Software Engineering - It is an engineering approach to develop software.

Types - System - Application - Utility

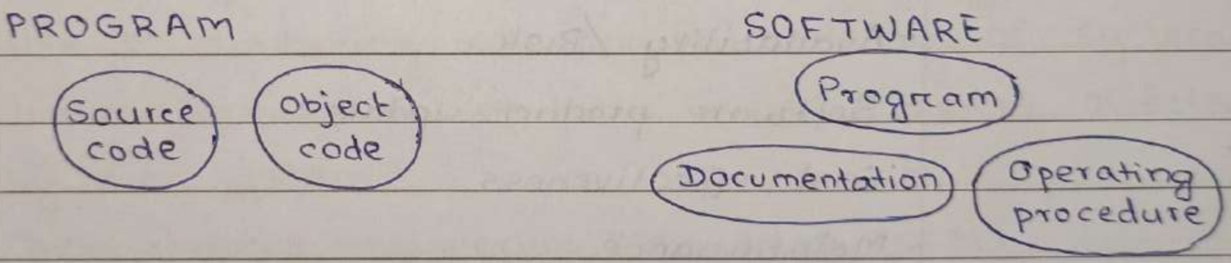
Role of software - is to perform task for the user by activating & controlling the comp. hardware.

Good software - Software which has no. of attributes which together can decide its nature.

For customer - cost effectiveness

For user - Usability and Reliability

For SEngineer - Maintainance & Efficiency



Program is a subset of software.

Program - set of instructions written for a specific task by an individual

- small in size with limited functionality

Software is a superset of program.

Software product - available in market. User friendly, portable, risk free, maintainable, cost effective. When demand changes w.r.t tools manufacturer need to modify the existing product.

Software crisis - "The term software crisis refers to a set of problem encounter in the development of software". "Inability to develop software on time, budget and within the requirement"

- Delaying process resulting out of schedule
- No proper methods to estimate software project
- Adequate communication bet^n user & developer
- Compatibility software
- Portability
- Documentation Staffing
- Statics & coordination
- Availability /Risk
- Software product update
- Cost effectiveness
- Maintainance

SOFTWARE ENGINEERING

It is a methodology that includes process/methods, tools and technique for the manufacturing of software product.

- Timely
- User friendly
- Reliable
- cost effective

- Portable
- Versatile
- Maintanable
- Reusable

2 components of Software Engineering

1> System engineering

2> Development engineering

__/__/__

Steps of System Engineering Method (SEM)

- Define objective of system
- Define the application boundaries of the System
- Factorisation of the system into different components for understanding the system functions & features
- Understanding the relationship between various components
- Understanding the role of forward software with the role of database and other software product used
- Identification of functional & operational requirement of the system
- Use of modelling software for modelling of system
- Interaction with customers, users & others affected by the system.

Development engineering Methodology (SEM)

- Requirement defination & specification
- Design solution to deliver the requirement
- Determine the architecture for deliver of the sol^n
- Software development planning
  - Software testing by components
  - Integration of system components
  - Integration testing for conformation of delivery of requirements
  - Implementation
  - Change management process
  - Maintainance of installed software

Software Engineering

System Engineering

- Business process engineering
- Business process modelling
- Determination of Key features & functions
- Assertion role of responsibility of Key func^n
- Analysis of the system for requirement
- Finalise the new scope and redesign the system

Development Engineering

- Requirement analysis and design
- Software requirement & specification
- System design
- Software development
- Software testing
- Implementation and maintainance

SOFTWARE DEVELOPMENT LOGIC CYCLE

Phases

Requirement gathering & analysis

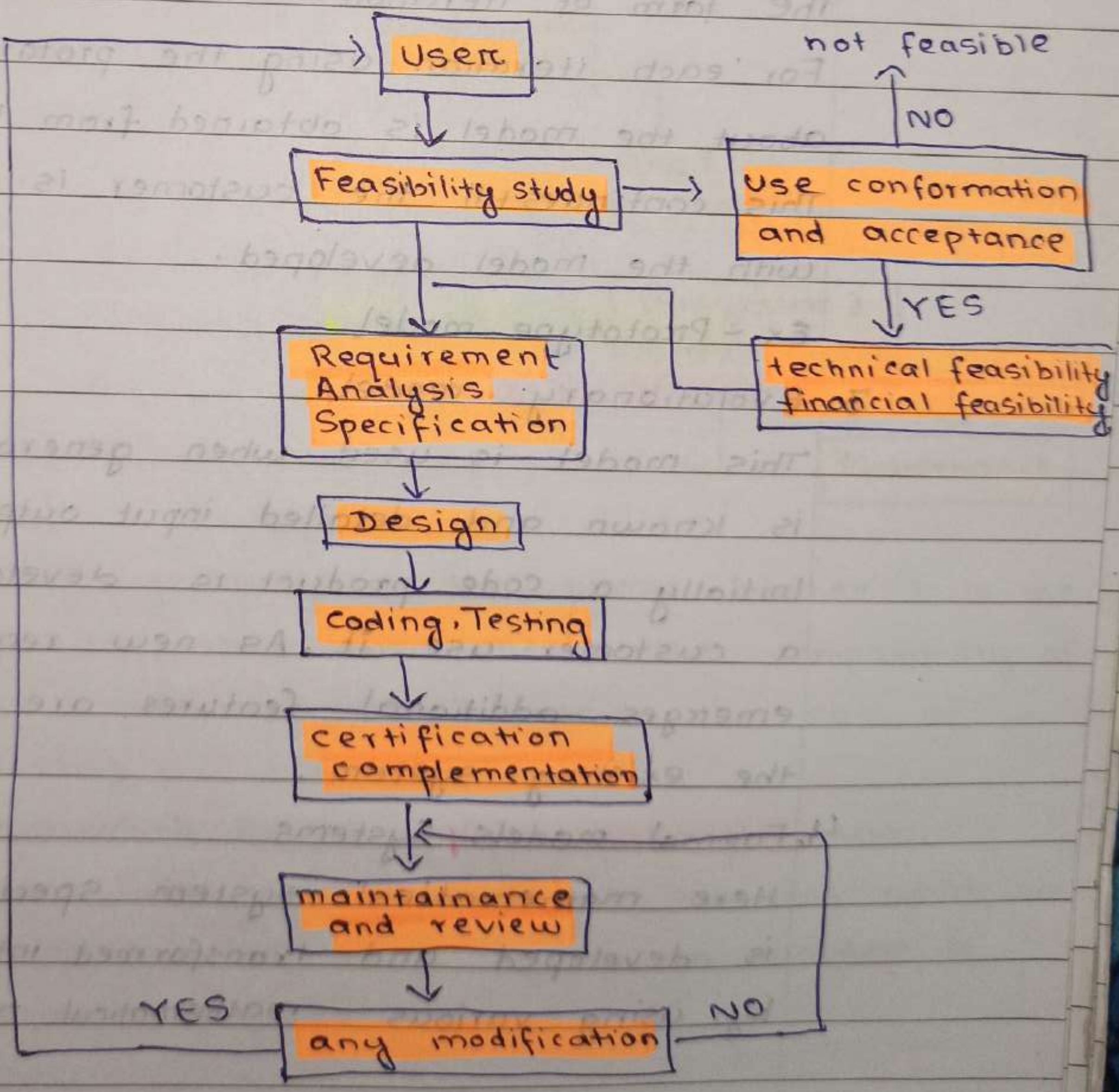
Design

Implementation & coding

Testing

Deployment

Maintainance



__/__/__

SOFTWARE PROCESS MODELS

1. **Linear sequential models**

This model proceed in a linear orderly fashion with transition of world define deliverable at stage.

Ex - Waterfall model, RAD model

2. **Iterative model**

In this model the process proceeds to ~~form~~ the form of iteration.

For each iteration using the prototype feedback about the model is obtained from the customer. This continues till the customer is satisfied with the model developed.

Ex - Prototype model.

3. **Evolutionary model**

This model is used when general objective is known and detailed input output is unknown. Initially a code product is developed and a customer use it. As new requirement emerges additional features are added to the existing system.

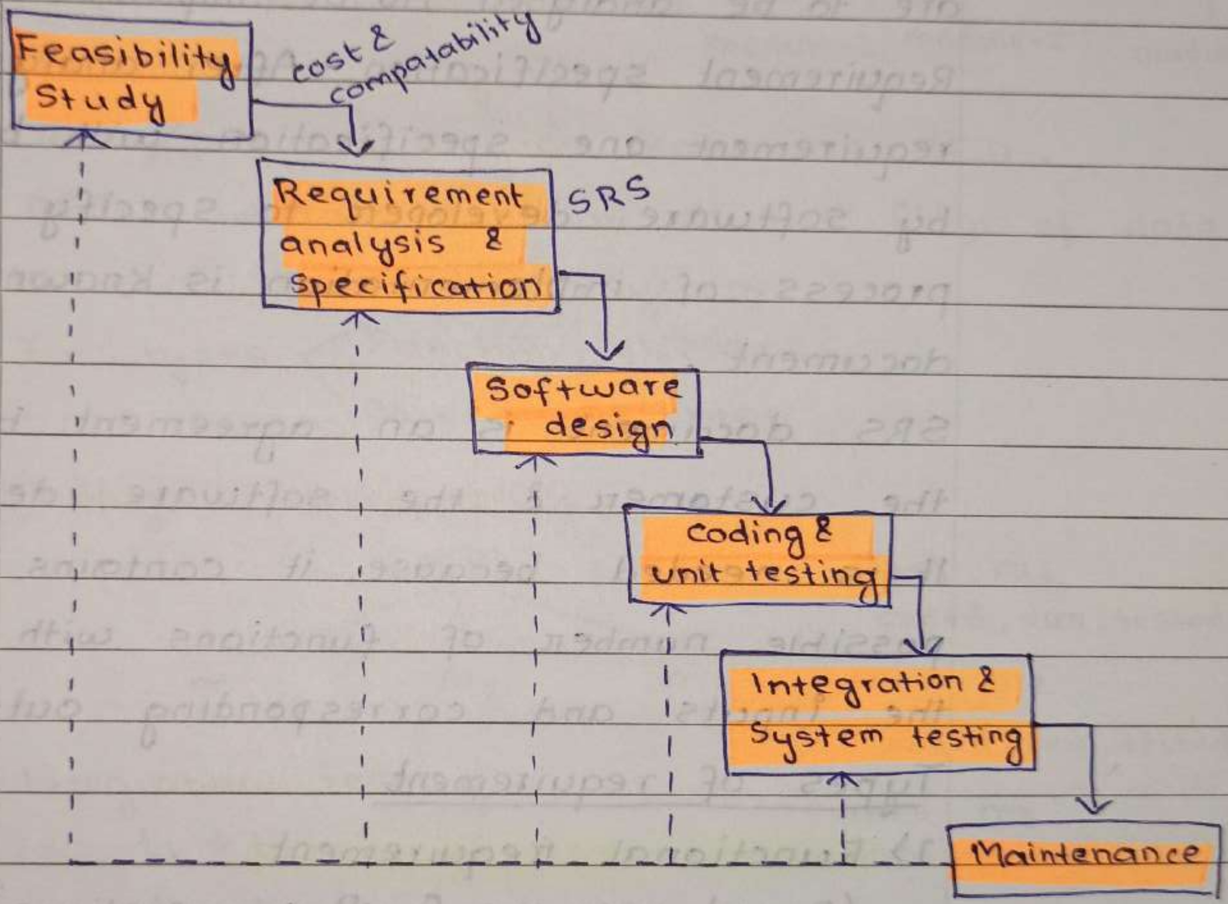
4. **Formal models** Systems

Here mathematical system specification is developed and transformed into a program by using various mathematical methods.

__/__/__

WATERFALL MODEL

- Winston Royce - 1970



- Feasibility study is the phase number 1 in the waterfall model who checks the compatability of each team member & decide the cost of planning.
- Requirement is the functional requirement. These are the number of functions which will be implemented in a system to complete a software project.

__/__/__

Requirement analysis is the first step of phase-2 where all the collected requirements are to be analysed, to be implemented further.

Requirement specification - After analysis of requirement one specification will be done by software developer to specify the total process of implementation is known as SRS document.

SRS document is an agreement between the customer & the software developer. It is needed because it contains all the possible number of functions with its all the inputs and corresponding outputs.

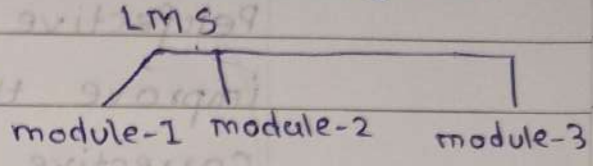
Types of requirement

1> Functional requirement
(Based upon the func^{n} or operations)

2> Non-functional requirement
(Not - functional)
Ex - Programming platform
- Hardware capacity
- Operating system

- **Software design (DFD) - data flow diagram**

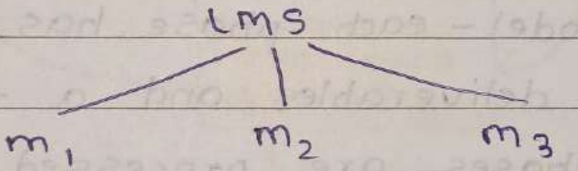
Software design is the implementation of the SRS document in a graphical view/pictorial diagram known as DFD.



DFD diagram explains all the flow of data to each module in a system.

Two types
- Function oriented
- Object oriented

- **Coding & Unit testing**



First m_1 coded, run, tested
then m_2 coded, run, tested
then m_3 coded, run, tested

Integration testing

$t_1 + t_2 = t$

$t + t_3 = ...$

Types of System testing

1. **α-testing** - Testing done by developer-own team
2. **β-testing** - Testing done by other team
3. **Acceptance testing** - Testing made by project coordinator - done before acceptao of project

__/__/__

- Maintenance

Perfective maintenance - check coding manually improve the perfectness

Corrective maintenance - correction of error increase the efficiency upto 100%

Adaptive maintenance - The project should adapt in any situation / environment

ADVANTAGES

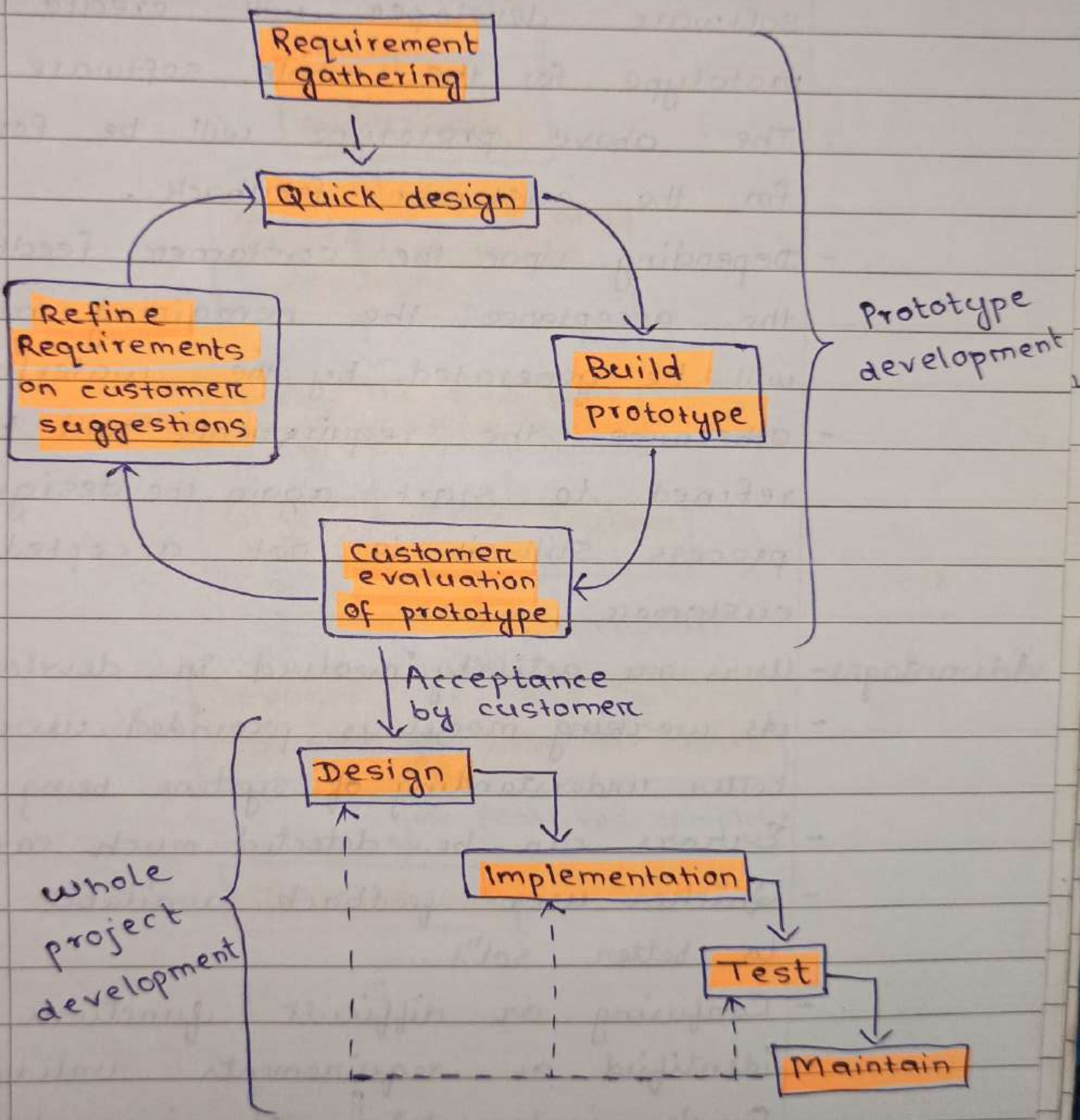
- Simple & easy to understand & use.
- Easy to manage due to the rigidity of the model - each phase has specific deliverables and a review process
- Here phases are processed & completed one at a time. Phases do not overlap
- Waterfall model works for smaller projects where requirements are very well understood.

DISADVANTAGES

- Once application is in testing stage, it is very dif. to go back and change something that was not well-thought out in the concept stage
- High amounts of risk & uncertainity
- Not a good model for complex & object-oriented projects

Prototype model is that instead of freezing the requirements before a design or coding can proceed a throwaway prototype is built to understand the requirements.

PROTOTYPE MODEL



Prototype means the sample copy of any software project.

