Software Engineering :-

According to the IEEE, software engineering is the application of systematic discipline and quantifiable approach to the development, operations and maintenance of the software.

Software Engg. is a discipline whose aim is the production of fault-free software that satisfy the user's need and that is delivered on time and within budget.

Software Engg. is the use of methodologies, tools and techniques to resolve the practical problem that arise in the construction, deployment, support and evolution of the software

A typical SDLC consists of –
↳ Software Development life cycle.

→ Requirement (Data gathering, data storage)

→ Design (Tools, software used to develop)

→ Implementation (coding part)

→ Verification (Testing)

→ Maintenance (updating codebase, bug fixes)

Software Crisis :-

It is characterized by inability to develop software on time, within budget and within requirements.

→ Factors ⇒

1. Lack of communication b/w developers.
2. Increase in cost and size of software.
3. Lack of understanding requirements.
4. Duplication of efforts.
5. Absence of automation.
6. Difficulty in maintaining the code.
7. Overbudget.

development.

Software Lifecycle Models :-

* SDLC - Software process / SDLC also known as software methodology. It is a set of related activities that leads to the production of the s/w. These activities may involve development of s/w from scratch or modifying an existing / adjusting systems.

(1) Feasibility study →

→ ~~Point out~~ Find abstract definition of the problem.

→ Checking financial and technical feasibility.

→ Analysis of cost.

→ Checking availability of infrastructure and HR.

→ Alternative solution strategy.

② Requirement Analysis and Specification →

→ Try to understand the exact requirement of the customer and document them properly.

→ Try to collect an analysis of all data related to the project.

SRS → software Requirement specification

large document written in natural language (what system will do) w/o describing how to do it.

③ Designing → [overall architecture and algorithm are chosen]

→ SDD (software designing Description) → how system will do functions.

→ We transform requirement into a structure that is suitable for implementing code in a specific programming language.

④ Implementation →

→ Translate the designing of the system into the code of programming language.

→ Readable or maintenable code.

(5) Testing →

→ S/w testing is a process of executing a program with intent of finding bugs in code.

(6) Deployment →

→ s/w is installed on the user site and training of user and hardware check is done.

(7) Maintenance →

→ Any change in the s/w after its official release is called maintenance.

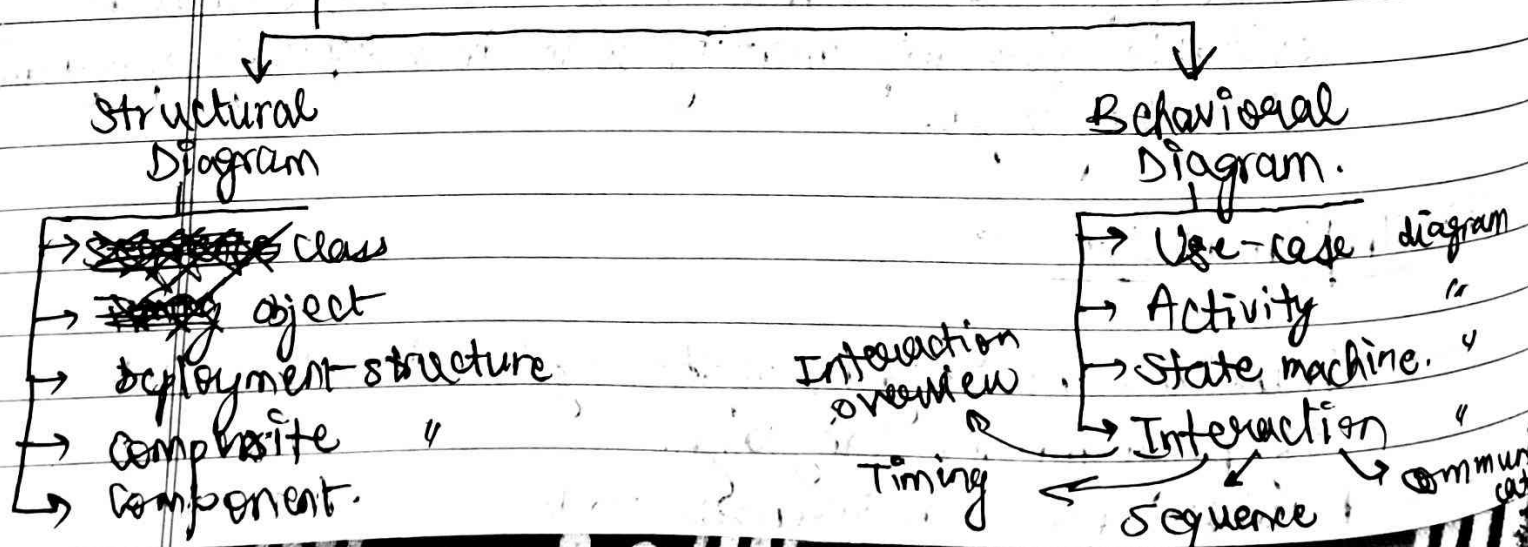
→ It could be because of various reasons.

(a) Adaptive maintenance. (Different devices)

(b) corrective

(c) Perfective.

UML :- (Unimodified Modelling language)



→ It is a modelling language in the field of software engineering that is intended to provide a standard way to visualize design of a system.

→ Usecase diagram:

* They model the behaviour of the system.
* Used to illustrate functional requirement of a system and its interaction with the external agents ~~and~~ (or, actors).
* High level view of the system without going into implementation details.

* components —

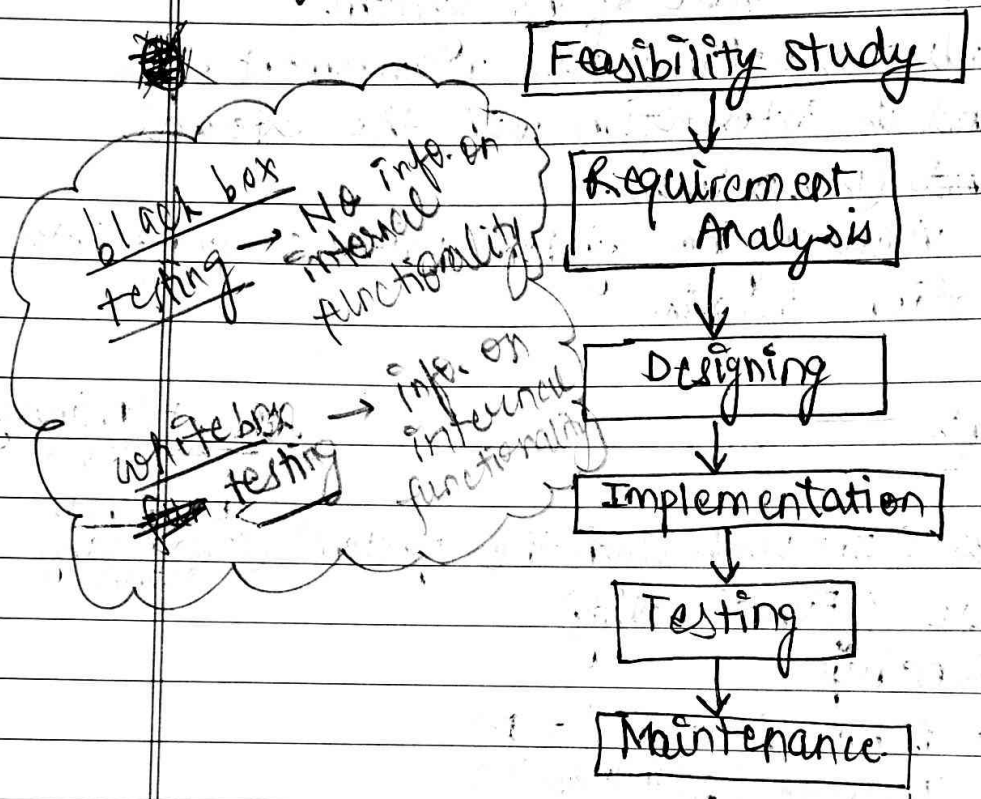
Client (Primary actor)
Backend (secondary actor)

① Actors (Interacts with the system)
② Usecase (Functionality)
③ Relationship
   - Generalization (→)
   - Include (<<include>>)
   - Extend (<<extend>>)
   - Association link (—)
④ System boundary

An actor or external agent lies outside the system model but interacts with the system in some way.

