Software Engineering Sessional-I

Chapter: Introduction to Software Engineering

What is software?

=) Software is a collection of programs, procedures, instruction that perform some task on a computer.

What is Engineering?

=) Engineering is all about developing product using well defined scientific principles & methods.

What is Software Engineering?

=) Software engineering is an Engineering Branch whose aim is the production of fault free software that satisfies the user needs and i.e. delivered on time and within budget.

Or

Software Engineering is a systematic approach of developing a software i.e. step by step analysis and development of a software.

Diff. b/w Software (Product) & Program

Software (Product)	Program
1) Product / Software is developed by a group of SW engineers working in a team.	1) Programs are developed by individuals, mostly for their own use.
2) S/W / product are too large in size and consists of many kilolines of code.	2) A program is small in size & have a limited functionality

3) It consists of program, code as well as many associated documents and manuals.	3) A program only have the coding statements & no user manuals.

SDLC (Software Development Life Cycle)

SDLC Phases

1) Requirement Gathering & Analysis:

During this phase all the relevant information is collated from the customer to develop a product as per their expectation. Any ambiguities must be solved in this phase only.

Business analyst and Project manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the Requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

2) Design:
In this phase the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation & Coding:

Implementation / coding starts once the developers gets the design document. The software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing:
After the product is tested, it is deploy
Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

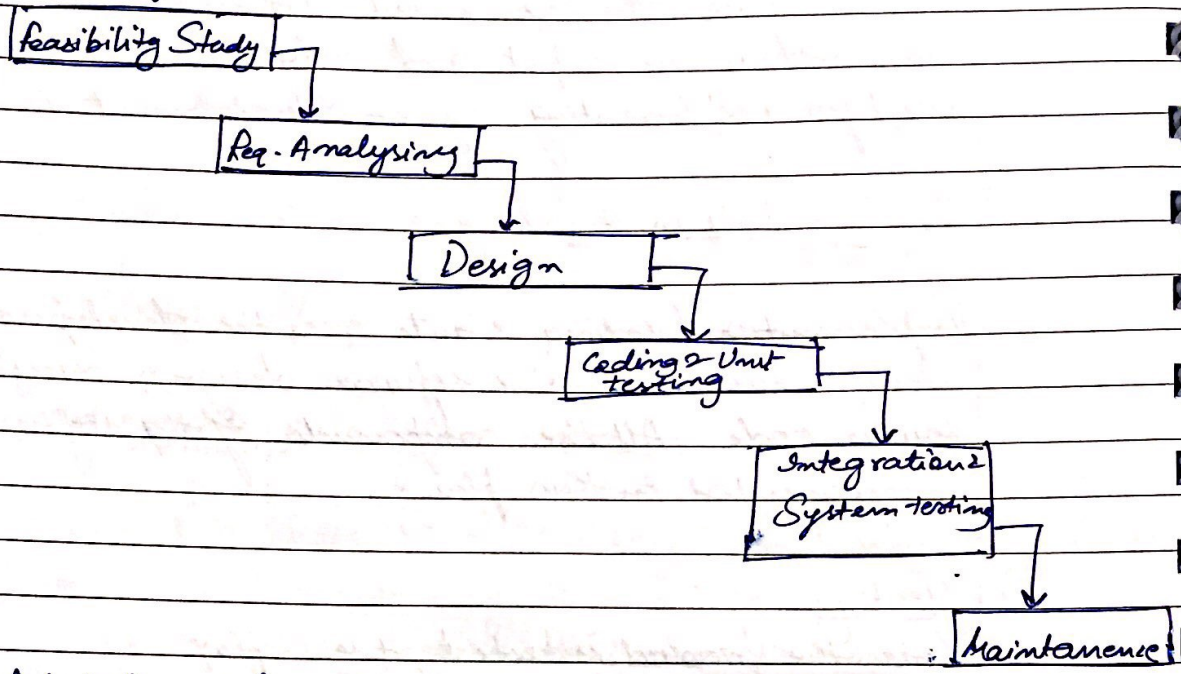
5) Maintenance
After the product has been tested by user and developers if any issue comes up and needs to be fixed or any enhancement to be done is taken care by the developers.

SDLC Models

1) Water fall Model / Classical waterfall model.
2) Iterative waterfall model
3) Prototyping Model
4) Evolutional model / Increment Model / Successive version model.
5) Spiral Model.

__/__/____

1) Water fall Model



Waterfall model is the basic model and other models are directly or indirectly based on this model. The waterfall is a conventional, linear, sequential a traditional waterfall S/W life cicle model.

It is a Sequential development approach in which development is seen as flowing downwards through the phases.

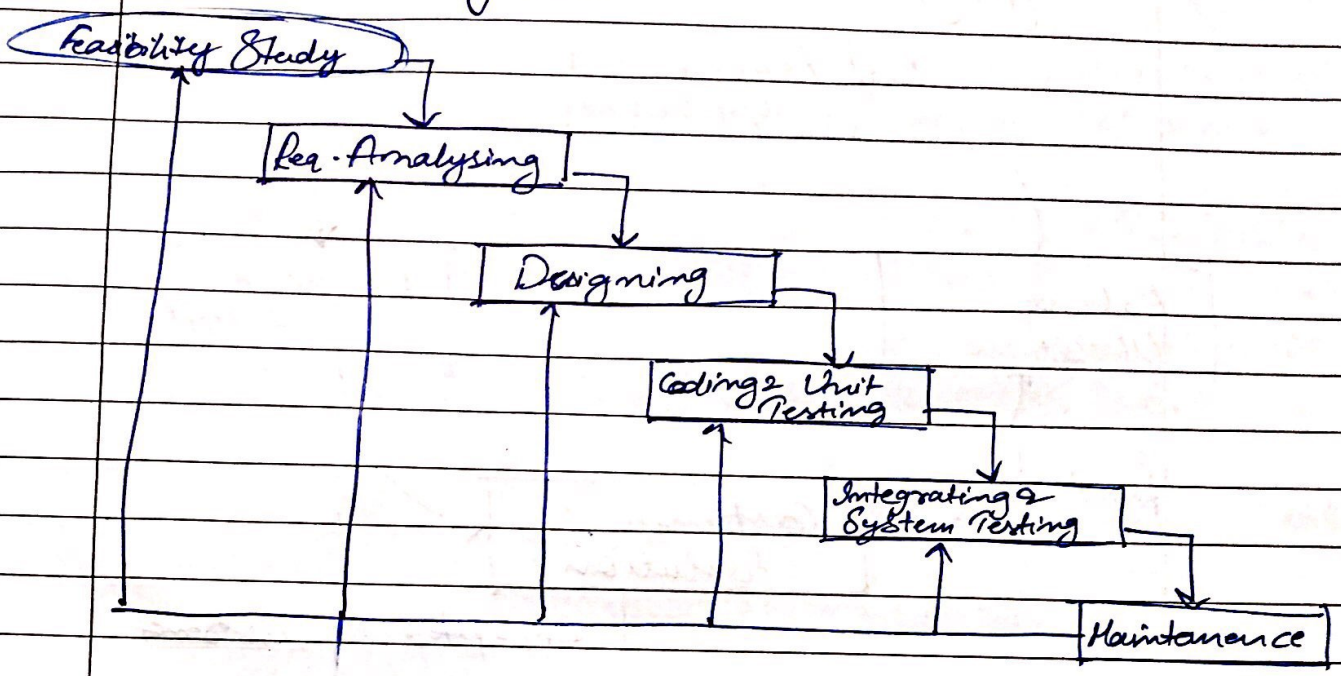
Advantage

Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.

Disadvantage

Back Tracking is not possible in this model.

2) Iterative Waterfall Model



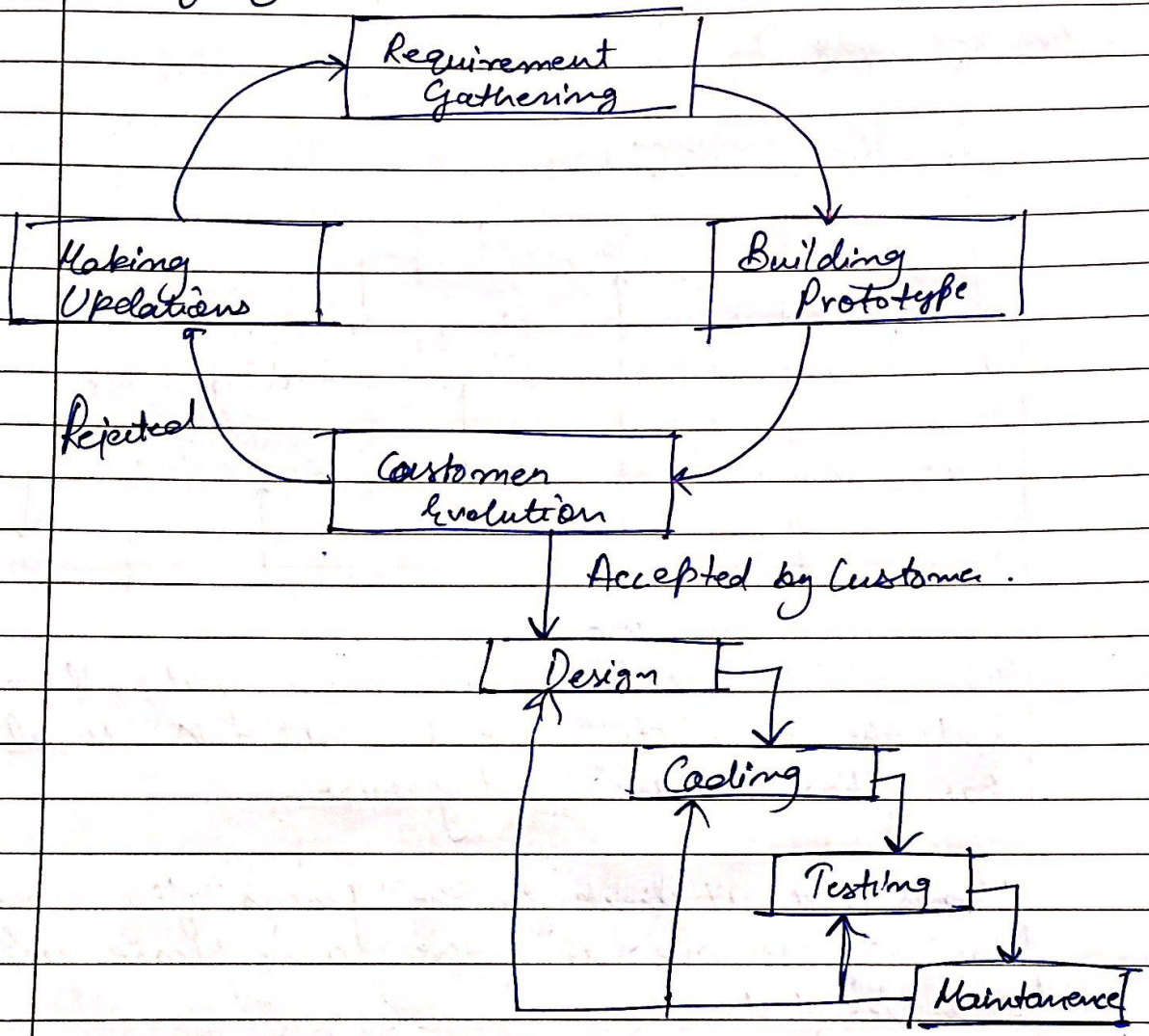
This model is very similar to classical waterfall model but the main difference is that this model has phase containment of errors.