__/__/__

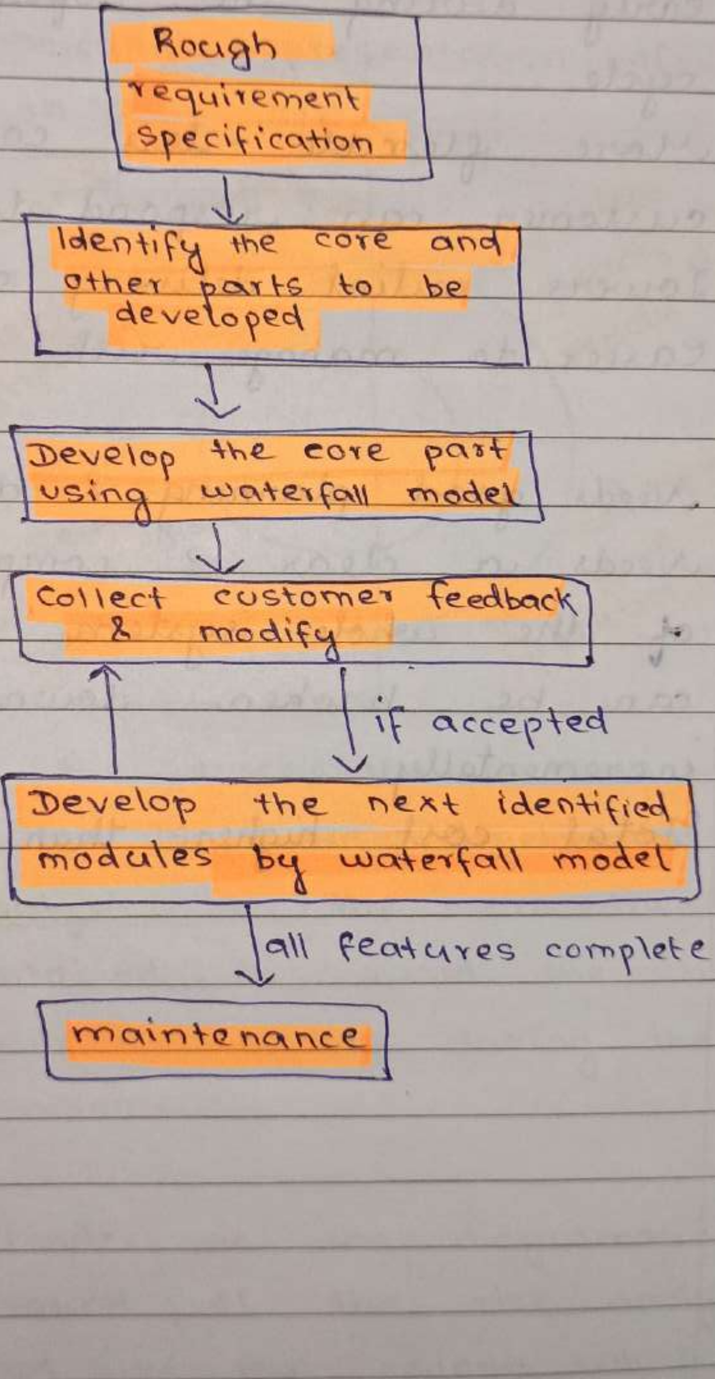
- According to prototypic model, the software developer will create a small prototype for the whole software project.
- The above prototype will be forwarded for the customer feedback.
- Depending upon the customer feedback or the acceptance the remaining modules will be generated by the waterfall model.
- otherwise the requirements will be further refined to start again the designing process still it is not accepted by the customer.

Advantages- Users are actively involved in development
- As working model is provided user gets a better understanding of system being developed
- Errors can be detected much earlier
- Quicker user feedback available leading to better sol^n^s.
- Confusing or difficult functions can be identified as requirements validation, Quick implementation of incomplete but functional application

Disadvantages- Leads to implementing & then repairing way of building
- This increase complexity as scope may expand beyond original plan
- Incomplete or inadequate problem analysis

Incremental model - the whole requirement is divided into various builds. Multiple development takes place here making the life cycle / a 'multi- waterfall' cycle.

Evolutionary / Incremental model

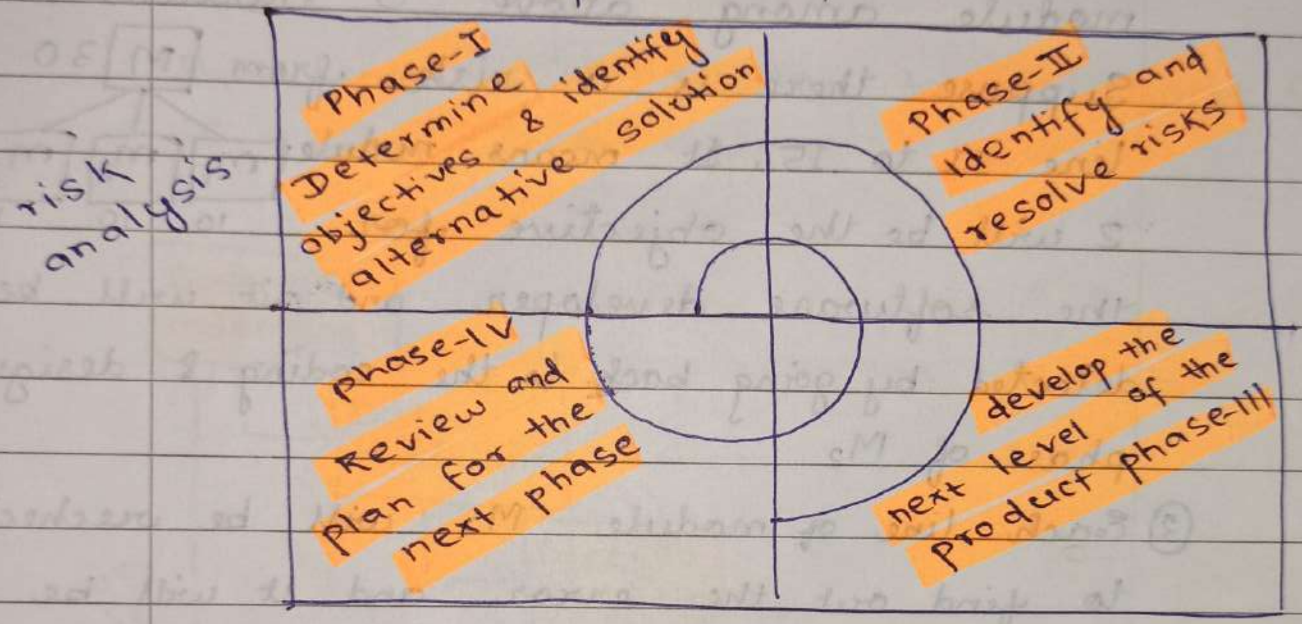

Rough requirement specification
Identify the core and other parts to be developed
Develop the core part using waterfall model
Collect customer feedback & modify
if accepted
Develop the next identified modules by waterfall model
all features complete
maintenance

Advantage - Generates working software quickly & early during the software life cycle
- More flexible - less costly
- customer can respond to each built
- Lowers initial delivery cost
- Easier to manage risk

Disadvantage - Needs good planning & design
- Needs a clear & complete defination of the whole system before it can be broken down and built incrementalley
- Total cost higher than waterfall

Spiral model

Diagramatically representation of risk analysis process in loop str.


risk analysis
Phase-I
Determine & identify objectives & alternative solution
Phase-II
identify and resolve risks
develop the next level of the product phase-III
phase-IV
Review and plan for the next phase

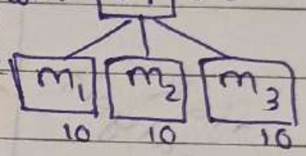
Risk is the uncertain error which comes during the software development.

Risk analysis is the procedure which will be maintained to avoid the number of risk which comes during the Software implementation.

Spiral model is the diagramatically representation of the risk analysis procedure in terms of four codons which creates loops like form.

__/__/__

① Determine objective and identify alternative solution - means we have to study for the exact program regarding the one module among above 3 modules. Suppose there is a risk from line 11 to 15. It means module 2 will be the objective for the software developer and it will be detected by going back to the coding & designing phase of M_2



② Each line of module M_2 will be rechecked to find out the error and it will be recoded & redesigned by using waterfall model.

③ Develop the next level of the product. In this level all the error will be checked one by one from line 11 to line 15 of module 2 by completing all the phase of iterative waterfall model. As a result now the risk will be avoided & there will be no blockage from line 11 to 15

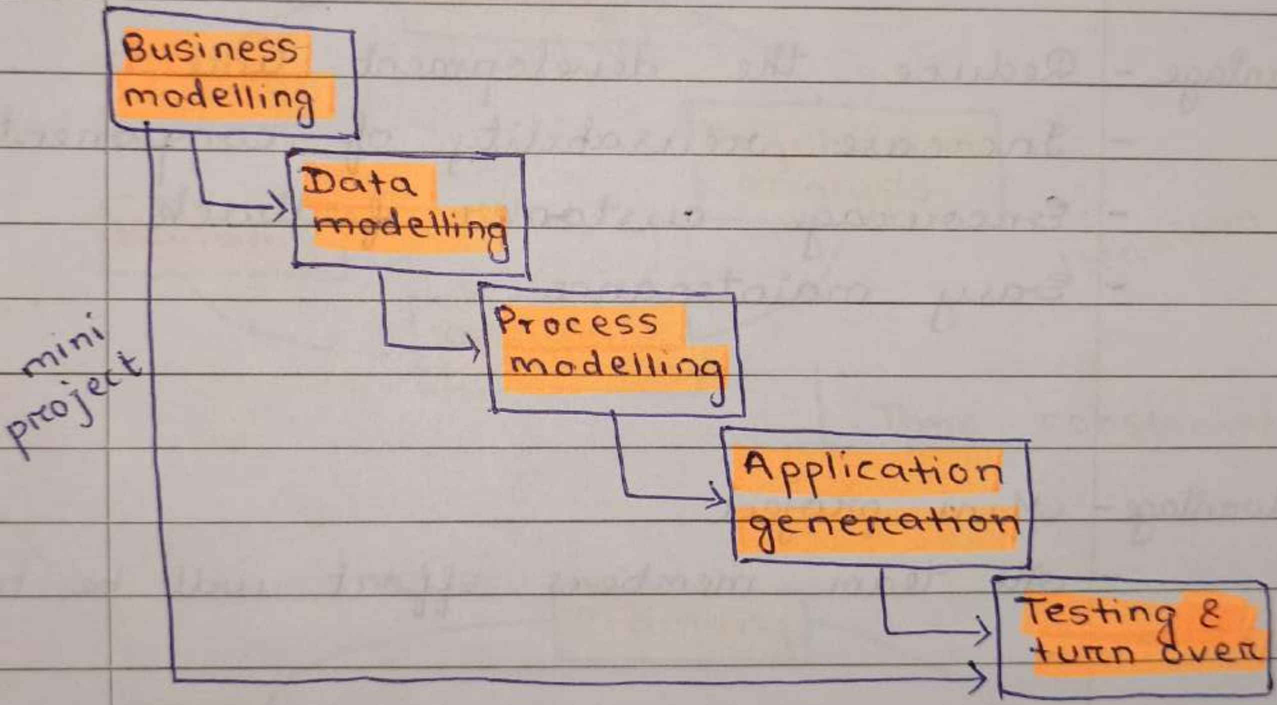
④ Continue from line 15 to line 30. If everything is proper, system is complete & implemented properly.

__/__/__

RAD model

RAD stands for rapid application developme model.

This model is similar to the some feature of incremental model.



It is based upon the parallel wise concept used for mini project Follows incremental model.

Here each module will run simultaneously with its some portion to make the sample for customer.

__/__/__

Business modelling - Planning for project
Data modelling - Write requirement
SRS document

Process modelling - Design
Application generation - Coding
Testing & turn over - Maintenance

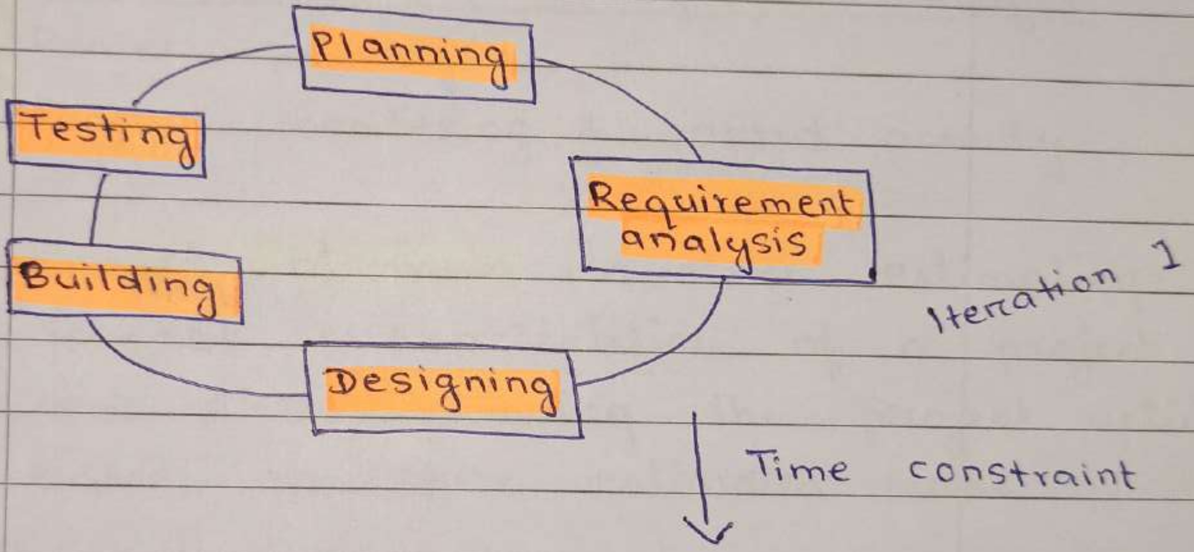
Advantage - Reduce the development time
- Increase reusability of components
- Encourage customer feedback
- Easy maintenance

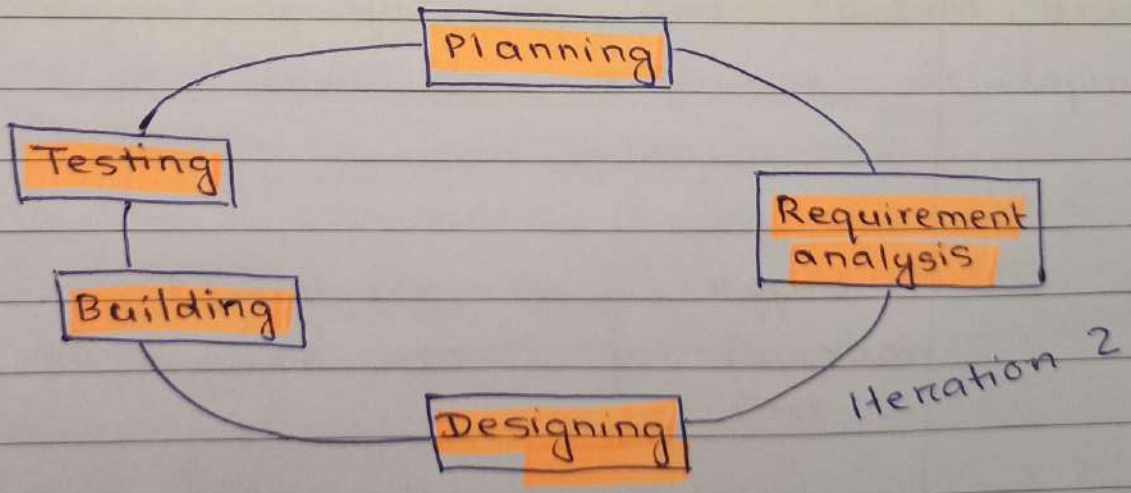
Disadvantage - Mini project
- All team members effort will be high

__/__/__

Agile model

Agile also follow the feature of incremental model with iterative process with prototype & time slot





SPM complexities - Changeability
- Complexity
- Uniqueness

SOFTWARE PROJECT MANAGEMENT

Main role is to enable a group of software developers to work efficiently towards a successful completion of the project.

Activities of software project manager

1. Project planning
2. Project monitering & control activity

(1) Project planning involves estimating several characteristics of a project and then planning the project activities based on these estimates made

(2) Project monitering and control activities ensures that the software development proceeds as per plan.

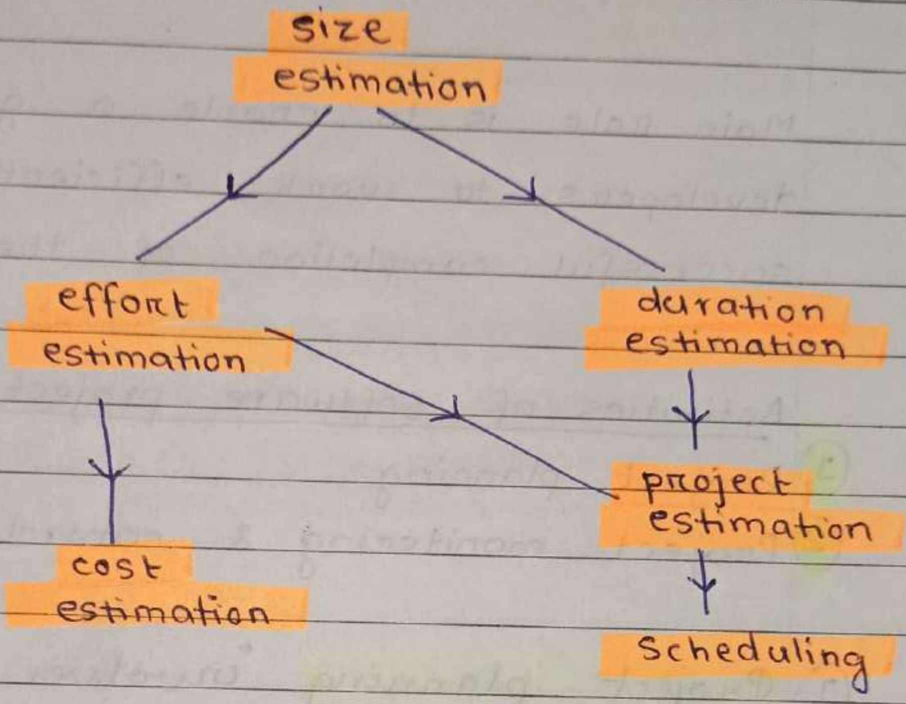
Project Planning

1) Estimation — Effort
— Duration
— Cost

2) Scheduling

3) Staffing

4) Risk management



Project monitering & control activities

Software Project Management Plan (SPMP) doc.

1. Introduction
   - Objectives
   - Major functions
   - Performance Issues
   - Management & Technical constraints

2. Project estimates
   - Historical data used
   - Estimation techniques used
   - Effort, resources, cost & project duration estimates

3. Schedule

- Work breakdown structure
- Task network representation
- Gantt chart - explains the individual task with their duration & the sequence of implementation of these task (module)
- PERT chart - explains each module with its corresponding operation by the each team member

4. Project resources

- People - no of group members having membership
- Hardware & software - system capacity
- Special resources

5. Staff organisation

- Team structure - Struc of all team leaders in single group
- Management reporting - Each team leader can produce corresponding reports of each team to the owner of organisation.

6. Risk management plan

- Risk analysis
- Risk Identification
- Risk estimation
- Risk abatement procedures

7. Project tracking & control plan

- Metrices to be tracked }
- Tracking plan
- Control plan -

→ Process based upon the management reporting done by the owner to know about the current status of the project.

→ Depending upon current status a control plan will be designed in such a way that the project will be completed in the time constant.

METRICES OF S.PROJECT

(1) LOC (Line of codes)

(2) FP (function point) - main functional model

8. Miscellaneous plans

- Process tailoring
- Quality assurance plan
- Configuration management plan
- Validation & verification
- System testing plan
- Delivery, Installation & maintenance plan

__/__/__

COCOMO Model (Proposed by Bohem)

Constructive cost effective model

(1) Organic

We can consider a dev. project to be organic if the project deals with developing a main understood application programme.

(2) Semidetached

The dev. team consists of a mixture of experienced and in.experienced

(3) Embedded

Developed for complex project with more no. of modules deals with experienced team members

__/__/__

Types of cocomo model

1) Basic cocomo

2) Intermediate cocomo

3) complete cocomo

Basic cocomo

1) Formula

Effort = $a_1 \times (KLOC)^{a_2}$ PM

$T_{dev} = b_1 \times (effort)^{b_2}$ months

Q Assume that size of an organic type software product is estimated to be 32,000 LOC. Av. salary = 15000 per month. Determine effort required to develop the software project.

Ans - Effort chart (const. parameter)

organic = $2.4(KLOC)^{1.05}$ PM

Semidetached = $3.0(KLOC)^{1.12}$ PM

Embedded = $3.6\ (KLOC)^{1.20}$ PM

Dev. time (T_{dev})

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

semidetached = $2.5\ (effort)^{0.35}$ months

Embedded = $2.5\ (effort)^{0.32}$ months

_/_/_

2) Intermediate COCOMO

Also calculate the 2 parameters effort & dev. time with other attributes like product, capacity of the system, personnel (efficient) and development environment.

Personnel -> Advanced programming platform with efficient team members.

Development environment -> Capacity of the project will be high

3) Complete COCOMO

Combination of basic & intermediate COCOMO.

