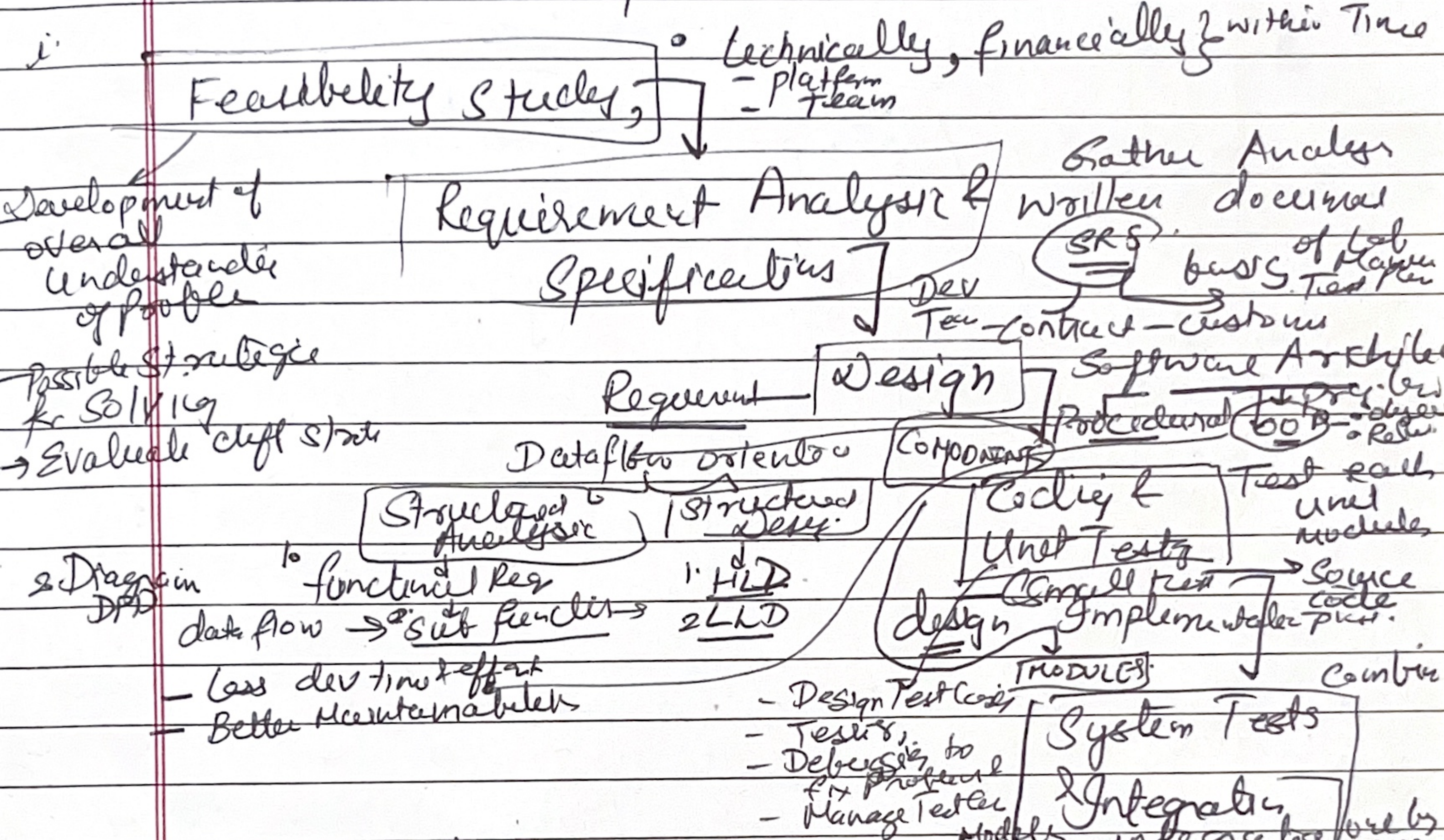


Q. No.

1 min

Classical Waterfall Model

SDLC - all phases are same in all modes.



Advantage	Disadvantage
- Basic	→ No feedback
Intuitively most obvious idea	Rigid - I will not change after goes out of phase
→ Simple	→ No Experiment / Change Request
Systematic - all steps	No Parallelism
Customer knows clear Requirement	Multiple Teams parallelly
Rigid (You can't change requirement)	High Risk
→ Small Project	No flexibility, no rigidity
	all problems you find in last phase.
	Inefficient Error Check
	Maintenance
	60% effort
	problem - bugs
	feature extension
	You enter house & start maintaining
	In between some feedback
	to remove bugs
	Extra Cost.

Iterative

Waterfall Model

Feasibility studies

→ Error propagates to further phase
I can give feedback to previous step
BUT NOT IN Feasibility studies

Req. Anal & Spec

Design

Coding & Unit Test

System Test & Integrate

discuss design
Base Model

check this document
no correct

How careful a programmer might be he can commit 4 mistakes

Error → feedback - Base

Advantage: Detect bugs

In some phase in which they are committed. Reduces time & effort - Error

Advantages

* Base Model

* Simple & Easy

* Small Project

* Feed back

Phase contains most of error

Detecting error close to point of commitment -

Inspection of errors are committed
Best Effort, errors escape detection in next phase already completed phase

Disadvantages

Slow code Fast code - BLOCKED PHASE

③ * No phase overlapping - P1 over P2

② * No intermediate delivery - Product delivery increment

① * Rigidity in b/w Req don't change (Problem) exposed in the end

* Less Customer Involvement

V-shaped Mode

- Also known as Verification & Validation Mode
 (Testing) Final product
 Right Proceed
 Step by Step ~~check~~ check own
 artifact is correct
 waterfall SD
- Extension of Waterfall Mode

- Testing is associated in every phase of life cycle
 1 unit - test
 2, 3 unit combine - test
 full system - test
Parallel Testing Full Build

- Verify — 1. Soft Req. Analysis, 2. System Design, 3. Architecture Design, Sub Module Design.
 Modules Floor Design By Room, Kitchen at each floor

- Validate — 1. Unit Testing → 2. Integration → 3. System
 1 Room Full Flow Full Build

- 1) Functions
- 2) Nonfunctional Req of SRS
- Acceptance Customer (User)
- are working as meet or fulfilled

Req. Analysis &
Specs

System Test Des

High level des
Integrates Test Des

Code's
Unit Test Des

System

Integrates

Unit Test

Adv

Time Save

TESTING IN EACH PHASE

(11 testing)

Concurrent

Good understanding of Project in Beginning
Req. → Testin Phase

Every Component must be testable

Progress tracked easily

Proactive defect tracking
(Not Reactive)

Disadv

No feedback

Iterative/Incremental

No Risk analysis

Evolutionary

Not good for Small Projects
Big or Object driven
Easy for Skilled Team
Urban Hospital road &