Waterfall Model :-

→ It is developed by the Winston W. Royce in 1970 inspired by manufacturing and construction process where each step relies on completion of previous step.



→ It is the simplest SDLC model in which phases are organized in linear fashion.

→ This type of model is used for small to medium sized projects with clear and well-defined requirements.

→ Disadvantages

- No scope for backtracking.
- Working version is produced in the last level.

- High amounts of risk and uncertainity.
- Not suitable for changes.

→ Advantages

- Easy to understand, implement.
- Clear milestone.
- well-defined stages
- well-defined I/O ~~end~~ for each stage.
- Easy to schedule as all staff do not work concurrently on diff. projects.
- low cost.

- High amounts of risk and uncertainty.
- Not suitable for changes.

→ Advantages

- Easy to understand, implement.
- Clear milestone.
- well-defined stages
- well-defined I/O ~~and O~~ for each stage.
- Easy to schedule as all staff do not work concurrently on diff. projects.
- low cost.

V-model.

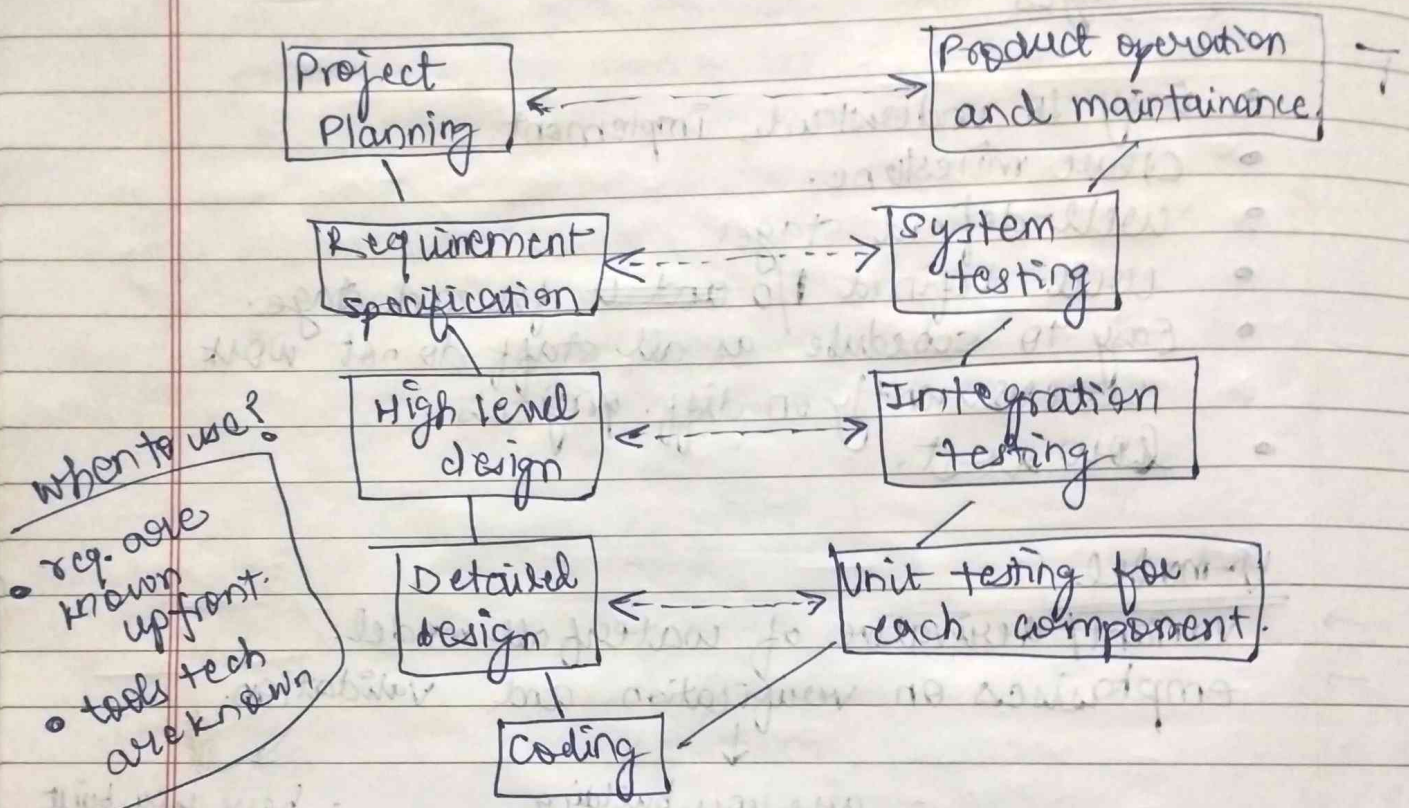
→ variant/derivative of waterfall model.

→ emphasises on verification and validation.

verification	validation
- are you building it right.	- have you built it right.
- each deliverable is testable.	- system testing
- unit testing integration testing	- by testers.
- by developer	- static, dynamic.
- involves walk through review, inspection	
- static dynamic both	

→ Named so because V and V activities are spread over entire lifecycle in this model.

→ in every phase of development testing activities are planned with development.



Advantages

* each deliverable stage is made testable.
* easy to use.
* emphasize planning for V and V of the S/W. starting from early phase of S/W development.

Disadvantages

* Doesn't support overlapping of phases (due to parallel work).
* Doesn't easily accomodate later changes to the requirement (variant of waterfall model)

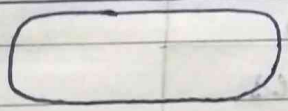
Iteration and Incremental model diff.?

Prototype Model :-

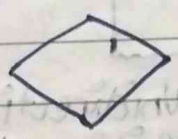
→ Working model with limited functionality.
→ eg. Tenskart trial, try game.

Activity Diagram :- [Advanced version of flowchart] {extension of workflow diagram}

1. Start State (●)
2. Final State (End Process) (◉)
3. Activity / Particular Action (operational process of the system)

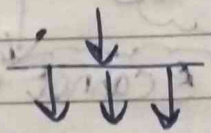


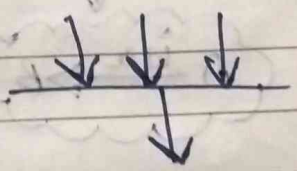
4. Decision Node (diamond-shaped box used to encounter a decision)
(TWO outcomes Yes & No).



5. Fork and Merge → used to combine 2 or more subprocesses into a process.

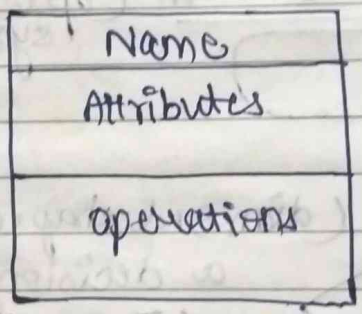
Fork → used to split a process into subprocess





Class Diagram :-

* **Class** → • Classes are entities with common features i.e. attributes and operations.
  • Represented as a solid outlined rectangle with compartments.
  • Compartments for name, attributes and operations.
  • Attributes and operations compartments are optional depending on the purpose of the diagram.