Means that it check for the errors after every phase and remove it in the same phase as soon as it is detected.

Advantage

Back Tracking is possible in this model.

3) Prototyping Model



This model aims at developing a working prototype (rough model or toy model) from the requirements defined by the user.

Advantages

1) This type of model is beneficial for a new developers to avoid the technical risk.
2) It is impossible to get the right product at the very first attempt. So the prototype model allows

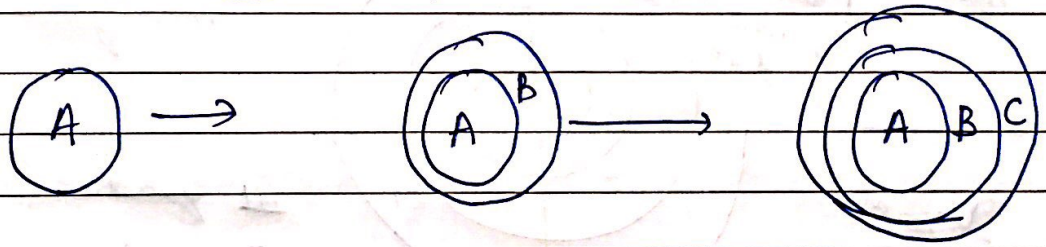
to make updation on prototype instead of throwing away the original software.

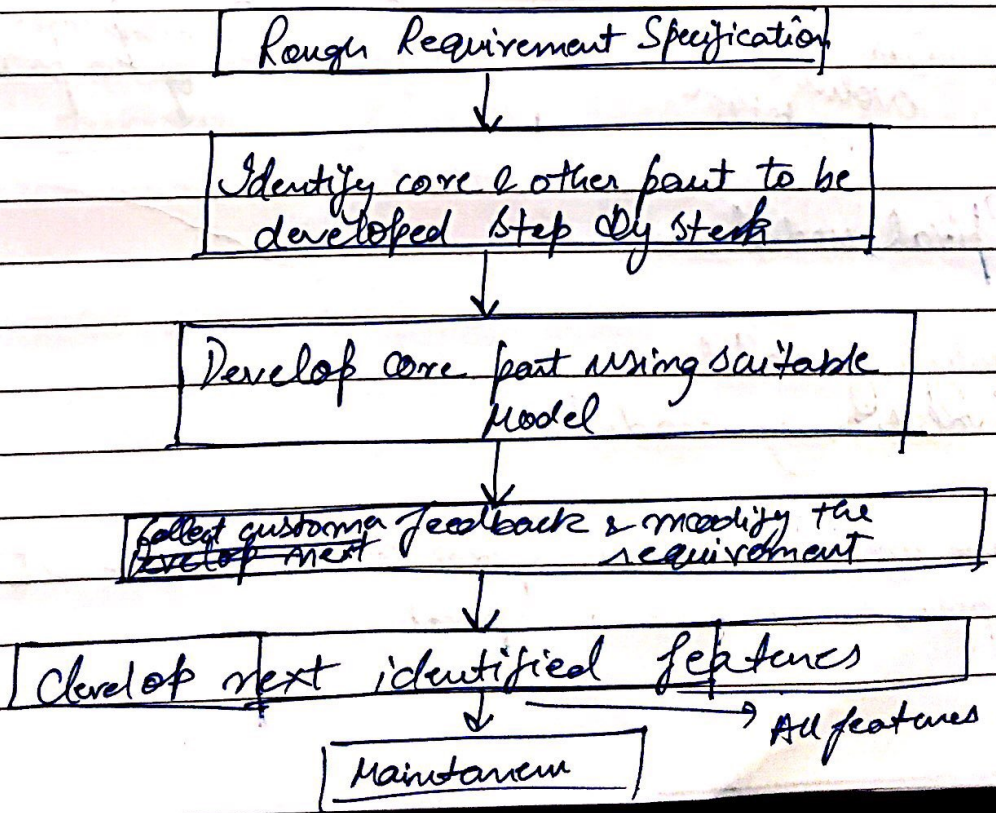
3) It helps the developers to gain experience in making particular application & then using this experience in making actual system.

4) It reduces the overall cost incurred in developing software. Although some cost is involved while making prototype but it is far lesser than making the wrong product which does not match customer needs.

5) The continuous evaluation & checking from the customer reduces the ~~risk~~ risk of change in the requirements of the customer. Thus there are less chances of customer ~~dissa~~ dissatisfaction.

4) Evolutionaly Model / Successive Version Model / Incremental Model

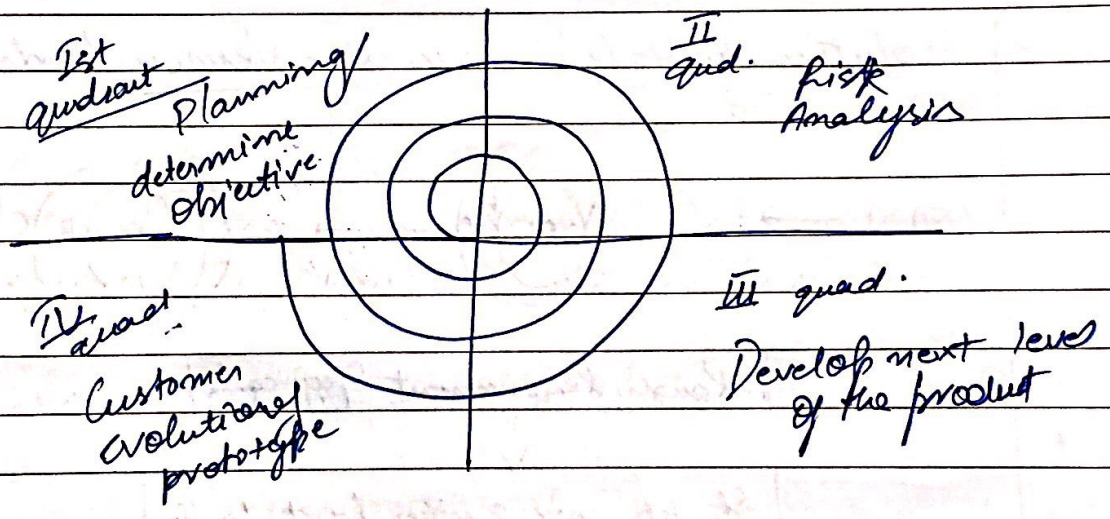




The evolutionaly model aims at fist developing the core features of a software and then adding new features to the existing software with time.
This model is also known as successive model / incremental model.

In evolutionary model only the core part is developed first because the users needs changes with time so the developer can use the latest technology to develop the software to meet the aspirations of the software.

5) Spiral Model / Meta Model



Spiral model use:-

1) Prototping model
2) Evolutionary model

(i) This model is called Spiral model because its structure appears to be like a spiral.

(ii) Each loop is called a phase of spiral model. Eg. innermost loop might be concerned with feasibility study. Next loop with requirement analysis & specification. The next one with design and so on.

(iii) It is also called as meta model because it can use many life cycle models discussed earlier for software development.

(iv) Each phase in this model is split into four sectors or quadrants as shown in figure.