It is a model which holds all the features of basic COCOMO (calculation of effort & dev time) & features of intermediate COCOMO (product, capacity of system, personnel & development environment)

__/__/__

Functional Requirement of document

Q R.1 : Return book()

Description : It is an operation in library management to return book()

R.I.1 : Select return book option

Input : Click on the "returnbook" option

Output : User promoted to enter the book details

R.1.2 . Select book details

Input : The details of the book will be entered by the user

Output : User select the book details

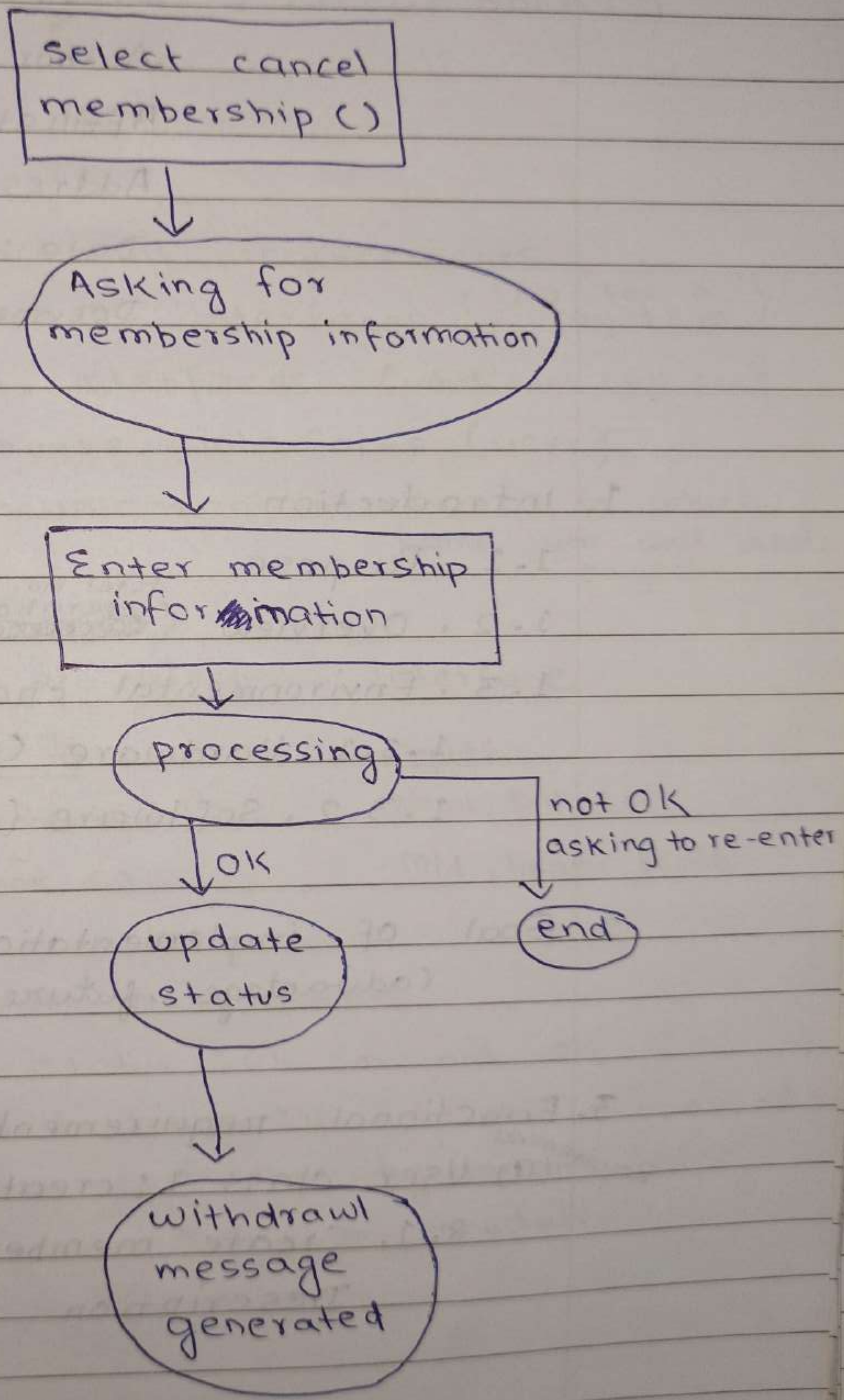
R.1.3 . Select the book

Input : Book selected by the user

Output 1 : Book return message generated

Output 2 : Update status

Cancel membership


select cancel membership ()
Asking for membership information
Enter membership information
processing
OK
not OK
asking to re-enter
update status
end
withdrawl message generated

__/__/__

Format of SRS document

Document Title:

Author:

Affiliation:

Address:

Date:

Document version: Edraw

1. Introduction
   - 1.1. Purpose
   - 1.2. Overview (total no. of operation)
   - 1.3. Environmental characteristics
     - 1.3.1. Hardware (system capacity)
     - 1.3.2. Software (software used)

2. Goal of implementation
   (advantage, future scope)

3. Functional requirement

Requirements (a) User class 1: create membership ()

R. 1. Create membership ()

Description

(b) User class 2 : issue book()
R.2. issue book()

4. Non functional requirements
(a) External interfaces (I/O for a system)
(b) User interfaces (what user will see)
(c) Software interfaces (user)
(d) Communication interfaces - in which comp. you will create the project

Examples of external interface

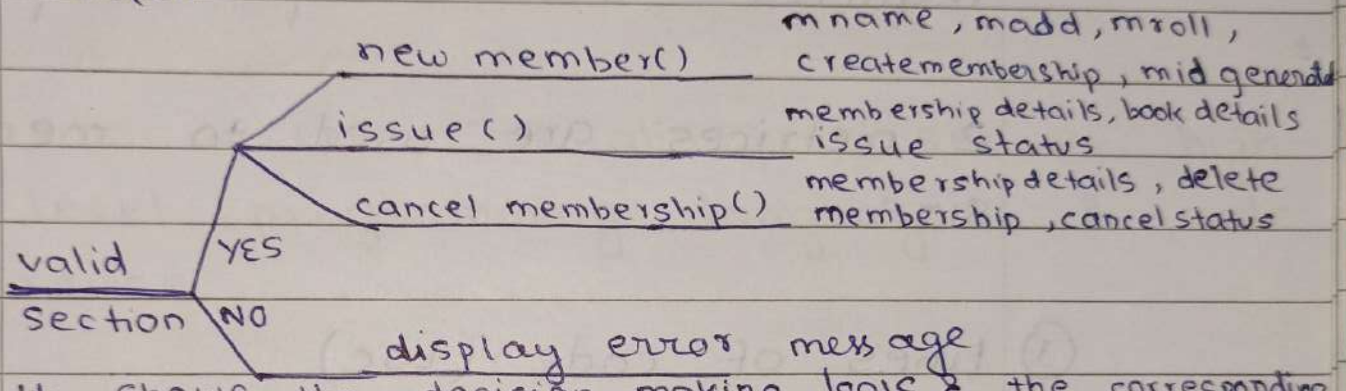
Module 1	Inputs
create membership()	Name, Roll, Add
issue book ()	mid, bname, bcode

5. Behavioural Description
(a) System status : OK or not ok
(b) Events & action : If not ok, processed further by means of time & module

__/__/__

Decision tree

Gives a graphical view of the processing logic involved in decision making and the corresponding action taken


mname, madd, mroll, createmembership, mid generated
new member()
membership details, book details issue status
issue()
membership details, delete membership, cancel status
cancel membership()
valid section
YES
NO
display error message

It shows the decision making logic & the corresponding action taken in a tabular or matrix form

Decision table

conditions.				
Valid section	NO	YES	YES	YES
New member	-	YES	NO	NO
Renewal	-	NO	YES	NO
Cancellation	-	NO	NO	YES

Actions				
display error msg	X			
Ask membership details		x	X	X
make member record		X		
ask membership detail			X	X
Update expiry date			x	
Delete record			X	X
Update		X		

__/__/__

Metrrices for project size estimation

The project size is a measure of the problem complexity in terrms of the efforrt & time required to develop the product.

2 metrices are used to measure size

① Lines of code (LOC)

- LOC is a measure of coding activity alone.
  - * The implicit assumption made by the LOC metrric is that the overall product development effort is solely determined from the coding effort alone is flawed.

- LOC counts depends on choice of specific instructions
  - * Even for same programming problem, diff programmers might come up with programs

having very diff. LOC counts.

- LOC measure correlates poorly with the quality and efficiency of the code.

- LOC metric penalises use of high-level programming languages & code reuse.

- LOC metric measures the lexical complexity of a program & does not address the more important issues of logical & structural complexities.

* From project managers perspective, the biggest shortcoming of LOC metric is that the LOC count is very difficult to estimate during project planning stage, and can only be accurately computed afterr software development is complete.

__/__/__

② Function Point (FP)

FP metric is based on the idea that a software product supporting many features would certainly be of larger size than a product with less no. of features.

Step 1

Compute the unadjusted function point using a heuristic expression

Step 2

Refine UFP to reflect the actual complexities of the different parameters used in UFP computation.

Step 3

Compute FP by further refining UFP account for the specific characteristics of the project that can influence the entire dev. effort.

Requirement Analysis & Specification

1. **Functional**
operations done by the user

2. **Non-functional**
requirements that are not functional
Ex - capacity of software, hardwar-

CHARACTERISTICS of SRS document

1. **Correctness** - The specification must define the desired capability's real world operational environment, its interface to that environment & its interaction with that environment. It means data comes sequeneially.

2. **Completeness** - A complete reqairements specificatn must precisely define all the real world situations that will be encountered & the capability's responses to them. (not included the unnecessary capability features)

3. **Unambigious** - A statement of a requirement is unambiguous, if it can only be interpreted one way.

4. **Modifiable** - Related concerns must be grouped together & unrelated concerns must be separated. Requirement document must have a logical str to be modifiable

5. **Traceable** - Traceability is a property of an element of documentation or code that indicates the degree to which it can be traced to its origin or "reason for being". The matrix is used to check whether the project progressed in the desired direction and for the right product

6. **consistency** - System functions & performance level must be compatible & require quality features must not contradict the utility of system.

7. **Structure**

8. Black-box view - method in which the internal structure /design/ implementation of the item being tested is NOT known to the tester

9. Conceptual Integrity - It is the principle that anywhere you look in your system, you can tell that the design is parat of same overall design.

10. Verifiable - It is the character in which the whole project will be checked by the customer.

Q Implement the SRS doc. for banking management system with basic func^{n}al requirements ie - create_account ()
- deposit_cash ()

1. Introduction

1.1 : Purpose: we are going to develop the banking management system which can be easily used by no. of users.

1.2 : According to development category two basic functional requirement i.e; create_account operation () will be implemented

1.3 : Environmental characteristics
(i) Hardware : RAM is 2GB
(ii) Software : C, C++

2. Goal of implementation
This shows the project developed currently with the less no. of modules can be further extended is future to complete the whole project.

3. Functional Requirements :

(a) User class 1 :

R.1 : create _account ( )

Description : This is the main operation to create_account_operation() in BMS

R:1:1 : Select create account type

Input : click on create account option

Output : Enter personal details.

R:1:2: Select personal details

Input : Give personal details

Output 1: wrong information & error message displayed

Output 2: New account is created & confirm me

(b) User class 2 :

R.2: Deposit _cash( )

Description : This is, the main operation to deposit_ cash() in BMS

R:2:1: select deposit - cash option

Input : Click on deposit cash option

output : Enter personal details

R:2:2: Select personal details

Input : Give personal details

Output 1: wrong info & error message displayed

Output 2: Cash is deposited & database is updated

4. Non-functional Requirements:

(a) External Interface

(i) create-an-account

Input	Output
Customer_name	Account no.
Address	
Ph. no.	
voter_Id-no.	

(ii) Deposit_cash

Input	Output
Account no.	credited_message
Amount with, no. of option	

(b) User interface:

The no. of bank employees are 100.

The no. of outside users are 100.

(c) Software interface: Java

(d) Communication interface: windows 10

5. Behavioural description

(a) System status : ok
    if not ok

(b) Event & action : This field will be implemented if the project denied for the constrain (in terms of time or terms of module)

__/__/__

Characteristics of a good SRS document

- Concise
- Structured
- Blackbox view
  It should specify the external behaviour of the system & not discuss the implementation issues
- Conceptual integrity
- Response to undesired events
- Verifiable

Bad SRS document

- Over-specification
- When we address the how to aspects in SRS document
- Forward references
- Wishful thinking

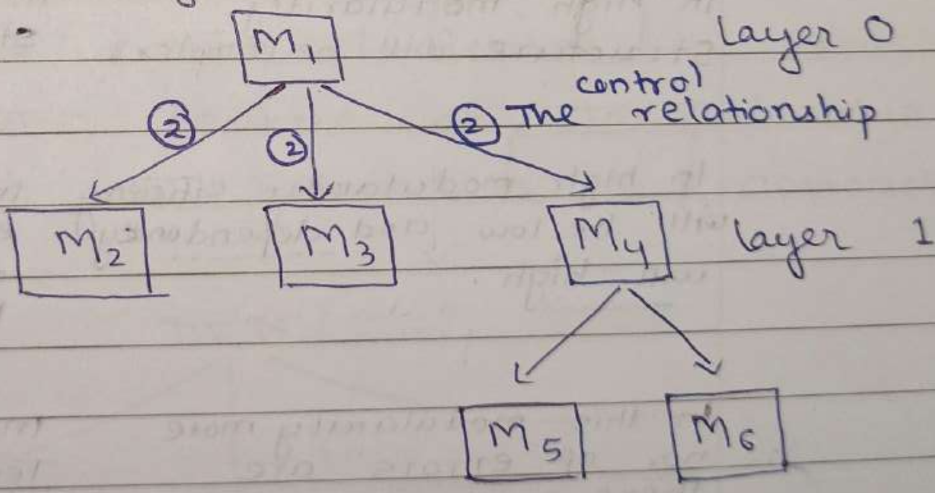
Software Design

Software design is a process where all the SRS document will be converted to a particular design document.

Steps

1. Different modules required
2. Control relationship among modules
3. Interfaces among different modules (Info. moves from one mod. to another)
4. Data structure of the individual modules
5. Algorithm required to implement each module

Layered Structure



Some part of each module will be interrelated to another module.

One module has one data structure.

Modularity - is a technique which divides the modularity means it main module into no. of sub-modules in layer structure.

It is of two types
- high modularity
- low modularity

High modularity	**Low modularity**
In high modularity no. of modules will be more in layer structure.	In low modularity no. of modules will be less in layer structure.
In high modularity structure will be complex.	In this modularity structure will be simple
In high modularity efficiency will be low and dependancy will high.	In low modularity efficiency will be high and dependancy will be low.
In this modularity more no. of errors are there.	In this modularity a less no. of errors are there.
The user can easily code the module and easy to understand.	The user can have to code the module with less coded.

No. of approaches for Software design process

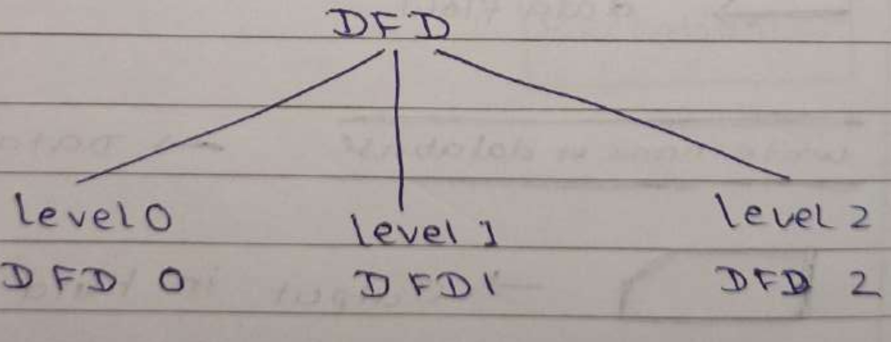
Software design
- Funcⁿ oriented design (DFD - data flow diagram)
- Object oriented design (UML design) Unified modelling language

01. Functⁿ oriented design

- As the name indicates it deals with the no. of functions with its corresponding input & output.
- It is a top to bottom approach.
- we will draw DFD (graphical view of SRS)

02. Object oriented design

As the name indicates it deals with the feature of class & object which is a bottom to top approach.



01.

__/__/__

Data flow diagram

DFD is a graphical model of a system that shows the different processing activities that a system performs & interchanges of the data among the functions.

It is a graphical representation which provides information flow between the input & output data.

○ bubble → process or function

▭ rectangle → external entity/user

→ data flow

write name or database → Datastore

▱ → Output in hard copy

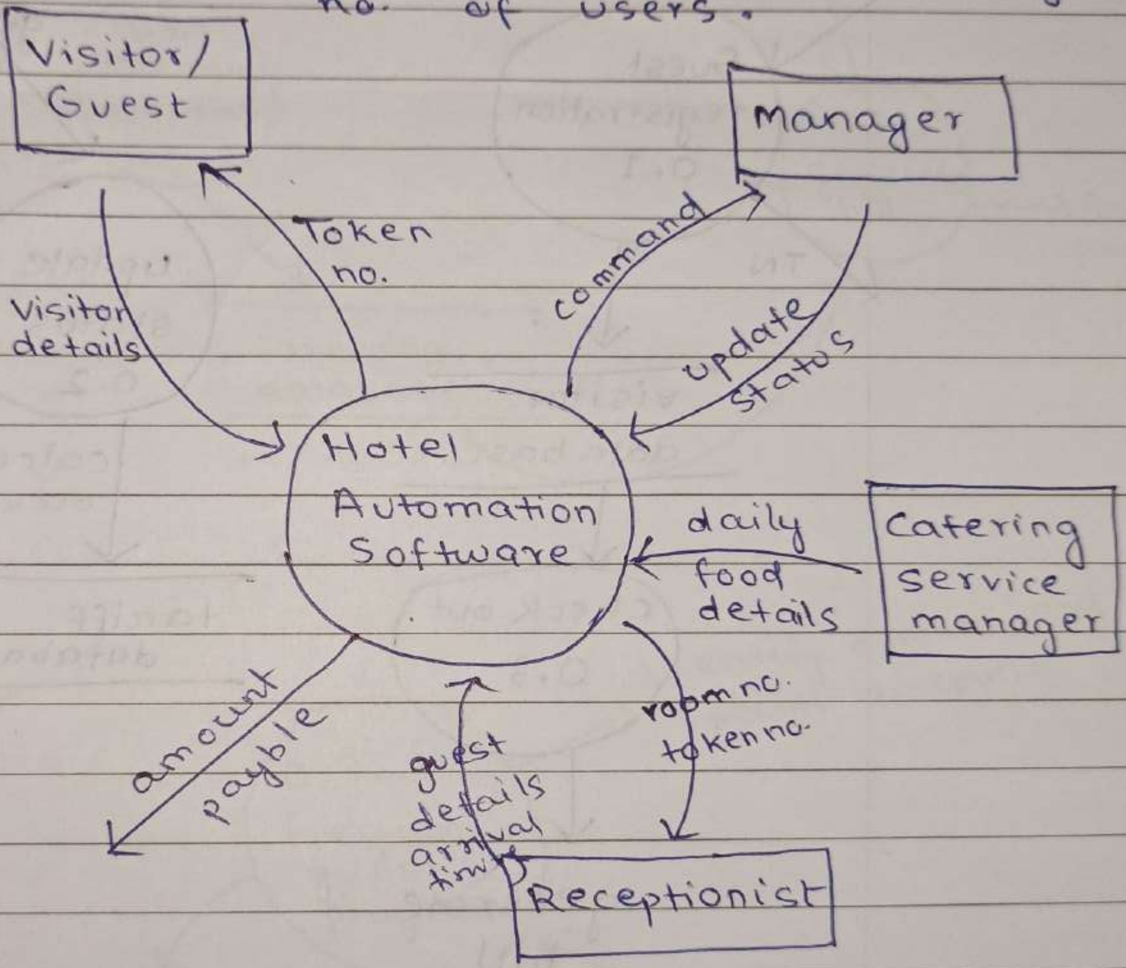
(1) Asynchronous data flow intermediate database → ○ → ○ → = → ○ →