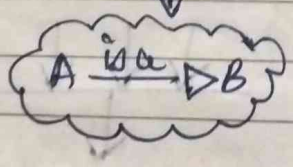


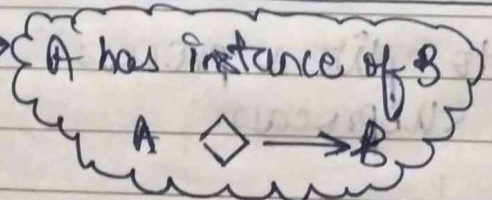
[[Symbols]]

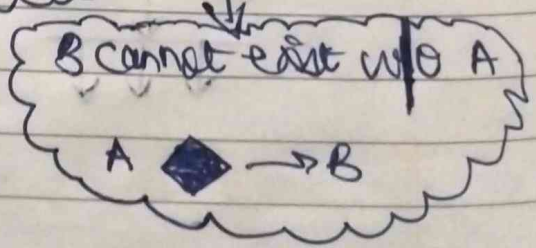
+ → Public
- → Private
→ Protected.

Relationship:

1. Association (—)
   → Unidirectional association (→)
   → Bidirectional " (—)
   → Multiplicity (1...*) many
   (1...2)
2. Aggregation and Composition.
3. Inheritance.



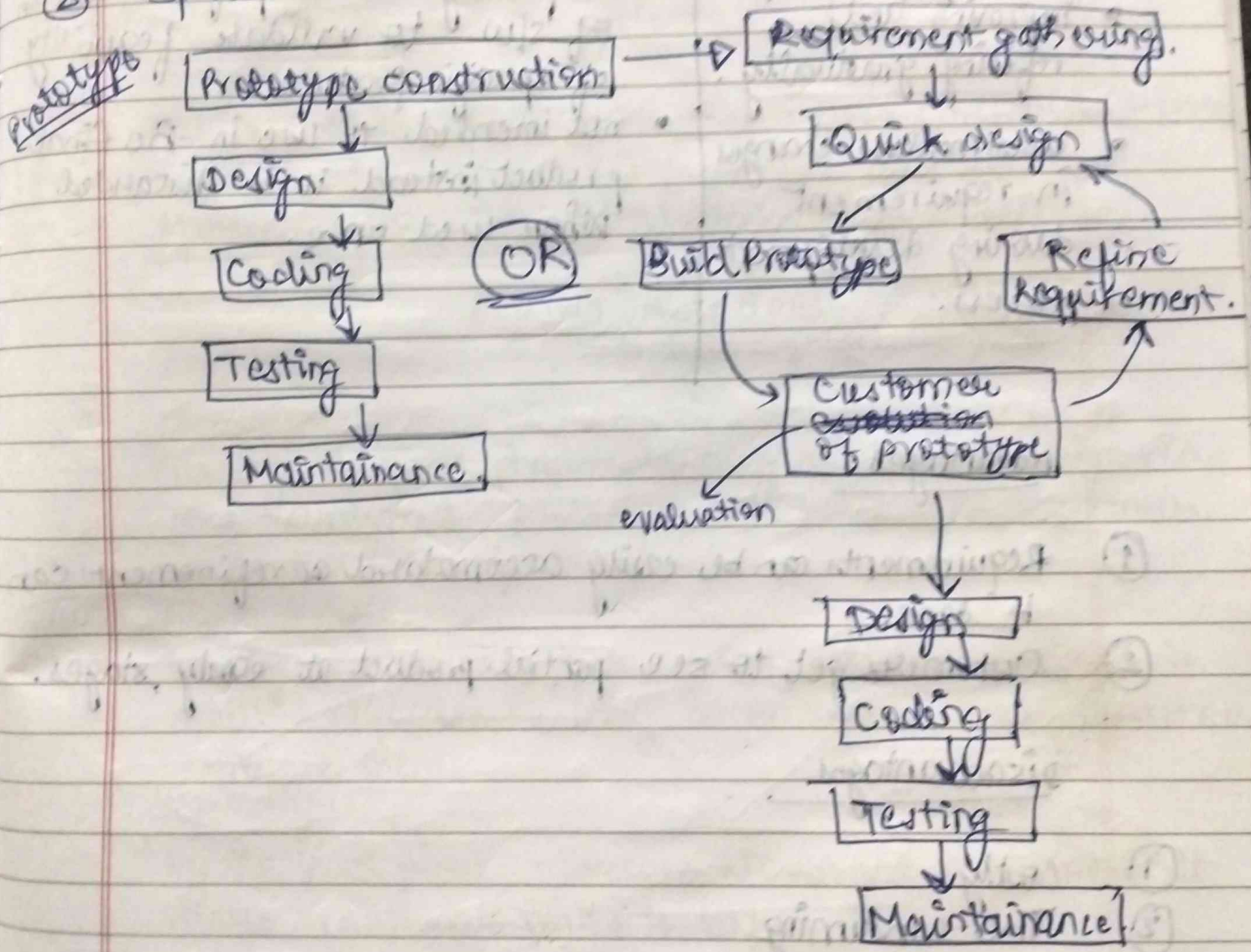




Evolutionary Process Model

→ It is iterative model
→ Characterized in a manner that enables you to develop more complete version of a s/w.

1. Prototype Model
2. Spiral Model.



Prototype Model

Evolutionary prototype model

- iterative / incremental
- working prototype
- improving and refining gradually!
- accomodate changes in requirement during development process.

Throw away prototype model

- rapid / exploratory prototyping.
- temporary or simplified version of s/w to validate feasibility of specific features.
- not intended to use in the final product instead it is discarded when used once.

Advantages:

1. Requirements can be easily accomodated as refinement can be done.
2. Customers get to see partial product at early stages.

Disadvantages

1. Costly.
2. Time-consuming
3. Poor documentation.
4. Difficult to accomodate requirements after every change.
5. Uncertainity in no. of iterations
6. Customer dissatisfaction.

→ Metamodel (combination of different models)

classmate

Spiral Model [Barry Boehm → Include risk factor in SDLC]

→ Each path around the spiral is indicative of the increased cost.

→ Radial dimension ⇒ cumulative cost.

→ Angular dimension ⇒ progress made in completing each cycle.

→ Each loop of the spiral from the x-axis clockwise (1-phase)

→ 1 phase divided into 4 sectors.

Planning ① risk analysis ② devel-opment ③ assessment. ④

→ During the first phase, planning are performed, risks are analyzed, prototype are built and customer evaluates the prototype.

→ During the second phase more refined prototype is built, requirement are documented and customer evaluate them.

→ By the third phase, risks are known and a somewhat more traditional route is taken.

Identification and elimination of risks before they threaten costs and operations.

Advantages

1. Risk Handling
2. Flexibility.
3. Good for large projects.
4. SW early in the lifecycle.
5. Suitable for high risk projects where business needs may be unstable.

