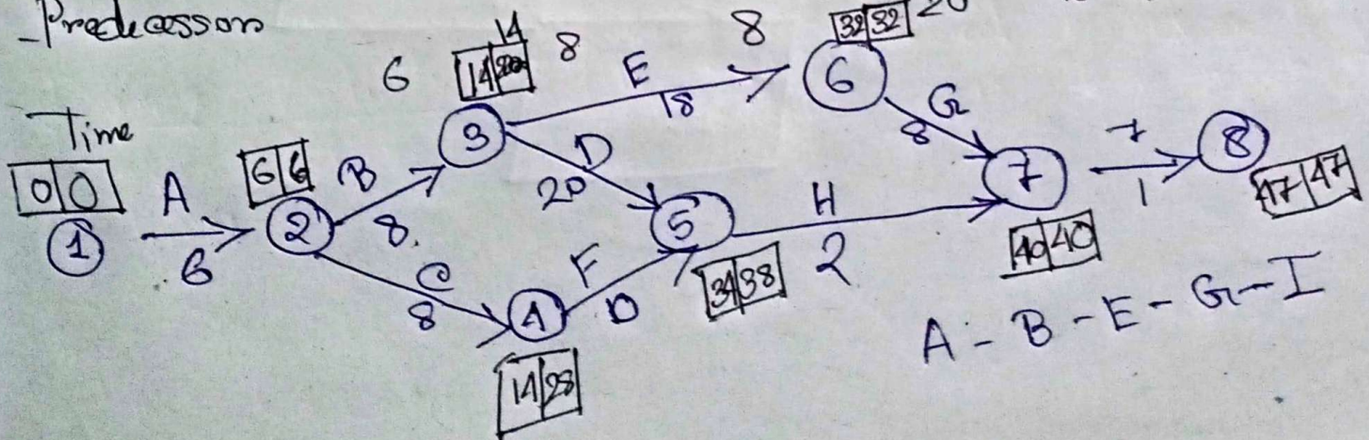


Critical Path Method

The Critical Path is the continuous chain of critical Activities in a network diagram. It is the longest path starting from first to last event and is shown by a thick line or double line.

Following are the information event about the project

Activities	A	B	C	D	E	F	G	H	I
Predecessor	—	A	A	B	B	C	E, F	D, F	G, H
Time	6	8	8	20	18	10	8	2	7



Float (Slack of an activity) The float is the length of time to which a non critical activity can be delayed or extended without delaying the total project completion time.

Activity (i, j)	Normal Times (t_i)	Earliest time Start E_i	Earliest time Finish $E_i + t_i$	latest time Start $L_i - t_i$	latest time Finish L_i	Total Float $L_i - E_i - t_i$	L_i

Data Flow Testing :

It focuses on two points.

i) In which statement the variables are defined.
ii) " " " " " are used.

It designs the test cases that covers control flow paths around variables def^n and their uses in the module

```
1. read a, b, c;
2) if (a>b)
3)     x = a+1
4)     print x;
5) else
6)     x = b-1
7)     print z;
```

Define		Use
a	1	2, 3
b	1	2, 5
c	1	NA
x	3, 5	4
z	NA	6

Advantages : i) variable that is defined but never used.
ii) " " is used but never declared.
iii) " " is defined multiple time before it is used.
iv) deallocating a variable before it is used

Activity Diagram

Flowchart to represent the flow of control among the activities in a system.

The flow of operations can be sequential, branched or concurrent.

