CO301 Software Engineering

2.5 → unified process
3.8 → Requirement management

Ch-1 Complete
Ch-2 (except 2.5)
SDLC models with Adv., Disadv., & Diagram
→ Waterfall, Spiral, Iterative Enhancement, Prototype, RAD
Ch-3 (except 3.8)
SRS R-E (5 Phases) DFD ER
Ch-5 (complete)

Unit-1 :- Intro :- Intro to SE, Software characteristics, Software components, Software applications, software engineering principles, Software metrics & measurement, maintain & control. SDLC models

Unit-2 : SRS :- Requirements Elicitation Techniques, Requirement analysis, models for Requirement analysis, requirements specification, requirement validation.

Unit-3 Design Principles : Problem partitioning, abstraction, Top-down & bottom up design, structural Approach, Functional vs Object oriented approach of design, design specification, Cohesiveness & coupling. Overview of SA/SD methodology, structured analysis, DFD, extending DFD to structured chart.

What is software :- It consists of programs, set of documentations, and the procedures used to setup & operate the software system.

Engineering :- It is the processes of designing & building something to serve the purpose by finding cost-effective solution to problems.

Engineering is a science, skill & profession of acquiring & applying scientific, economical, social, & practical knowledge in order to design & also build structures, machines, devices, systems, materials & purposes.

Software Engineering :- It is an engineering discipline which concerned all aspects of software production.

OR

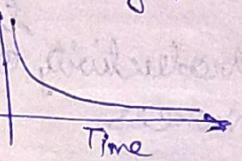
The practical application of scientific knowledge in the design & construction of computer program & its associated documentation in order to ~~main~~ operate & maintain them.

Engineering uses → methods → different approaches you can take
→ tools → automated support
→ procedure → sequence of methods.

Software Components :- ① **Program :-** set of instructions which tells the computer what to do
② **Documentation :-** source information about the product contained in detailed code comments, design documents
③ **Operating procedures :-** set of step by step instructions compiled by an organization to help the workers to carry out complex routine ops.

Software Characteristics :-

(1) Software does not wear out – like Hardware But it may be retired due to environmental change, requirements or new expectations.



(2) Software is not manufactured in the classical sense. Its life is from concept exploration to the retirement. It is one time process of development efforts & continues maintenance efforts in order to keep it operational. However, making 1000 copies is not issue like hardware bcz there is no problem of raw material.

(3) Software is logical rather than physical.

(4) Software is flexible

(5) Reusability of components.

(6) Most software is custom built, rather than being assembled from existing components.

Software Applications :-

- → Real Time Software [Monitor, control & analyze real world events as they occur. eg. weather forecasting]
- → Embedded software [ROM of products controls different functions]
- → Business software [Payroll, file monitoring, employee mngmt]
- → Personal computer software [Multimedia computer graphics, Video games]
- → Artificial Intelligence software (Non-numerical algo to solve complex problems eg. signal processing, artificial neural networks)
- → Web based software → Web application related
- → Engineering & Scientific software → Huge computation of data MATCAD, CAD, Circuit analysis
- → System software (Infrastructure Software) OS, editors, drivers, compilers

---

Documentation Manuals

- **Analysis Specification**
  - DFD
  - Context diagram
  - Formal specification
- **Design**
  - E-R diagram
  - Flow-charts
- **Implementation**
  - Source Code Listings
  - Cross-Reference Listings
- **Testing**
  - Test Data
  - Test Results
- **Operating Procedures**
  - **User Manuals**
    - Beginner's Tutorial Guide
    - System Overview
    - Reference Guide
  - **Operational Manual**
    - Installation Guide
    - System Administration Guide

Measurement :- The act of evaluating a measure.

Measure :- A measure provides a quantitative indication of the extent, dimension, size, capacity, efficiency, productivity or reliability of some attributes of product or process.

Metrices :- A metrice is a quantitative measure of the degree to which a system, component or process possesses a given attribute.

Module/Subprogram :- A work assignment for an individual developer.

Role of Management in S.D

Dependency bcoz

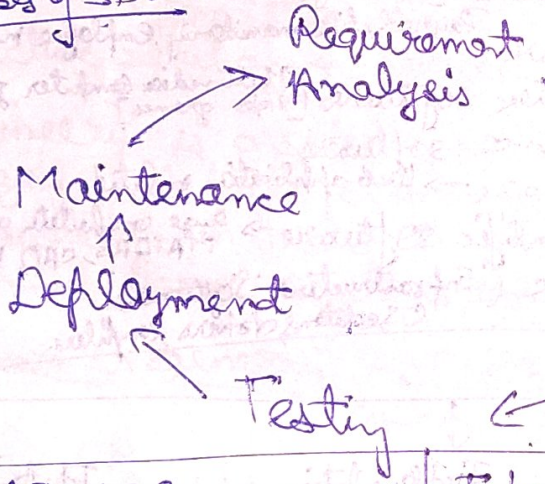
(1) People
(2) Product
(3) Process
(4) Project

Software Development Life Cycle (SDLC)

SDLC is a systematic process for building software that ensures the quality & correctness of the software built.

SDLC models take care of the order of activities performed on a software product from its inception to retirement.

Phases of SDLC :-



Need of SDLC :-

(1) Each phase defines entry & exit criteria for it.
(2) Development of software in a systematic & disciplined manner.
(3) Clear understanding among team members about when & what to do.

Importance of SDLC :-

(1) It provides visibility for the engaged parties.
(2) It allows to control the project.
(3) Predictable deliveries throughout an entire development process.
(4) Eliminating risks like overbudget, deadline breach.
(5) Process goes on until all requirements are met.