Prototyping Model

Build Prototype of system before development of actual system

Dummy
Type

Crude implementation

Requirement Gathering

Build quickly, shatters

Dummy functions

Rapid Prototyping

Quick Design

Refine Requirements

Incorporating

customer suggestions

Build Prototype

Customer
Evaluate
problem

Got part
of
S/W
system

Acceptance by Customer

Design

Implement

Test

Monitor

is
almost
always
developed

users
prototype Technical Issues

Req = UNCLE

Technical Issues

Solution

Customer Requirement

Technical Issues
Higher cost

Threats

Prototyping

Iteration

• Requirements which are related to functional working aspect of software, fall in this category. Can - $\pi(\alpha) \leq \Sigma$ 1, 2, 3 gear, Brake Exh

- Security (Can crash test) spw launch, hook, DOS attack

Configuration: Which type of S/W, DB, Server ^{old} - Fast
Performance Slow ← Respons ^{Time}

Cost - Rival has low cost Eg ~~Elm~~ ^{Elm} ~~Heck~~ ^{Heck}
 ~~Cost~~ ^{Cost} $\rightarrow$ NASA

Interdependence lib. - Comm. with

Interoperability - Comm. with other systems
Flexibility - Scalability

Disaster Recover - Problems - Outage - 10 days
Sheet 100

Google Down for 10 min

Loss of Revenue $\rightarrow$ Trust Creditable

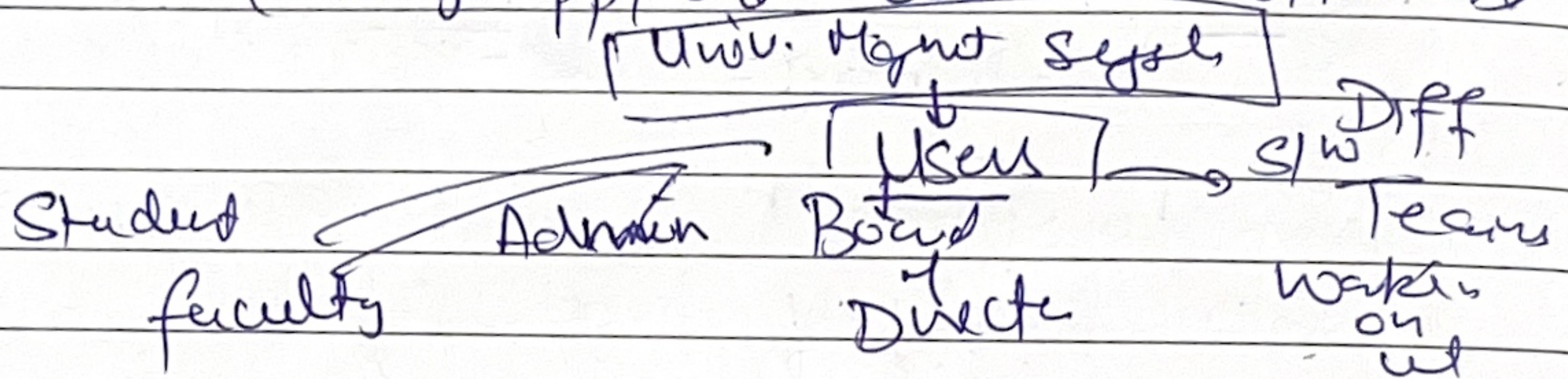
Accessibility - Valid users are access
DBA Low level HOT Data

Software Requirements Specification

Date: _____
Page No.: _____

1. SRS is description of a software system to be developed (before entering Design, Technical documents)
(Many ppl are attached with SRS)

E.g.



Doc — all things are well defined
↑
all CONSISTENCY

3. It lays out both functional & non-functional

4. It may include a set of Use Cases (situation) that describes user interaction that S/W must provide to user

Ex

Students
↓
forgot
password

Context
Diagram

Students
discuss doubts
Faculty
apply a lesson

SRS Structure

Date :
Page No. :

1. Introduction

- 1.1 Purpose — Why you are making Univ Mgmt Sys
- 1.2 Intended Audience → Student, faculty, Admin. who all will use.
- 1.3 Scope — in future what you will gain = The big market
- 1.4 Definitions → Principal Director Administrative. HOD of Dept
- 1.5 Reference → Taken help from some online other university or offline media.

2. Overall Description

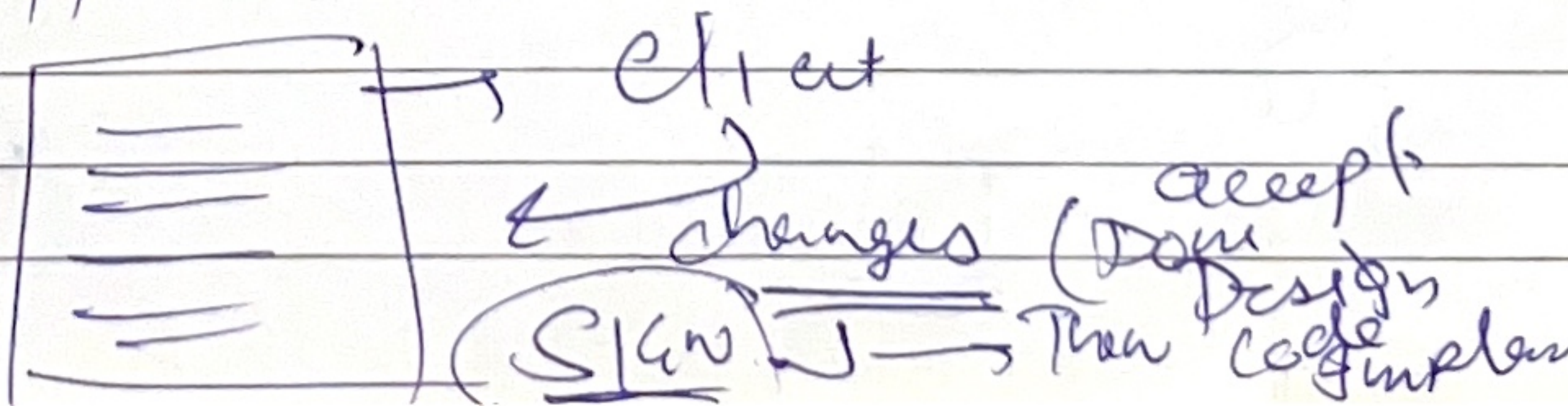
- 2.1 User Interface — Students Interface — Navigator which gives visible
- 2.2 Constraints, Assumptions, Dependencies HTTP, App, Check
- 2.3 User Characteristics → Student final year — participate in. HOD, Faculty, Payment — transfer. Power, Star

Communication of Front & Back

3. System Features & Requirement

- 3.1 Functional Requirements — Register for user ID
- 3.2 Use Case
- 3.3 External Interface → Password, Homepage, New
- 3.4 Logical Database → Third party, Student — Pay fees, Fees, Payroll
- 3.5 Measuring Requirements where data will be seen Storage Disk
- 3.6 Non functional RAM Safety, Security, Data, Browser, CSS

4. Deliver for approval



User Requirement

Web site
web applications
Date: _____
Page No.: _____
Flex Search
Interface
Windows, Mac
76% share 2-3%
Android - iOS