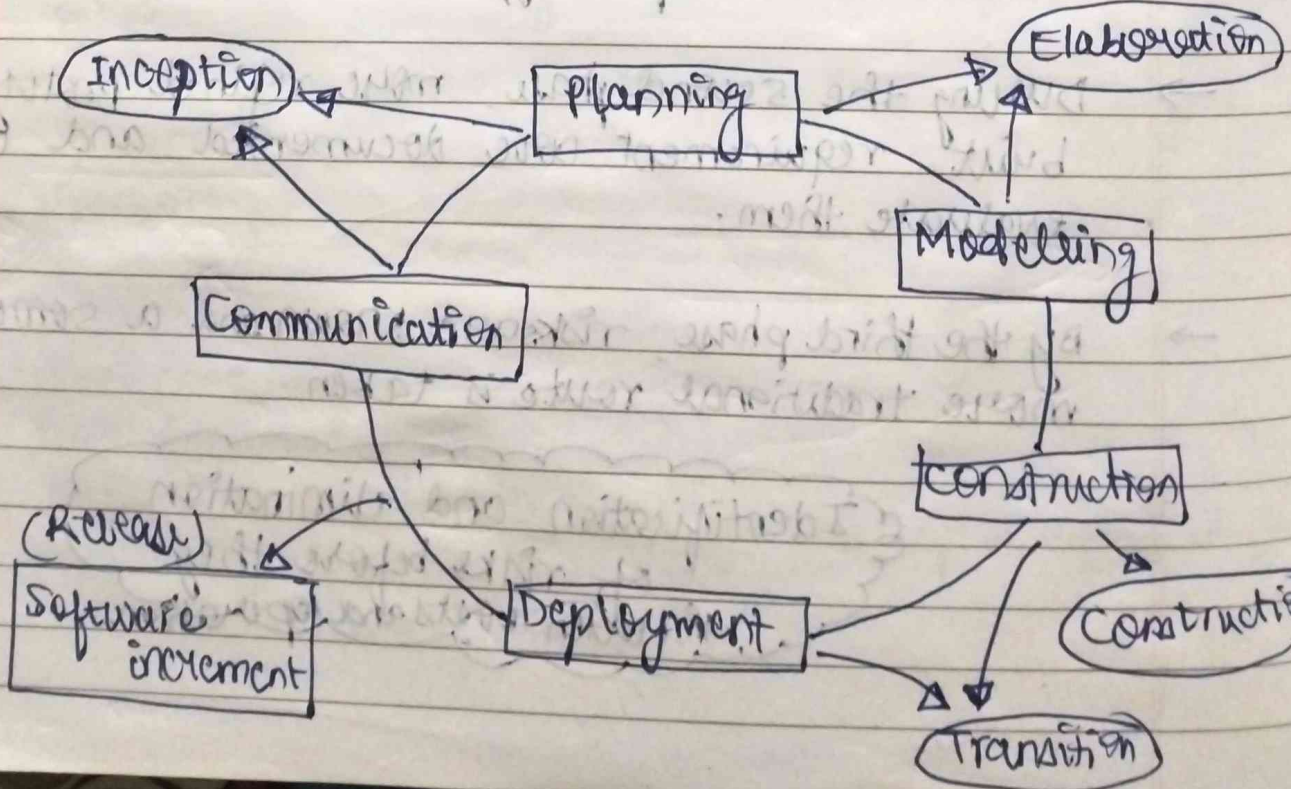
Disadvantages

1. Time, cost or complexity
2. Too much facilities for risk analysis.

---

* Unified Process Model :-

object-oriented system.
4 steps :
1. Inception 2. Elaboration 3. Construction 4. Transition



① Inception

→ Initial business case, Initial risk list, project plans, prototype.

② Elaboration

→ Detailed use case model
→ Revised risk list
→ Analysis model

③ Construction

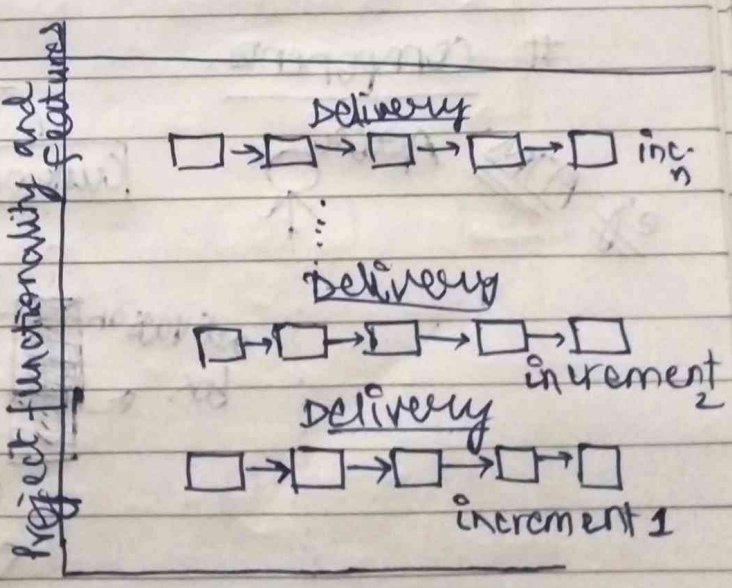
→ Design model
→ Test plan
→ Procedures
→ Test cases
→ Instruction models.

④ Transition

→ Software increment.
→ Beta testing (by user)
→ User feedback

Incremental Model :-

- communication
- Planning
- Modeling
- construction
- Deployment.



→ This model is used when the delivery of entire working project at the committed date seems to be impossible.

→ First delivery of increment model is core product. (generally) and supplementary features are delivered in subsequent incremental deliveries.

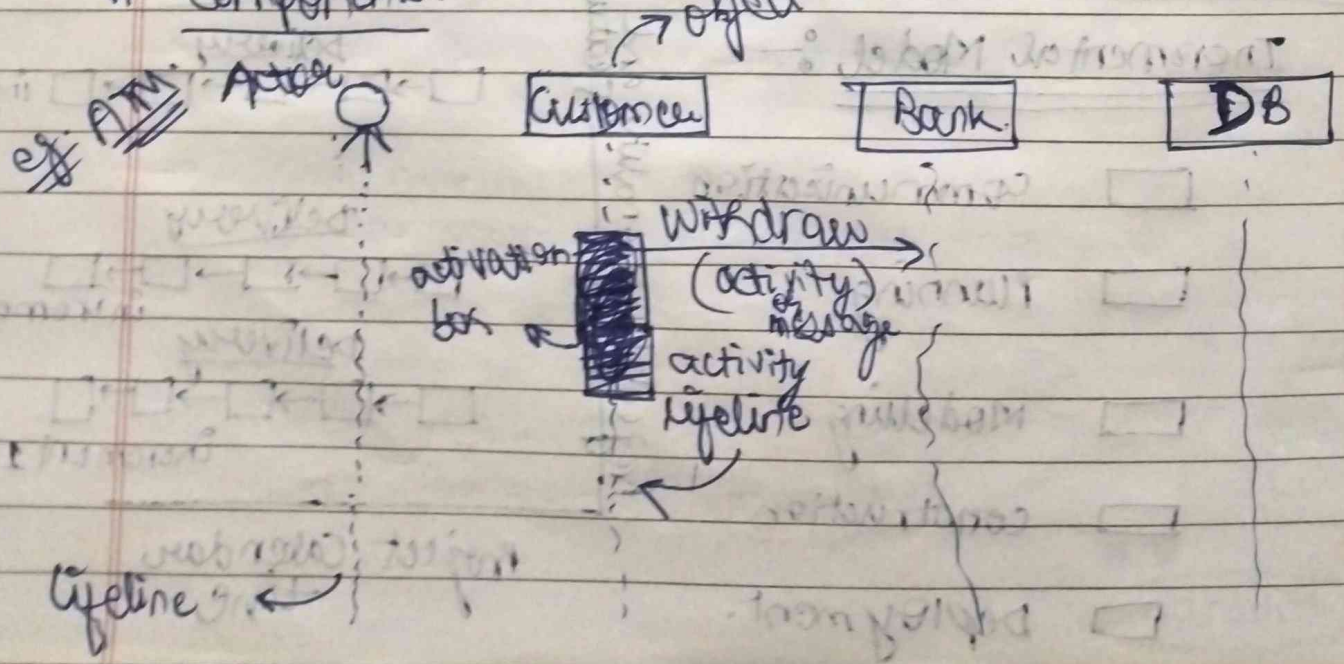
4 Approaches
- Top-down
- Bottom-up
- Middle-out.
- Usecase driven

Sequence Diagram:- (Event Diagram / scenarios)

It is an interaction diagram used to show the interactive behaviour of the system.

It depicts interaction between the objects in a sequential order.

Components.