→ Requirement Analysis :- Collection of all relevant info from customer. Remove ambiguity SRS is made

→ Designing :- Involves the design of entire system & its components
LLD → Design of its component
HLD → overall plan of the system
Software design document are prepared as per requirement & specification (SDD)

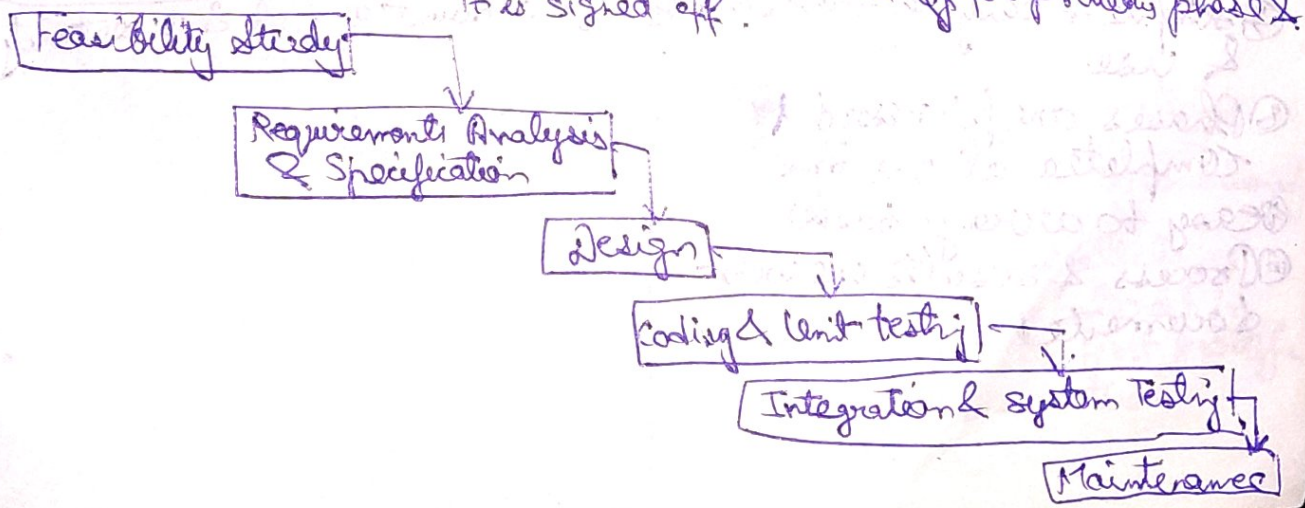
→ Coding Actual development starts.
Longest phase of the SDLC process.
↳ tasks are divided into units or modules & assigned to various developers for writing codes.

→ Testing to rectify any defects/bugs

→ Deployment : software is shifted to market for B-testing & collects feedback of the first user, if any bugs come up, fix them & final version is rolled out.

→ Maintenance
- Bug-fixing → bugs are reported bcz of some scenarios which are not tested at all
- Upgrade → upgrading the application to newer version of software
- Enhancement → Adding some feature into existing software

(I) Waterfall Model :- One phase can be started after completion of the previous phase.
⇒ (Output of one phase) → input of other phase.
Development process can be considered as a sequential flow in waterfall.
Phases do not overlap : Next phase is start only after the defined set of goals are achieved for previous phase & it is signed off.



Feasibility Study :- Rough understanding about what is to be done | resources req. development cost & time
whether the soln is feasible financially & technically

Requirement Analysis & Specification

Design :- SRS → SDD (Software design docs) Transform SRS doc into structure that is suitable for implementation in some programming language

Coding & Unit Testing : SDD → Source code modules have been ind. tested to determine the correct working of all individual modules.

Integration & system testing :- Integration of different modules is undertaken

- α-testing → Performed by development team
- β-testing → Performed by set of friendly customer
- Acceptance testing → By customer itself

Maintenance :-
- → Corrective Maintenance
- → Perfective Maintenance
- → Adaptive Maintenance

Can be used

1. when requirements are well documented, clear & fixed.
2. No ambiguous requirements
3. Product definition is stable
4. Tech is understood & not dynamic.
5. Ample resources are available with required expertise
6. Project is short

Problems

1. Difficult to define all requirements in beginning
2. Not suitable for accomodating any change
3. Real projects are rarely sequential
4. Large project
5. Working version is not seen until late in project & type

7. Simple & easy to understand & use.
8. Phases are processed & completed at one time
9. easy to arrange tasks
10. Process & results are well documented.

11. High amount of risk & uncertainty

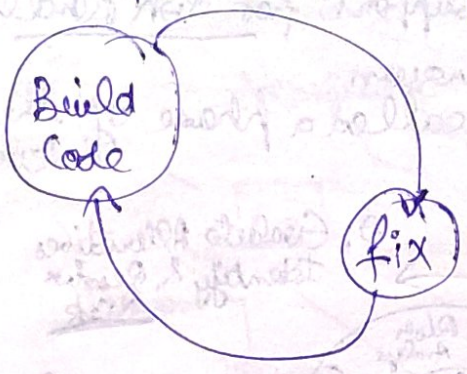
Build & fix models

Adhoc approach & not well defined

100 or 200 lines of code project.