- Easy & simple to Operate
- Quick Response
- Effectively handling
- Customer Support

Operational errors

Payment Page -
Do not Refresh

5s - 5000, 5000
Focus on
Don't want to wait

how our system handles errors

Customer Support { Whatsapp
Call

Time Bound to be

✓ End User

Money Benefit = Client

Company - developer
taken his share

Online gateways

User Requirement Specification

1) URD — URS what the user expects the SW to be able to do

2) It is a CONTRACTUAL document

- Client can't ask Extra

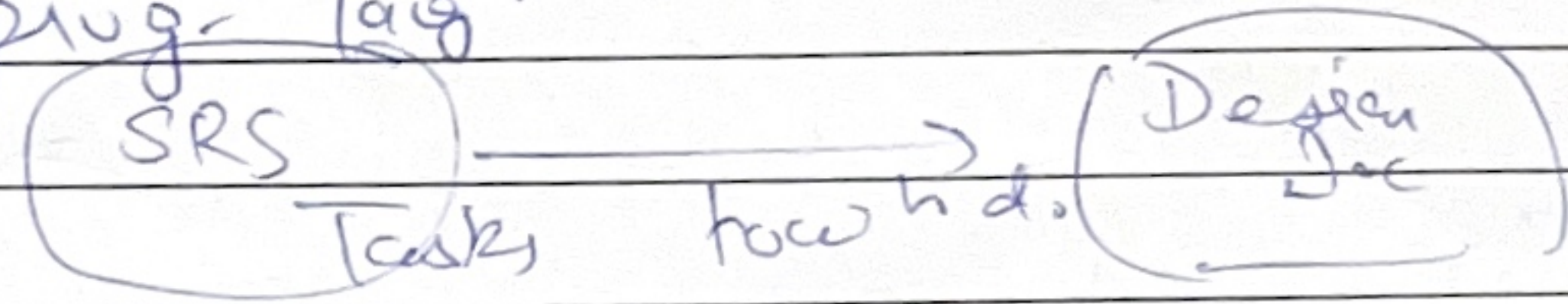
- If company says 1 feature else it is left

Relationship b/w 2 matters. Company can see

Quick in all phase Test & code

Design Phase = ^{Transfer} SRS (all problem) documents

into a form easily implementable in some prog. lang.



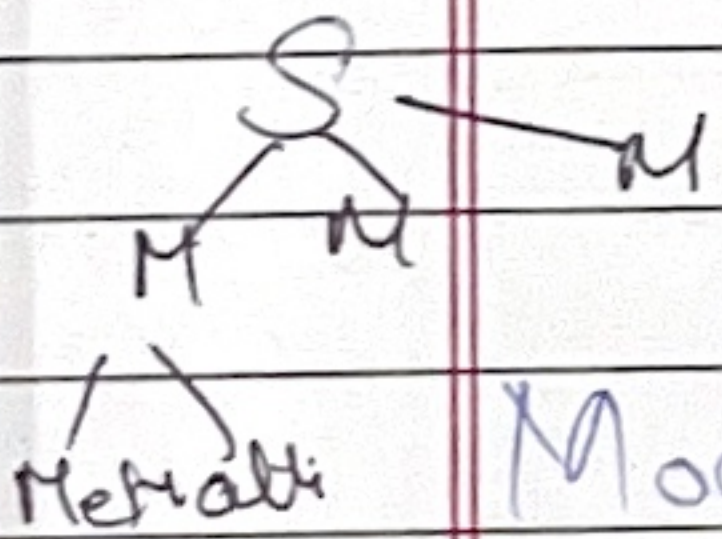
Module

1. Several functions
2. associated data structures design

Class = methods attributes

Cohesion

Coupling



Modularity - fundamental attributes of good ds

Decomposition of a problem clearly into modules.

→ Reduce dependencies

1. Modules are independent of each other.
2. Divide & Conquer Principle.

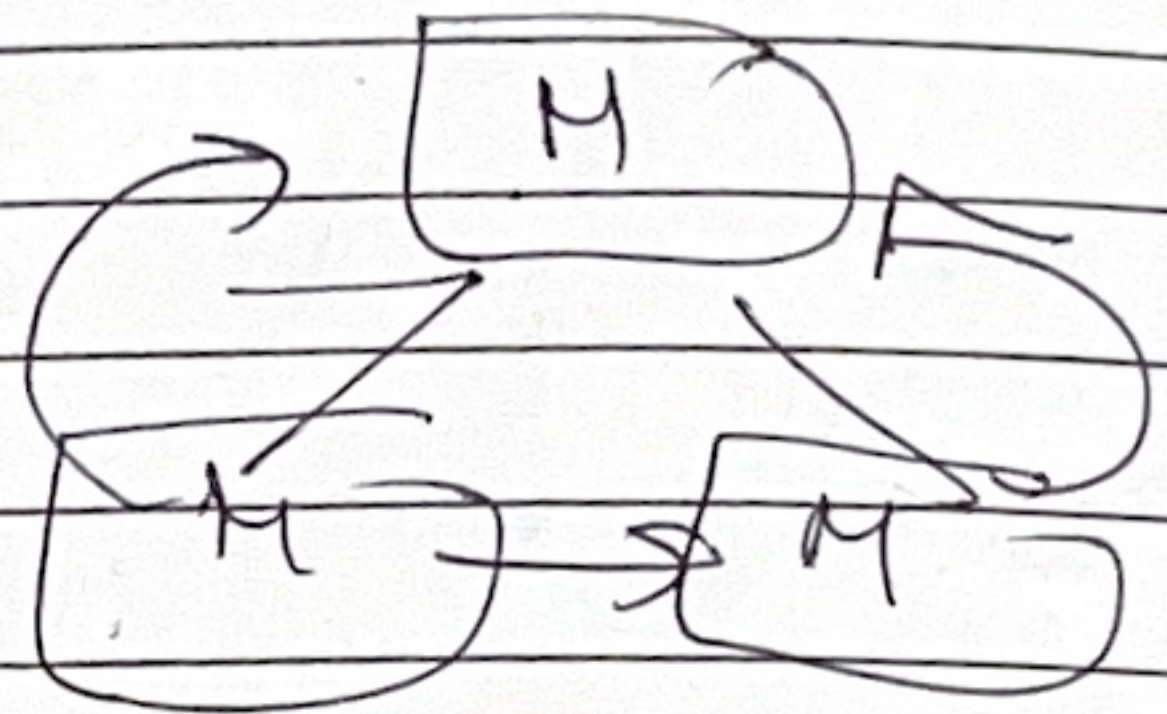
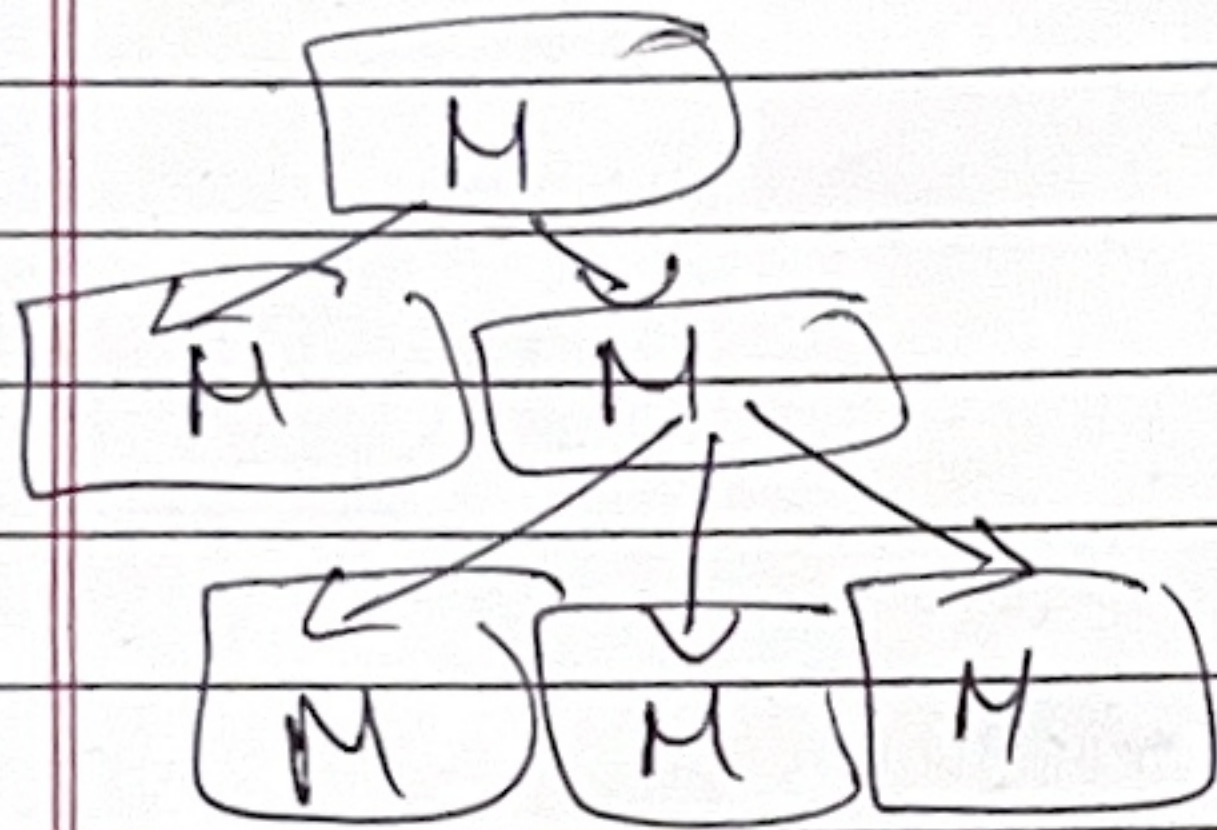
Small problem → small/pd solution → combine → final solution