(2) Synchronous data flow without intermediate database → ○ → ○ → ○

Hotel Automation Software

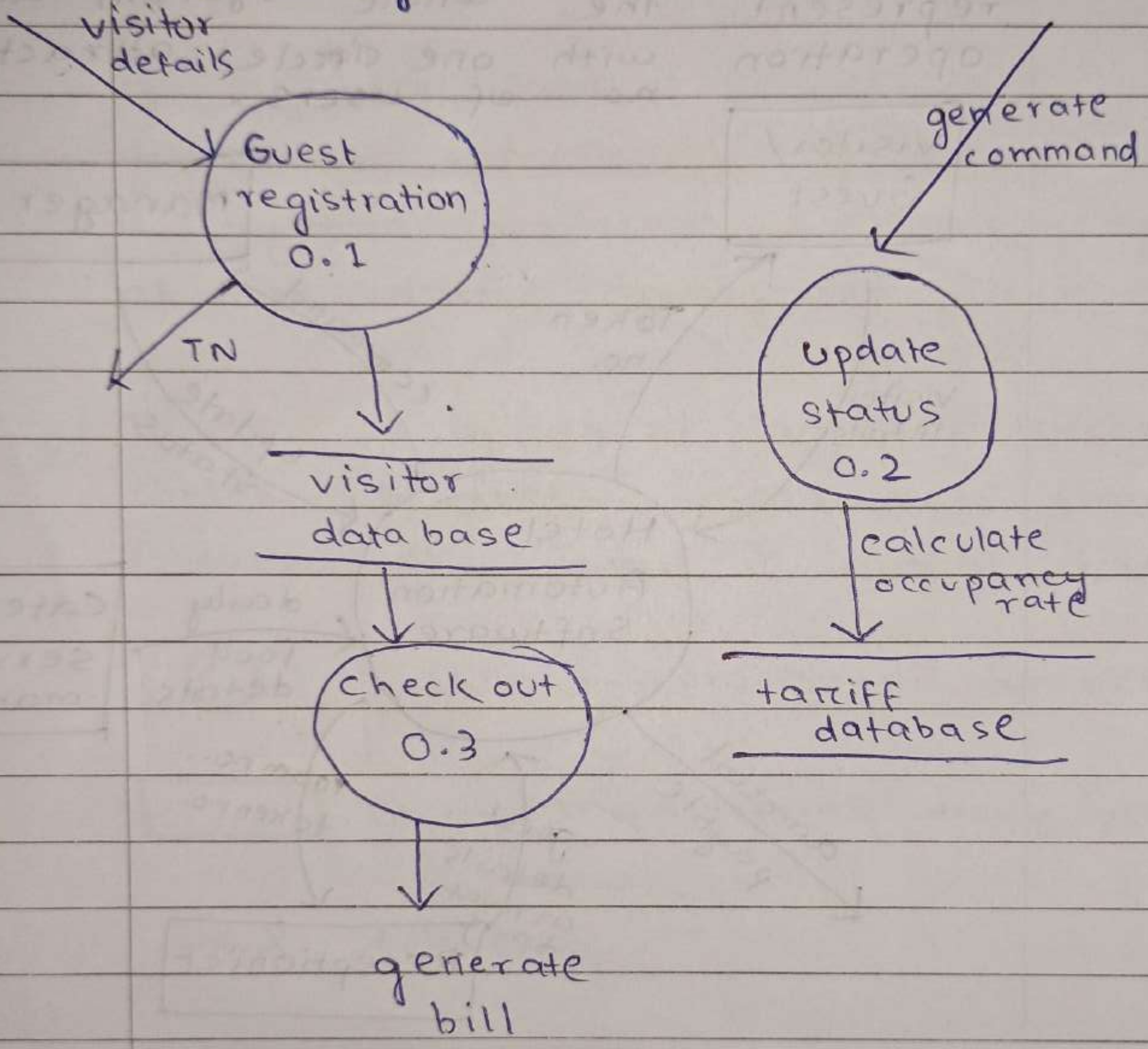
__/__/__

DFD-0 - The 1st level diagram which represent the whole system as single operation with one circcle attracting with no. of users.

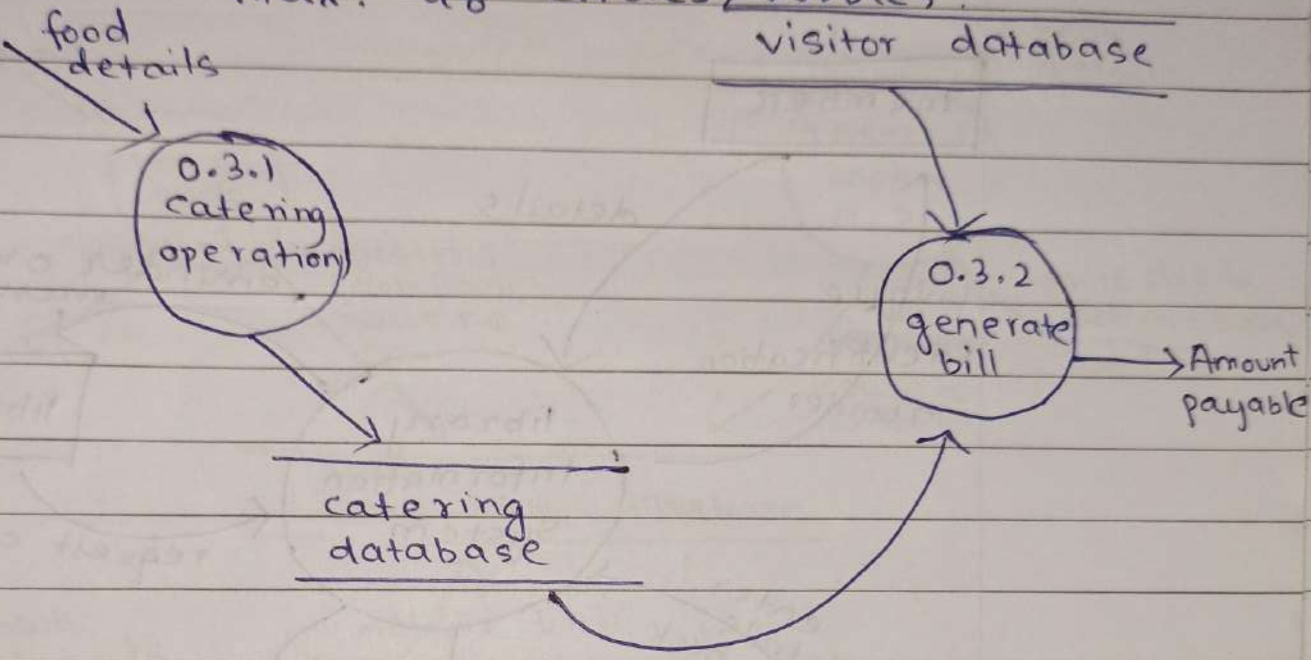


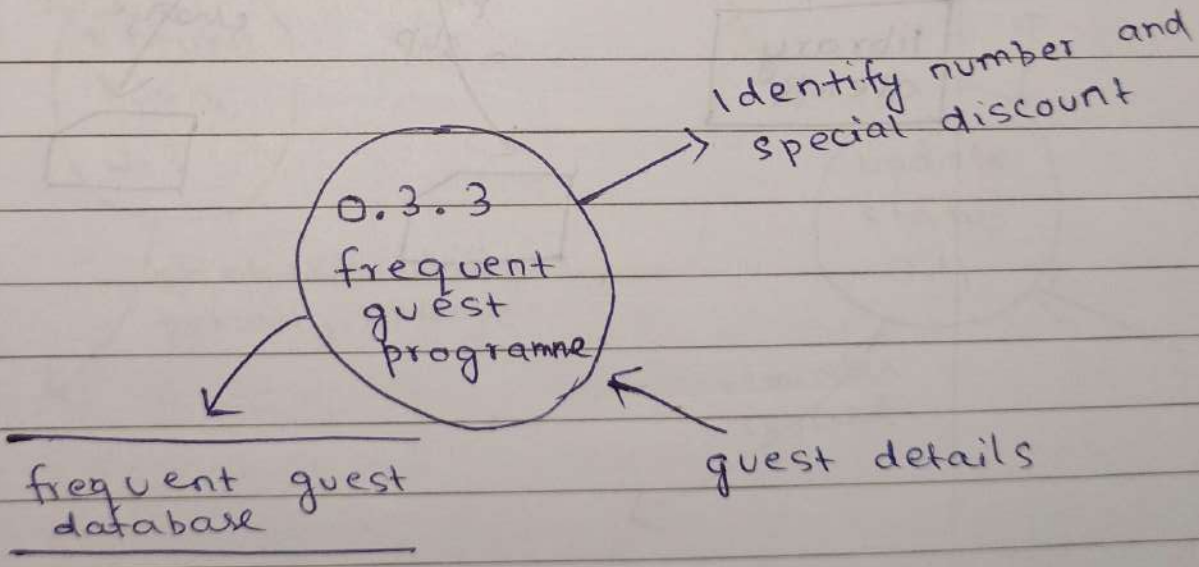
__/__/__

DFD - 1 - The 2nd level diagram which represents the no. of operation of a system with min. 3 & max 7 circle



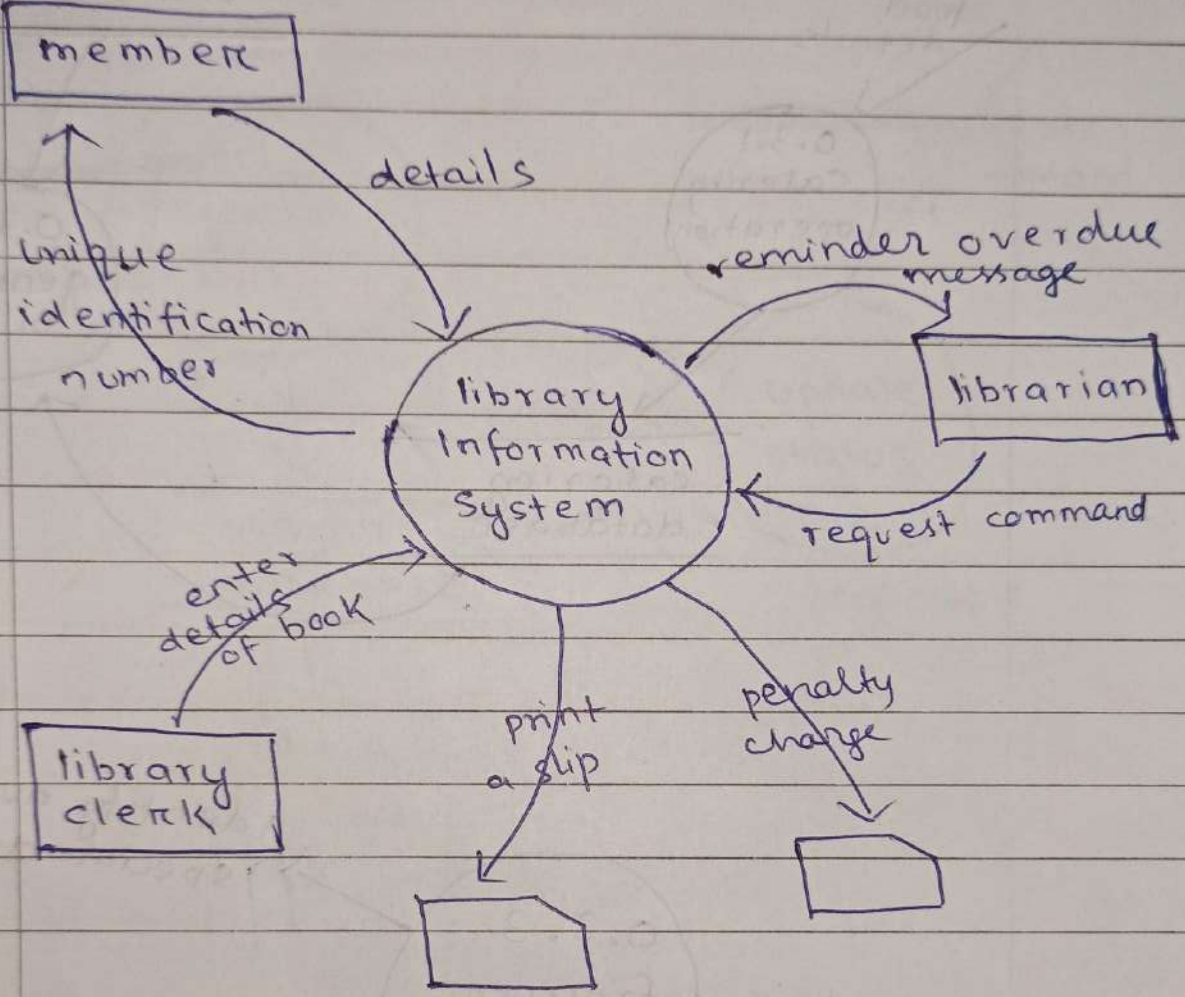
DFD-2 - The 3rd level diagram which explains the sub-operation for a particular operation which contains max. 48 circles/bubbles.



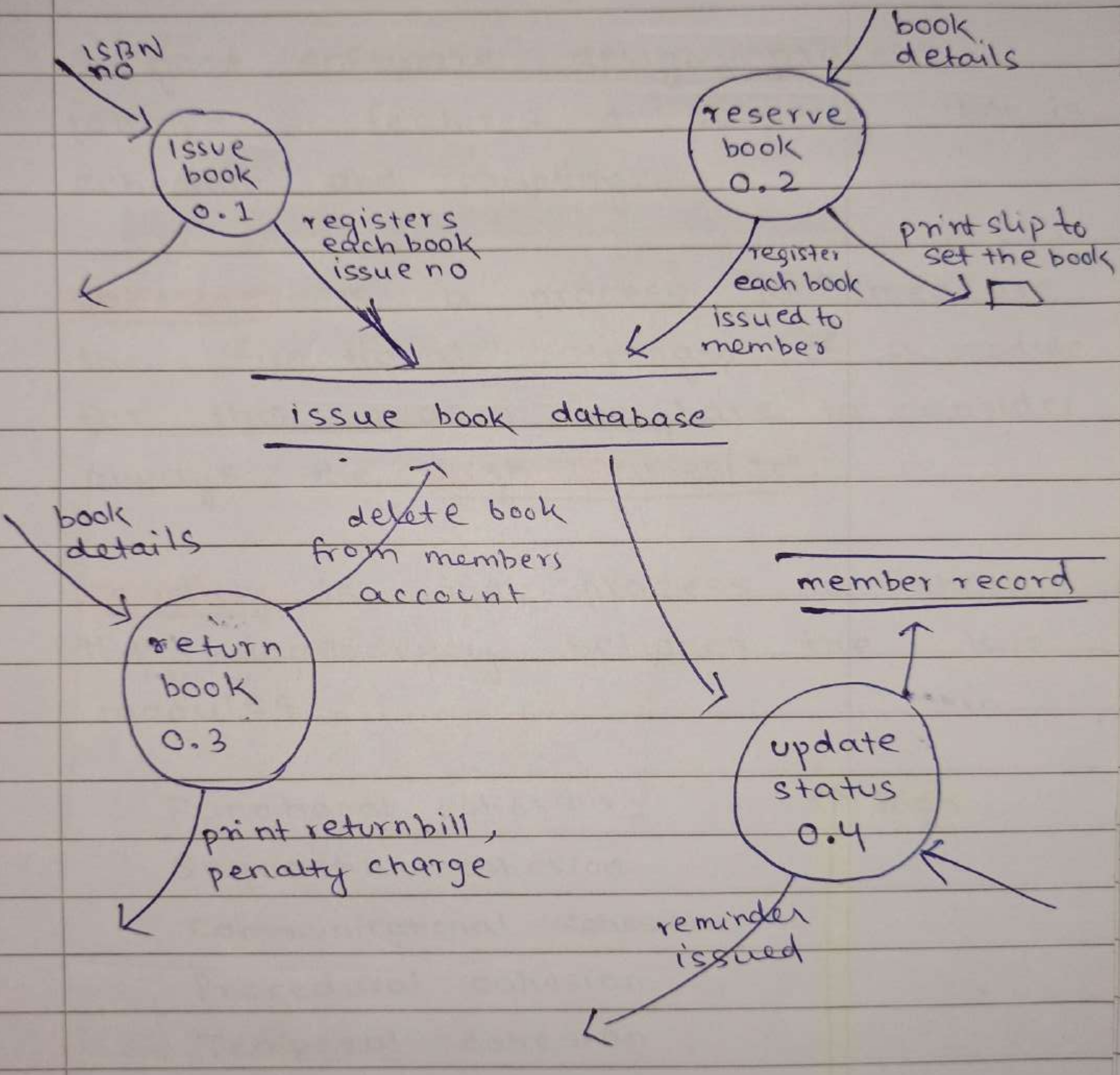


Library Information System __/__/__

DFD - 0



DFD - 1


ISBN no
issue book 0.1
registers each book issue no
book details
reserve book 0.2
register each book issued to member
print slip to set the book
issue book database
book details
delete book from members account
return book 0.3
print returnbill, penalty charge
member record
update status 0.4
reminder issued

Cohesion and Coupling

A good software design process follows 2 features or property that is cohesion and coupling.

Cohesion is a process to measure the functional strength of a module for this reason we have to consider always the **high cohesion**.

Coupling is the process to show the dependency between the two modules.

high ↑

- Functional cohesion
- Sequential cohesion
- Communicational cohesion
- Procedural cohesion
- Temporal cohesion
- Logical cohesion
- Coincidental cohesion

low

7. Coincidental cohesion

instructions have no relationship with each other.

It means they have just coincidentally fall in the same module.

It is the worst type.

Ex –

module name :
LHS

Functions :

create member ();
issue book ();
Calculate total sale ()

6. Logical cohesion

A module is said to be logically cohesive if all elements of the module perform similarr operations such as data input & data output.

_/_/_

5. Temporal cohesion

In temporal cohesion all the functions present in a module if distributed depending upon its input & output for a particular timespan.

4. Procedural cohesion

A module is said to process procesural cohesion if the set of functions of the module are running one after another.
So that we can design the whole module in the particular time period.

```
module name :
  order processing
Functions :
   login () ;
   place order();
   place order on vendor();
   update();
   logout();
```

3. Communicational cohesion

A module is said to have communicational cohesion if all the functions in the module to refer to the same data structure .

2. Sequential cohesion

A module is said to possess sequential cohesion if the different modules run in sequence that means the output of the one function will be input to other function.

Module name :
Library management System
Functions :
create membership();
issue book ();
return book();
Delele membership();

1. Functional cohesion

Functional cohesion which possess the high cohesion for the module who is the combination of both sequential cohesion & communicational cohesion in a proper manner to implement a system.

So for the above reason the functional cohesion is the best cohesion among all the cohesion available for software development.

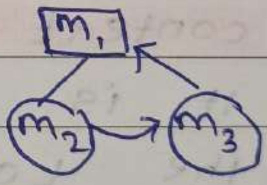
—/—/—

coupling

There are 5 types of coupling.
coupling is the interaction between 2 modules at a time which shows the dependency among the modules.

So for this reason we have to consider low coupling.

1. context coupling (2+3+4)
2. common coupling (LOC are same)
3. control coupling (with logic of one another)
4. Stamp coupling
5. Data coupling



5. Data coupling:

It is the coupling in which only one variable transfers from m_1 to m_2.

4. Stamp coupling:

It is the coupling in which more than more than one variable is transfered from m_1 to m_2

__/__/__

3. Control coupling

Control coupling is the coupling in which the logic of one anotherr be totally control on another module.