No space for requirement & design & maintenance

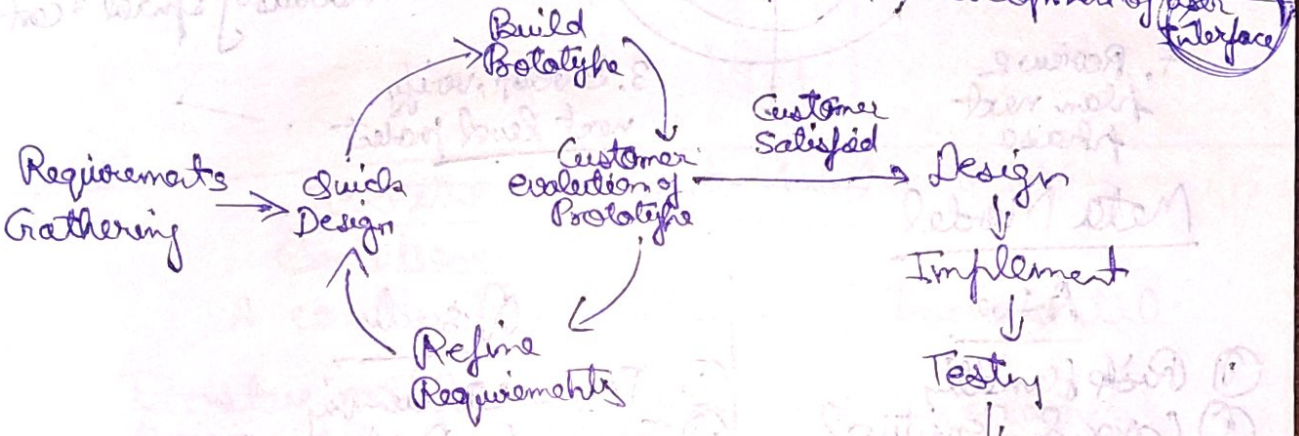
Code soon become unfixable & unenhanceable



Prototype Model :

Suitable for projects which either the customer requirements or the technical solutions are not well understood.

Popular for development of user interface



- develop the incomplete high-level paper model
- how the screens must look like
- how the user interface must behave
- how the system would produce output
- In Quick design, more intricate details of internal design are ignored

→ Basic Requirement identification
→ Develop the initial prototype
→ Review of the prototype
→ Revise & enhancement of prototype.

Advn

1. Customer get to see working version of system early in the life cycle.
2. New requirements can easily be accomodated
3. Missing functionality can be figured out.
4. Flexible in design
5. Errors can be detected much earlier
6. Developed prototype can be reused by developer for more complicated project

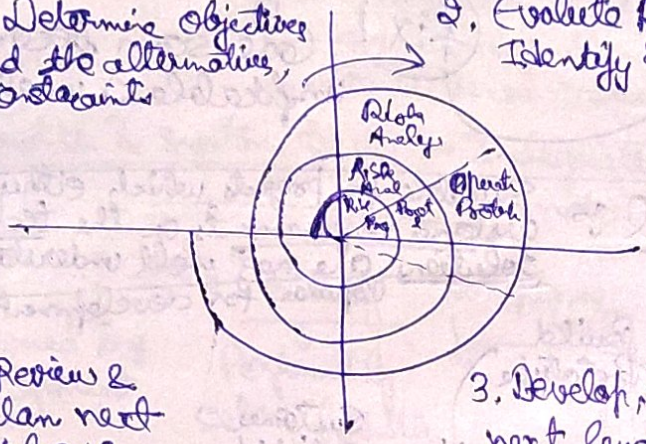
Disadv

i. Costly wrt time & money.
ii. There may be too much variation in requirement each time
iii. Poor documentation due to continuously changing customer requirements.
iv. Difficult to accomodate all changes
v. Customer might lose interest in ~~the~~ product after first prototype
sometimes demand the actual product to soon.

Spiral Model

provides support for Risk Handling

No. of spiral is unknown
& each spiral is called a phase of the software development process



Angular dimension = Progress
Radius of spiral = cost

Meta Model

Advantages

(i) Risk handling
(ii) Large & Sensitive & Projects.
(iii) Flexibility in Req.
(iv) Customer Satisfaction

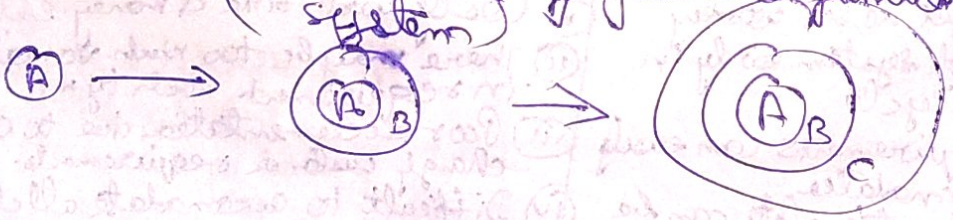
Disadvantages

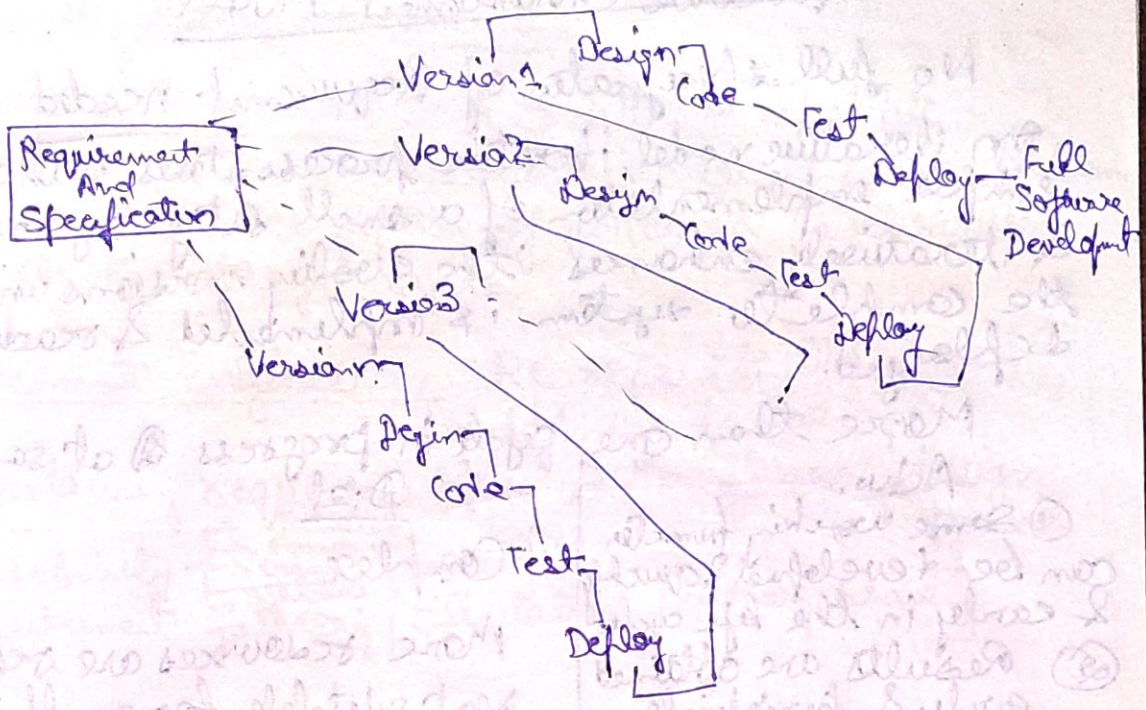
(i) Time consuming
(ii) Too much Risk Analysis
(iii) Complex
(iv) Expensive