Start

Final node

Control flow

Join

Fork

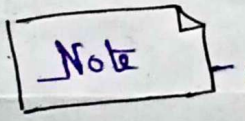
Task

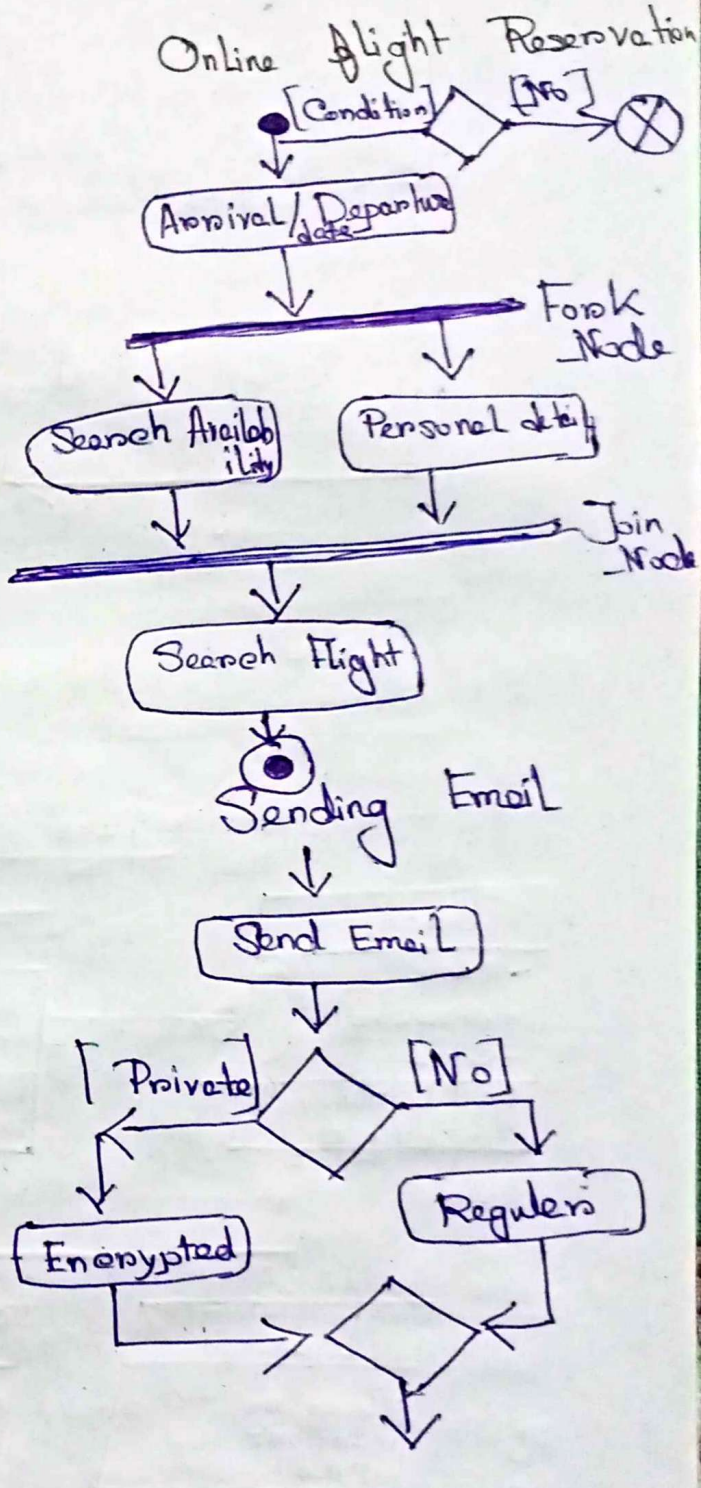
Decision Node

Merge Node

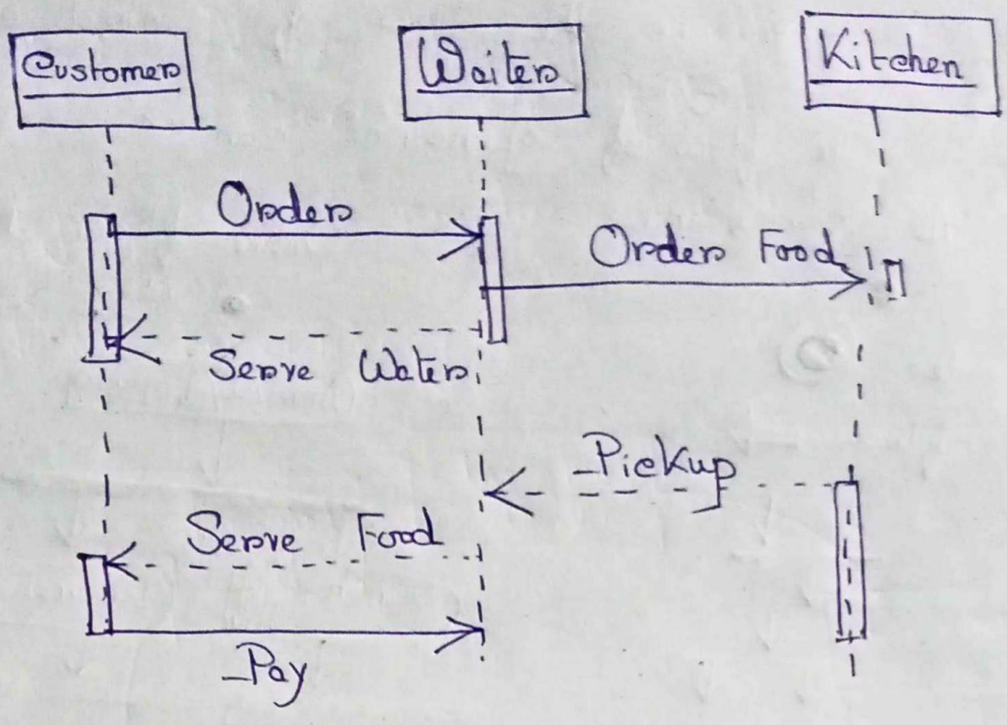
Final Flow Node

Note

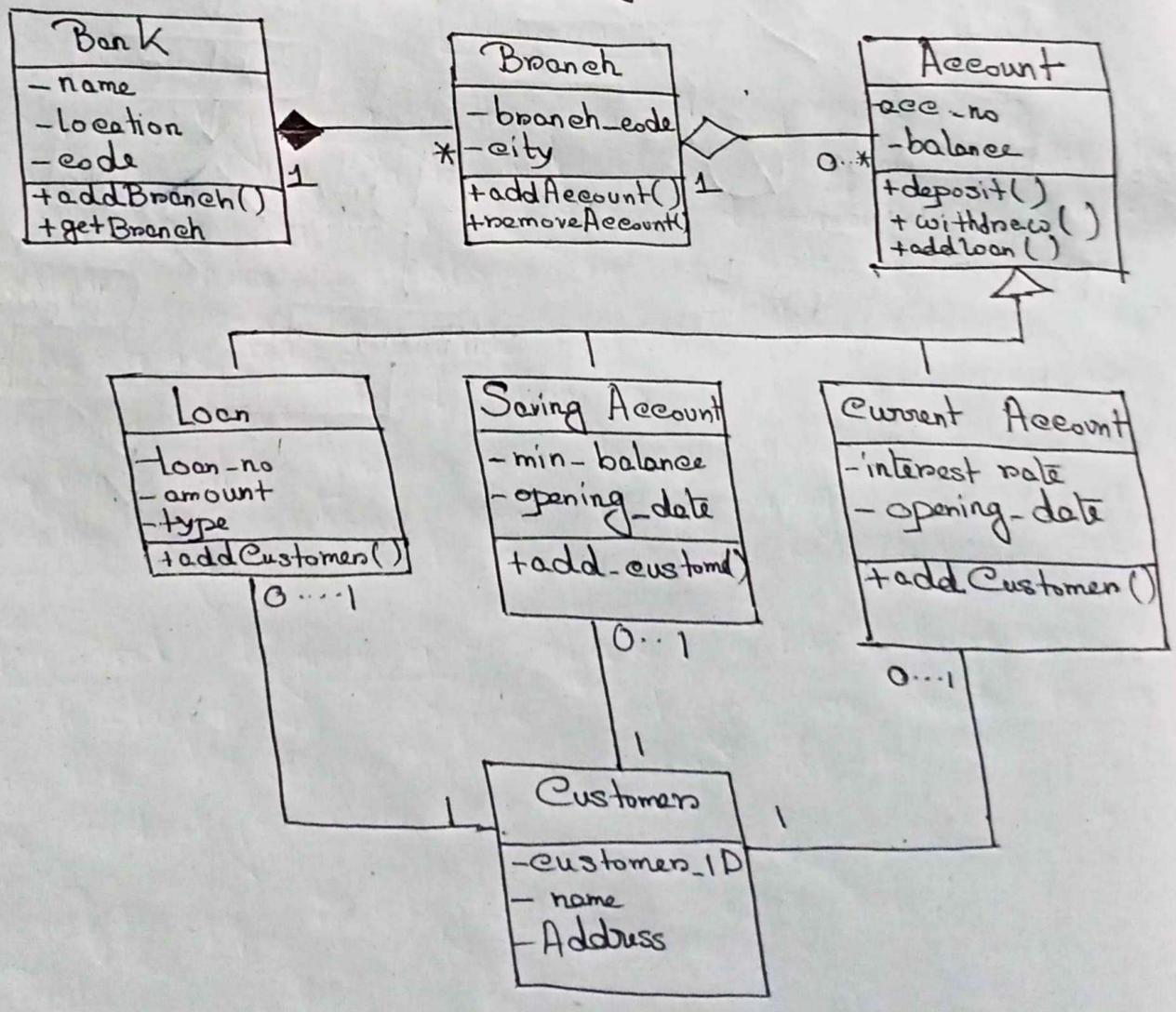




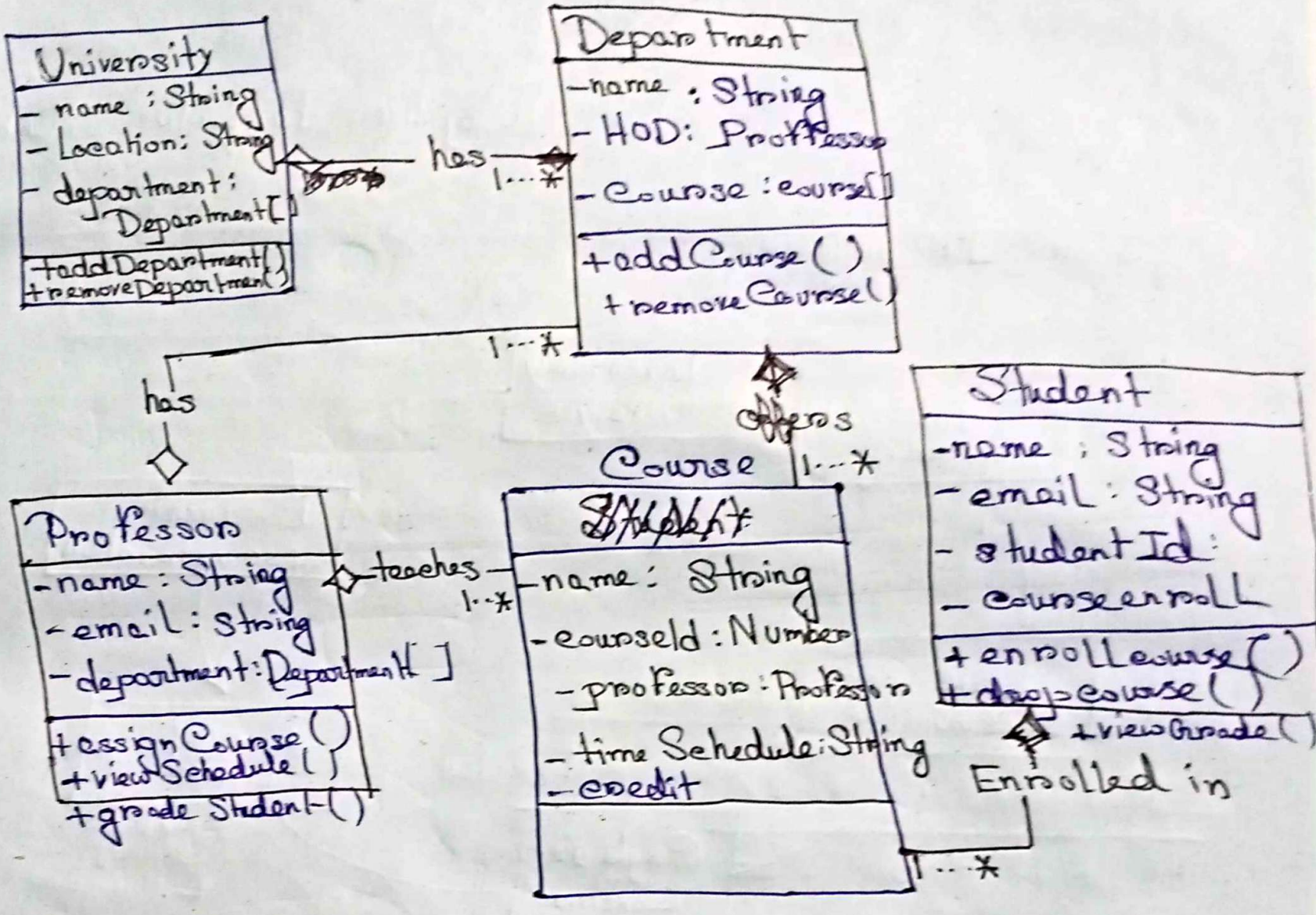
Sequential Diagram



Class Diagram



Class Diagram for University Management

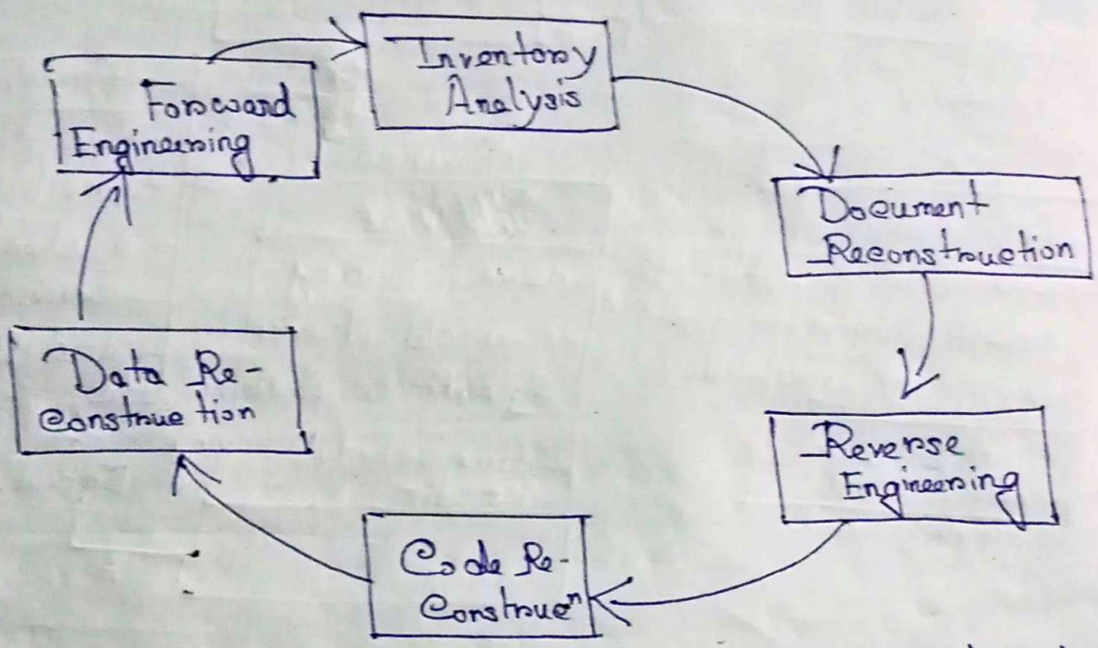


Software Project Maintainance

usage : Always changing, Getting more complex, Survival for fittest.

Categories - Corrective, Adaptive, Perfective Maintainance

Software Re-engineering



Reverse Engineering: This is the process of deconstructing a system to understand its components and their relationships, often to create documentation for a system where none exists or is out of date.