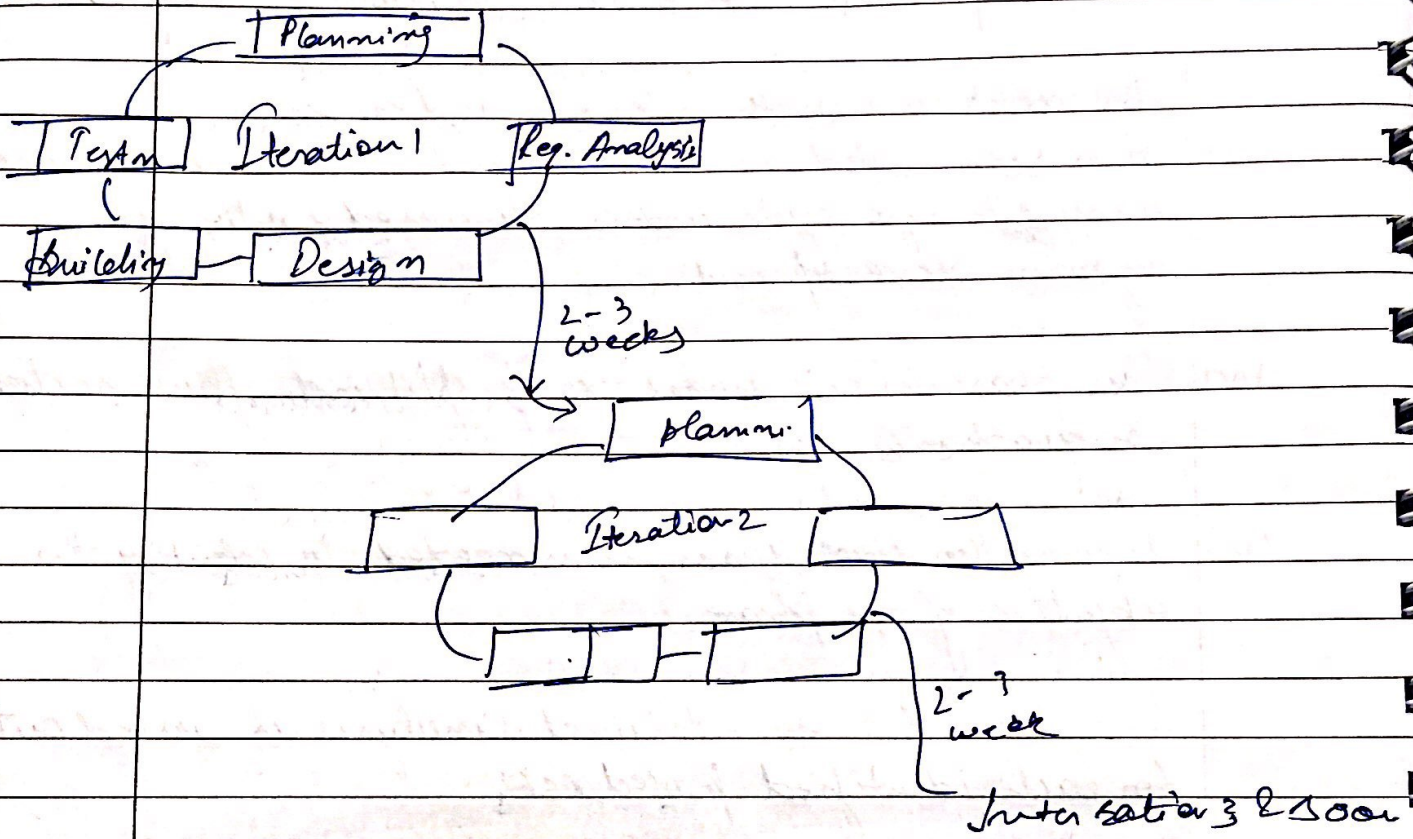
(v) During the first quad, it is needed to identify the objectives of the phase.

Second quad, a detailed analysis is carried out for each identified project risk.

Third quadrant, develop & validate the next level of the product after resolving the identified risk.

Fourth quadrant, review the results achieved so far with the customer & plan the next iterations around the spiral.

Agile SDLC Model



1) In Software development, the term 'Agile' means the ability to respond to changes.
2) It is iterative & incremental process.
3) Direct collaboration with the customer
4) Each Iteration lasts from 2-3 weeks.

Comparison of different life cycle models

1) The classical waterfall model can be considered as the basic model and all the other life cycle models are made of this model. However, this classical model cannot be used in practical development projects.
Since this model supports not mechanism to handle the errors committed during any of the phases.

2) Iterative waterfall model :- The above problem is overcome in this model. The iterative waterfall model is probably the most widely used software development model involved so far. This model is simple to understand & use. But this model is not suitable for very large projects & for projects that are subject to many risks.

3) Prototyping model is suitable for projects for which either the users requirements or the underline technical aspects are not well understood.

4) Evolutionaly model approach is suitable for large problems which can be decomposed into a set of modules for incremental development & delivery.

5) Spiral model is called the meta model. Since, it incompasses all other life cycle models.
Risk handling is inperibly build into this model.
Spiral model is suitable for development of technically challenging software products that are prone to several kinds of risk.

Chapter — Requirement Analysis & Specification

- Requirement Gathering & Analysis
- Requirement Specification

Requirement Gathering & Analysis

1) Requirement Gathering

Requirement Gathering is the first step before implementing or starting any project.

So Requirement Gathering has we have study or discuss the following:-

(i) Observation of existing software
(ii) Studying existing procedure
(iii) Discussion with the customer.

2) Requirement Analysis

After Requirement Gathering, analyze it

(i) clearly understand the user requirements.
(ii) Detect Ambiguities, incompleteness and inconsistencies.

(i) Ambiguity:- Ambiguity means having more than one meaning.

Eg. Was the cake good or bad.

(ii) Incompleteness :- When information is missed out.

Eg. the analyst has not recorded :- When temp. falls below 90°F heater should be turned on & water showers are off.

(iii) Inconsistency :- Some part of the requirement contradicts with some other parts.

Requirement Specification (SRS Document)

SRS is a sort of an official agreement b/w the user and developer. The SRS document is useful in various contexts.

1) Statement of user needs
2) Contract document
3) Reference document

All the final needs of the user are organized in the form of SRS document.

Users of SRS document

1) Customer
2) Developer
3) Project Manager.

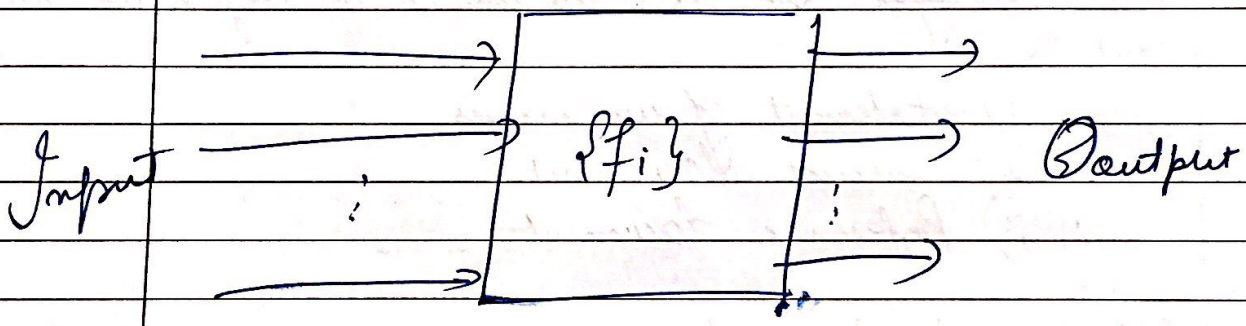
PTO

Contents of the SRS document

1) Functional Requirement
2) Non-Functional Requirement
3) Goal of Implementation.

Function requirement

It specifies something the system should do. It describes the functions a software should perform. A function is nothing but inputs its behaviours and outputs.



Non-functional requirement

Eg. Consider the case of the library system where.

f_1 :- Search Book fn
Input :- An author's name
Output :- Details of the authors books and the location of these books in the library.

Non-functional Requirement

These types of requirements includes additional functions and parameters like reliability issues, security, efficiency, maintanence ability, portability etc.

Goal of Implementation

This part of the SRS document provides some new ideas and suggestion for the future maintainence of the project.

Characteristics of SRS

1) Coincide ~~Coincide~~ Concise
2) Structured
3) Black Box View: Also know as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is ~~be~~ not known to the tester.

This method is named so because the software program in the eyes of the tester is like a black box inside which one cannot see.
This method attempts to find errors in the following categories:

(i) Incorrect or missing fnc.
(ii) Interface error
(iii) Errors in data structures
(iv) Behaviour or performance errors
(v) Initialization & termination errors

4) Response to undesired events.

Organization of SRS document

1) Introduction
   a) Background
   b) Overall Description

2) Environmental characteristics
   (i) Hardware
   (ii) Peripherals.

3) Goal of Implementation
4) Functional Requirement
5) Non-Functional Requirement.

Chapter:- Software Design

Following Items are designed and documented for the design phase.

1) Different Modules Required } High level Design
2) Control Relationship
3) Interfaces among Modules
4) Data Structure of the Individual module } Detailed Design
5) Algorithms/Techniques to be used.

Characteristics of Good S/w Design

1) Correctness
2) Understandability
3) Maintainability.

Two types of Designs

1) High level Design
2) Detailed Design

Understandability Concept of a Design

Following steps can be taken to increase understandability of a software design:-

1) We can use Layered & Modular structure to develop a software design.

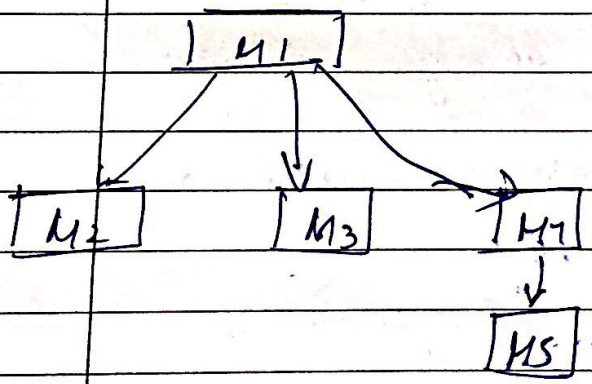
2) We should assign meaningful (unique) names to the different components of a design according to the function performed by them.

3) It should use decomposition & abstraction in order to keep the design simple & less complex.

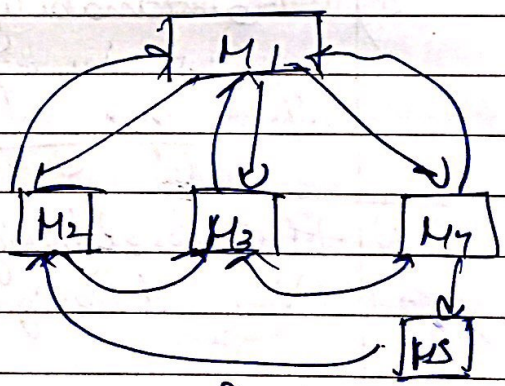
Implementation of Modular & Layered Design

1) **Modularity** :- Modularity uses the concept of decomposition of a problem into various modules.