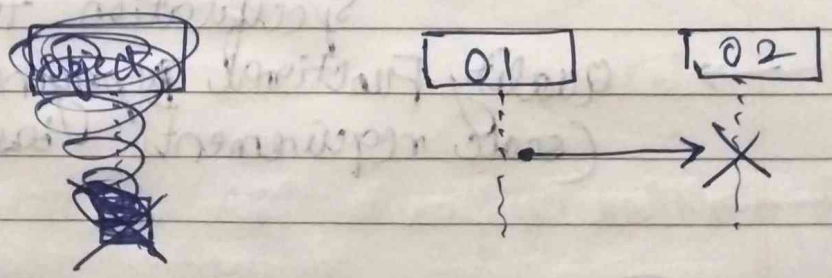
① Lifeline →
It is a named element which depicts an individual participant in a seq. diagram. So, each instance in it is represented by lifeline.

② Activation box →
Activates an element for a particular amount of time.

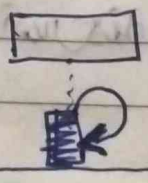
③ Message →
communication between objects depicts a message
a) Synchronous: (→)
Waits for a reply before the interaction can move forward.
b) Asynchronous: (→)
Doesn't wait for a reply from the reciever. The interaction moves forward irrespective of the reciever processing the previous message.
c) Create message:
Used to create a message to create an object in the sequence diagram.

(<<create>> ⇢ message.)

d) Deleting an object:



e) Self-message:



f) Found message:
Unknown source was found.

Q) Create a sequence diagram for an emotion-based song playing application.

Requirement Analysis :- ① Feasibility study.

② Requirement gathering →
- Requirement elicitation.

③ Requirement analysis

④ Requirement documentation / specification

⑤ SRS (Software Requirement specification) / Requirement review.

⇒ Requirement Engineering.

(can succeed only through an efficient customer - developer partnership)

(It is the activity that helps to understand the problem to be solved)

★ Requirement elicitation Methods

→ Usecase approach
→ Interview (one to one talk)
→ Brainstorming (group discussion)
→ Delphy technique. (write on a single piece of paper)
→ FAST (Facilitated Application Specification technique)
→ Quality Functional Deployment.
(scale requirements based on quality)

Requirements are gathered by writing down the key points and discussion.

(a) Usecase approach (describes 'what' not 'how')

→ Uses a combination of text and pictures in order to improve the understanding of requirements.
→ Functional view of system.

(b) Interview.

→ objective is to understand the customer expectation from the s/w.
→ may be open ended (like discussion) or structured (predefined agenda).

(c) Brainstorming.

→ Group discussion that may lead to new ideas and promotes the creative thinking.
→ All participants can share their ideas.

(d) Delphy Technique.

→ write requirements on a single piece of paper.

(e) FAST

→ similar to brainstorming.
→ objective is to bridge expectation gap between the developer and customer.
→ Each FAST attendies are asked to make.

(IN) • part of environment that surround system.
(OUT) • " " " " is produced by system.
(USER) • " " " " used "

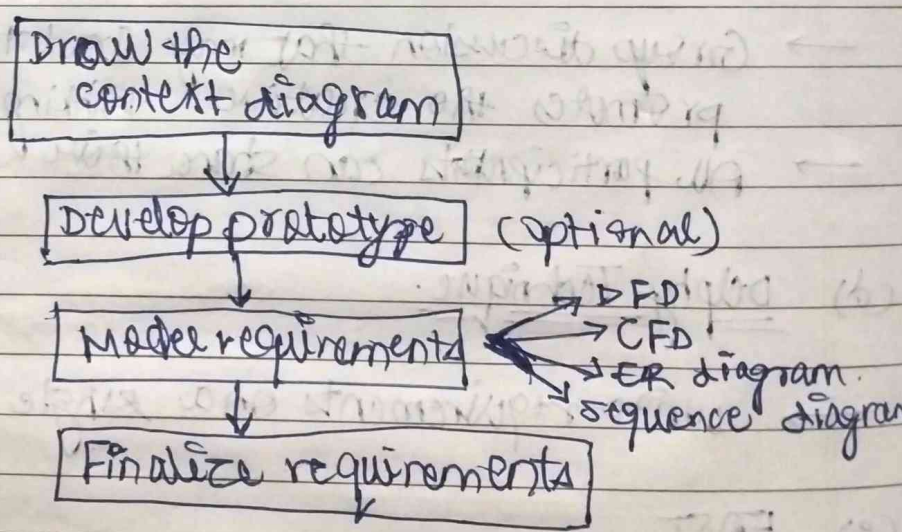
verified

(f) Quality Functional Deployment.

→ Quality management technique.
→ Incorporate voice of customer.
→ A value indicating degree of importance is assigned to the requirements.
→ Determines importance of each requirement on a scale of 1-5.

1	2	3	4	5
Not practical	Not Imp.	Not Imp. but nice to have.	Imp.	V. Imp.

* Requirement Analysis



→ It is after the requirement elicitation.

→ We analyze, fine and scrutinize the gathered req. in order to make unambiguous and consistent requirements.

① Draw the context Diagram.

→ System overview
→ High level view of system.
→ It is a simple model that define the boundaries and interface of the proposed system with external world.

② Prototype

→ Developed quickly and at a relatively low cost to better understanding of requirements.

③ Model Requirements.

→ This process usually consists of graphical representation of the functions, data entities and external entities and relationship b/w each other.

→ DFD, ER diagram, State Transition diagram.

④ Finalized Requirements

→ Inconsistencies and ambiguities are removed in this stage.

* Requirement Documentation/Specification.

→ consistent format for representing req.
→ SRS

Imp: IEEE Format for SRS (IEEE 830).

→ Standard for SRS document
→ Using it, we can a SRS document more readable, identifiable and universally acceptable.

1. Introduction
   - 1.1 Purpose
   - 1.2 Intended audience.
   - 1.3 Definition, Abbreviations and Acronyms
   - 1.4 References [redacted] / contact information
   - 1.5 Overview

2. Overall description
   - 2.1 Product Perspective.
   - 2.2 Product Functions
   - 2.3 User characteristics
   - 2.4 General constraints
   - 2.5 Assumptions and dependencies.

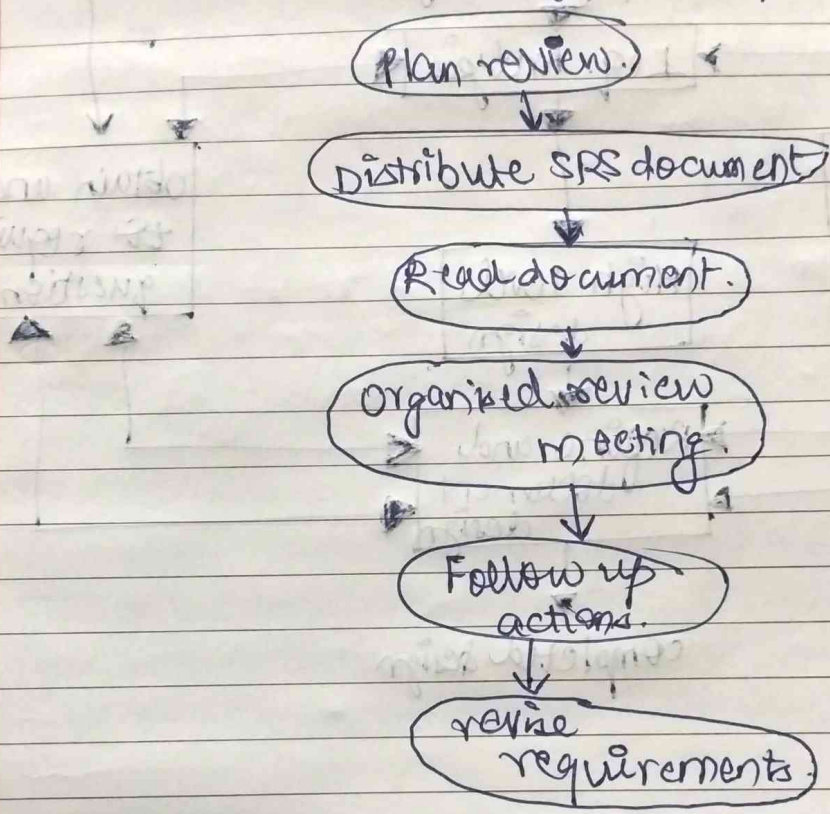
3. Specific Requirements.
   - 3.1 External Interface requirement.
   - 3.2 Functional requirements
   - 3.3 Performance
   - 3.4 Design constraints.
   - 3.5 Logical database
   - 3.6 Software system attributes.