If Modules are independent:

- modules can be understood separately
- reduces complexity greatly
- to understand the

Bunch of sticks — difficult to keep individual sticks

Cleanly & Non cleanly decomposed Modules



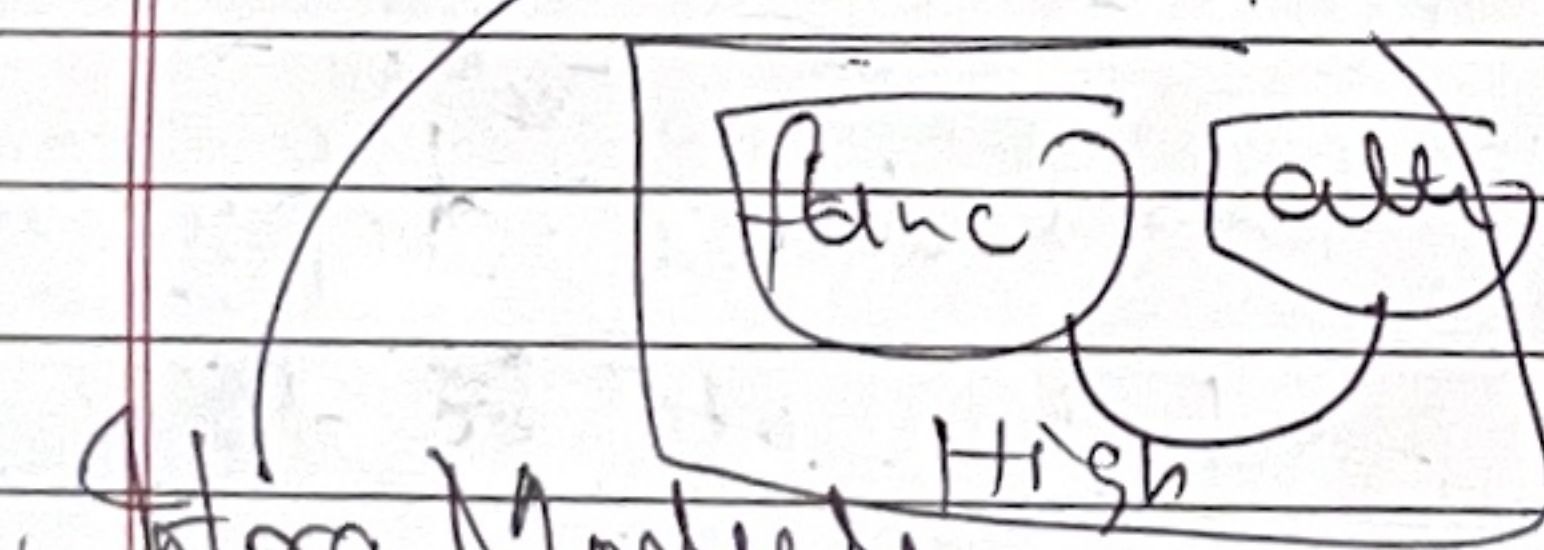
Modules should display

High Cohesion

— Togetherness

— within a module,

functions & attributes

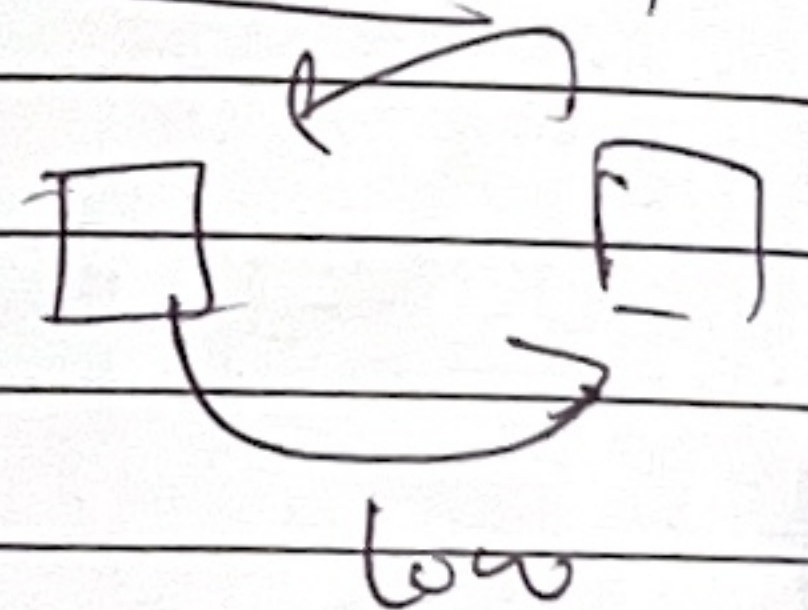


Intra Module

— Cohesion is a measure of functional strength of module

— It performs single task & function

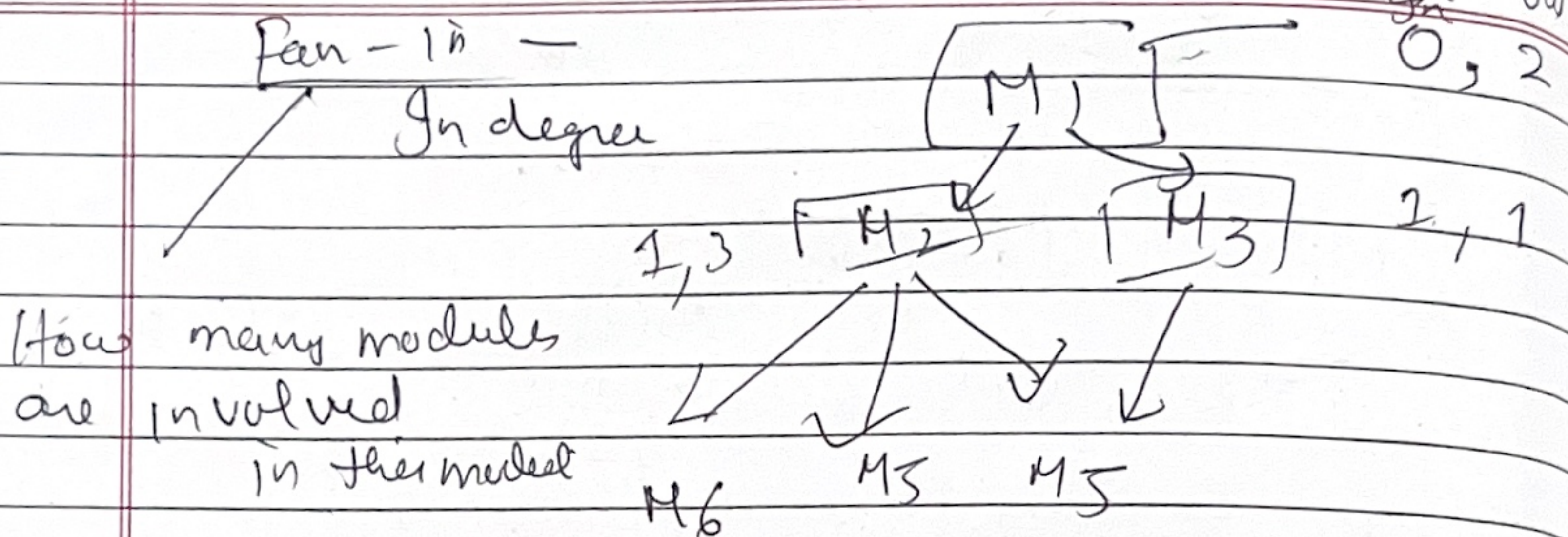
Low Coupling



Inter module

— Coupling b/w 2 modules

— measure of degree of interdependence or interaction b/w 2 modules



Fan-out → No of modules directly controlled by given module

Object Modeling Using UML

Date:
Page No.:

Object Oriented S/W development style - popular
oo Programming lang oo Design Skill

CASE Tool
generate code from design

first ^{good} design
then code

What is Modelling

Model is constructed from Problem
by focusing only on few aspects of problem
& ignore rest
The Model of a Problem = Analysis Model

But the Model of the Solution (code) = Design Model
by carrying out iterative refinement to the
Analysis model using Design Methodology

How to document a Model using Modeling Language
Set of notations using which
design & analysis models
are documented.

Methodology

Design Process — Given a problem description,
how to systematically work out the design
solution to the problem.

Step by step procedure — using which problem
description can be converted into design sol.