Decomposing the problem helps to reduce the complexity and to understand each module individually. This is very much similar to divide and conquer rule.



Neat Modularity



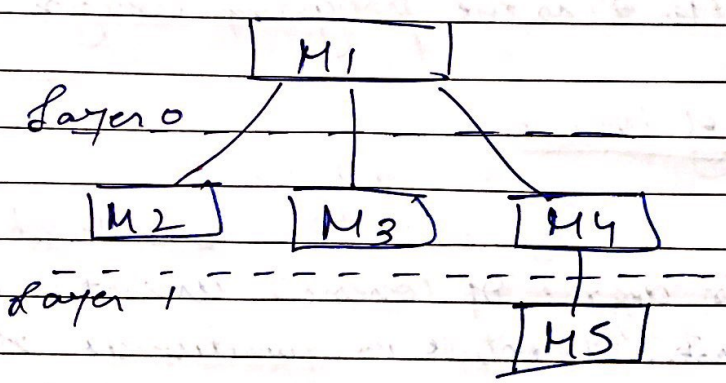
Poor Modularity

2) **Layered Design** :- A layered design is one that implements control abstraction i.e. the lower layer is unaware of the functions performed at higher layer.

A lower layer is not allowed to call a function

from higher layer.

The whole design is organized in such a way that the control need not be transferred from bottom to top.



Layered Design with good Abstraction.

Most researches and engineers agree that a good S/w design implies when decomposition of the problem into modules & the neat arrangements of these modules in the hierarchy.

The primary characteristics of a neat module decomposition are:-

(i) High Cohesion
(ii) Low Coupling

P.T.O

__/__/__

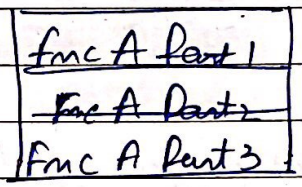
Cohesion — Cohesion is the functional strength of a module i.e. the degree to which different functions of the module cooperate to work for a single objective

Types of Cohesion — Or

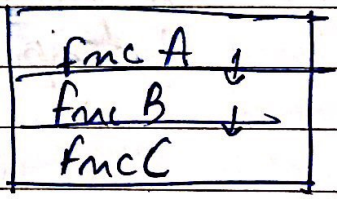
Cohesion is a measure that defines the degree of inter-dependability within elements of a module. The greater the cohesion, the better is the program design.

Types of Cohesion

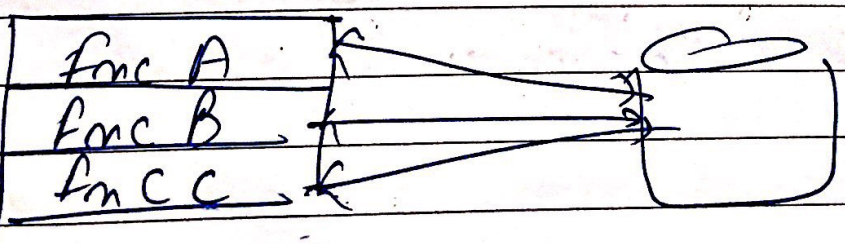
1) **Functional Cohesion** :- It is considered to be the highest degree of cohesion. and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well defined function.



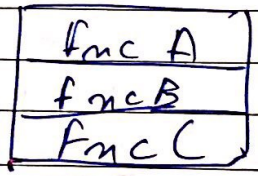
2) **Sequential Cohesion** :- When elements of module are grouped because the output of one element serves as input to another & so on.



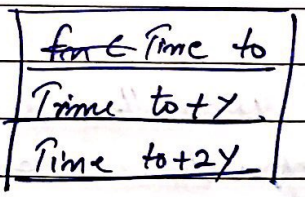
3) **Communicational** :- When elements of module are grouped together, which are executed sequentially and work on same data (information).



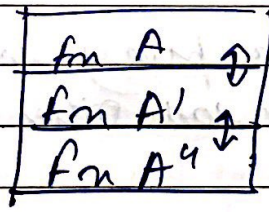
4) Procedural :- When elements are grouped together in order which are executed sequentially to perform a task.



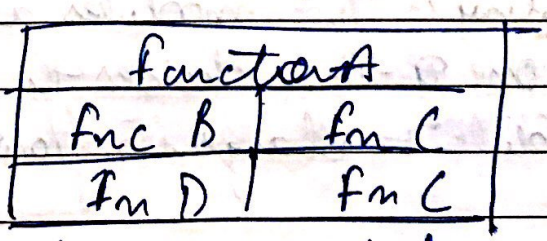
5) Temporal :- When elements of module are grouped organized such that they are processed at a similar point in time.



7) Logical :- When logically categorized elements are put together into a module, it called logical cohesion.

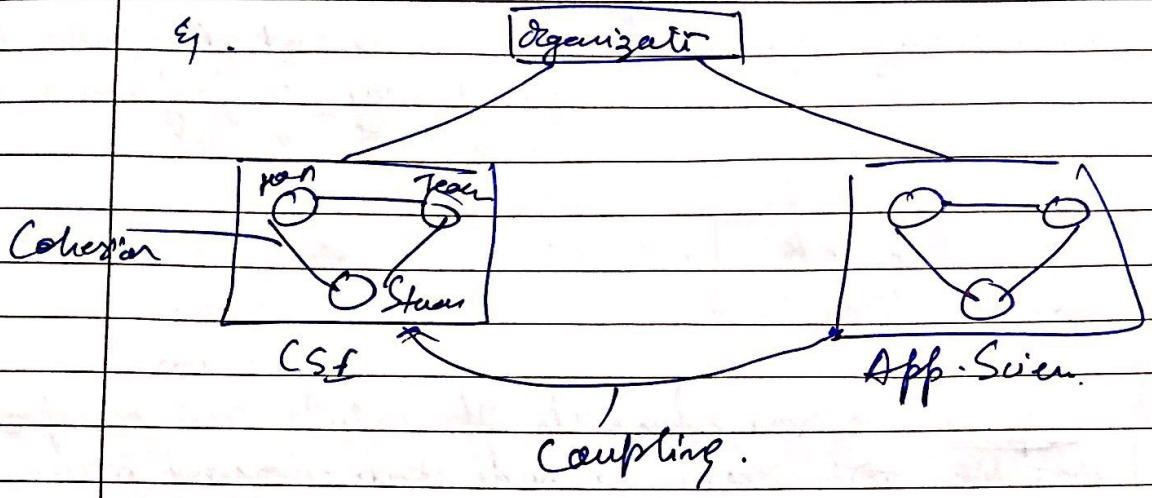


8) Co-Incidental :- It is unplanned & random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularity.



Co-incidental

Eg.



Coupling

Coupling b/w two modules is a measure of the degree of inter-dependence or interaction b/w the two modules.

Types of Coupling.

1) Data Coupling :- Data coupling is when two modules interact with each other by means of passing data.

2) Stamp Coupling :- When multiple modules share common data structure & work on different part of it.

3) Control Coupling :- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

4) ~~Common~~ Stamp Coupling :- When multiple modules have read & write access to some global data.

5) Content coupling:- When a module can directly access or modify or refer to the content of the another module.

Approaches to S/w Design

- Function Oriented Design Approach
- Object Oriented design Approach

Function Oriented	Object oriented.
1) It uses function to execute the logic	1) It uses objects To call the fnc.
2) It is a conventional method	2) It is a modern approach.
3) It is top-down decomposition approach.	3) It is botton-up approach.
4) The data is centralized & shared by all the modules.	4) The system is decentralized as each object has its own copy of data & no global data is shared.
5) It uses structured analysis & design to model the requirements Eg. DFD diagram.	5) It uses object oriented analysis & design to model the requirements. Eg. UML diagram.

Chapter :- Structured Analysis and Design

Structured Analysis (SA)

The aim of structured analysis activity is to transform a textured problem into a graphical model.

During Structured Analysis function decomposition of the system is achieved.

The various principles that are followed during SA are :-

1) Top down decomposition.
2) Each function is decomposed.
3) Graphical representation of the analysis result using DFD's.

Tools of Software Analysis

1) DFD (Data flow diagram)
2) Data Dictionary
3) Decision Trees
4) Decision Tables

Structured Design (SD)

During SD all functions identify during SA are mapped to a modular structure

Modular structure is also known as software architecture

Tools of SD

ii) Structured Chart

DFD

DFD is also known as Bubble chart. It is simple graphical representation that can be used to represent the system in terms of: input data to the system, various processing carried of these systems, output data generated by the System.

Or

★ DFD is a graphical representation of data flow in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data ~~from~~ flows through the system.

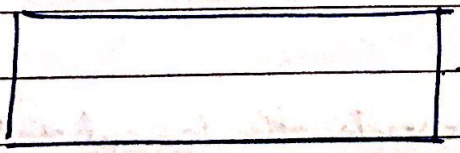
Types of DFD:

(i) Logical DFD:- This type of DFD concentrates on the system process and flow of data in the system.

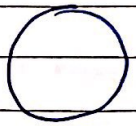
(ii) Physical DFD:- This type of DFD shows how the data flow is actually implemented in the system.

Various Symbols used in DFD

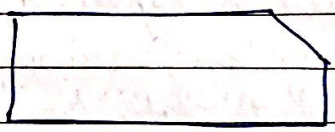
1)

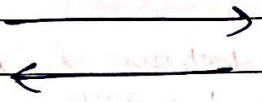


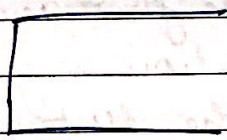
→ Entity

2)  → Process or function Symbol

3)  → Data Store or Database

4)  → Output

5)  → Dataflow

6)  → Data Store or Database

(i) Function Symbol :- Activities and action taken on the data are represented by Circle.