Incremental Process Models

when requirements defined precisely & there is no confusion about functionality of final product.

After every cycle a useable product is given to customer. (Quick delivery of limited functionality system)





~~Iterative Enhancement Model~~

Evolutionary Model

A combination of Iterative & Incremental

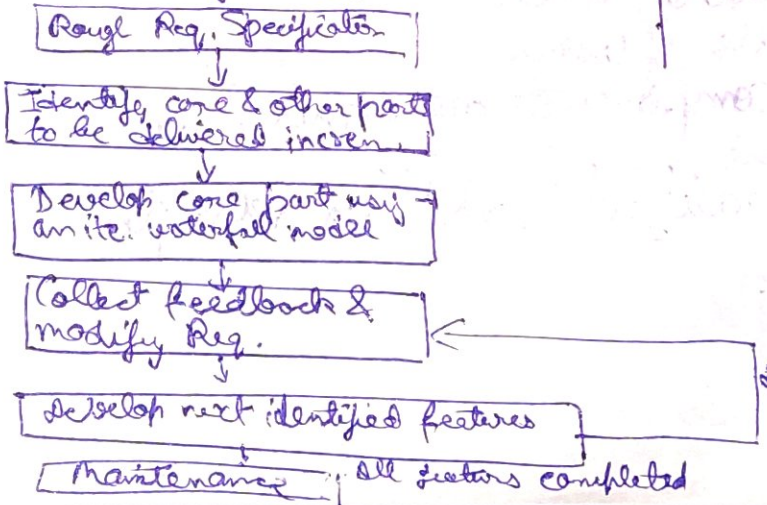
Iterative model main advantage is its feedback process in every phase

Advantages

1. Customer Requirements are clearly specified
2. Risk analysis is better
3. It supports changing environments
4. Initial operating time is less.
5. Better for large mission-critical process

Disadv.

1. Not suitable for small project
2. Cost
3. Highly skilled resources are required



Iterative Enhancement Model

No full specification of Requirement needed

In iterative model, iterative process starts with a simple implementation of a small set of software req. & iteratively enhances the evolving versions until the complete system is implemented & ready to be deployed.

More than one cycle in progress @ at same time

Adv.

1. Some working functions can be developed quickly & early in the life cycle
2. Results are obtained early & periodically
3. Progress can be measured.

Disl

Complex

More resources are required

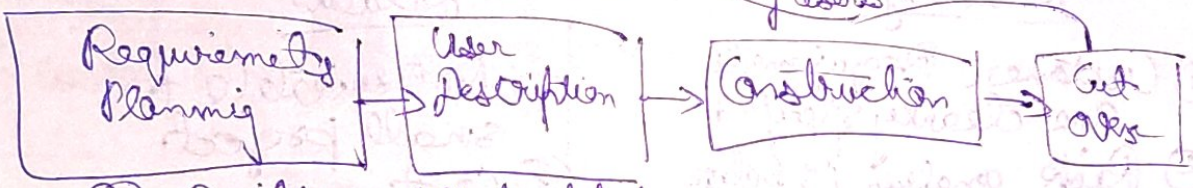
Not suitable for small project

Time & Cost Constraint

RAD

Rapid Application Development

with active participation of users



1. Build a rapid prototype
2. Give it to user for evaluation & obtain feedback
3. Prototype is modified

Cons:- 1. Not a suitable model in absence of users
2. Reusable components are required to reduce time
3. Highly specialized & skilled developers required

~~Coupling & Cohesiveness~~

Software Requirements :- A condition or capability needed by a user to solve a problem.
It is the description of features & functionalities of the target system.

Requirement Engineering :- RE refers to the process of defining, documenting & maintaining requirements in the engineering design process

→ Feasibility → Technical, Economical, Legal, Scheduled → operational
→ Requirement gathering / Elicitation [Gathering phase]
→ SRS
→ Software Requirement validation (Checking)

Tool supports for Requirement Engineering :-

(i) Observation reports
(ii) Questionnaires
(iii) Use case
(iv) Use stories
(v) Requirement workshops
(vi) ~~Use case~~ Mind mapping
(vii) Role-playing
(viii) ~~Prototyping~~

Functional	Non-functional
related to working/functional aspects of software	are expected characteristics of target software (security, storage, configuration, Flexibility, Cost Performance, Disaster recovery, interoperability, accessibility)
Product features	Product properties

Interface Requirements → User
→ Communication
→ Software
→ Hardware

SRS

SRS is a description of software system to be developed.

It lays out functional & non-functional requirements of the software developed.

It may includes a set of use cases that describe user interaction that the software must provide to the user for perfect interaction.