2. Common coupling :

Common coupling is the coupling in which the lines of code are the same in m_1 & m_2.

1. context coupling

It is the coupling in which the LOC, variable and all logic are transferred from m_1 to m_2.

Q2. Object oriented Design – 5 views

(a) User view -> use case diagram

(b) Structural view → class diagram, Object diagra

(c) Behavioural view → sequence diagram
Activity diagram
State chart diagram
Collaberation diagram

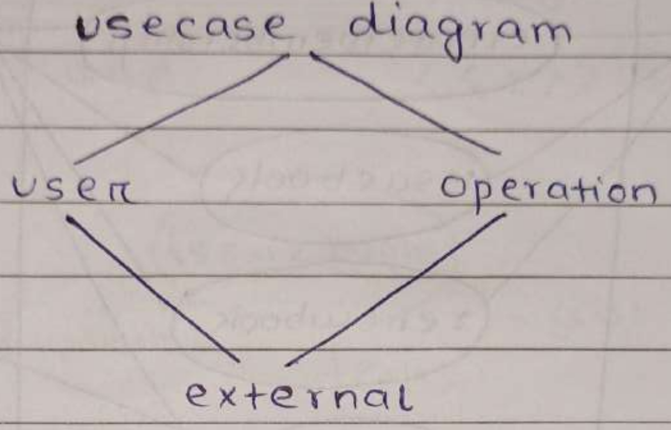
(d) Environmental view → Deployment diagram

(e) Implementation view → component diagram
(part of database

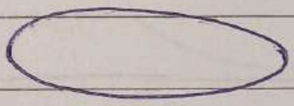
1. User view

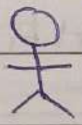
__/__/__

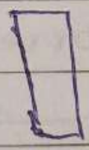
usecase diagram

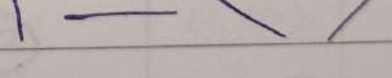


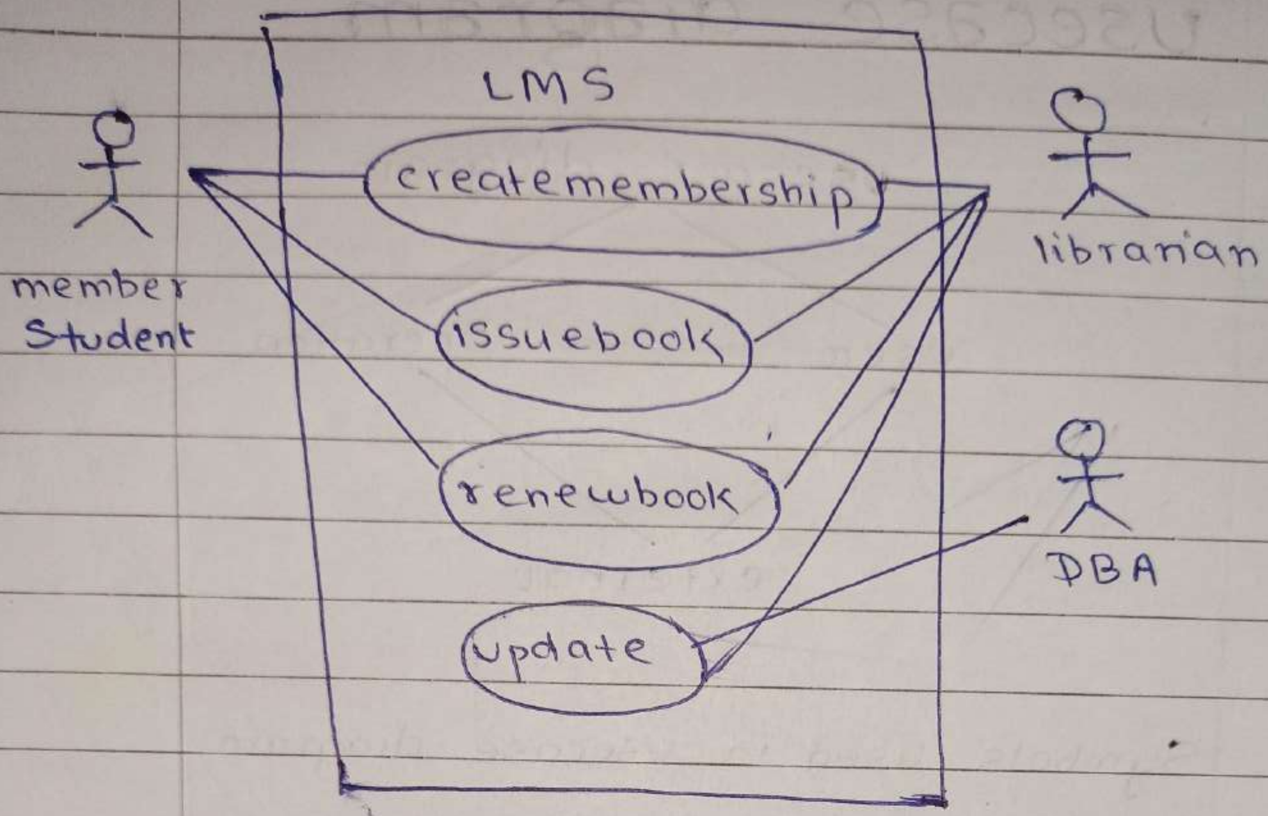
Symbols used in usecase diagram

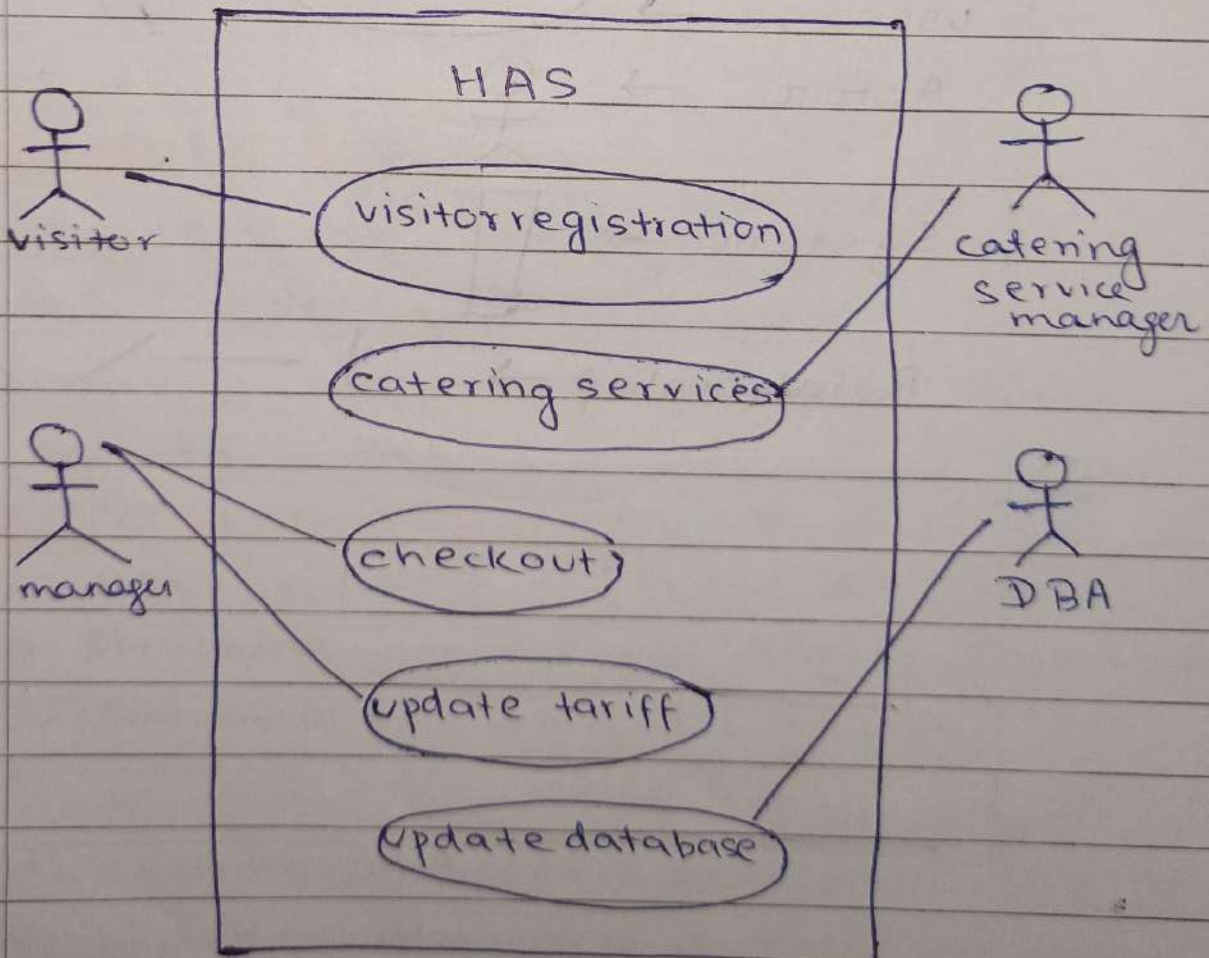
usecase → 

Actor → 

System → 

Relationship → 

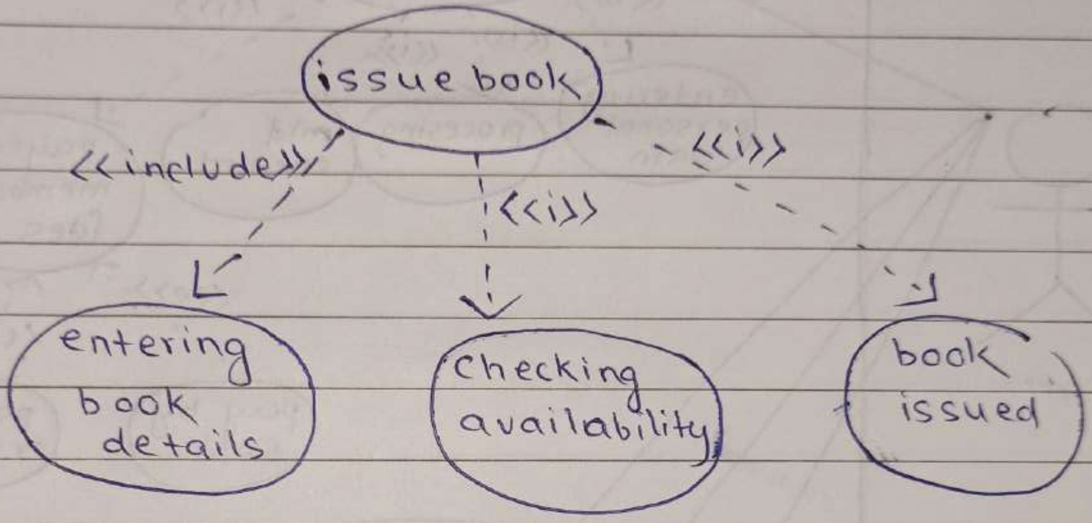

LMS
createmembership
issuebook
renewbook
update
member
Student
librarian
DBA

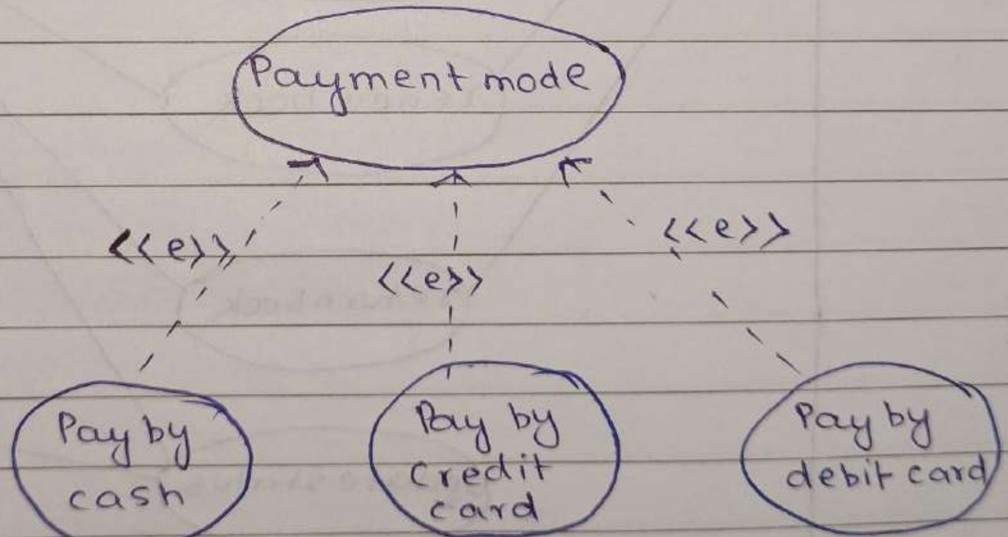

HAS
visitor
visitorregistration
catering services
catering service manager
manager
checkout
update tariff
update database
DBA

__/__/__

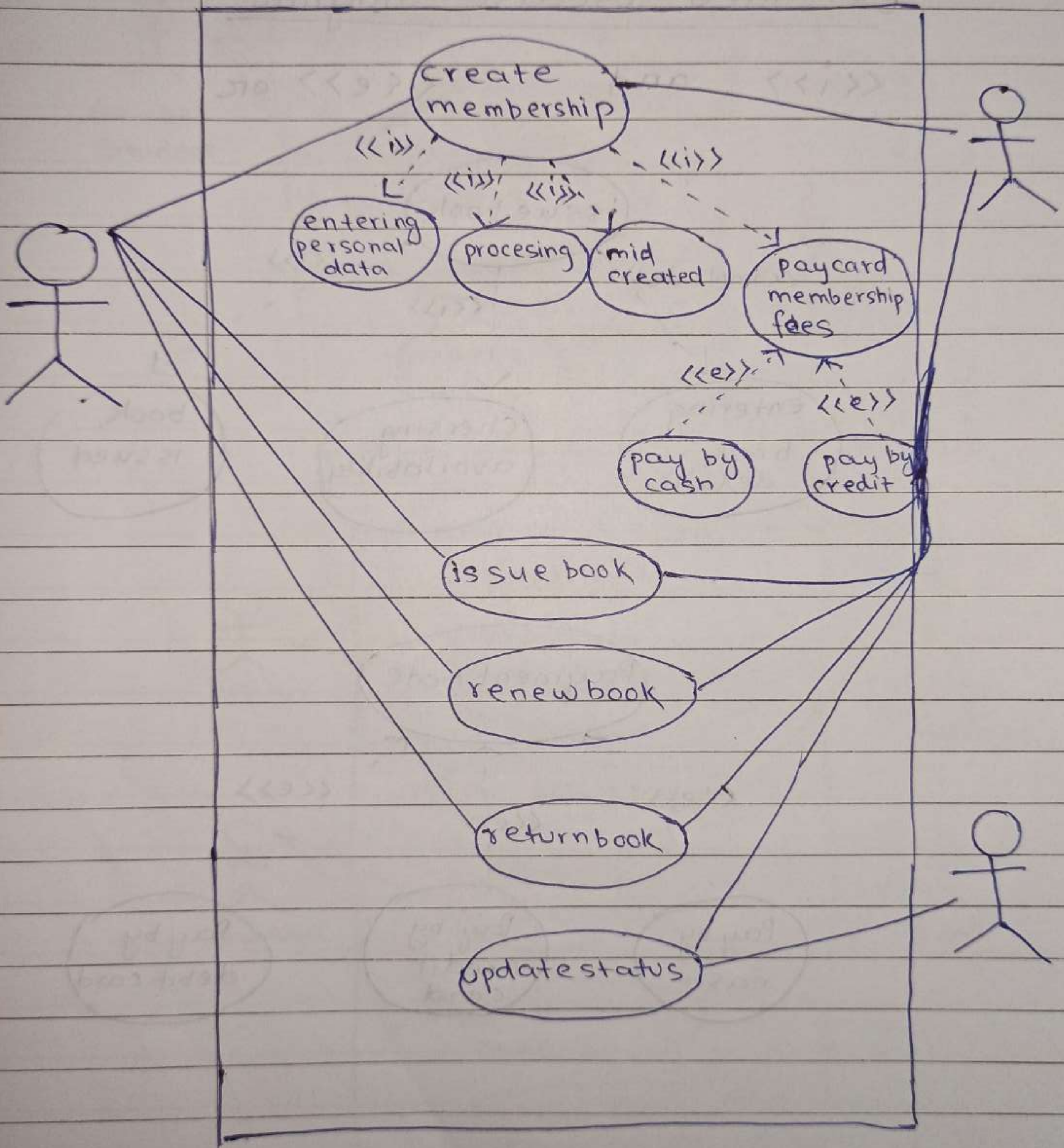
Extended usecase diagram

- <<i>> and <<e>> or





__/__/__

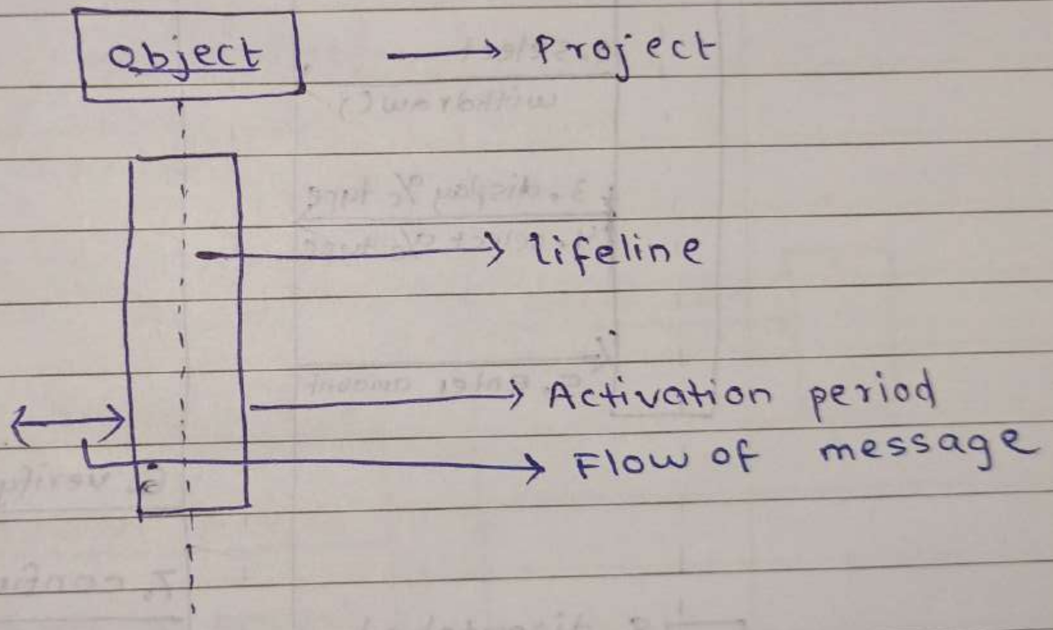

create membership
<<i>>
<<i>>
<<i>>
<<i>>
entering personal data
procesing
mid created
paycard membership fees
<<e>>
<<e>>
pay by cash
pay by credit
issue book
renew book
returnbook
updatestatus