4. Change Management Process.

5. Document Approval.
   - 5.1 Diagram, Table and Flowchart.
   - 5.2 Index

* Requirement Review.

→ validation technique.
→ check the document for completeness, consistency, tactical errors, ambiguous req., req. conflicts.



Design :- [Modularity] → modules.

Data
Stem control
External
common
content

→ coupling (Intermodular communication | dependency)

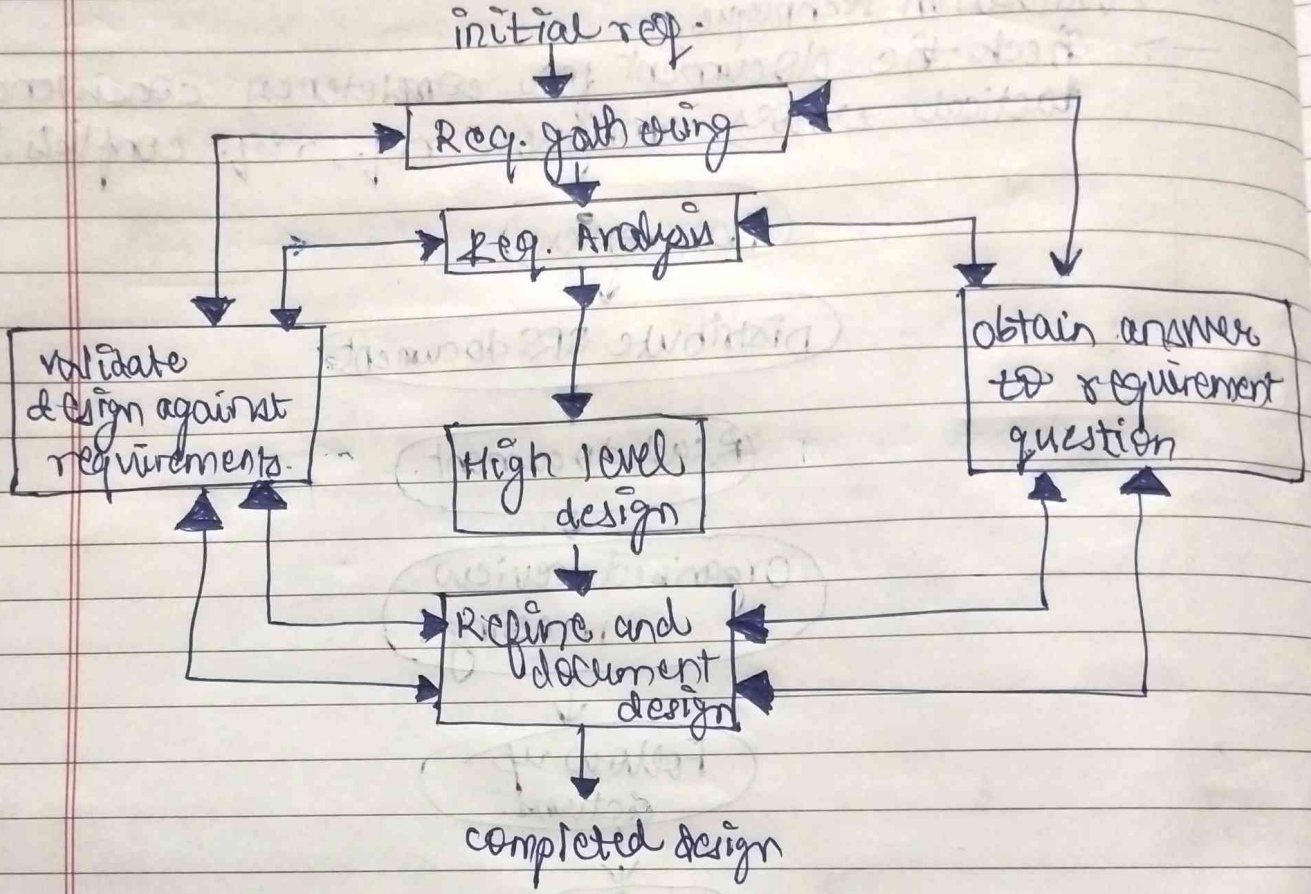
1. Data coupling (weakest coupling)
2. content " (strongest coupling)
3. Stem "
4. External "
5. common "
6. control "

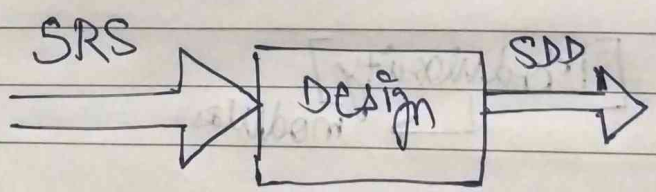
Weak coupling is good coupling

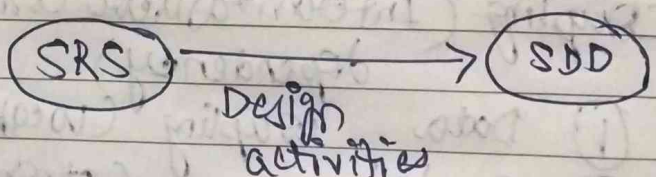
call by value → good coupling

call by ref. → bad coupling

Cohesion → Intramodular dependency.







→ Output of software design process is process diagram and detailed description of functional and non-functional requirements.

→ Software design is a 3 step process —
1. Interface design.
2. High-level or preliminary design / architatural design
3. detailed or low level design

→ Items designed during software design:—
- Module structure.
- control relationship among modules.
- Interface among modules (Data item exchange)
- Data structure for each module.
- Algorithm for each individual module

→ Good software Design.

1. Easily understandable.
2. Easy to change
3. Efficient.
4. Implement all functionality of the system correctly.

(1) Interface Design.

⇒ We treat system as a black box and don't focus on how the system will function / is implemented.

(2) High Level Design.

⇒ Major modules that must be implemented and what are their responsibilities and how they will communicate with each other.

③ Low-level design

⇒ Internal functions of each module.
⇒ Data structures, algorithms, functions of each module is defined properly.

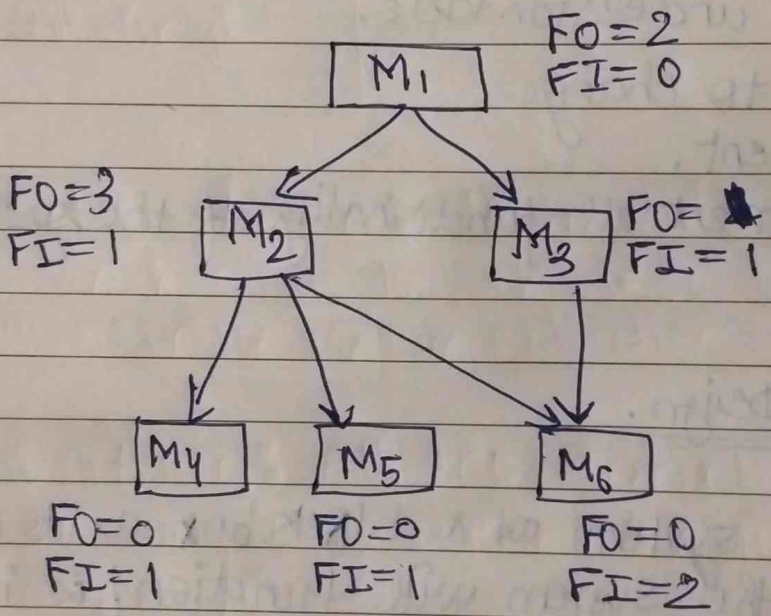
Fan-in → • Indicates how many modules directly invoke a given module.
• High fan-in indicates code reuse (encouraged)

Fan-out → • It is a measure of no. of modules directly controlled by a given module.
• Low fan-out is encouraged.