SRS Structure

1. Introduction
   - 1.1. Purpose
   - 1.2. Intended Audience
   - 1.3 Scope
   - 1.4 Definition
   - 1.5 References
2. Overall Description
   - 2.1 User Interface
   - 2.2 System Interface
   - 2.3 Constraints, assumptions & dependencies
   - 2.4 User characteristics
3. System features & Requirements
   - 3.1 Functional Req.
   - 3.2 Use-Cases
   - 3.3 External Interface Requirements
   - 3.4 Logical database requirement
   - 3.5 Non-function Req.
4. Deliver for Approval

User Requirements :- URS specifies what user expects the software to be able to do

1. Easy & simple to operate
2. Quick Response
3. Effectively handling operational errors
4. customer support

Data-flow Diagrams

A graphical representation for communicating with users, ~~for~~ managers & other personnel.

Useful for analyzing

Data can flow from
- entity to process
- process to entity
- process to store & back
- process to process

Data can't flow from
1. External entity to entity
2. External entity to store
3. Store to external entity
4. Store to store

FAST
Interviews, brainstorming session, task analysis, Delphi technique, prototyping

5 phase of RE

- Requirement elicitation → Gathering of info/ to know about project domain
- Requirement analysis → analyzed for conflicts, ambiguity, missing or extra req., inconsistency
- Requirement specification
- Requirement validation
- Requirement management

changing nature of req.

Verification
set of tasks which ensures that the software correctly implements a fun.

Validation
It ensures that the software which is built is traceable to customer requirements.

Concerned with demonstrating that the requirements define the system that the customers really want.

Use-case diagram

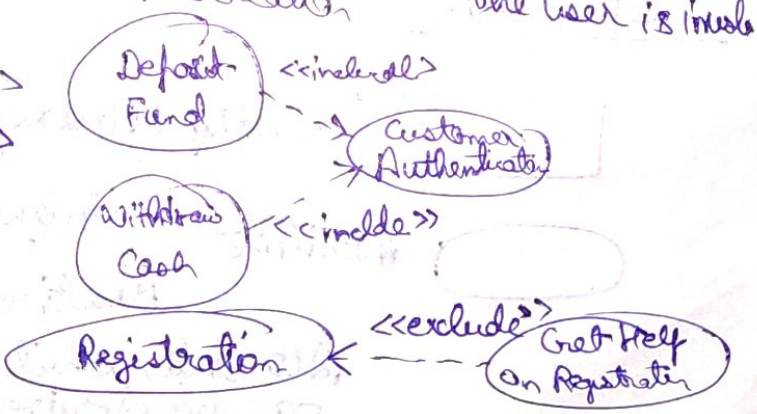
Representation of user's interaction with system that shows the relationship b/w user & different use cases in which the user is involved

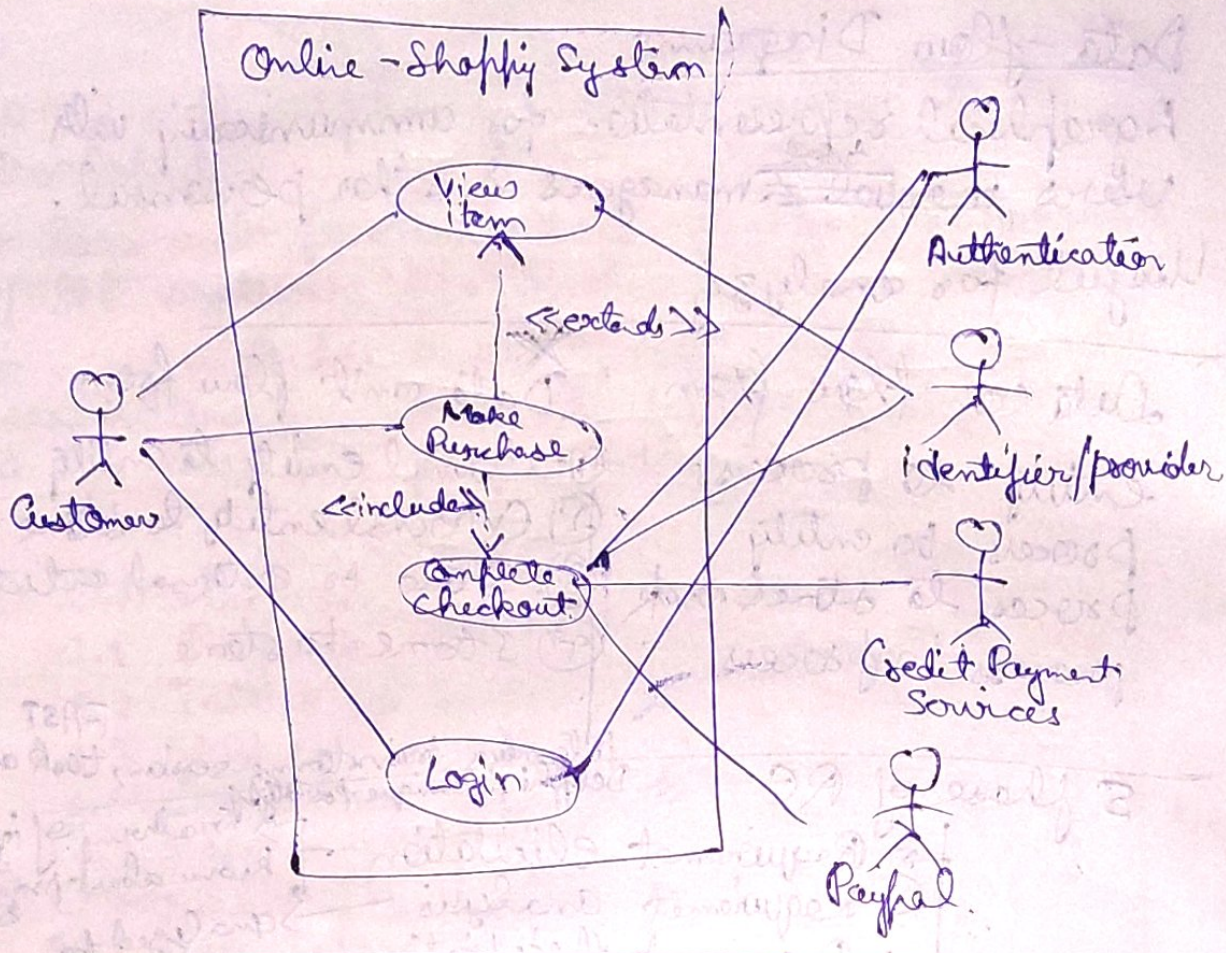
Actor

Use-case

Association

<<include>>
<<exclude>>





DFD (Data-flow diagram) [Bubble chart]

It is a traditional visual representation of the info flows within a system. It is used as a communication tool b/w a system analyst & any person who is interacting with system.