(ii) Entity symbol :- Entities are source and destination of information data such as employee, customer, patient etc. which are represented by rectangle.

(iii) Data flow Symbol :- It is a directed arc or arrow which is used to represent a data flow occuring b/w 2 two processes.

(iv) Data Store Symbol :- It represents the logical file that is shown by two parallel lines.

__/__/__

DFD levels

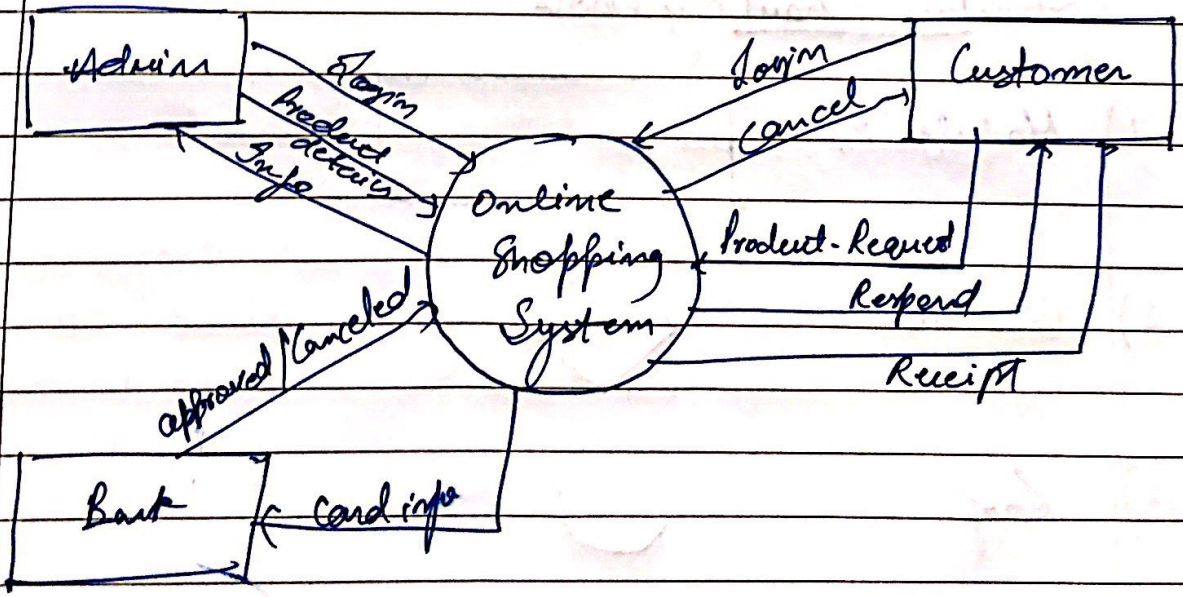
1) Level 0 / Context Digram
2) Level 1
3) Level 2

Context Digram

The context diagram is the most abstract data flow representation of a system. It represents the entire system as a single bubble. The bubble is labelled on to the main function of the system. The data input to the system and the data output to from the system are represented as incoming or outgoing arrows. The context diagram is also known as level 0 DFD.

Eg. DFD of online shopping

level 0 / Context Diagram.



__/__/____

Structure Chart

Structure chart in Software Engineering and organization theory is a chart which shows the break down of a system to its lowest manageable levels.

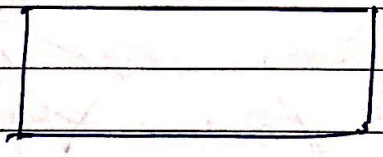
Structure chart is a chart derived from DFDs, it break down the entire System into lowest functional modules, describes functions and sub functions of each module of the system to a greater detail than DFD.

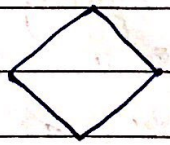
Structure Chart uses:-

(i) Describe functions and sub-functions of each part of system.

(ii) Show relationships b/w common & unique modules of a computer program.

Structure Chart Symbols

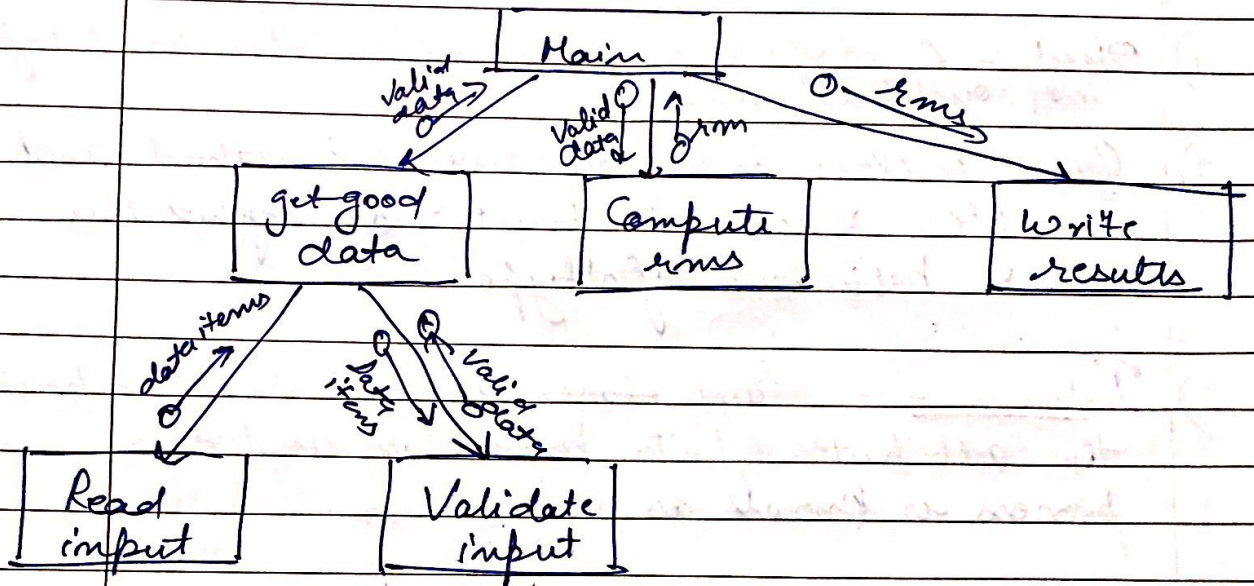
1) Modules :- 

2) Condition :- 

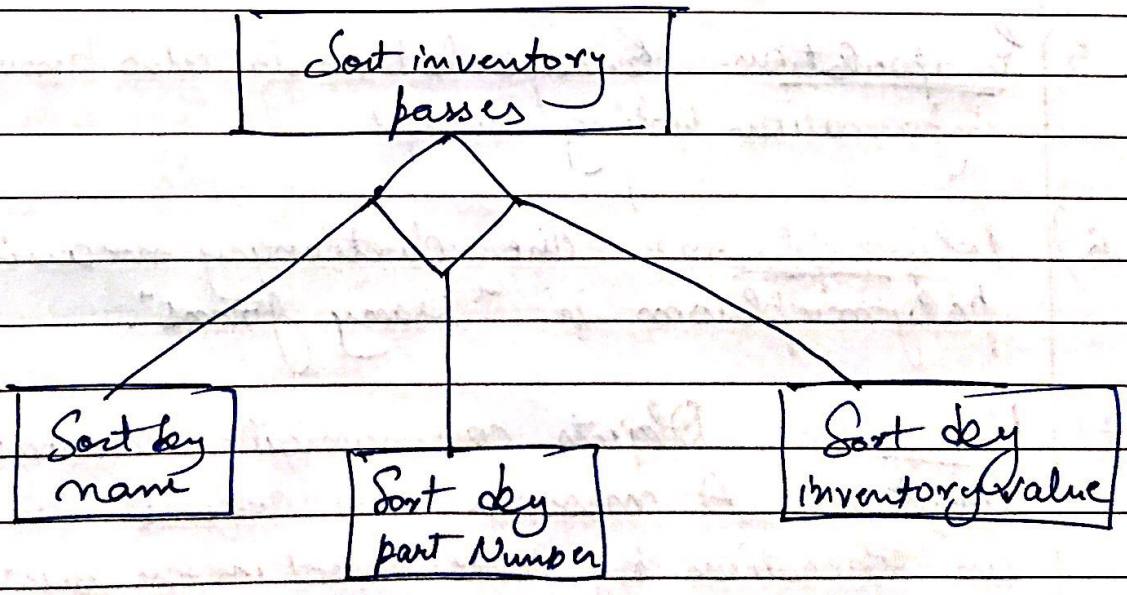
3) Loop :- 

4) Data couple : ○→

Eg. Structure chart for RMS calculating Software.



Eg. Structured chart condition eg.



Chapter :- Object Oriented Analysis & Design.

Object Oriented Concepts

1) Object :- A set of objects with similar behaviour & info. may constitute a class.

2) Class :- A class is a collection of method and variable. It is a blue print that defines the data & behaviours of a type.

3) Inheritance :- Each ~~class~~ derived class inherits the attributes of its base class and this process is known as inheritance.

4) Abstraction :- Data Abstraction is to collect essential elements composing to a compound data.

5) Encapsulation :- Encapsulation is also known as information hiding concept.

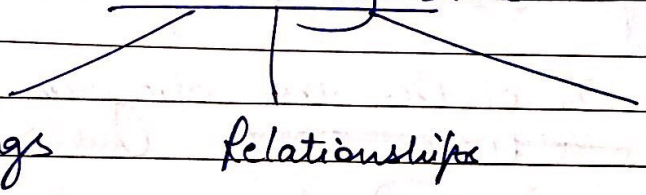
6) Polymorphism :- The dictionary meaning of polymorphism is "many forms".