Basis Object Oriented Concepts

1. Abstraction

4. Method Overriding

2. Polymorphism

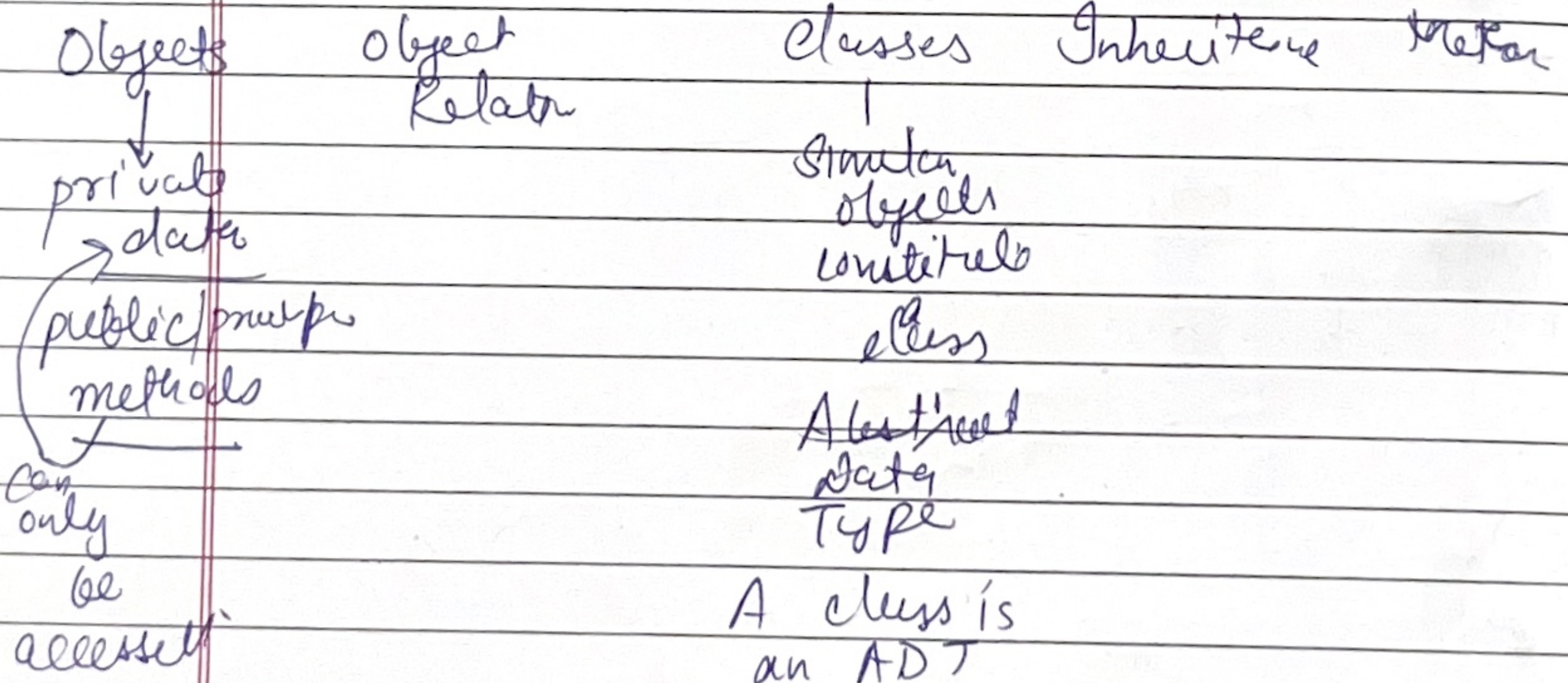
5. Generics

3. Encapsulation

6. Composite Objects