Restructuring: This involves transforming the existing source code into a more maintainable form, often while preserving its functionality. This can include activities such as code re-factoring, where the code is recognized to be more understandable or efficient.

Forward Engineering: After understanding and possibly restructuring the system, changes are made to the system to improve it or adapt it to new requirements or technologies.

Benefits of Re-Engineering:

Cost

Lower Risk

Better use of existing staff

Incremental Development.

Software Configuration Management Activity

Primary Reasons

- There are multiple people working on software which is continually updating.
- changes in user requirement, policy

objectives: Remote System administration, Reduced user down-time, Reliable data backups, Easy Workstation setup, Multiuser Support.

Tasks in SCM Process

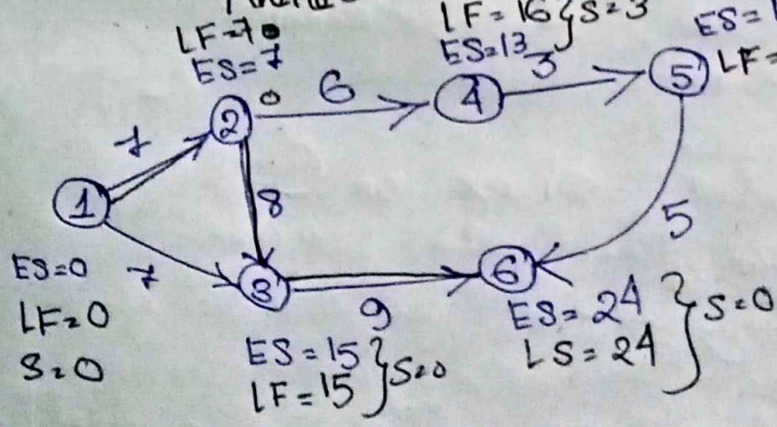
- Configuration Identification
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews.