7) Messages :- Objects communicate through passing messages. A message is a request for performing an operation by some object in the system.

UML (Unified Modeling Language)

1) It is not a programming language.
2) It is a pictorial language used to make software blueprints.

UML Building Block



3) UML is a graphical language for specifying, visualizing, constructing and documenting the artifacts of software system.

UML Diagrams

UML includes the following nine diagrams:-

1) Use case diagram
2) Class diagram
3) Object Diagram
4) Sequence diagram
5) Collaboration diagram
6) Activity diagrams
7) State Chart diagram
8) Deployment Diagram
9) Component diagram.

__/__/____

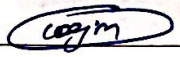
Use Case diagram

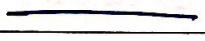
Use case are nothing but scenarios by which, the user can use the System.

Victorically we represent these scenarios by using use case diagrams.

Components of Use case diagram:-

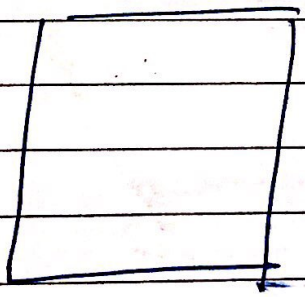
1) Actors 

2) Usecase 

3) Link 

4) Generalization 

5) - - - - - -> Relationship b/w two usecase

6)  -> System Boundary

__/__/__

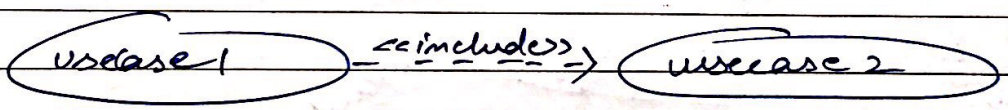
Actors :- Who interacts with the System.
Actor is of two types:-
1) primary actors (left side)
2) secondary actors (right side)

Primary Actors:- Primary actors is one who takes assistance (help) from the System. Eg. Customer is using ATM to withdraw cash.

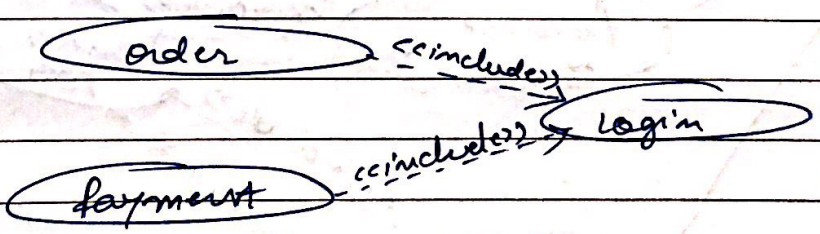
Secondary Actor:- Secondry Actor is one who gives assistance to the system. Eg. Bank is helping ATM by validating the card & pin no.

Relationship used in Use case Diagram

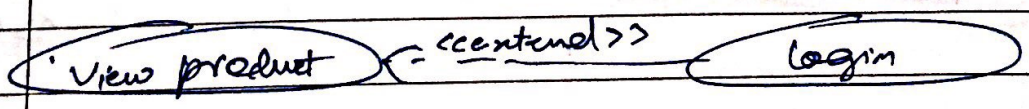
1) Include (Mandatory)



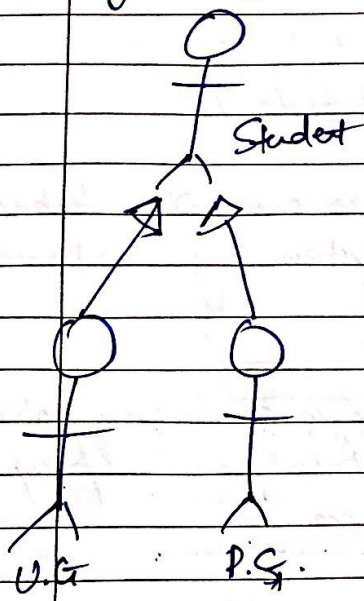
2) ~~Extend~~ Eg.

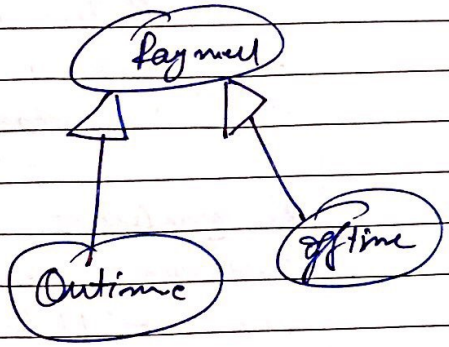


2) Extend (Optional)