Concepts

Mechanisms



No object can access data of any other directly but

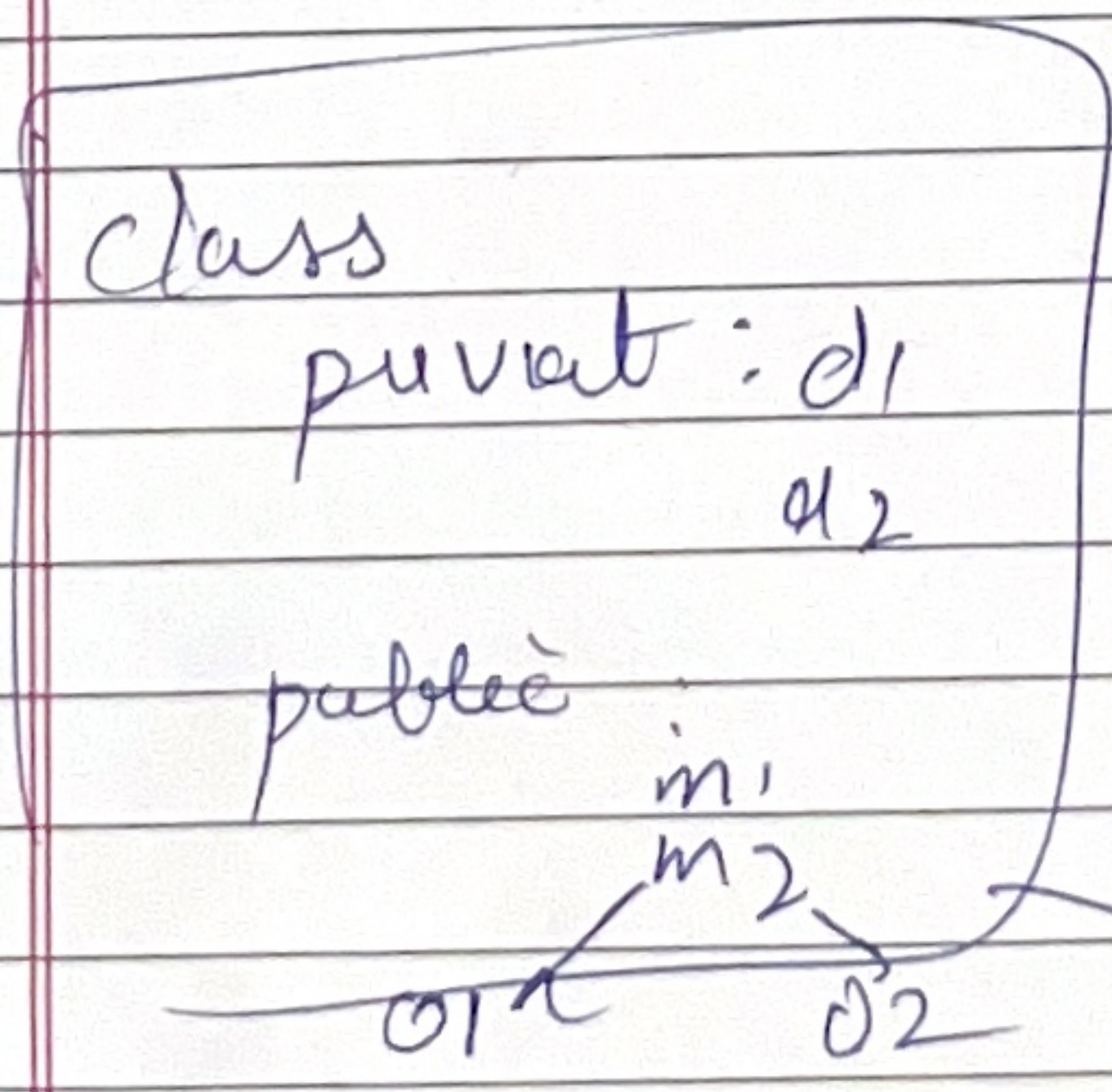
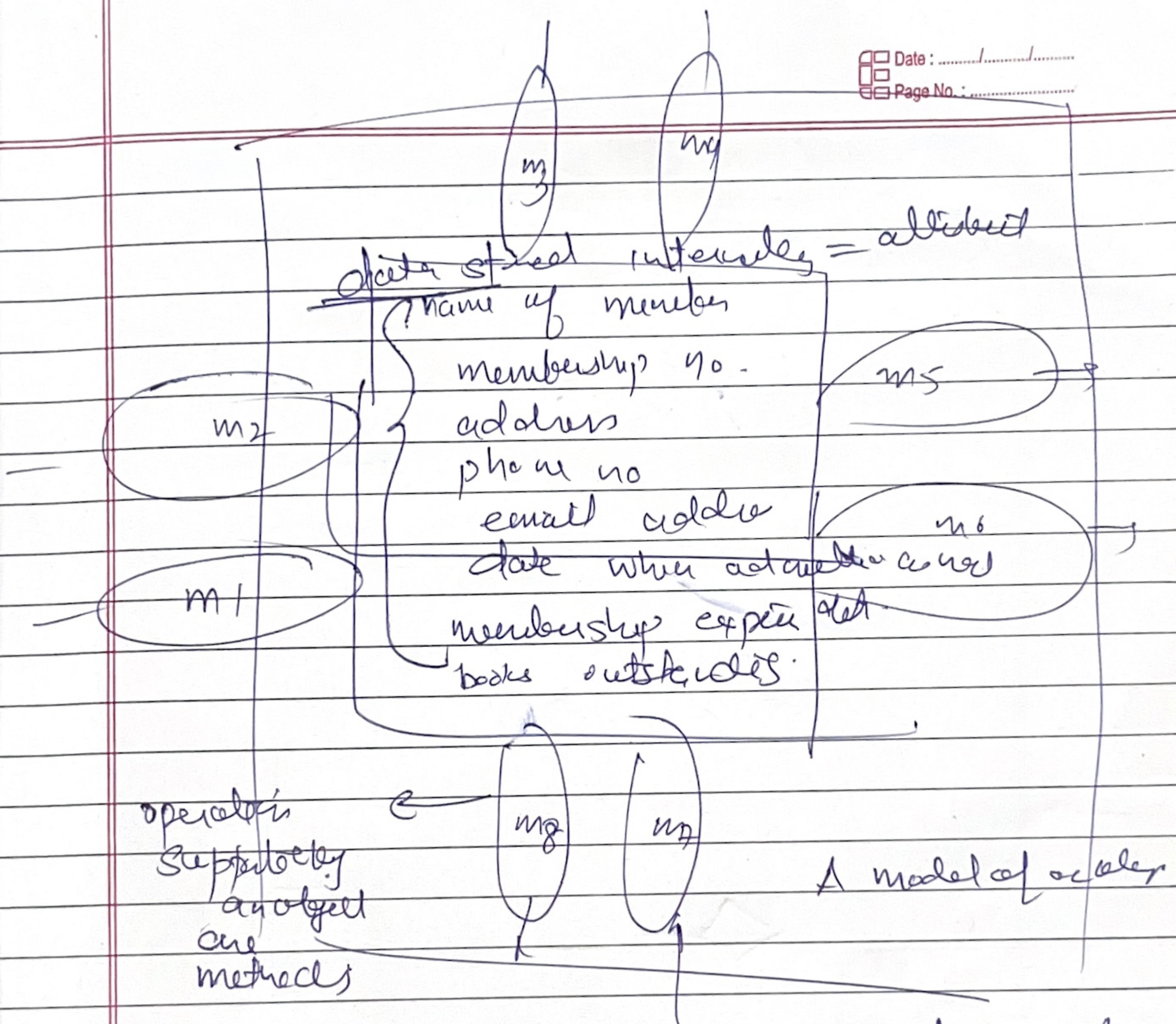
Abstract Def :