Resource Allocation Models:

Activity	Day
1-2	7
1-3	4
2-3	8
2-4	6
3-6	9
4-5	3
5-6	5

Resources Machine	Man
2	20
2	20
3	30
1	30
2	20
2	20
2	20
2	20
4	40

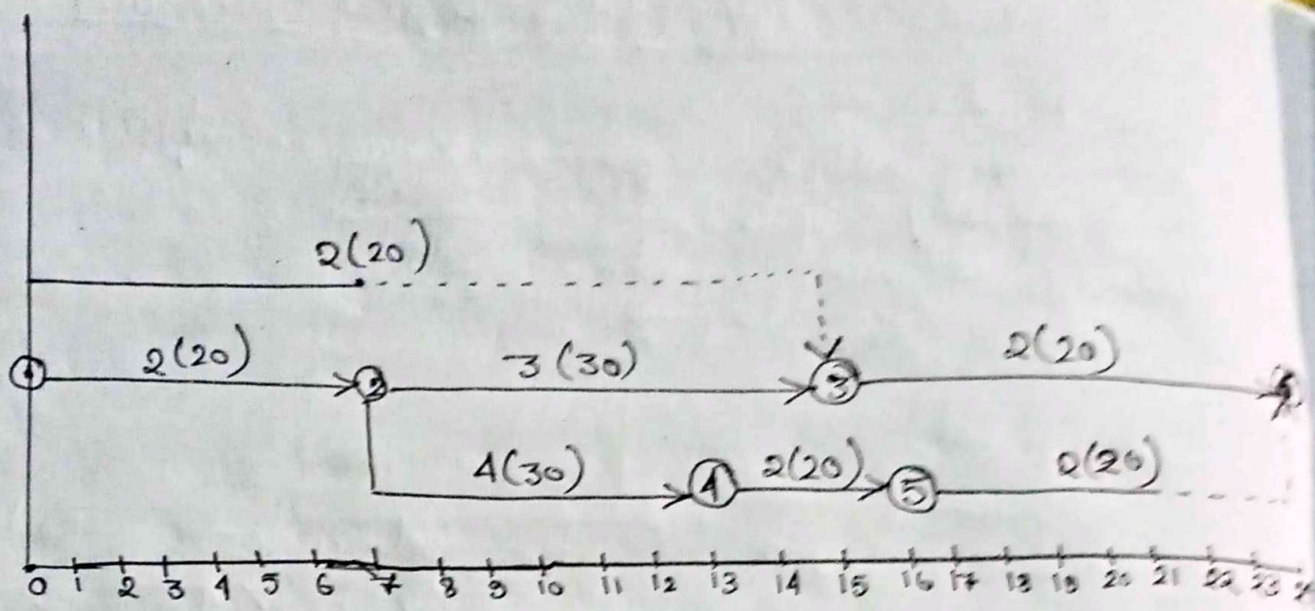
Available Resource -



Earliest start = highest
Lowest Finis = Lowes

Critical Path = 1-2-3-6

duration = 24 days



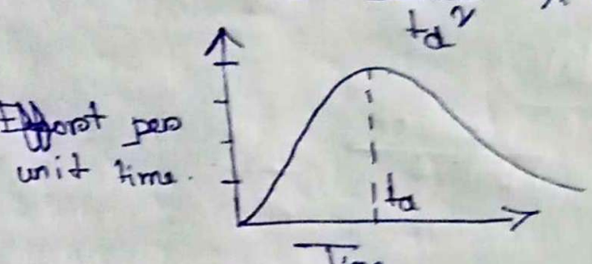
Squared Network

Staffing Level Estimation:

Norden's Work: Norden found that the staffing pattern can be approximated by Rayleigh Distribution Curve.

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

E = effort req. at time t.
K = area under the curve.
t_d = time at which the curve attains its max value.



The Results of Norden are applicable to general R&D project aren't as meant to model the staffing pat

Putnam's Work:

$$L = C_k K^{4/3} t_d^{4/3}$$

L = product size in KLOC
K = total effort
t_d = time of system, integ and testing
C_k = constant.

$C_k' = 2$ for poor development environment
$C_k' = 8$ for good "
$C_k = 11$ " excellent "

Effect of Schedule change on Cost

$$L^3 = \{C_k K^{1/3} t_d^{4/3}\}^3$$

$$L^3 = C_k^3 K t_d^4$$

$$K = L^3 / C_k^3 t_d^4 \quad \left[\left(\frac{L}{C_k}\right)^3 = \text{constant}\right]$$

$$K = \frac{c}{t_d^4}$$

Project Estimation Techniques

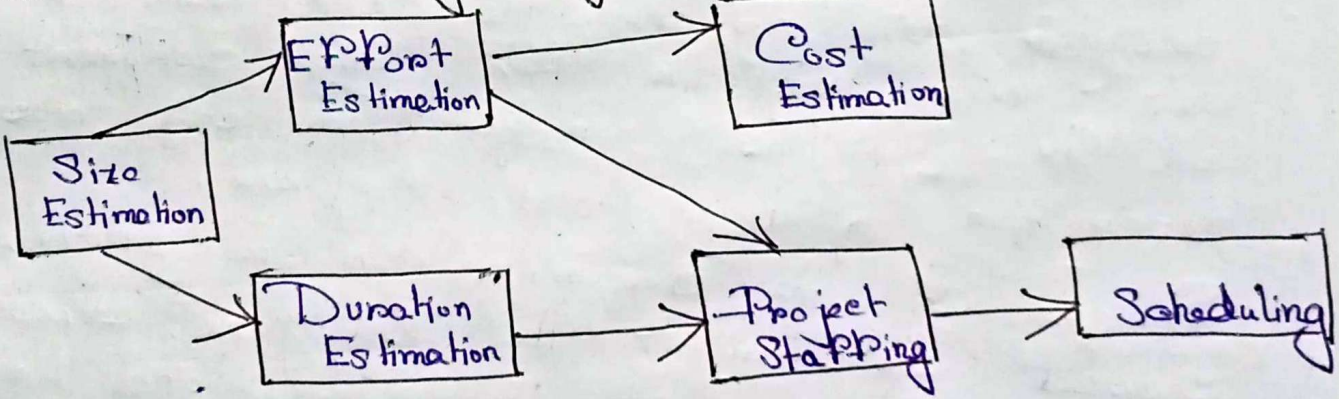
Estimation of various projects parameters is an important project planning activity. The different parameters of project that need to be estimated include —

- project size
- Effort required to complete the project
- Project Duration
- Cost.

- Empirical Estimation Techniques → Expert judgement tech (Alone)
  → Delphi Cost Estimation (team of expert)
- Heuristical Estimation Technique → COCOMO
- Analytical Estimation Technique.

Task of Project manager → Project Planning
→ Project Monitoring
→ Control Activities

Precedence ordering among Project Planning Activities



Software Documentation

⇒ Software documentation is an important part of software process. A well written document provides a great tool and means of information repository necessary to know about software process.

⇒ A well maintained documentation should involve the following documents.

- Requirement documentation.
- Software Design documentation.
- Technical Documentation.
- User Documentation.

Requirement documentation:

i) This documentation works as key tool for software designer, developer and the test team to carry out their respective task. This document contains all the functional, non-functional and behavioral description of the intended software.

ii) Source of the document can be previously stored data about the software, already running software at the client's end, client's interviews, questionnaries and research. Generally it is stored in the form of spreadsheet or word processing document with the high-end software management team.

iii) This documentation works as foundation for the software to be developed and is majorly used in verification and validation phases. Most test-cased are built directly from requirement documentation

User Documentation: All other previous documentations are maintained to provide inform^n about the software and it development process. But user documentation explains how the software product should work and how it should be used to get the desired result.

Software Design Documentation

These documentations contain all the necessary information, which are needed to build the software.

It contains:

(a) High-Level software architecture.
(b) Software design details.
(c) Data-Flow Diagram.
(d) Database Design.

These documents work as repository for developers to implement the software. Though these does do not give any details on how to code the program, they give necessary information that is req for coding and implementation.

Technical Documentation:

i) These documentation are maintained by the developers and actual coders. They does as a whole represent information. While writing the code the programmers also mention objective of the code, why he wrote it, where will it be required, what it does and how it does, what other resources the code used etc.

ii) The technical documentation increases the understanding betn various programmers working on the same code. It enhances re-use capability of the code. It makes debugging easy and traceable.

iii) There are various automated tools available and some comes with the programming language itself. Eg. Java comes JavaDoc tool to generate technical documentation of code.