3. Behavioural view

__/__/__

(a) Sequence diagram

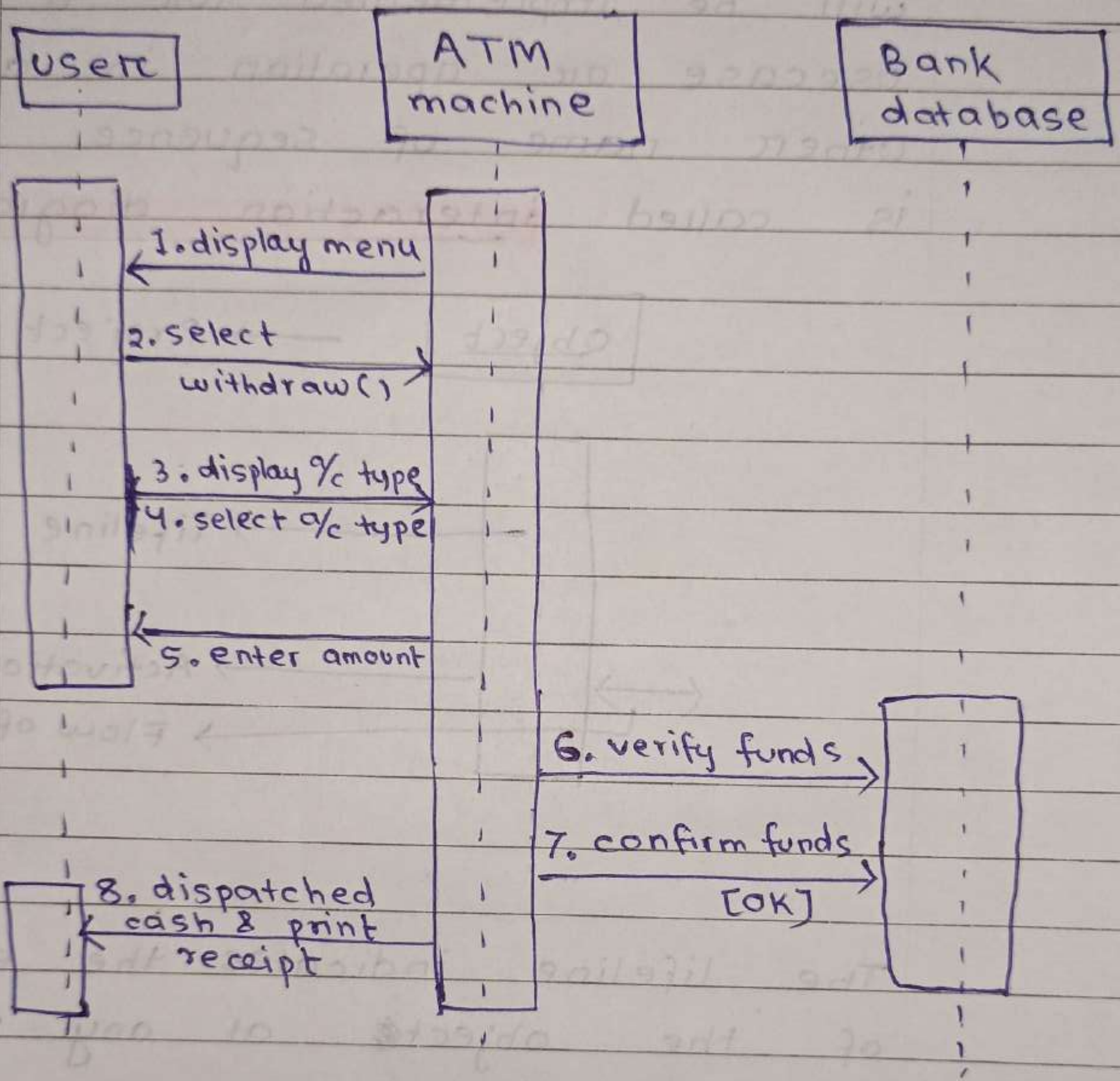
will be implemented for a single usecase or operation in a system. Other name of sequence diagram is called interaction diagram.



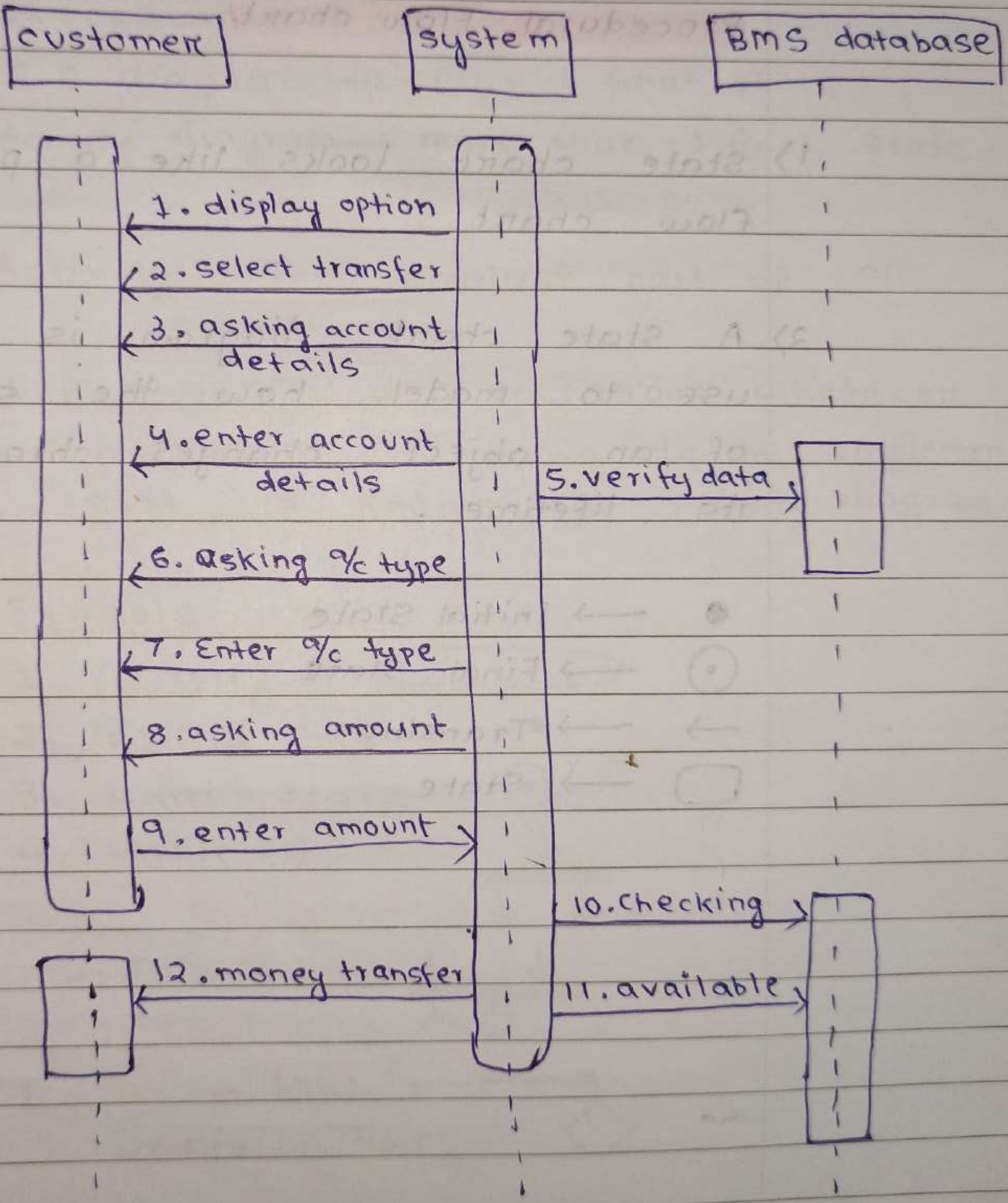
The lifeline indicates the existance of the object at any particular point of time.

It indicates the active period of an object to communicate with another object.

withdraw cash()



Transfer amount ()



3 (b) State-chart diagram

↓

Procedural Flow chart

1) State chart looks like a procedural flow chart

2) A state chart diagram is normally use to model how the state of an object changes throughout its lifetime .

● ⟶ Initial State

⊙ ⟶ Final State

→ ⟶ Transition

▢ ⟶ State

3 (c) Activity diagram

Both the activity diagram & state-chart diagram is : both are coming under procedural diagram.

S-C diagram → only 1 final state

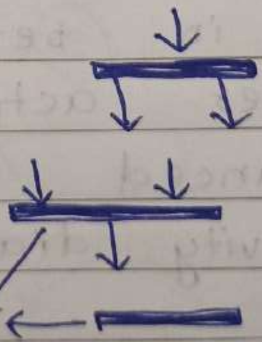
Activity diagram → more than 1 final state

Activity :- The smallest part of an operation.

Defination: To do operations whatevers processes is going to be implement that is known as activity diagram.

Symbols

1. Activity - ▭
2. Initial State - ●
3. Final State - ⊙
4. Fork ( ) -
5. Join ( )
6. swim lane
7. Decision box - ◇



ForK ( ) - when upper single activity is divided into more than one activities.

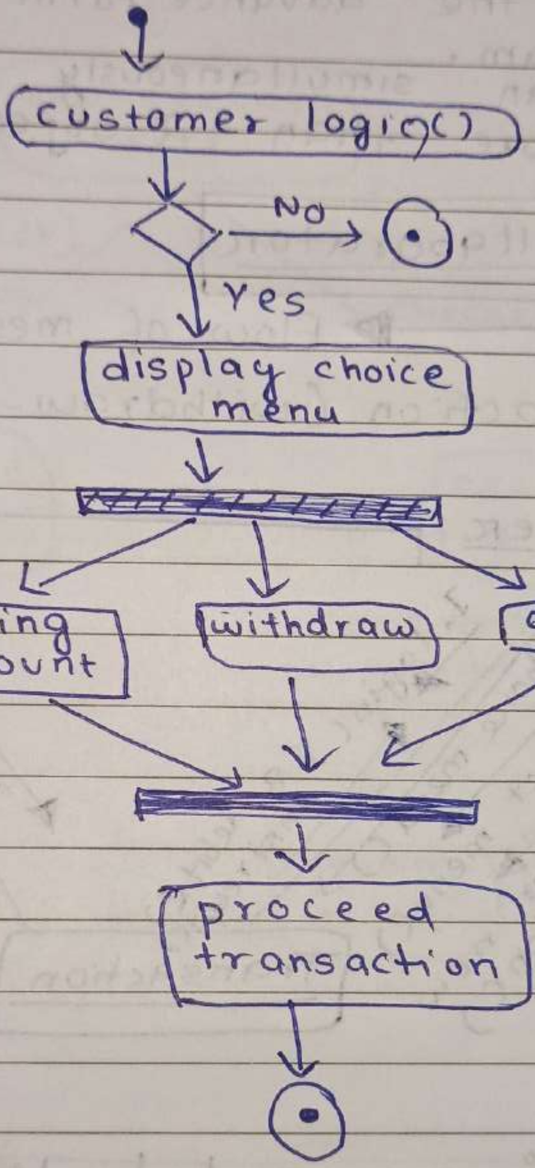
Join ( ) - when more than one activity will be combined / joind to form a single activity

Swim lane -> It is a thin bold rectangle which balance the synchronised line upper and lower activity in the fork and join operation.

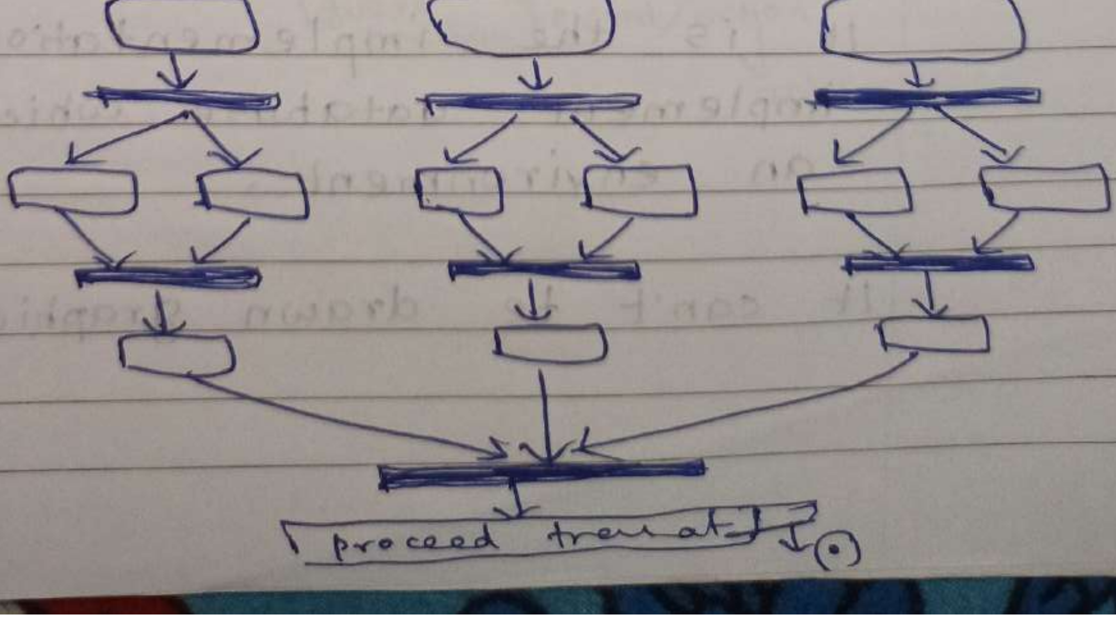
OR

This is the Synchronisation line in between upper & lower activity to show the balanced structure in the activity diagram.

Q Draw activity diagram for BMS ?



Adw. form



Interaction diagram

3. (d) Collaberation diagram

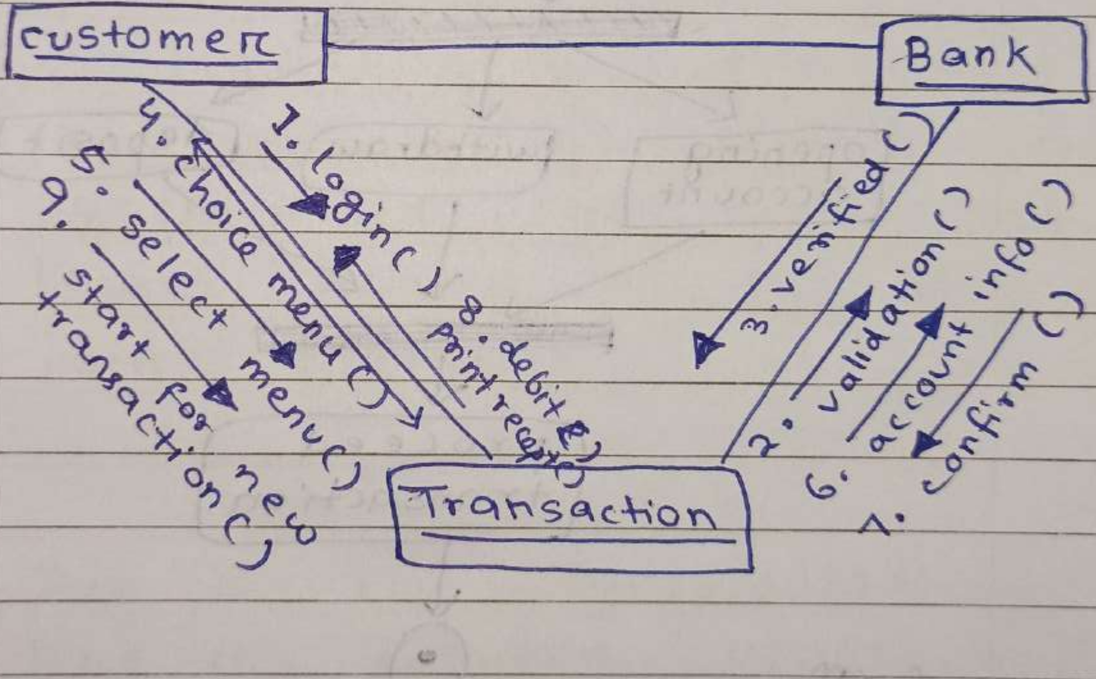
- It is one for one operation.
- It is the advance form of sequence diagram.
- It can simultaneously collaberate / work on more than 2 objects.

Symbol

1. collaborator

2. ———▶ Flow of message with the no.

Q Transaction (withdraw-cash) ATM system

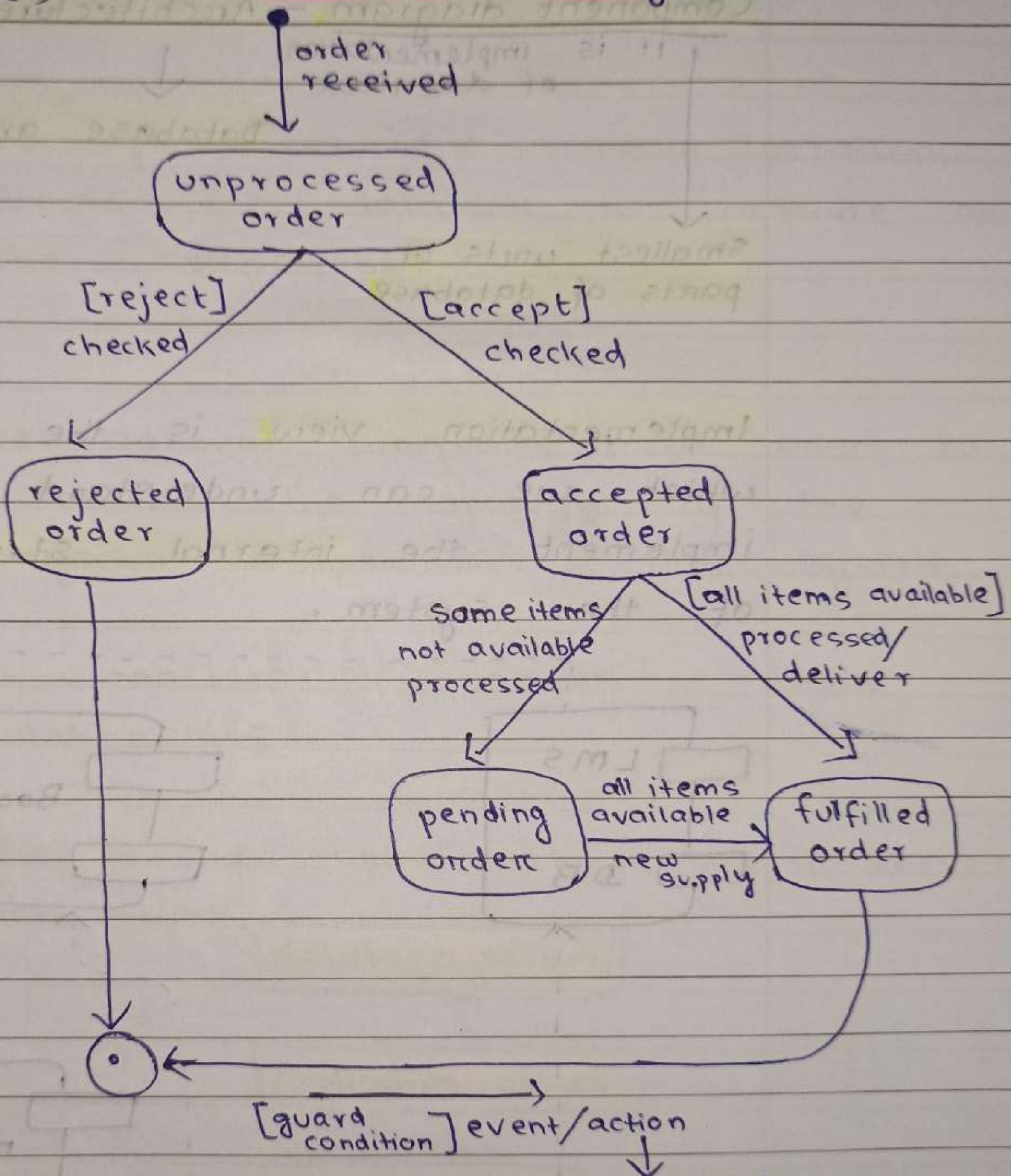


4. Environmental View

It is the implementation of implement database which needs an environment.

It can't be drawn graphically.

3 (b) State chart diagram



5. Implementation view

Component diagram - Architecture diagram

It is implementation of databare.