There is no order of events, → represents flow of data.

0-level DFD (context diagram)

1-level DFD

2-level DFD

ER Model

- external entity
- Process
- data flow
- Store

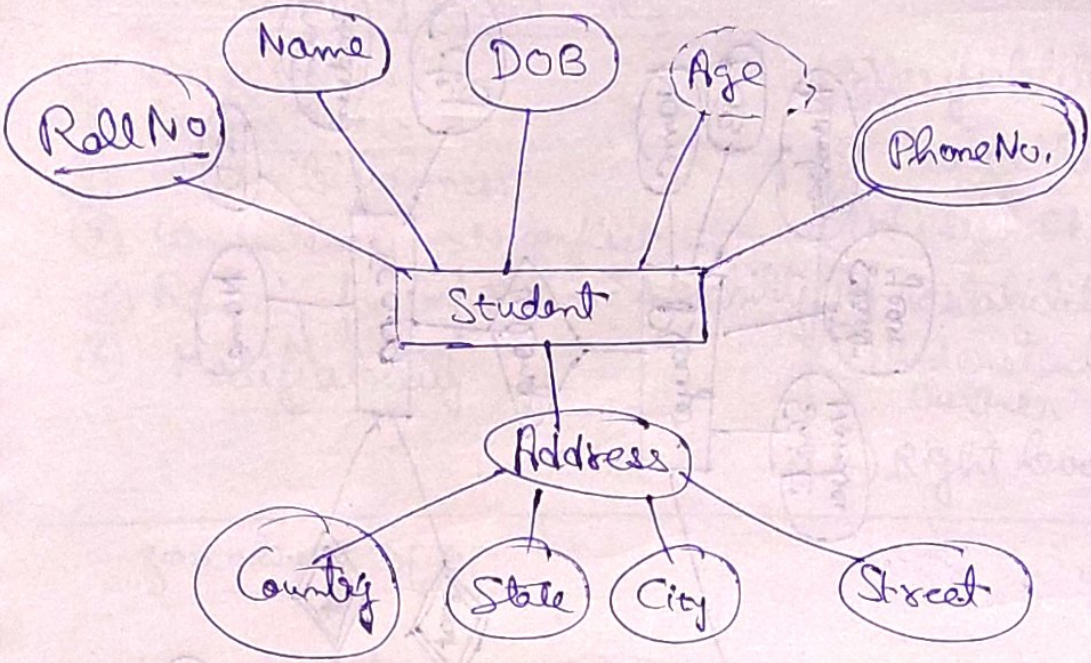
Entity (Person / Object / physical / conceptual e.g. job, salary)

Attributes

Unique

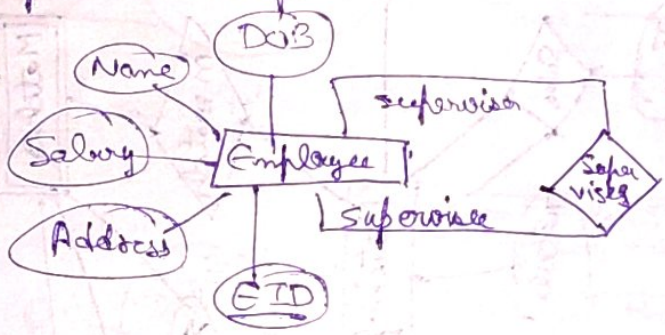
Multi-valued

Attribute, which can be derived from other attrs.