User Documentation:

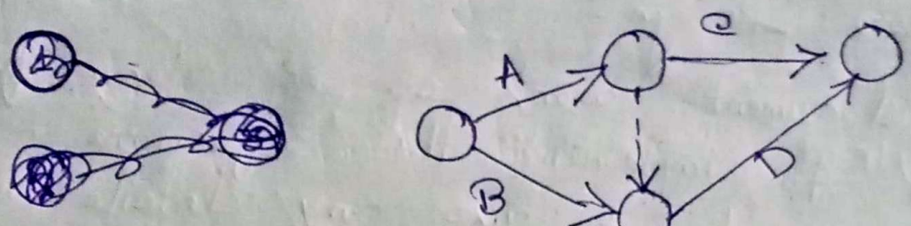
These documentation may include software installation procedures, user guidelines, uninstallation method and special references to get more information like licence updation etc.

CPM And PERT

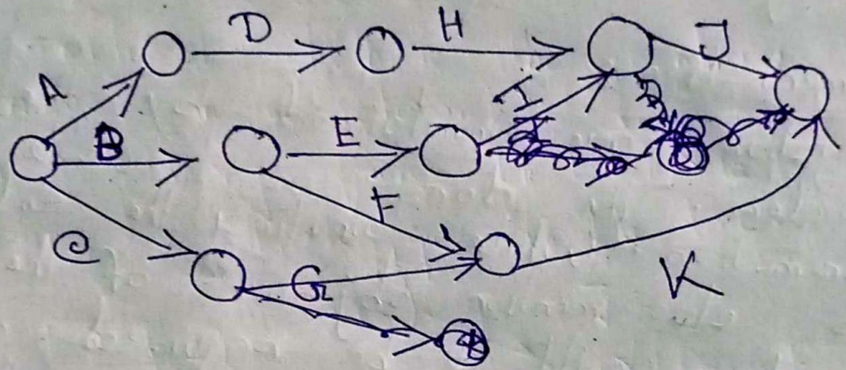
Activity on Arrow

Draw Network Diagram.

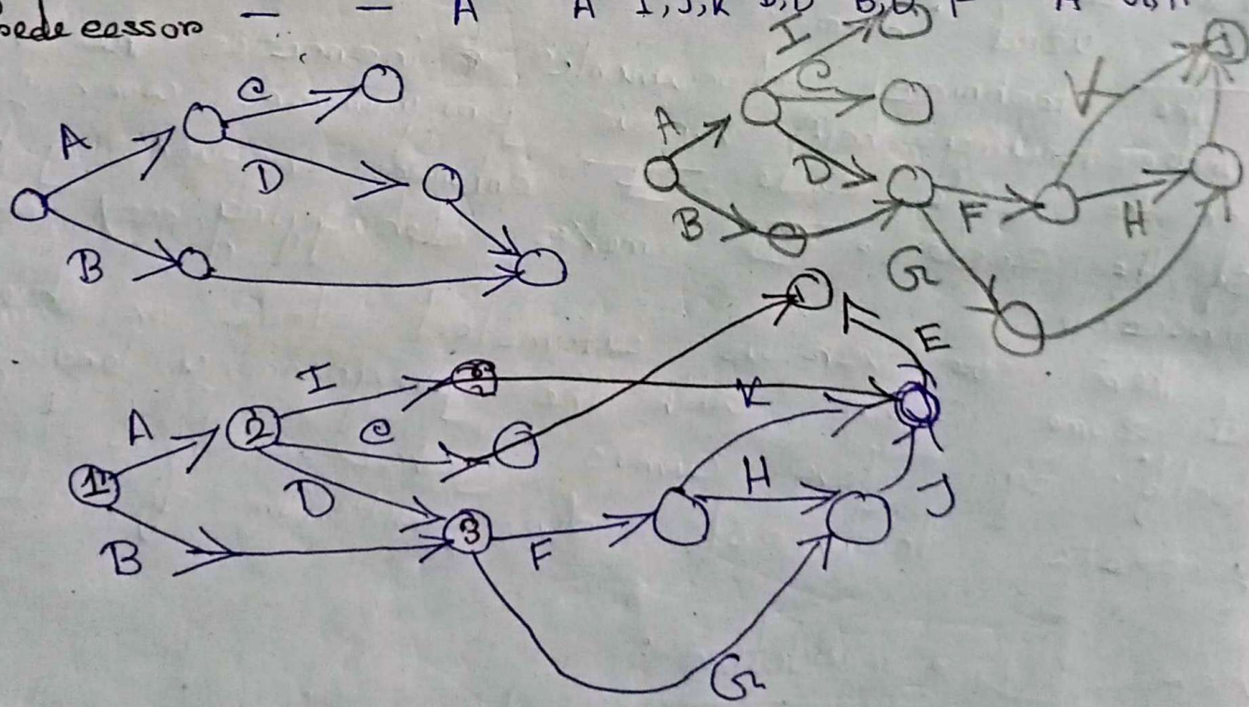
Activity	Immediate Predecessor
A	—
B	—
C	A
D	A, B