data accessed through methods
data stored internally
stack access

by invoking methods
Principle of Data Hiding or Abstraction

Data Type = variable
into collection of values or objects
class = objects

High Cohesion Low Coupling



create - method
 & issue book
 & return book

create

int create()

int create(int)

int create(float x, float y)

int create...

Implementer of a Responsibility Overloaded
 of a class through multiple methods with same method name = Method Overloading

1

Class Diagram

Structural Diagram $\rightarrow$ S/W Proceed

Structure of all component

oops

real life objects

class

attribute

method

Bank

Composition

Aggregation

Branch

has

Account

— name
— location
— code

+ add Branch()
+ remove Branch()
+ get Branch()
+ get All Branch()

branch code
city

+ add Account()
+ remove Account()

— acc no
— balance

+ deposit()
+ withdraw()
+ add loan
+ remove loan

Verification
Author

Generalization

Loan Account

— loan no
— amount
— type

Saving Account

— min - balance
— opening - date
+ add Customer()
+ remove Customer()

Current Account

— int - rate
— opening - date
+ add s

0..1

0..1

0..1

1

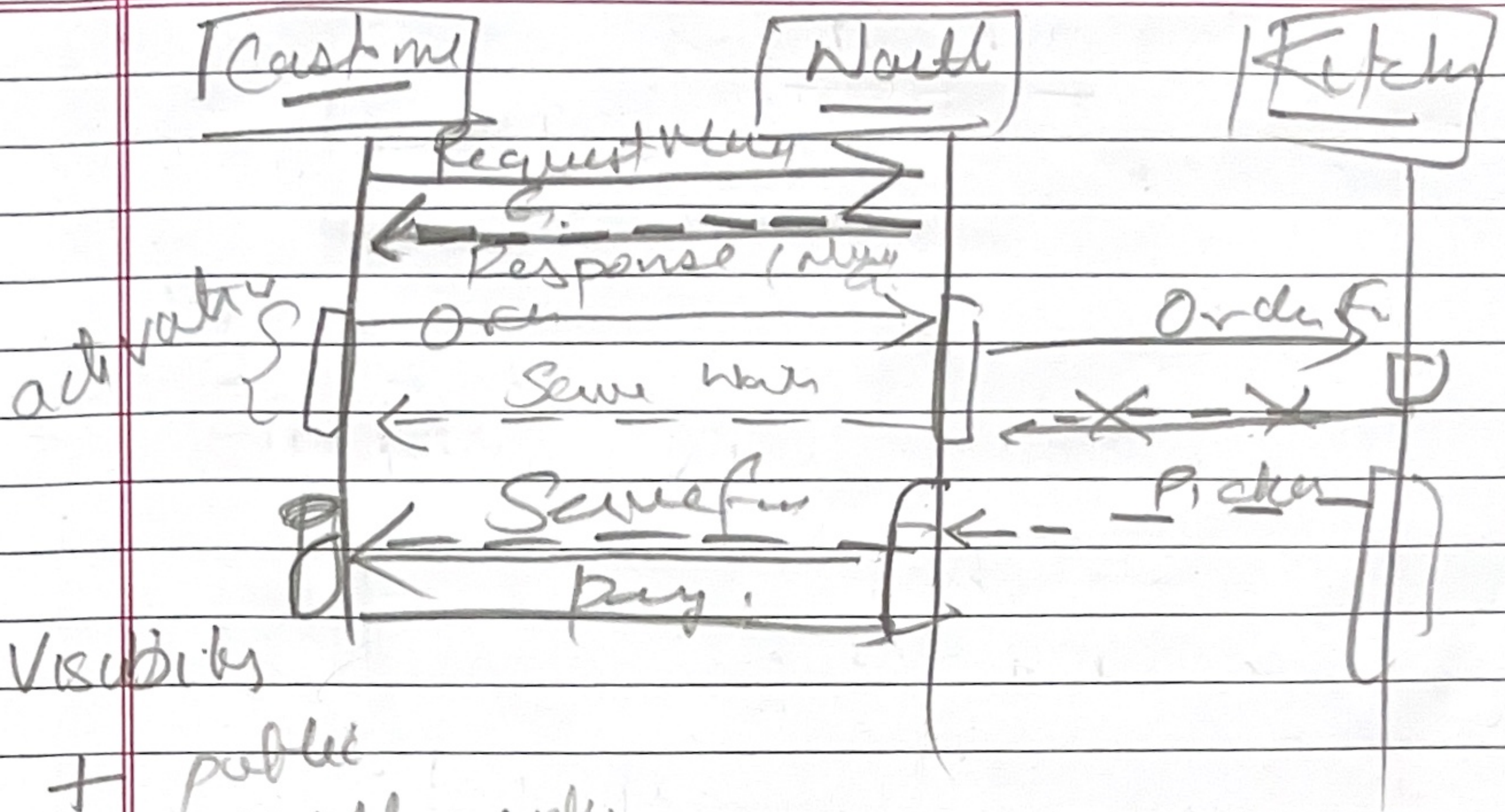
Customer

— cust ID
— name

1

Dynamic ~~of~~ ^{process} Behaviour ACTIVITY DIAGRAM Interaction = Message Passing
 b/w Multiple objects in Time