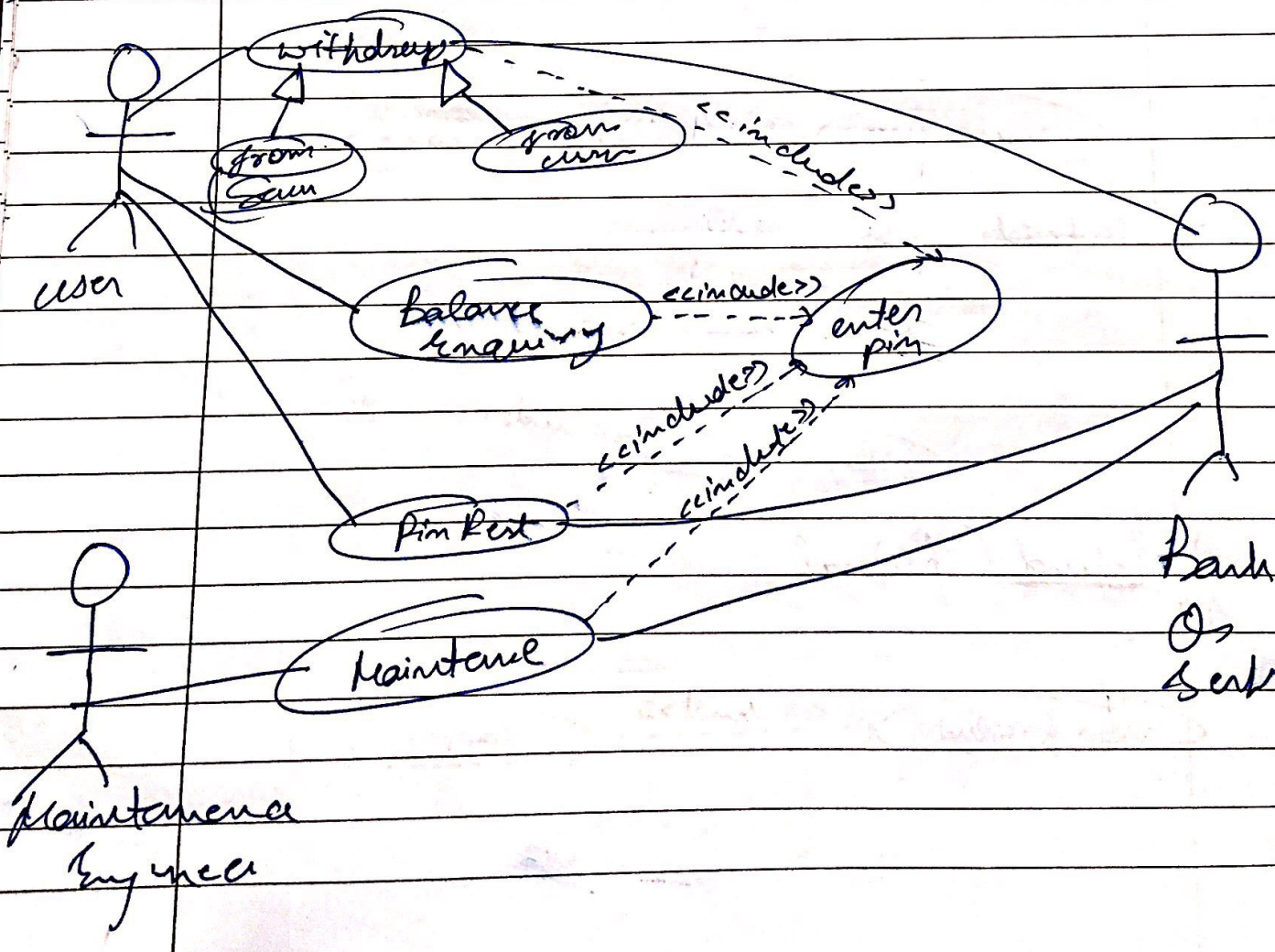


3) Generalization





ATM System Example

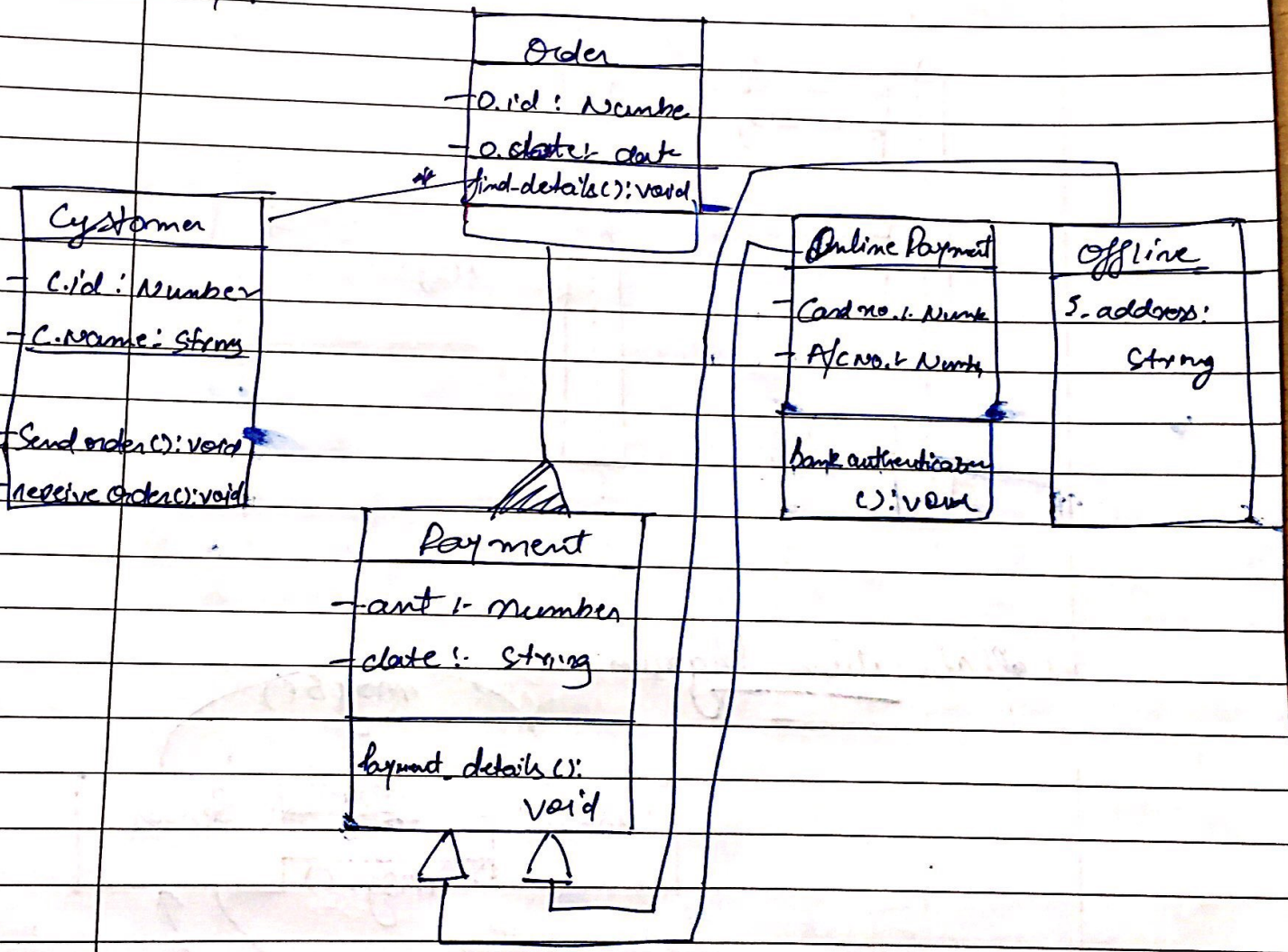


__/__/__

Class diagram

The classes represent entities with common features i.e. attributes & operations.

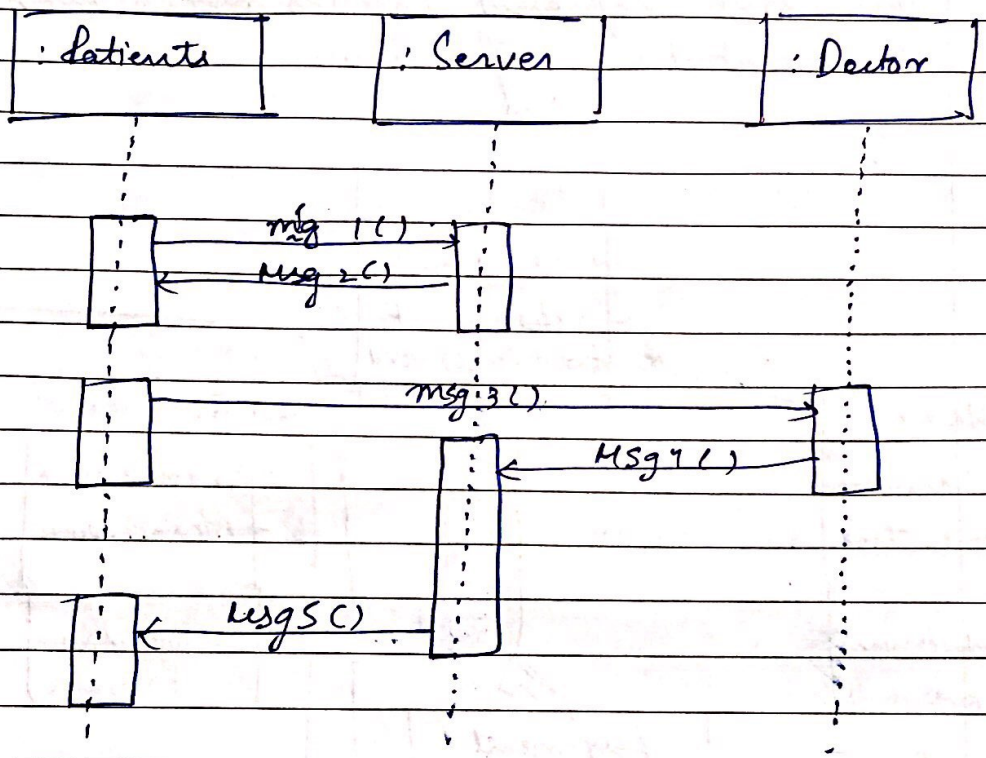
Eg.



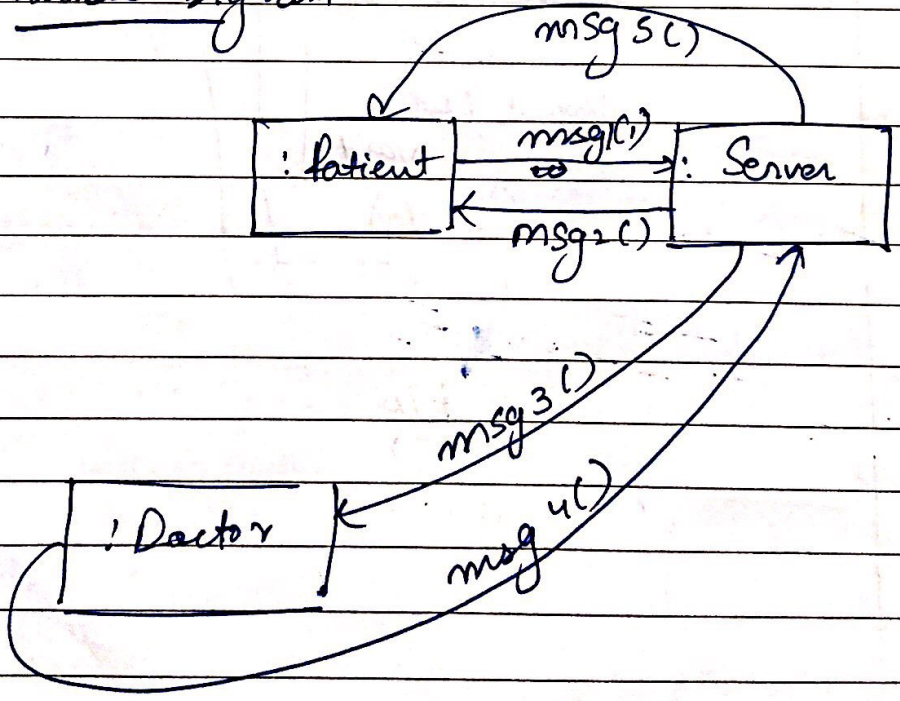
to
→)

__/__/____

Sequence diagram



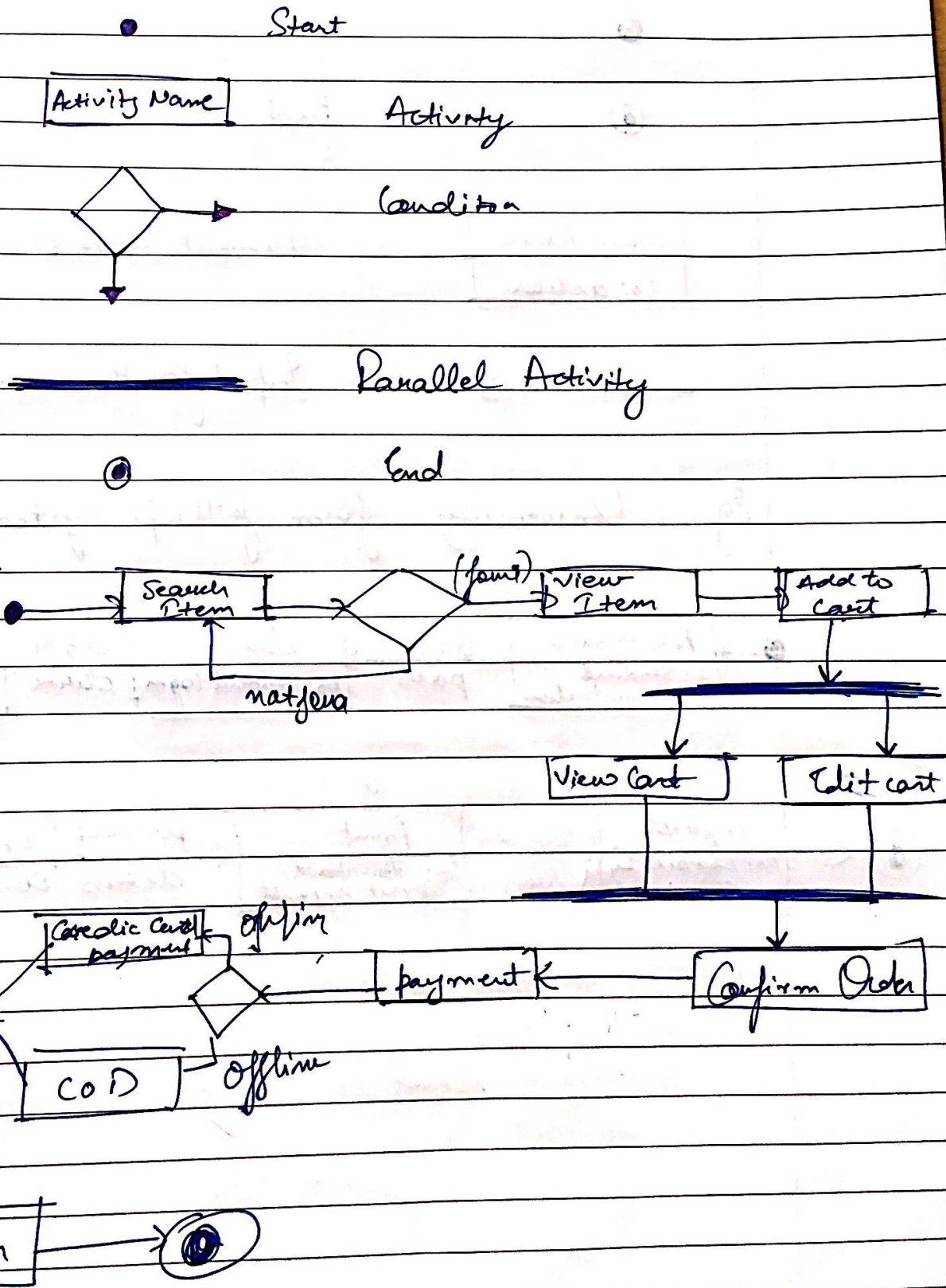
Collaboration Diagram



__/__/____

Activity Diagram

Symbols used in Activity diagram.