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

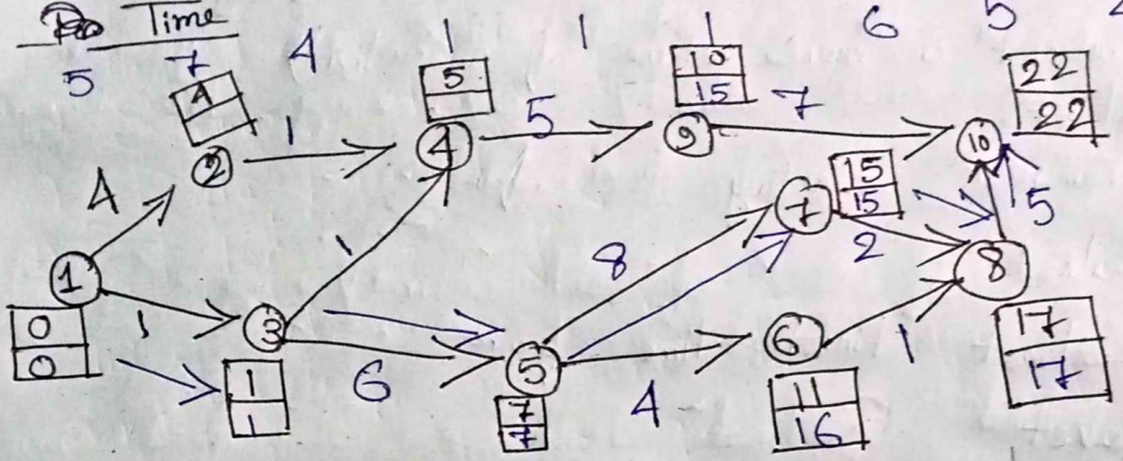


Activity	A	B	C	D	E	F	G	H	I	J	K
Predecessor	—	—	A	A	I, J, K	B, D	B, D	F	A	G, H	F



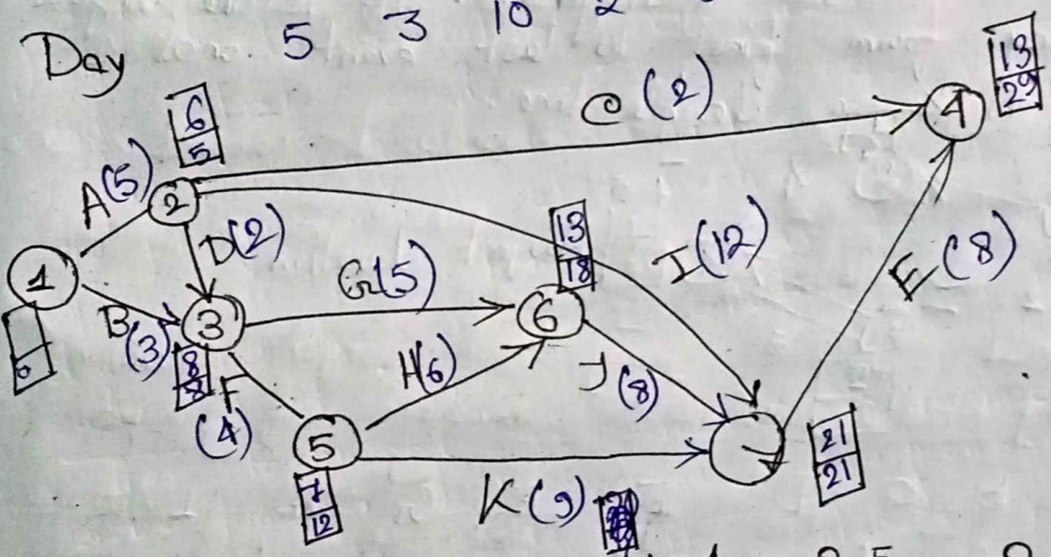
Activity on Node

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	5	7	4	1	1	1	6	5	4	8	1	2

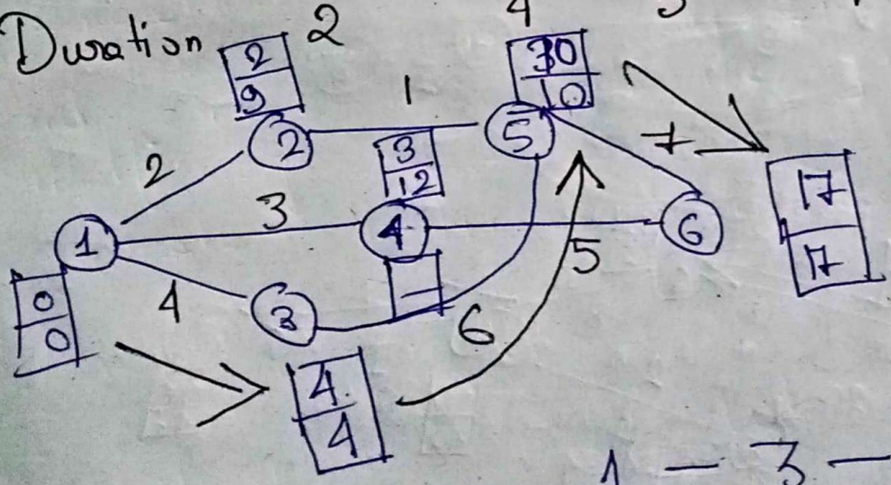


1-3-5-7-8-10 = 22

Activity	A	B	C	D	E	F	G	H	I	J	K
Precedence	—	—	A	A	I,J,K	B,D	B,D	F	A	G,H	F
Day	5	3	10	2	8	4	5	6	12	8	9



Activity	1-2	1-3	1-4	2-5	3-5	4-6	5-6
Duration	2	4	3	1	6	5	7



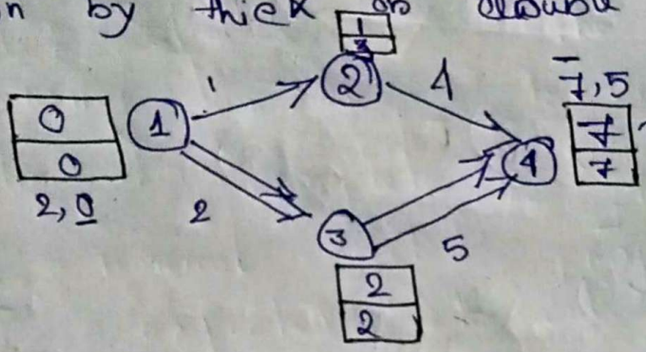
1 — 3 — 5 — 6

CPM

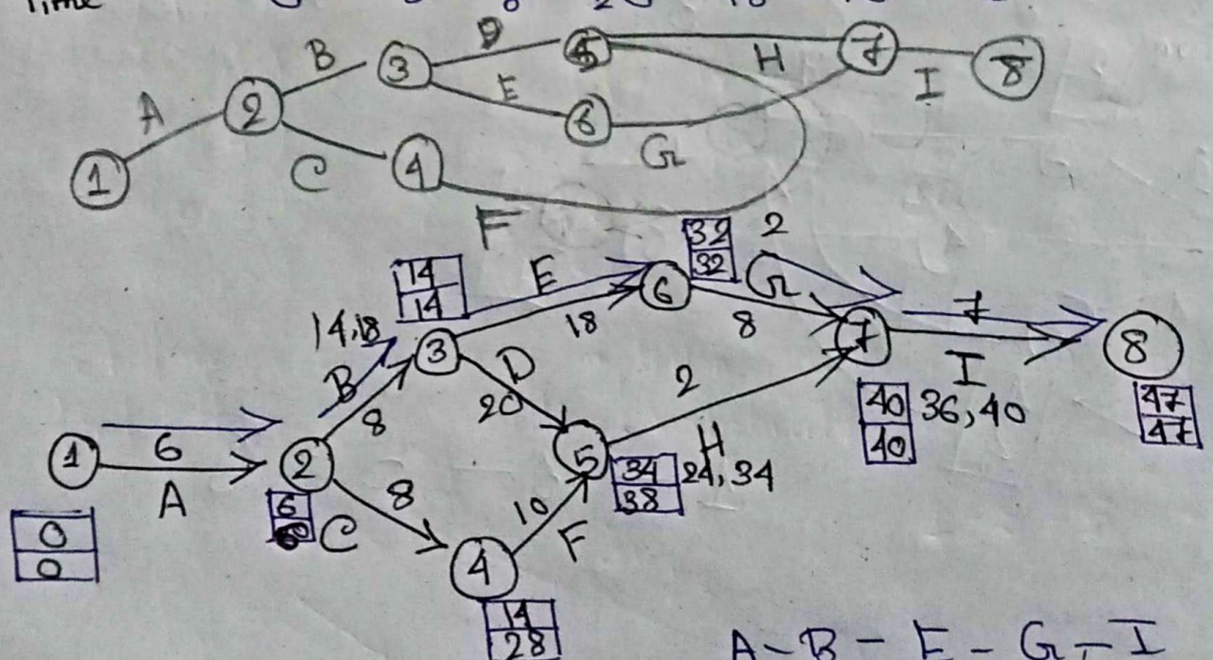
The objective of Critical Path analysis is to estimate the total project duration and to assign starting and finishing time to all activity involved in the project.

E_i = Earliest occurence time of an activity.
L_j = Latest " " " " "
ES_{ij} = Earliest starting " for " "
LS_{ij} = Latest " " " " "
EF_{ij} = Earliest Finish time for " "
LF_{ij} = Latest " " " " "
t_{ij} = Duration of an activity.

The critical Path is the continuous chain of critical activities in a network diagram. IT is the longest Path starting from first to last event and is shown by thick or double line.