⇒ Internal functions of each module.
⇒ Data structures, algorithms, functions of each module is defined properly.

Fan-in →
- Indicates how many modules directly invoke a given module.
- High fan-in indicates code reuse (encouraged)

Fan-out →
- It is a measure of no. of modules directly controlled by a given module.
- Low fan-out is encouraged.



Cohesion :-

→ Cohesion is a functional strength of a module.
→ A cohesive module performs a single task or function.

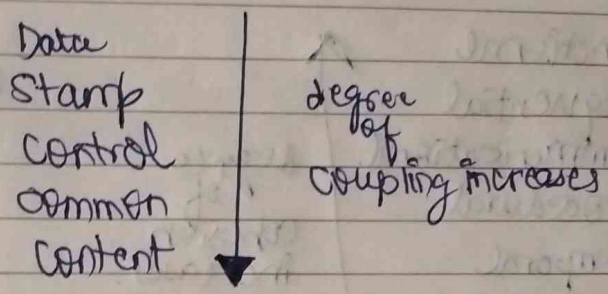
Coupling :-

→ It is a measure of interdependence or interaction b/w the two modules.

Functionally independent :-

→ A module having high cohesion and low coupling is called functionally independent.

→ Functionally independent modules need very little help from each other, ∴ has minimal interaction.



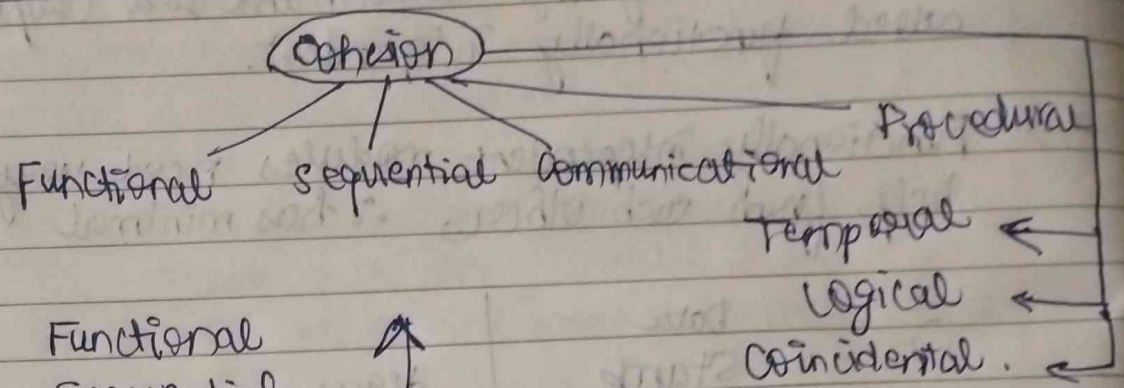
* **Data coupling.**
Two modules are data coupled if they communicate via parameter (like elementary data item). Data item should be program-related not control related.
eg. call by value.

* **Stamp coupling.**
Two modules are stamp coupled if they communicate via composite data item (data structure).
eg. call by reference.

* **Control coupling**
Data from one module is used to direct execution in another module.
eg. a flag set in one module and tested in another module

* **Common coupling.**
Two modules are common coupled if they share some global data.

* Content coupling.
Modules are content-coupled if they share code.



Functional
Sequential
communicational
Procedural
Temporal
Logical
Coincidental

degree of cohesion increases.

→ Coincidental cohesion: The module perform a set of tasks which relate to each other very loosely. (Modules contain random collection of functions)

→ Logical cohesion: All the elements of a module perform similar operation [example: Data input, data output]

→ Temporal cohesion: The module contain operations such that all the operations are executed in the same time span.

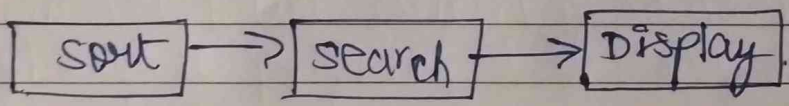
```
module initialization {
    initializevariables();
    checkmemory();
    checkports();
}
```

→ Procedural cohesion: The set of functions in a module that follows a procedure / are a part of algo.

certain sequence of steps are to be carried out in a certain order for achieving an objective.
eg. decoding of a message.

→ communicational cohesion: All functions of module reference or update the same data structure.
eg. Stack module.

→ Sequential cohesion: Element of a module form diff. part of a sequence. Output from one element is the input of the sequence's element.

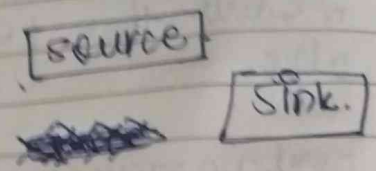


→ Functional cohesion: Different elements of the module cooperate to achieve a single function.
eg. We can describe the module in a single sentence.

How data flows in a system

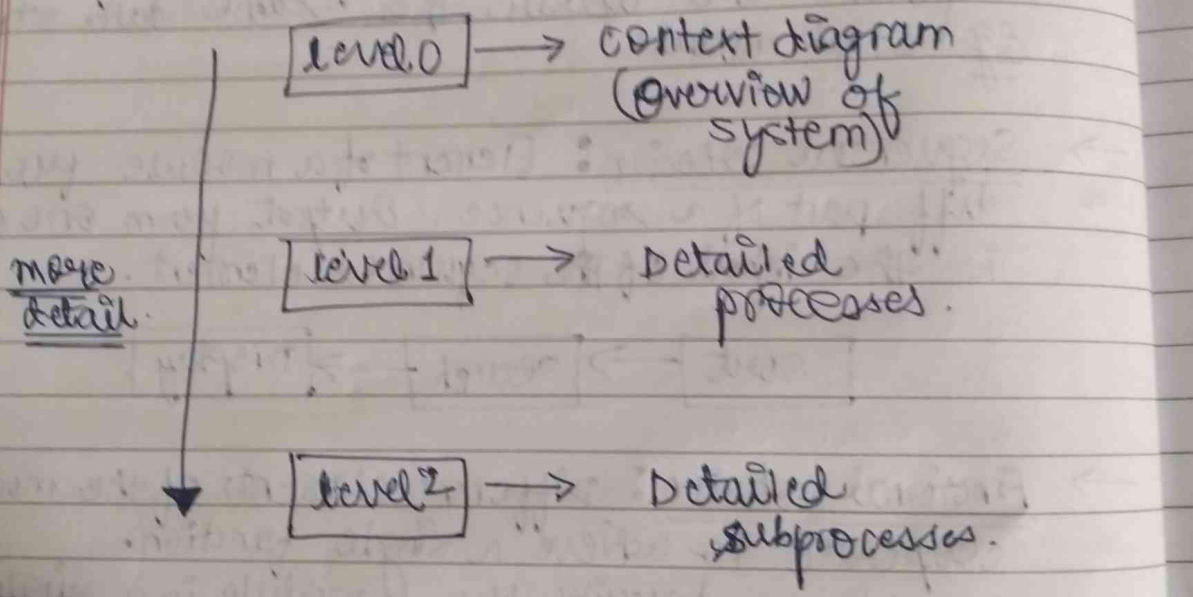
DFD ↔ Flowchart → Buddechart

Data Flow Diagram :-



1. source/sink
2. Data Flow (→)
3. Process

4. Data store (DB) (Warehouse) ═══



Agile

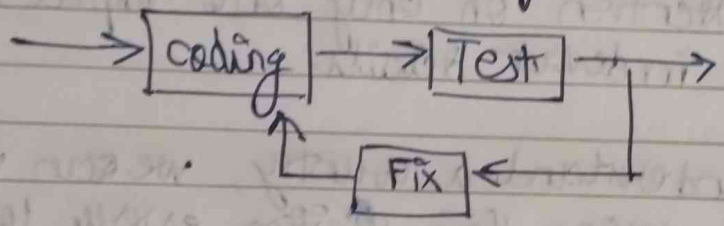
Imp Ques.