__/__/__

State Chart Diagram

Symbols :-

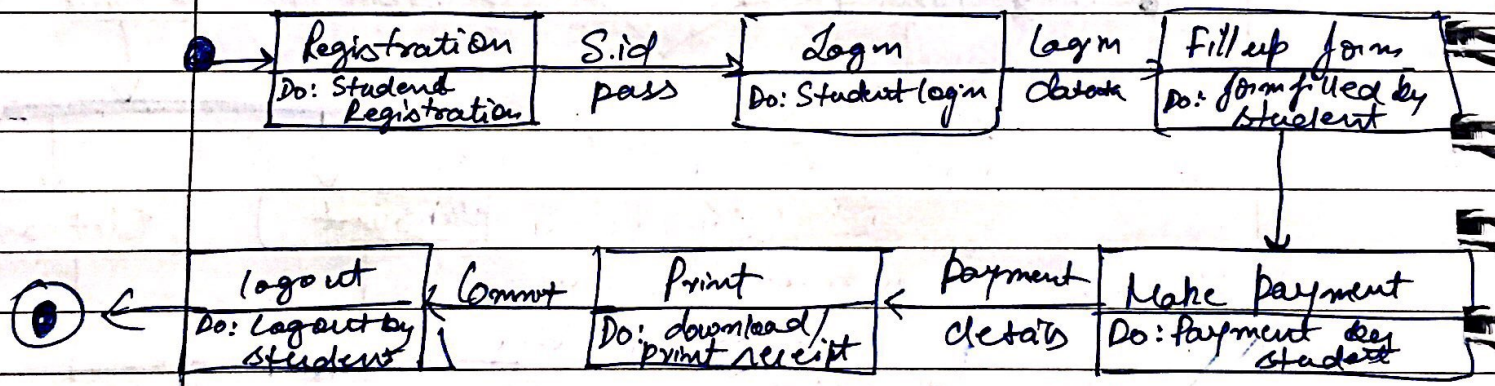
- Start
- End

State Name
Do: action

Present State & Activity.

→ Input for the next state

Eg. University form fillup system.



Chapter :- Software Management (SPM)

Software Size Metricces / Project Size Estimation techniques

The project size estimation is the measure of the complexity in terms of effort and time required to develop the software.

Techniques used are:-

(i) LOC (Lines of code)

- Simplest and most widely used metric
- Comments and blank lines should not be counted.

Disadvantages

(i) Different coding style :- The coding style of different programmers vary from each other, so the same program can be written in multiple ways & therefore the no. of lines may vary from each other. So we can't define a specific complexity level for a software.

(ii) LOC do not consider the quality & efficiency of the code written by the programmer.

(iii)

(ii) Function Point Metric

FPM overcome the drawbacks of the LOC method as it do not depend on the number of lines to calculate the size of program.
FPM estimate the size of a project by the total number of functions or features supported by it.

The program that support large number of features will be of larger size than a S/W that include lesser nol of functions.

FPM is calculated in three steps:-

(i) UFP (Unadjusted function point)
It is the weighted sum of 5 categories:-
UFP:- (Number of inputs)x4 + (Number of outputs)x5 + (Number of inquiries)x4 + (Number of files)x10 + (Number of interfaces)x10

(ii) TCF (Technical complexity factor)
TCF = 0.65+0.01 * DI
where DI is the Degree of Influence that varies from 0° to 84°

(iii) FP (Function Point)

Function point is yelded by multiplying UFP to TCF

FP = UFP * TCF

(iii) Feature Point Metric

Feature point metric assumes that all the functions do not have same complexity, therefore the overall complexity cannot be calculated only by counting the number of functions.

Feature point metric thus include the complexity of each function separately. It aims at calculating the algorithm complexity to determine the overall size & effort.

feature point metric thus include the complexity of each function separately. It aims at calculating the algorithm complexity to determine the overall size & effort.

Project Estimation techniques

Project Estimation is necessary to have an estimate of the parameters like cost, time and efforts.

We can broadly classify these methods as follows:

(i) Empirical estimation techniques
   - (i) Expert judgement
   - (ii) Delphi Cost Estimation

(ii) Heuristic Estimation Techniques
   - (i) Cocomo model
   - (ii) a) Basic COCOMO model
     b) Intermediate COCOMO model
     c) Complete COCOMO model.

(iii) Analytical Estimation Techniques
   - (i) Halstead Software

(A) Empirical Estimation Techniques:-

Empirical estimation is based on making an educated guess of the parameters involved by using the experienced developers. In this technique we mostly take help of the developers who have an experience of many years and have developed similar projects before.

This is further classified into two types:-

(i) Expert judgement Method:- In this only a single expert developer is the incharge and take all the decisions after analyzing the problem.
Drawbacks:-
(i) It is subject to human error as only a single person take all the decisions.
(ii) Single expert may overlook some features i.e. may miss some functions while planning for S/w Development.

(ii) Delphi Cost Estimation:- Though its name is cost estimation technique, but it has to measure the duration & efforts required for development for estimating the cost of the project.
This is different from expert judgement because it involves multiple experts that are authorized to make collective decision.

(B) Heuristic Estimation Technique :-

It is an estimation technique based on the mathematical expressions and modelling all of the relationships among different project parameters.

(i) COCOMO model :- Constructive Cost Estimation Model

It was proposed by Boehm in 1981. COCOMO model mainly aims at classifying the software into 3 broad categories:-

(i) Organic
(ii) Semi-detatched
(iii) Embedded

Organic :- Software under this category are application programs and are usually developed by small team of developers.
The team members are very experienced for developing organic software.

Semi-Detatched :- They involve development of utility softwares (antivirus, backup softwares). Team member are combination of experienced & inexperienced staff.

Embedded :- It uses complex hardware and new/modern techniques for developing system softwares. It involves the fresh and experienced staff to work with the new technology.

P.T.O.

It can also be further classified into 3 categories:-

(i) Basic COCOMO model:- This gives an approximate estimate of the project parameters by using the following formulas:

$$\text{effort} = a_1 \times (\text{KLOC})^{a_2} \text{ PM}$$
$$T_{dev} = b_1 \times (\text{effort})^{b_2} \text{ Months}$$

where KLOC is estimated size of project expressed in kilolines of code.

a_1, a_2, b_1, b_2 are constants.

T_{dev} is development time in months.

Effort is the total manpower required with respect to time. It measured in units person month.

To estimate effor and time for Basic COCOMO in the above formula are calculated as follows:-

	Effort	Development
Organic	effort = $2.4 (\text{KLOC})^{1.05}$ PM	Time = $2.5 (\text{effort})^{0.38}$ months
Semi-Detached	effort = $3.0 (\text{KLOC})^{1.12}$ PM	Time = $2.5 (\text{effort})^{0.35}$ months
Embedded	effort = $3.6 (\text{KLOC})^{1.20}$ PM	Time = $2.5 (\text{effort})^{0.32}$ months.

Q. 32000 lines of code. Assume that Rs. 15000/month determine the effort required to develop the software product & the nominal development time.

=)

KLOC = 32

$$\text{effort} = 2.4 (\text{KLOC})^{1.05} \text{ PM}$$

P.T.O.

$\Rightarrow 2.4(32)^{1.05}$ PM

$= 2.4(38.05)$ PM

$= 91.33$

$\approx$ 91 PM / Person Month.

Time $= 2.5(\text{effort})^{0.38}$ months

$= 2.5(91)^{0.38}$ months

$= 2.5(5.55)$ months

$\approx$ ~~13~~ 13.87

$\approx$ 14 Months.

∴ Cost required to develop the product $= 14 \times 15000$

$\approx 210,000$

(ii) Intermediate COCOMO model:-

Basic cocomo model only consider effort and time but actually many more factors are involved for estimating the actual size of the project.

Intermediate COCOMO recoganize this drawback and fix it by adding many more factors/parameters for estimating project size.

Some of the parameters are:-

(i) Product:-
   (1) Complexity of software
   (ii) Reliability issues

(ii) Computer:- hardware req., memory space needed or processing speed are checked.

(iii) Personal:- Skills, experience or capability of developer.

(iii) Complete COCOMO model :

It states that not only the software are of different types but also the different modules / parts / components of a software can be of different types with different complexities.
Basic and intermediate consider the whole model as a single entity but complete COCOMO divide the whole software system in sub systems and determine their individual effort and time. These individual estimates are then combined to calculate overall cost of the project.

The major components or sub system into which the main system is divided are:

(i) Database part:
(ii) Graphical user interface part
(iii) Communications part.