Activity	A	B	C	D	E	F	G	H	I
Predecessor	–	A	A	B	B	C	E	D, F	G, H
Time	6	8	8	20	18	10	8	2	7



A – B – E – G – I

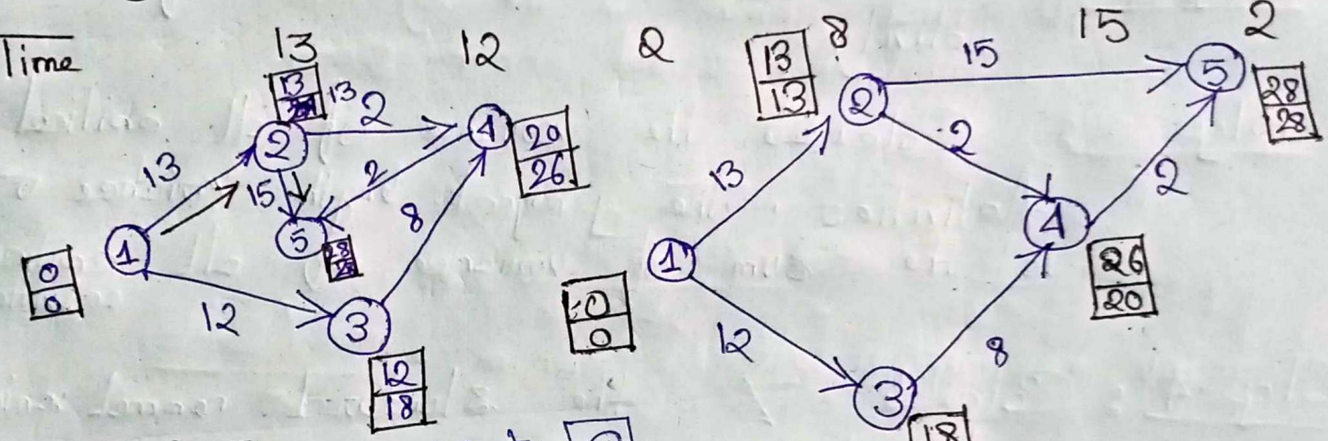
1 – 2 – 3 – 6 – 7 – 8 = 47

Float of an Activity and Event:

The float is the length of time to which a non-critical activity or an event can be delayed or extended without delaying the total project completion time.

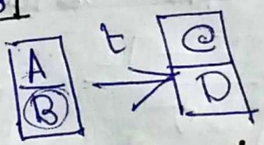
$$\text{Total Float } (TF_{ij}) = (L_j - E_i) - t_{ij}$$
$$= LS_{ij} - ES_{ij}$$
$$= LF_{ij} - EF_{ij}$$

Activity	1-2	1-3	2-4	3-4	2-5	4-5
Time	13	12	2	8	15	2

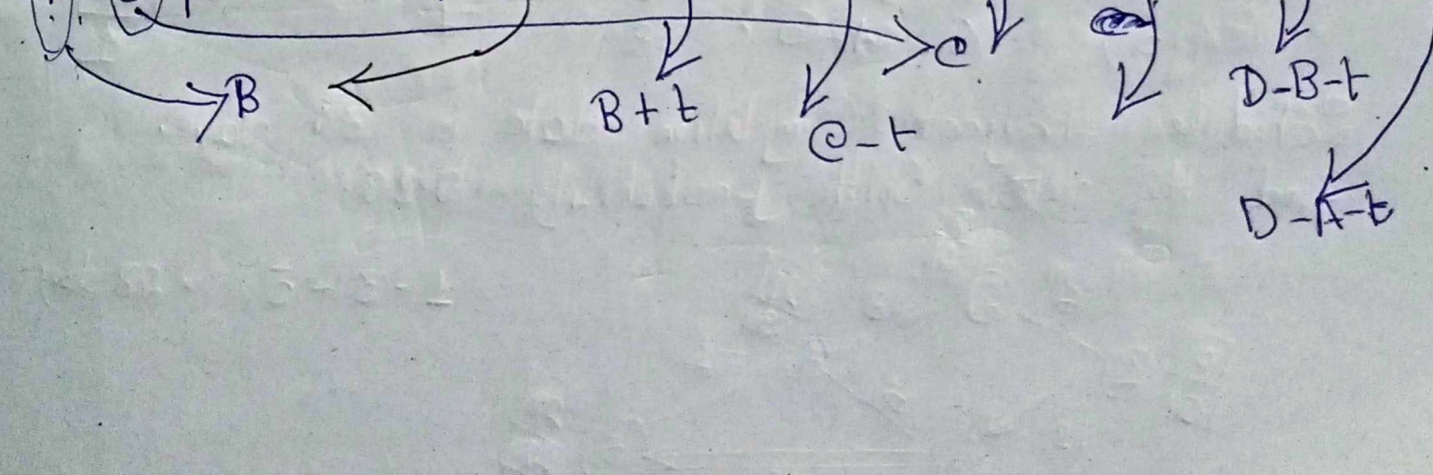


1-2-5

T = 28



Activity (i, j)	Normal times t_{ij}	Earliest Time Start E_i	Earliest Time Finish $E_i + t_{ij}$	latest time Start $L_j - t_{ij}$	latest time Finish L_j	Total Float $L_i - E_i - t_{ij}$	E_j	Free Float $E_j - E_i - t_{ij}$	L_i	Independent Float $E_j - L_i - t_{ij}$
1-2	13	0	13	0	13	0	13	0	0	0
1-3	12	0	12	6	18	6	12	0	0	0
2-4	2	13	15	24	26	11	20	5	13	5
3-4	8	12	20	18	26	6	20	0	18	6=0
2-5	15	13	28	13	28	0	28	0	13	0
4-5	2	20	22	26	28	6	28	6	26	0



optimistic time estimation - t_o Most likely time estimate = t_m
Pessimistic time estimate = t_p. Expected time of activity = t_e
variance of the activity = σ^2.

Step: Draw the project network and expected duration of each activity by using the formula

$$t_e = \frac{1}{6}(t_o + 4t_m + t_p)$$

and variance of σ^2 of each activity using $\sigma^2 = \left[\frac{t_p - t_o}{6}\right]^2$

Step-2: Find Critical Path and also find Critical Activity.

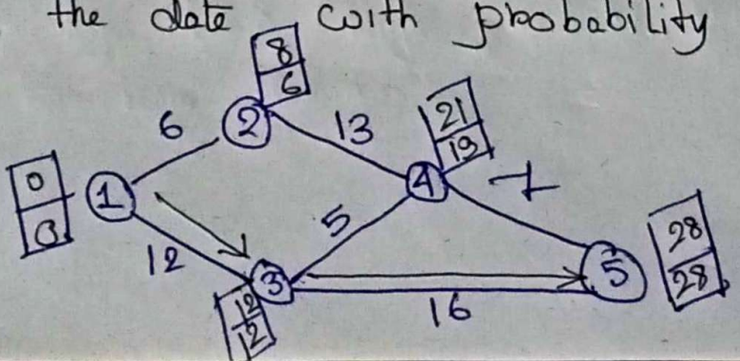
Step-3: Calculate the variance of all critical Activities. The project length variance σ^2 is the sum of variance of all critical activities

Step-4: Calculate Z, the standard normal variable using formula $Z = \frac{T_s - T_e}{\sigma}$ where T_s is scheduled time to complete the project.

T_e = normal expected projected length duration.
σ = expected standard deviation of project length

Activity	t_o	t_m	t_p	Expected Time: $1/6(t_o + 4t_m + t_p)$	Variance $\left(\frac{t_p - t_o}{6}\right)^2$
(1,2)	2	$20 = 5 \times 4$	14	6	4
(1,3)	9	$48 = 12 \times$	15	12	1
(2,4)	5	$56 = 14 \times$	17	13	4
(3,4)	2	$20 = 5 \times$	8	5	1
(3,5)	8	$68 = 17 \times$	20	16	4
(4,5)	6	$24 = 6 \times$	12	7	1

Schedule Project Completion date is 30 days.
find the date with probability 0.90.