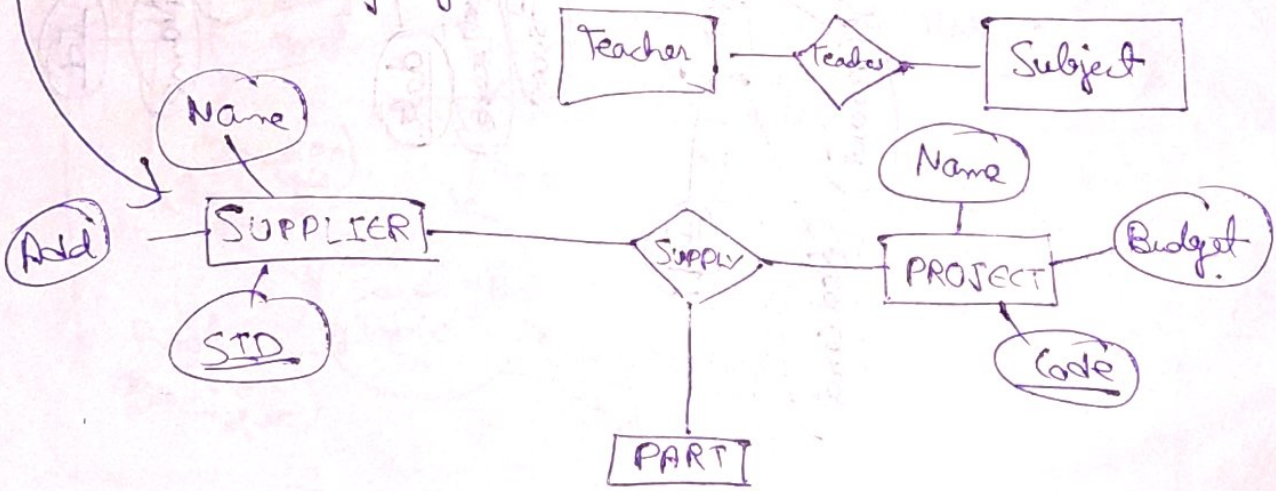


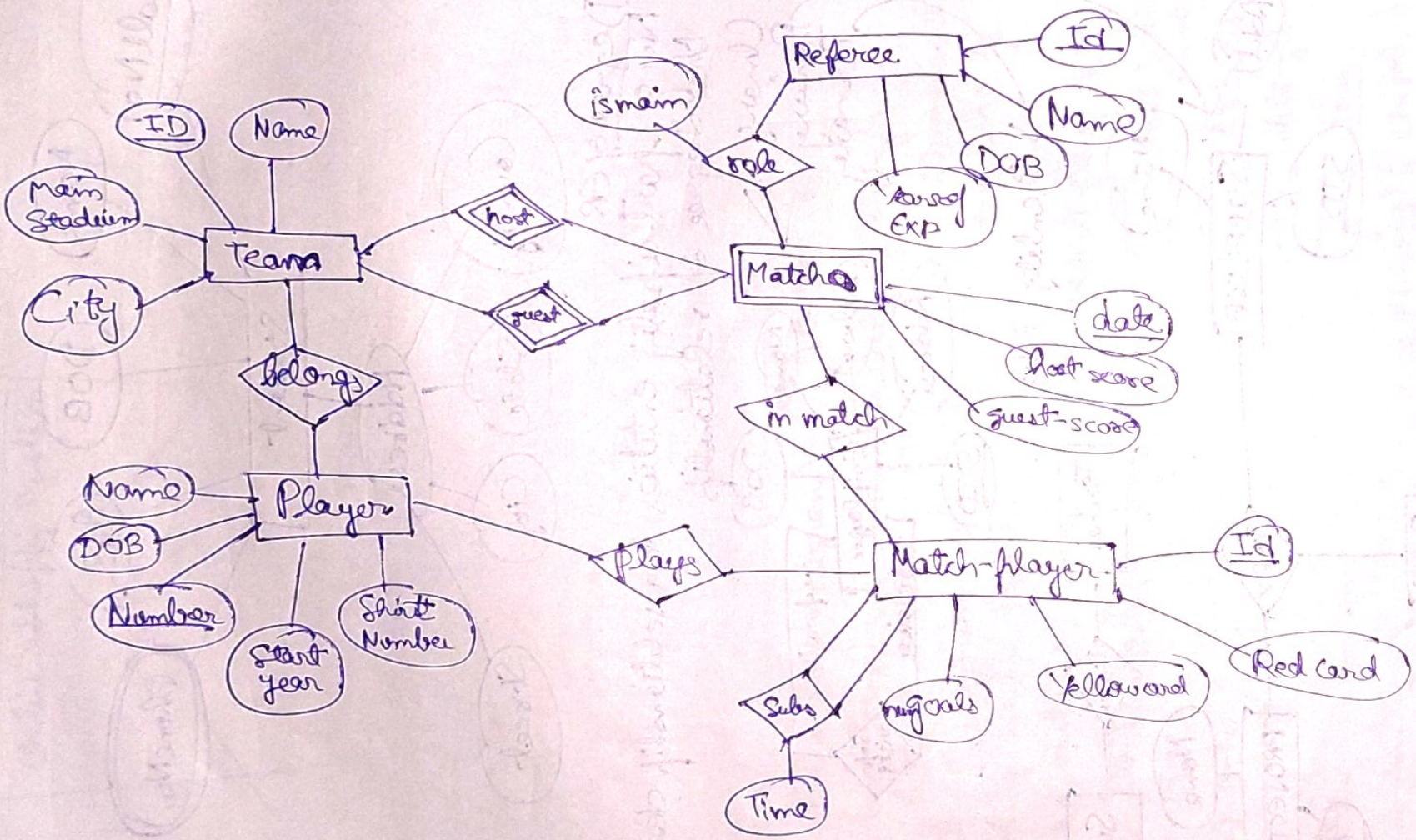
Degree of ER
No. of participating entities in a relationship describes the degree of relationship

Unary
Binary
Ternary



2 entity type

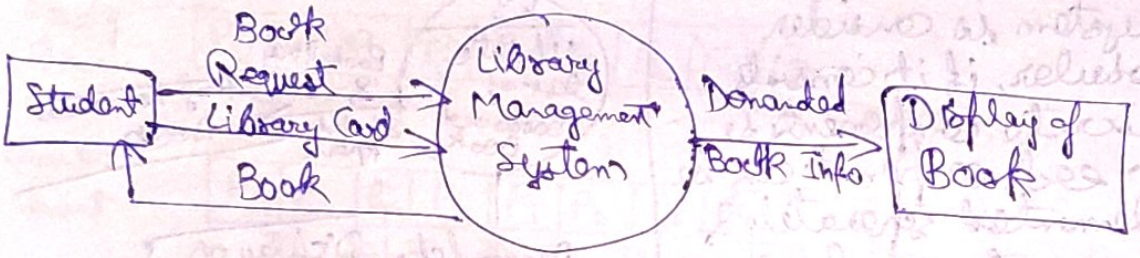



Referee
Id
Name
DOB
Years of Exp
ismain
role
ID
Name
Main Stadium
City
Teams
host
guest
Belongs
Matches
date
Host score
Guest-score
in match
Name
DOB
Number
Start year
Shirt Number
Players
plays
Match-player
Id
Red card
Yellow card
goals
Subs
Time