Date: / /
 Page No:



Visibility

- + public
- private → only within class
- # protected
- child can access

Parent → Inherited

inherits Power
 adds extra

child

Multiplicity
 diff classes associated

1 - 1 OR 0 .. 1
 1 - many 0 .. *
 many - 1 1 .. *

Use Case Diagram

- They model the behaviour of system
- They illustrate functional requirements of the system & its interaction with external agents.
- high level view of system w/o going into details

book my show

filter

Movie Name

Engles

Punjabi

Search

Movie 1

Movie

Date

Time of show

Book & Cancel

System

Bar

CONNECTOR

Search & Extra functionalities

Movie 1 Book

Movie 2 Check

Search & Adv Rep

Show Time

implied << includes >>

Bookings

Payment

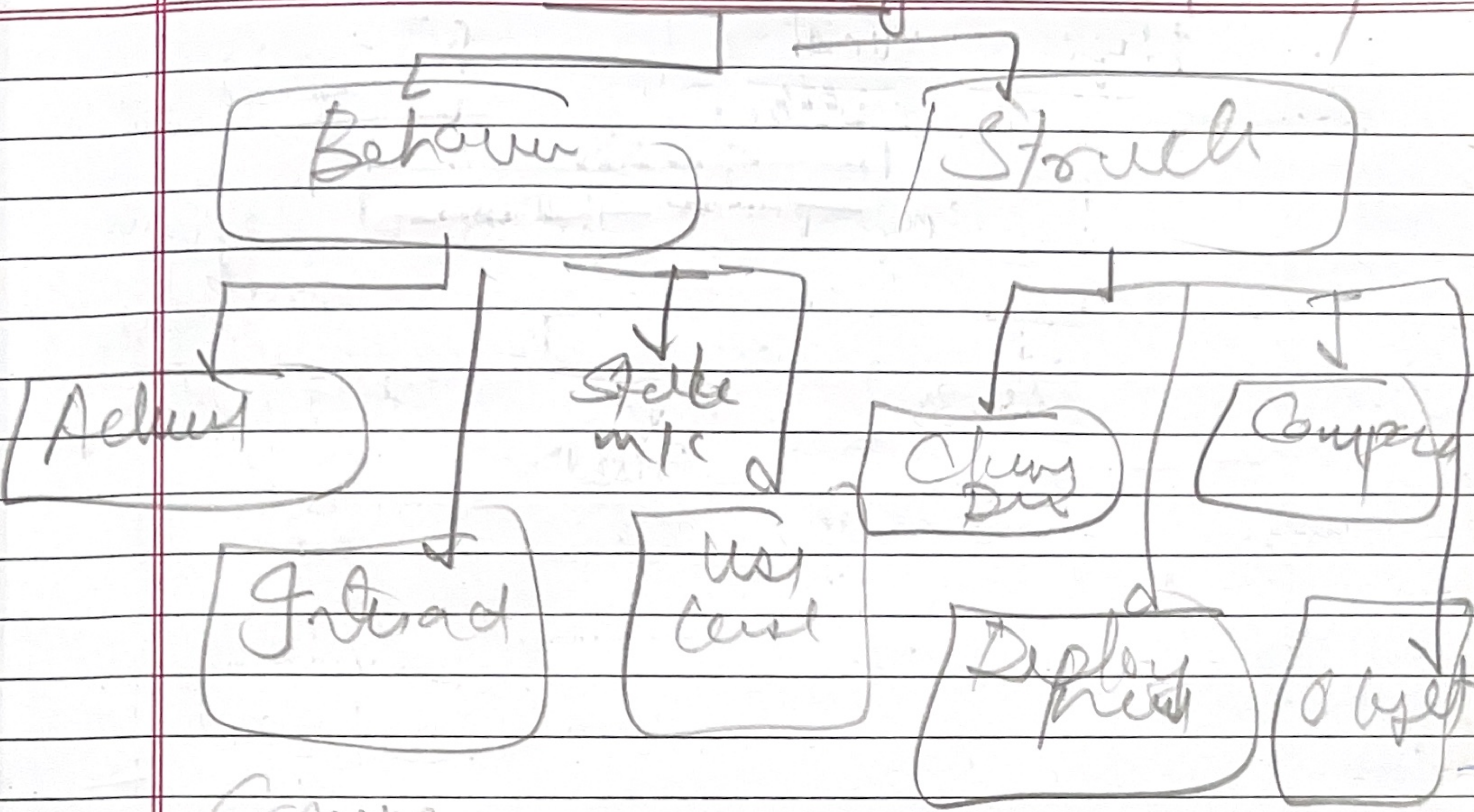
Cancel Book

login

necessary to do login (Email ID)

Customer
Actor
machine
90% of ppl

UML Diagram



Common
Seq.
Timing

Types of Testing / Levels

Date : _____
Page No. : _____

Software Testing

Coding Phase

Testing Phase

Maintenance Phase

Unit Testing

Integration Testing

System Test

Regression Testing

Module Testing

Bang Bang
Top-Bottom
Bottom up

Based on who is testing

Performance
Nonfunctional

Smallest entity

Lowest / first level

A Type

B gears

C engine

F123
F12
A1 F2 F3

combination
Module

Volume
Load
Stress

Security

Configuration

Compatibility

Recovery

Installation

Designing Test Cases } 2 approaches

White Box Black Box

New edits of Book

slow update

Changes Bugs New features

already exist app should not be impacted

White Box

Black Box

1. Developers perform white box tests

Test Engineers do Black box tests

2. What the S/W is supposed to do & how it does it

What S/W is supposed to do. But it is NOT aware of how it does it

3. To perform WBT we should have understanding of programming lang. Arrays Control structures

No need to know prog. lang.

We will look into the source code & test the logic of source code

Verifies functionality of application based on Requirements Spec.

We should know internal design of code

No need to know internal design

Test Design techniques

- Control Flow Tests
- Data Flow Tests
- Statement Coverage
- Decision Coverage
- Path Tests

- Decision Table Test
- All pairs Test
- Equivalence partition
- Boundary Value analysis

Can be mainly applied heavily at Unit Testing less at Integration

Cause effect graph

- Unit
- Integrates
- System Acceptance

Statement Coverage Technique

- It is used to design White Box Test cases.
Programmer knows Internal structure
- It involves execution of all statements of the source code at least once.
- Calculated $\frac{\text{Total Statement Executes}}{\text{Total Stmt}}$
- Detects Dead code, Unused Code, Branch

1	main()	n1	n2	n3	
2	{	30	20	10	
3	int n1, n2, n3;	10	20	30	
4	if (n1 >= n2 && n2 >= n3)				5/10
5	printf ("n1 is largest", n1);				30/100
6	else if (n2 >= n1 && n2 >= n3)				
7	printf ("n2 is largest", n2);				
8	else				
9	printf ("n3 is largest", n3);				30/100
10	}				8/10 = 80%

Called as Predecide Coverage in which each of the Boolean exprs have been evaluated as both true & false

• Head