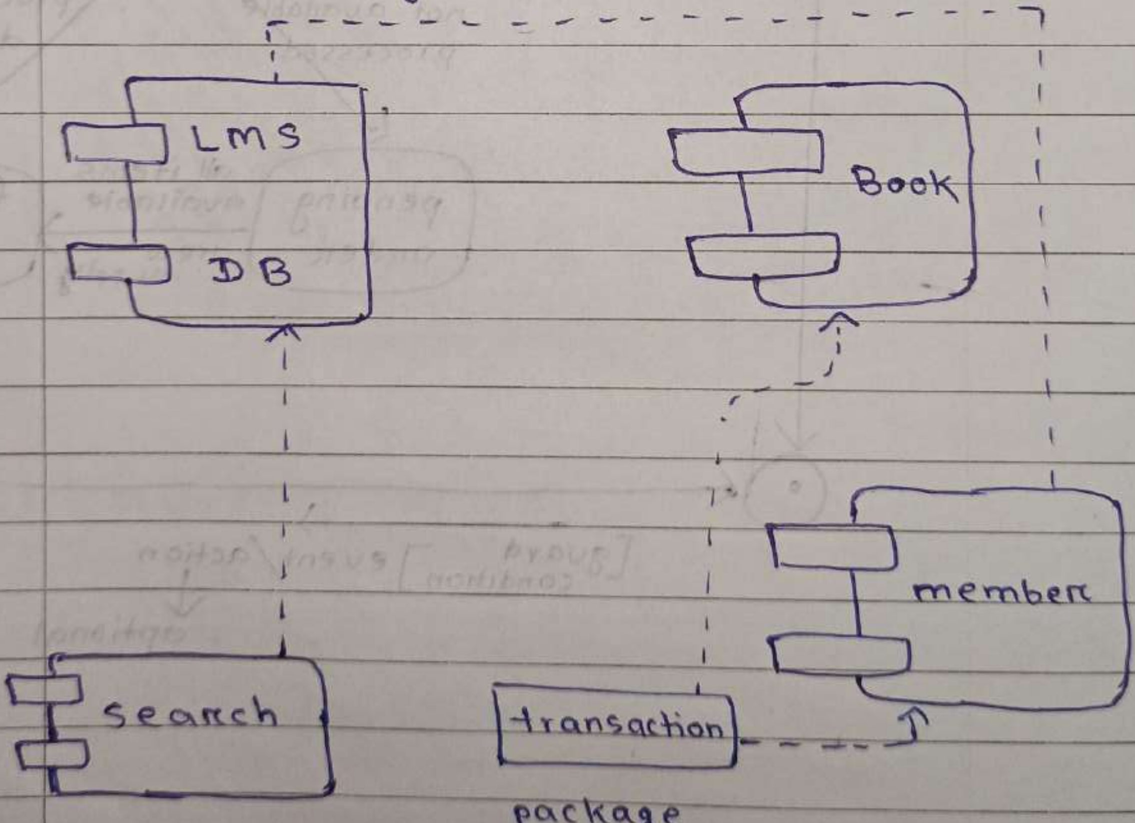
↓

Database architecture

↓

Smallest units or parts of database

Implementation view is the way by which we can understand how to implement the internal structure of the system.



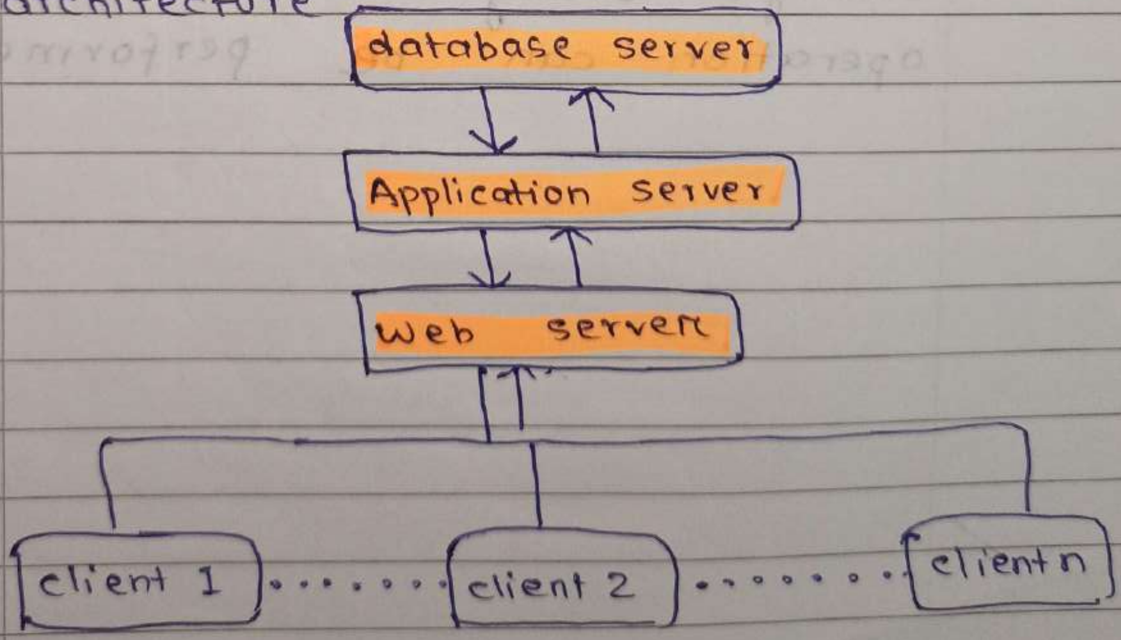
package symbol

Package - It's the main operation inside the database to activate all the components.

Development diagrams are used to visualize the topology of the physical components of the system.

It consists of nodes & their relationships.

It's based upon the client-server architecture



Characteristics of a good SOFTWARE DESIGN

1. Correctness

A good design should correctly implement all functionalities identified in the SRS document.

2. Understandability

A good design is easily understandable Any user can understand it easily.

3. Efficiency

It should be efficient

4. Maintainability

It should be easily available to change. Easily added & delete operation can be performed.

- semantic error
- server error
- logical error
- runtime error

Testing

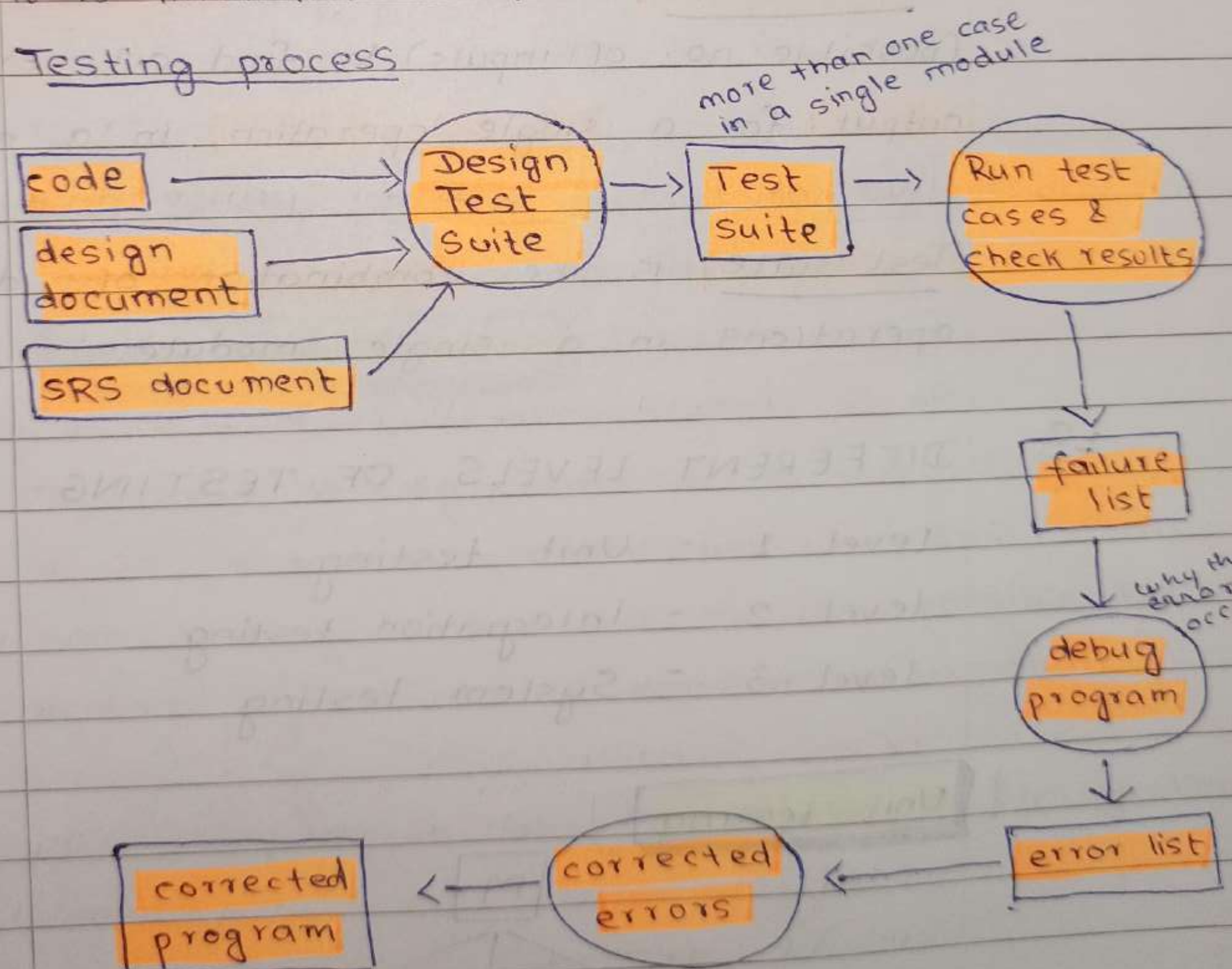
Software testing is the process of locating exceptional error during the implementation of a system by using testing software.

Objective - To test the codes and uncover all errors.

It also demonstrate whether the components of a software product is working satisfactorially according to SRS w.r.t the functionality, facility and performance.

It is intended to find out the errors.

Testing process



__/__/__

Test case [I, S, O] - No. of cases/condⁿ to run the program

I = Input to the system

S = State of the system

O = Expected output of the system

$t_1 = \langle (2,3), (2.5, 4.5), (2.395, 4.69) \rangle$

$t_s = \{t_1, t_2, t_3\}$

A test case is the set of no. of cases (possible no. of inputs) to find out expected output for a single operation in a module.

Test suite is the combination of different operations in a single module

LQ

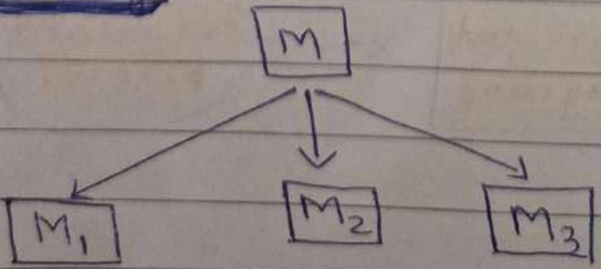
DIFFERENT LEVELS OF TESTING

level 1 - Unit testing

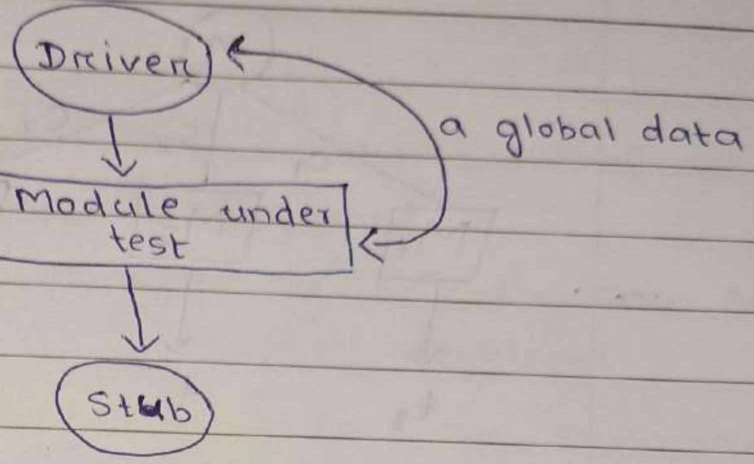
level 2 - Integration testing

level 3 - System testing

Unit testing



__/__/__



Driver

It is a module who drive outs the global data from driver to module under test and same manner in the reverse.

Stub

It is another module present in the testing software who only stimulate the unit testing process.

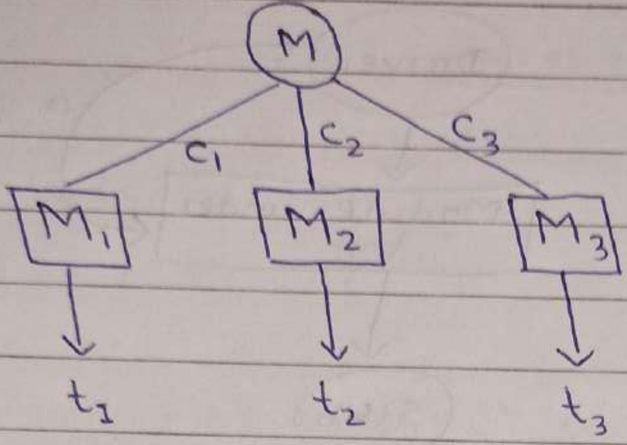
Integration testing

To avoid linking error we do integration.
It is a process which will be done after unit testing to find out the all possible linking errors in between 2 unit tested module.

Two approaches are there to do integration testing

1) Incremental approach
2) Non-incremental approach

__/__/__



1) Incremental approach

In this approach we can combine all the unit tested module.

There is a chance of error.

So, we can move to non-incremental approach.

Integration testing → Integration testing is a process which leads to linking of two unit tested modules to test it to find the final tested result.

It checks calling of global data.

2) Non-incremental approach

Only two unit tested module can be combined at a time then the result combine with another module.

System testing → It is a process by which the total output of integration testing will be tested at a time for a system.

Three types of testing

1) α-testing - done by own set of software development

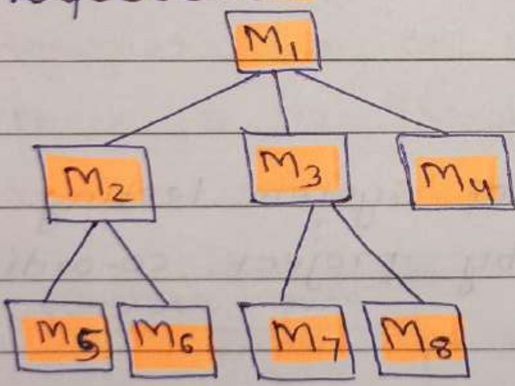
2) β-testing - done by friendly set of software developer

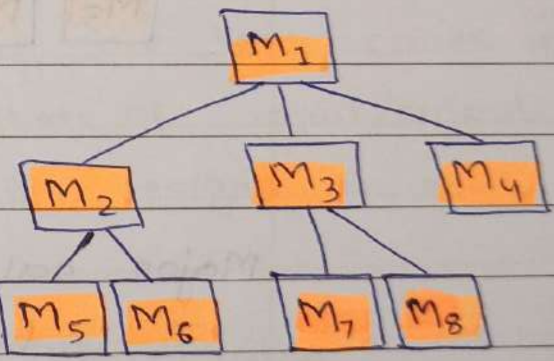
3) Acceptance testing - done by customer

Types of integration testing

1. Top down testing
2. Bottom-up testing

Top-down





Breadth first search

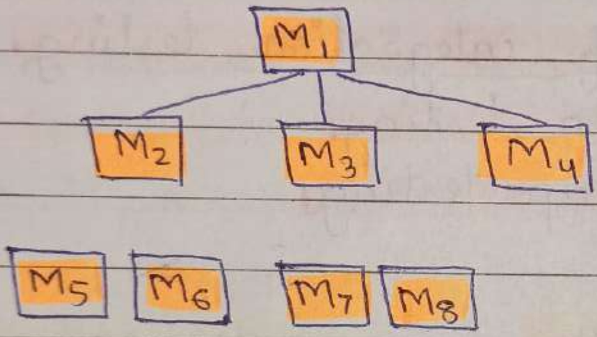
$M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4 \rightarrow M_5 \rightarrow M_6 \rightarrow M_7 \rightarrow M_8$

Depth first search

$M_1 \rightarrow M_2 \rightarrow M_5 \rightarrow M_2 \rightarrow M_6 \rightarrow M_3 \rightarrow M_7 \rightarrow M_3 \rightarrow M_8 \rightarrow M_4$

Bottom up

As the name indicates it is a method where two lower level modules will be integrated to create a temporary module which will be then forwarded to find out the original content of the main module.



Major category of system testing
(done by project co-ordinator)

1) Volume testing
It checks data structure of the module

2) Configuration testing
It checks the designing or programming style of the module

3) Compatibility testing
It checks the changing of the environment
It checks the platform independency of the module.

__/__/__

Imp 4) Regressing testing

$t_s = \{t_1, t_2^6, t_3\}$ – regressive

$t_1 = \langle (2,3),(4.5,6.5)\rangle$

$t_2 = \langle (\ ),(\ )\rangle$

$t_3 = \langle (\ ),(\ )\rangle$

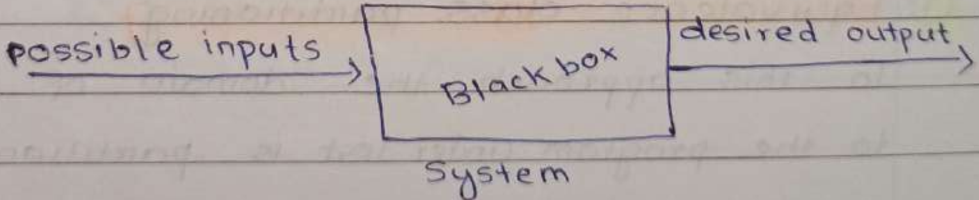
LQ TYPES OF TESTING

1) Black-box testing

2) White-box testing

Black box testing

- In these types of testing the test cases are designed from an examination of input/output.
- There is no knowledge of design or code to be required



	Black box testing	White box testing
Defination	BBT is a software testing method in which the internal structure/ design/implementation of item being tested is not known to the tester.	White box testing is a Software testing method in which the internal structure /design/implementd of item being tested is known to the tester.
Levels Applicable:	Higher levels Acceptance testing System testing	Lower levels Unit testing Integration testing
Responsibility	Independent software tester	Software developers
Prog. Knowledge	Not required	Required
Implementation knowledge	Not required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

TYPES OF BLACK BOX TESTING

① Equivalence class partitioning

In this approach the domain of input varies to the program under test is partitioned into a set of equivalence classes.

The partitioning is done such that for every input data belonging to the same equivalence classes.

Disadvantage - Large partitions leads to risk of missing defects.

__/__/__

② Boundary value analysis

Testing based on test cases.

This technique is applied to see if there are any bugs at the boundry of the input domain.

It helps in testing the value of boundary between both valid and invalid boundary partitions.

Disadvantage - It can not test all combinations of inputs and therefore can't identify problems resulting from unexpected relationships between Input types.

- It can not detect Algorithm errors, Array size and specification errors.

WHITE-BOX TESTING

As the name indicates it's the best procedure to handle the testing for a particular system.
In this situation the software developer has lot of Knowledge regarding design.

- Error detection for white box testing

→ Incorrect function - linking error betn modules

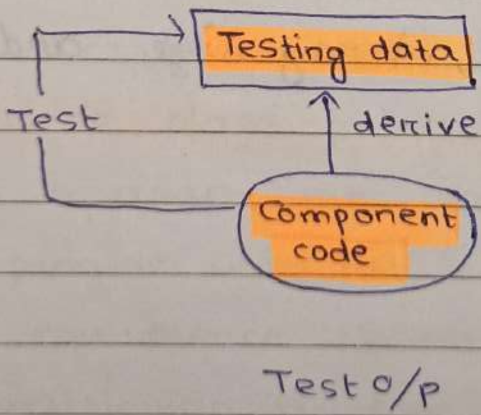
→ Data structure errors - given input not properly run by all modules

→ Missing error - module has been declared but it's not running

→ Performance error - not showing 100% efficiency

→ Database error - giving the data and taking the data from database

→ Initialising & termination error
error regarding input given & output found



White box testing is also known as glass box testing, logical testing, path oriented testing, clear box.