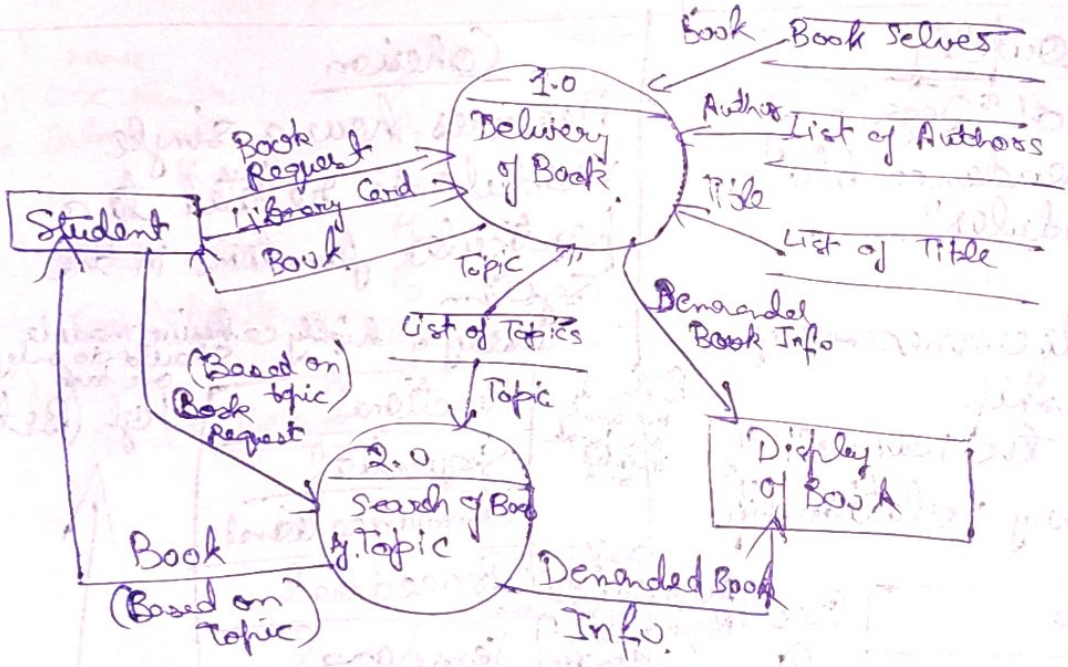
Characteristics of Good SRS

1. Correctness
2. Completeness
3. Unambiguousness
4. Consistency → No conflict
5. Ranking for importance & stability
6. Modifiability
7. Verifiability
8. Traceability
9. Design Independence
10. Testability
11. Understandable by customer
12. Right level of abstraction

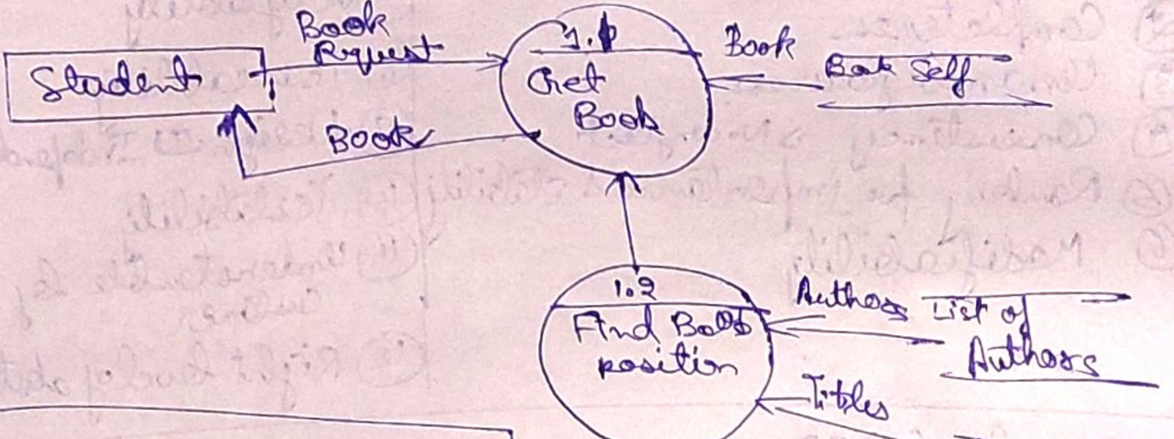
Zero Level DFD

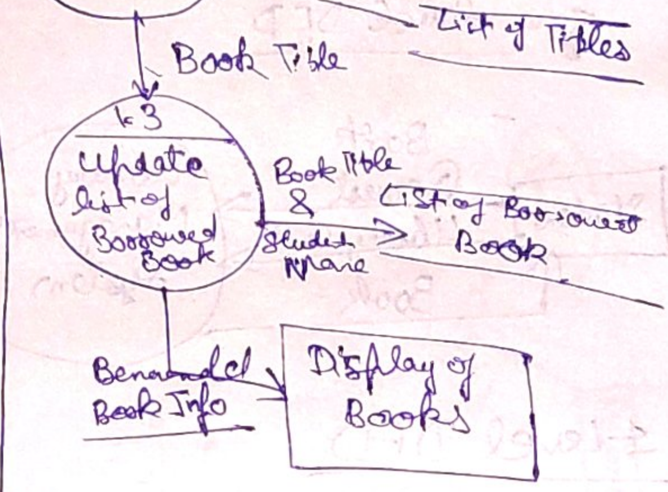


1 Level DFD



2-level DFD :-





Modularity

A system is consider moduler, if it consists of discrete components so that each component can be implemented separately & a change to one has minimal impact on other.

Coupling

Measure of "Degree of interdependence b/w the modules".

↓

⇒ eliminating unnecessary relationship

⇒ reducing the number of necessary relationship

Best (Most Required)

- Data
- Stamp (data structure)
- Control
- External
- Common
- Content

Worst (Least Required)

Cohesion

Measures how a single module is related to a particular functions in the System

- ideally a highly cohesive module should do only one task

High (Best)

- Functional — only to execute one task / one problem related task
- Sequential
- Communicational
- Procedural — specific order in which tasks are to be followed
- Temporal — same time-span
- Logical — Because they fall in same category
- Coincidental

Worst (low)

⇓ (Contain instruction that have no relationship to one another)