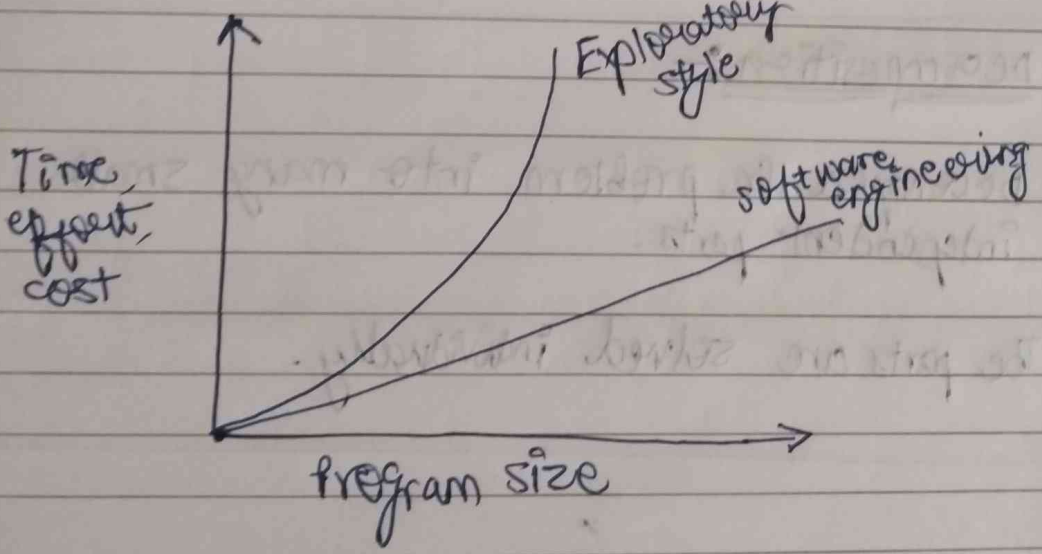
- → Burn up (work done)
- → Burn down. (Work remaining)
- → velocity of sprint (speed of doing work)
- → Diff. b/w incremental and iterative delivery
- → sprint backlog and product backlog.
- → Types of agile methodology.
- → responsibilities of product owner, Scrum master, dev. team.
- → Daily standup meetings.
- → what type of project is suitable for agile.
- → 4 values and 12 principles.

→ Epic, User story and Task.
→ Advantage and disadvantage of using agile.
→ Diff. b/w traditional method and agile methodology.
→ Sprint.

Exploratory software development.
(Build and fix)

→ Bugs are fixed when they are rectified or noticed.





Why software development curves linear?

Chunking:
- chunking by groups
- semantic chunking
- chunking by patterns

eg. (phone numbers)

⇒ grouping data into chunks.
eg. encapsulation.

Abstraction:

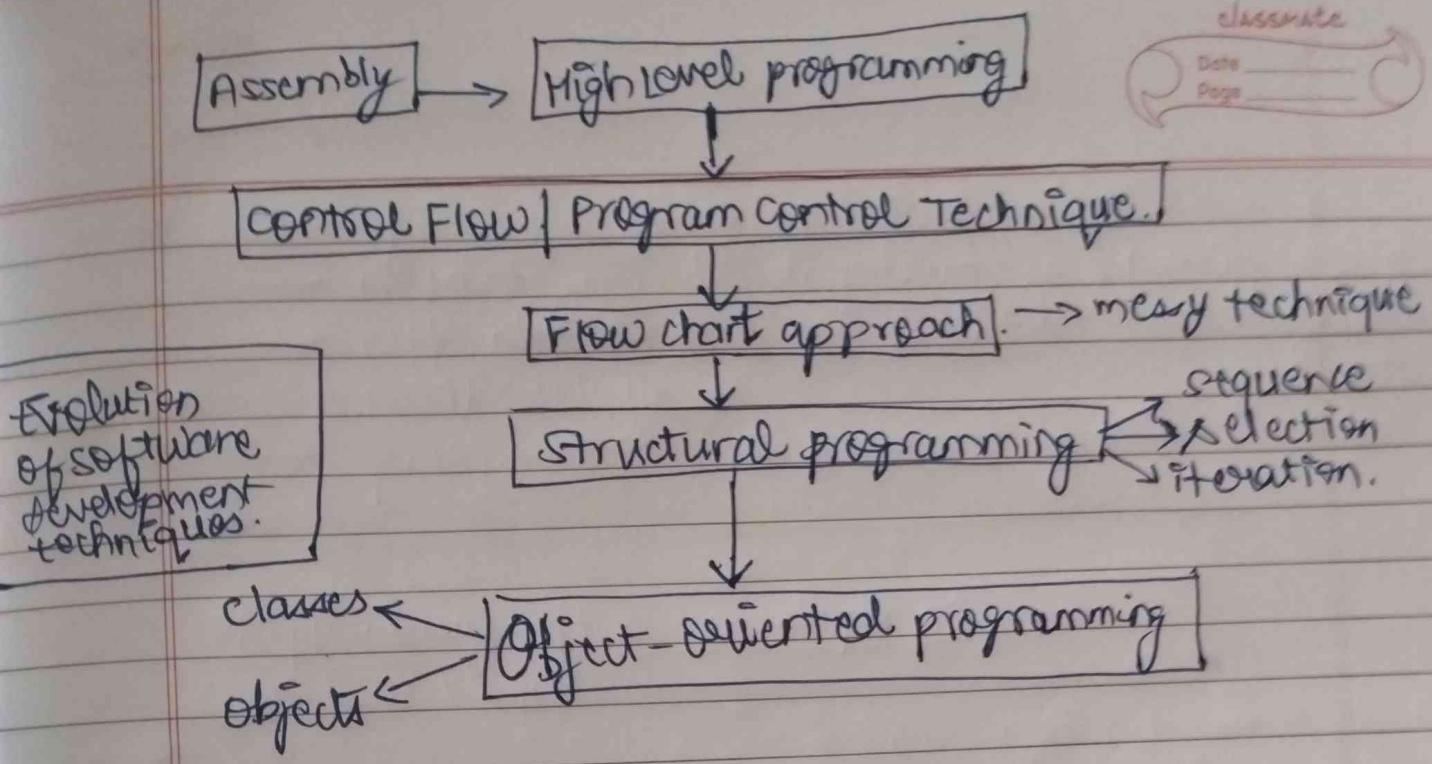
⇒ Hiding unnecessary details.
⇒ Also called "Model Building"
⇒ Focused attention on only one aspect of problem ignoring irrelevant details.

eg. To understand a country, we can use a map rather than visiting every location.

Decomposition:-

⇒ Decompose the problem into many small independent parts.

⇒ The parts are solved individually.


Assembly
High Level programming
CONTROL FLOW | Program control Technique.
Flow chart approach. → messy technique.
Structural programming
sequence
selection
iteration.
Evolution of software development techniques.
classes
objects
Object-oriented programming