1 - 3 - 5 = 28 day

Project duration = 28 day.

variance ~~and~~ of critical path $\sigma_{1-3}^2 + \sigma_{3-5}^2 = 1 + 4$

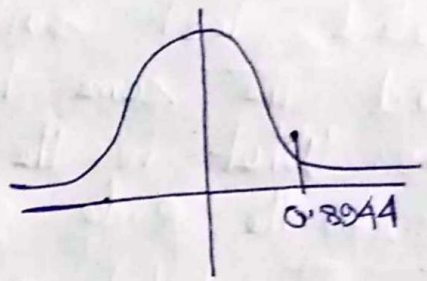
$\sigma^2 = 5$

$\sigma = \sqrt{5}$

$$Z = \frac{X - \mu}{\sigma}$$

$$= \frac{X - 28}{\sqrt{5}}$$

$$= \frac{30 - 28}{\sqrt{5}} = 0.8944$$



$P(Z < 0.8944)$

$= 0.5 + P(0 < Z < 0.8944)$

$= 0.5 + 0.3133$

$= 0.8133$

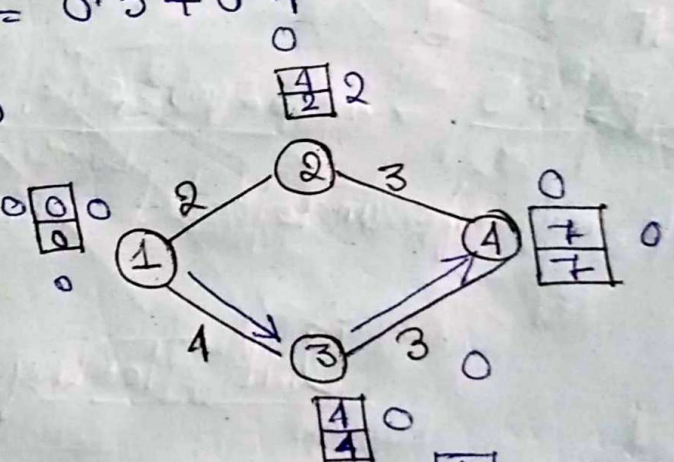
(ii) $P(Z < Z_1) = 0.9 = 0.5 + 0.4$

$P(Z < 1.29) = 0.9$

$$\frac{X - 28}{\sqrt{5}} = 1.29$$

$X = 30.88$

$= 31$ days.



Total Float = difference between $\frac{A}{B}$ A-B

Free Float = $\frac{A}{B}$ 1 ~~2~~ t $\frac{4}{4}$ 3 A-C-~~B~~t

Independent Float $\frac{A}{B}$ ~~B~~ t $\frac{C}{D}$ D-A-~~B~~t

Testing

A software was ~~seeded~~ tested using the errors seeding strategy in which 20 errors were seeded in the code. When the code was tested using the completed test suite, 16 of the seeded errors were detected. The same test suite also detected 200 non-seeded errors. What is the estimated no. of undetected ~~code~~ error in the code after this testing?

S = total errors seeded in code. = 20

s = total seeded errors detected during testing
= 16

N = total errors in the system

n = total non-seeded errors = 200

$$\frac{n}{N} = \frac{s}{S}$$

$$N = \frac{nS}{s} = \frac{200\ (20-16)}{16} = 15$$

total no. of undetected error $= \frac{n \times (S-B)}{s}$

Unit Testing

Unit

Integration Testing

Integration test is conducted to evaluate the compliance of a system or component with specified function requirements. It is a white box test.

- It occurs after unit-testing and before system testing.
- Type —
  - Big-bang
  - Mixed (Sandwich)
  - Top-down (High priority function will be tested first)
  - Bottom down.

System Testing

- System Test 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 specification.
- System Testing is a black-box testing.
- System testing categories based on : Who is Doing the testing? α, β,
- System testing categories based on : Functional / Non-functional Requirements?
- performed by a testing team that is independent of the development team that helps to check quality of the system impartial.

Type

- Performance Testing : type of s/w testing that is carried out to test the speed, scalability, stability and reliability of s/w product of applica.
- Load Testing : to determine the behaviours of s/w product under extreme load. Like Max Capacity of a Web Server.
- Stress Testing : To check the robustness of system under varying load.
- Scalability Testing : to check the performance of a s/w system in terms of its capability to scale up or scale down the no. of users request load.

Black-Box Testing

Boundary Value Testing : It is helpful for detecting any errors or threats that happened at the boundary values of valid or invalid partitions rathers than focusing on the center of the input data.

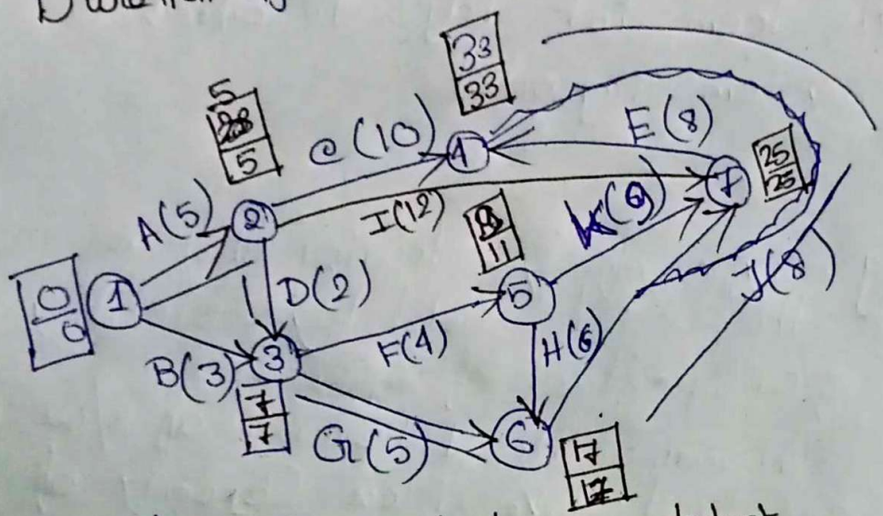
Test Case 1

Let's assume test case that takes the speed of a car from 40 to 80 to get the best fuel efficiency.

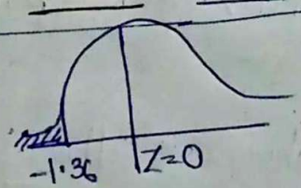
Boundary Value Test Case

Invalid Test Case	Valid Test Case	Invalid Test Case
(Min Value − 1)	(Min, +Min, Max, −Max)	(Max + 1)
39	40, 41, 80, 79	81

Activity	A	B	C	D	E	F	G	H	I	J	K
Precedence	—	—	A	A	I,J,K	B,D	B,D	F	A	G,H	F
Duration tij	5	3	10	2	8	4	5	6	12	8	9



	tij	Earliest Start	Earliest Finish	latest start	latest Finish	Total Float	Free Float	Indep Float
A	5	0				latest start − Earliest Start	4	4
B	3	0					18	18
C	10	5					0	0
D	2	5						
E	8	25						
F	4	7						
G	5	7						
H	6	11						
I	12	5						
J	8	17						
K	9	11						



$$Z = \frac{X - \mu}{\sigma}$$

$\mu = 61.5 \quad \sigma = 3.3$

$$Z = \frac{X - 61.5}{3.3}$$

$$P(X < 60) = P\left(z < \frac{60 - 61.5}{3.3}\right)$$

$$P(Z < -1.36)$$

$$= 0.5 - P(-1.36 < Z < 0)$$

$$= 0.5 - 0.4131$$

$$= 0.0869$$