__/__/__

Imp

Types of white box testing

① Fault based testing / Mutation testing

② (Code) coverage based testing

① Fault based testing

It is used to detect certain types of faults in the model.

Fault - gives rise to error (logical error like in coding)

Error - Many faults present& after compilation, mistake comes

Failure - lots of error leads to failure

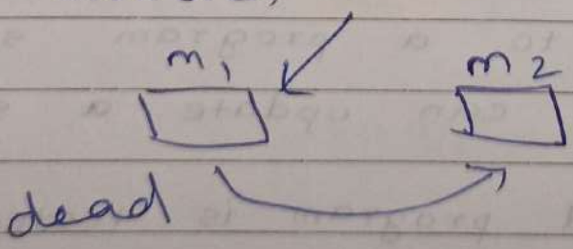
- Mutation testing is used to make a few arbitary changes to a program at ~~a time~~
- Each time it is changed it is called a mutated program & the change affected is called mutant.
- The mutation operator makes specific changes to a program such one mutation operator can update a single program
- A mutated program is tested against the test suite of the program
- If all the test cases will be accepted by the mutated program

then it is called Alive.
Otherwise it is called dead
[For a single particular module]

- If a mutant remain alive even after the testcases having exhausted then the testsuite will be enhanced to kill the mutant.
(when therre is control relationship among the module)

[This will work when it has control relationship betn modules]

Killing - It happens when there is control relationship among modules



(2) coverage based testing

code coverage - $\frac{\text{no. of statement executed}}{\text{no. of statement declared}}$

(a) Statement coverage

It aims to design test cases so as to run every statement in a program at least ones.

Every statement will be run to find out the error present is that statement

```
1. print sum (int a, int b) {
2. int result = a+b
3. if (result >0)
4. print col ("red", result)
5. else if (result <0)
6. print col ("blue", result)
7. }
```

a = 3
b = 9

a = -5
b = -8

```
1> Print sum (int a, int b){
2> int result = a+b
3> if (result>0)
4> else if (result <0)
5> print col (blue, result)
6> }
```

```
1> print sum (int a, int b) {
2> int result B = a+b
3> if (result >0)
4> print col ("red", result)
5> }
```

(b) Branch coverage

(How many looping are there)

Branch coverage is a method by which we can understand how many number of branches covered in the program

$$BC = \frac{\text{no. of executed branches}}{\text{Total no. of branches}}$$

Control flow graph (CFG)

By the help of CFG branch coverage is testing.

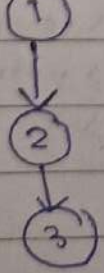
CFG shows the how many statement running inside the graph.

CFG is of 3 types of ways

1. Sequence
2. Selectection
3. Iteration

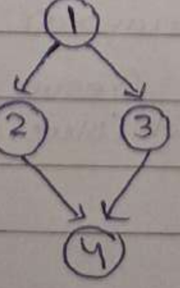
Sequence

```
1. a = 5;
2. b = a*2-1
3. c = b-4
```



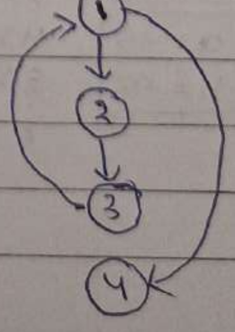
Selection

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

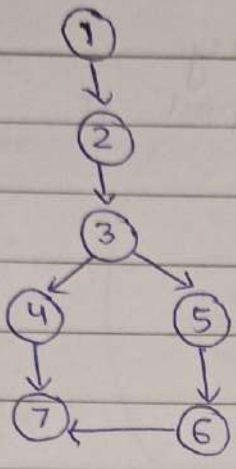


Iteration

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



__/__/__



12347 (Branch 1)
red

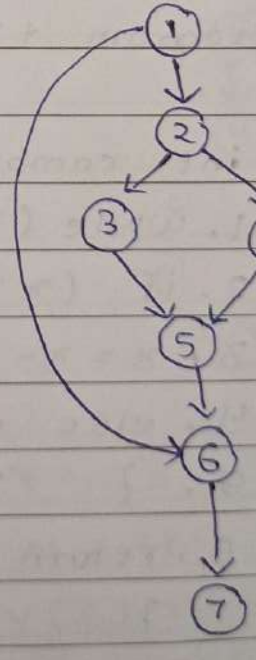
123567 (Branch 2)
blue

12357 (Branch 3)
a = 0
b = 0

(c) Path coverage

Path coverage refers to designing test cases such that all linear independent paths in the program will be run at least once.
A linearly independent path can be defined in terms of control flow graph application (CFG)

```
1. if a=50
2. then if b>c
3. then a=b;
4. else a=c;
5. end if;
6. end if;
7. print a;
```



Method for measurement of white box testing using

Cyclomatic complexity

- Mccabe's developed it

Method

① $\boxed{V(G) = E - N + 2}$

V = vertex

G = Graph

N = No. of nodes

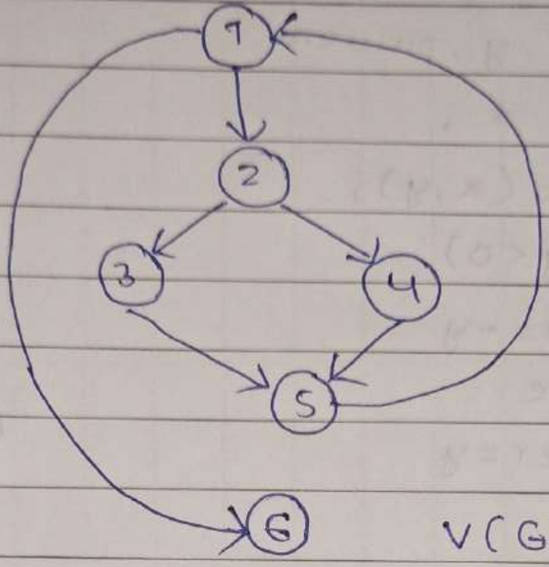
E = No. of edges

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

③ No. of decisions & loop statement of the program + 1

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

__/__/__



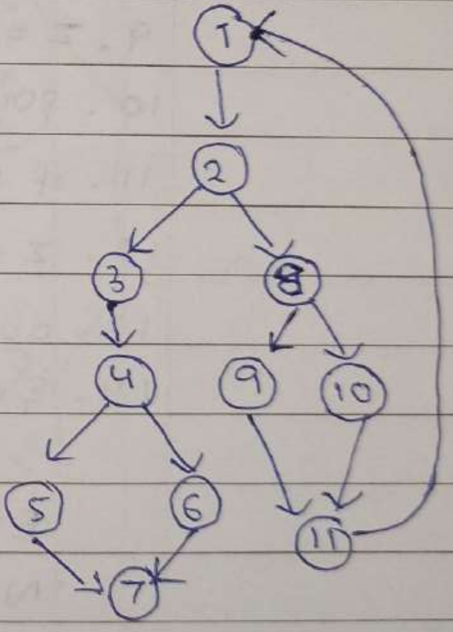
E = 7

N = 6

$V(G) = E - N + 2 = 3$

Q

```
1. Input  A, B, C
2. while (A < 20) {
3. print (A+B);
4. if (A == C)
5. print A;
6. else
7. print C
8. if (C <= 100)
9. print (A+B)
10. else
Print (A-B)
11. end;  }
```



- E = 13

  N = 11

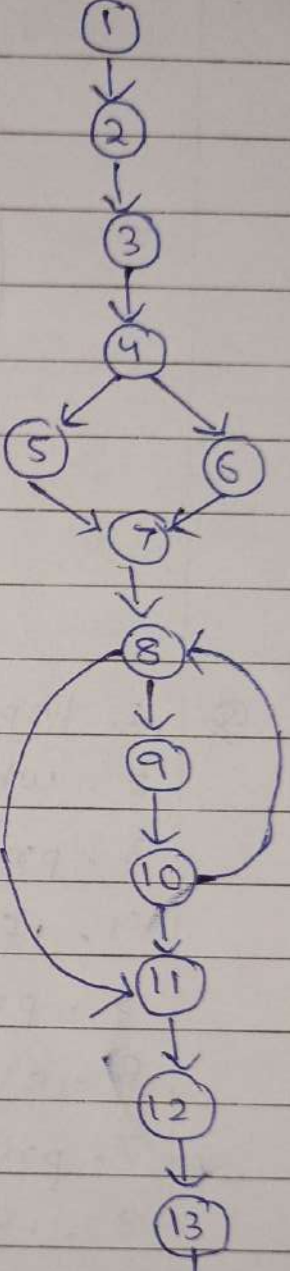
  $V(G) = E - N + 2$

  $= 13 - 11 + 2 = 4$

- $V(G) = 3 + 1 = 4$

Q

```
1. int x, y, power
2. float x ;
3. input (x,y);
4. if (y<0)
5. power=-y
   else
6. power=y
7. z = 1
8. while power != 0 {
9. z = z * x
10. power = power-1 ; }
11. if (y<0)
12. z = 1/z
13. output(x)
14. End
```



$E = 15$

$N = 14$

$$V(G) = E - N + 2$$

$$= 15 - 14 + 2$$

$$= 3$$

$$V(G) = 2 + 1 = 3$$

Software maintainance

Process by which s/w product can be uttered by any updation before delivering the customer.

3 basic procedure

1) Corrective maintainance

- Updation done by the project cordinator
- where the customer will check the updated module

2) Adaptive maintainance

- Changing of the environment done infront of the customer
- the project adaptable in any kind of environme

3) Perfective maintainance

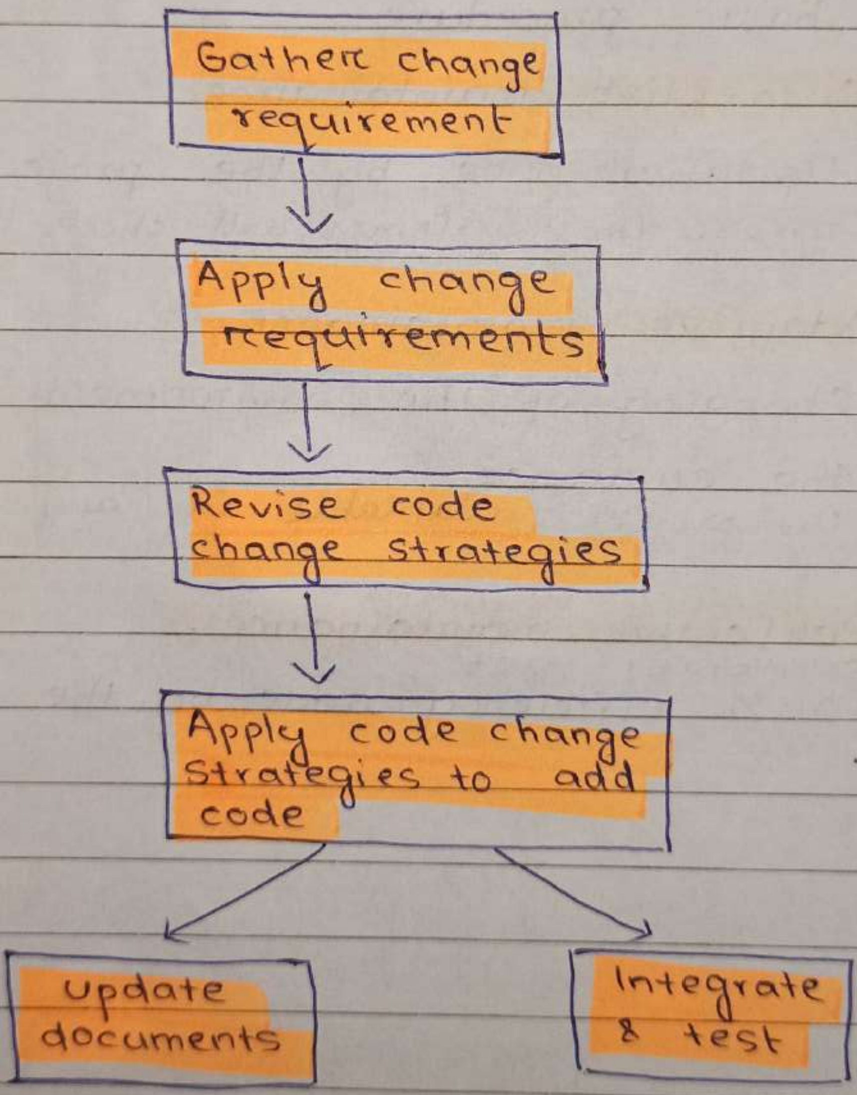
customer get
100% efficiency done by the software develope
.

Maintainance Process :-

Boham's maintainance process models

Process model 1:

Based on concept of forward engineering in software engineering.

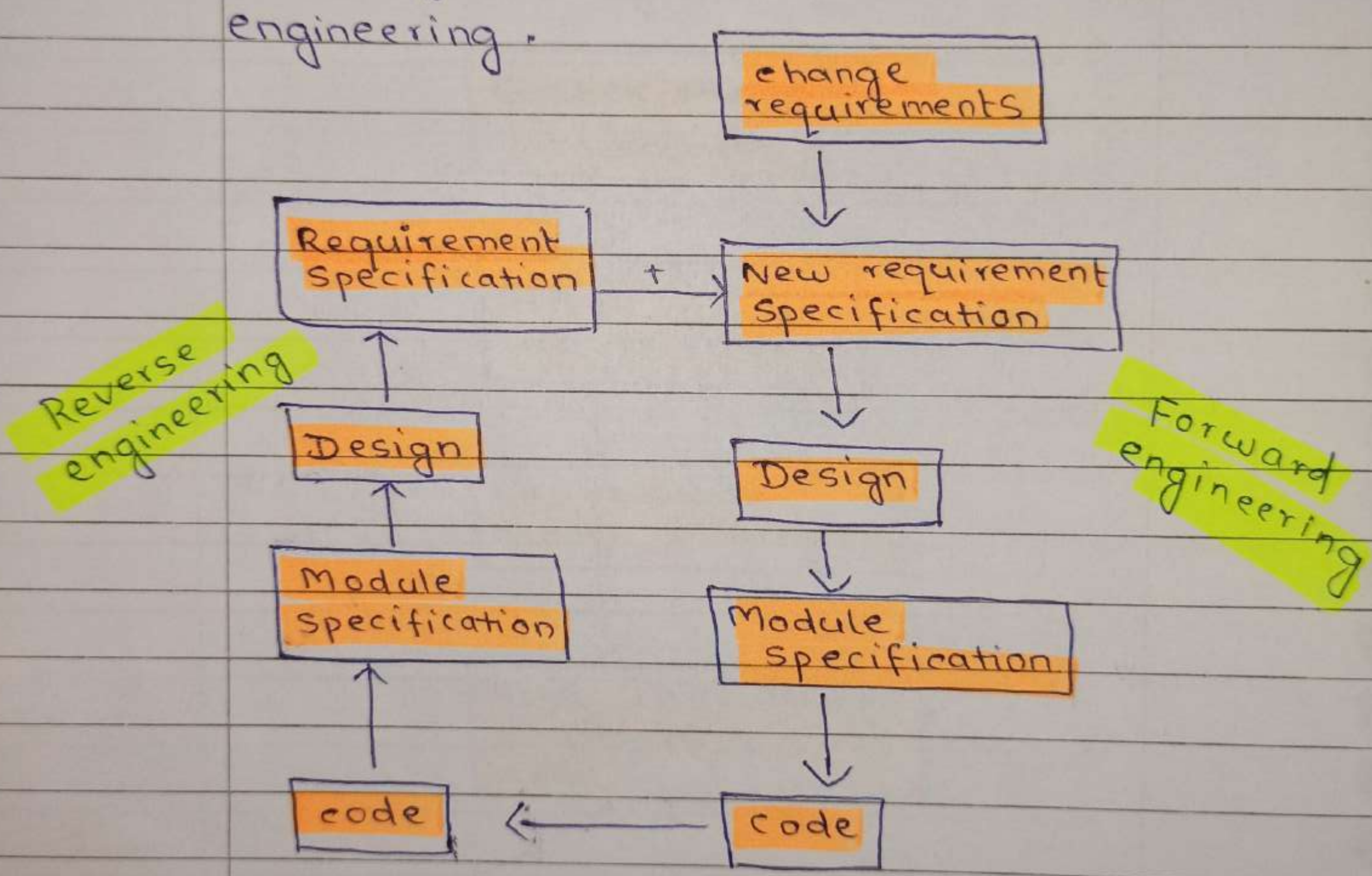


Gather change requirement
Apply change requirements
Revise code change strategies
Apply code change Strategies to add code
update documents
Integrate & test

Process model 2:

(Maintainance process model)

It is based upon the concept of both forward and reverse engineering by following the rules of software engineering.



Module Specification

To specify the code of the new module over the old module.

__/__/__

Estimation of m.cost

Imp ACT : Annual Change Traffic
It is the way of changing the capacity of the code.

$$ACT = \frac{KLOC\ added + KLOC\ deleted}{KLOC\ total}$$

Maintainance cost = ACT x dev. cost

K = Kilobyte
LOC = line of code

Estimation of maintainance cost

If a software product cost is 10 lakh for development, compute the annual maintainance cost every year app. 5 % of code needs modification. Identify the factors which gives the maintainance cost estimation.

$$\text{Annual Charge Traffic (ACT)} = \frac{\text{KLOC (added)} + \text{KLOC (deleted)}}{\text{KLOC (total)}}$$

Maintainance cost = ACT x development cost

SOFTWARE REUSE

Software reuse

Process where the existing composed will be reused in the new environment to develop a new system/new software product

What can be reused ?

- Design
- Requirement specification
- code
- Test cases
- Knowledge

Advantages

- Increase productivity
- Reduce development time
- Reduce maintainance cost
- Produce better quality software product

Basic steps in any reuse progress

- component creation
- component indexing & sorting
- component searching
- component understanding
- component adaption
- repository maintainance

component - It is a module which will be choosen from existing modules

component creation - It is the process for the modules which will be choosen from the existing modules and that creation process is component creation.

component Indexing & Sorting - Here the modules are combined together one after another and then it is sorted one by one.

component searching - Here it means searching for a particular module which the developer wants to search.

component understanding - Here the software developer understand the whole program for which it is done & also understand its each & every module why it has been added in the program

component adaption - changing to new environment. Suppose 1st program is being done in 1 env. & developer want to design a new program by taking the Reference of the 1st one. So that the 1st one is platform independent so that it can adjust in any environment & can be executed.

Repository maintainance

It is used to store the whole (final) program created by developer by taking reference of previous modules to create a new module & make ready for execution process.

__/__/__

Software reliability & quality management

Reliability of a software product can be defined as the probability of the product working correctly for a given period of time.

2 types of reliability
- Hardware reliability
- Software reliability

Reliability metrices

1. ROCOF - Rate of Occurrance of Failure
2. MTTF - Mean time to failure
3. MTTR - Mean time to Repair
4. MTBF - Mean time between failure
5. Availability

1. It measures the frequency of occurance of failure.

$$\text{ROCOF} = \frac{\text{Total no. of failures observed}}{\text{The duration of observation}}$$

2. It is the time between 2 successive failure.

__/__/__

3) It measures average time it takes to track the error causing the failure and to fix them.

4) MTBF = MTTF + MTTR

5) Availability is a term or a factor by which a software developer make the availability of the system to the next user.

Types of failure which affects the reliability

1> Transient
2> Permanent

} Programatic error

3> Recoverable
4> Unrecoverable

} System Error

Software quality

The quality of a software product is defined in terms of it's fitness of purpose.
It means how much it is fit to do its purpose.

Factors of software quality

(1) Portability - adjust in any kind of operating system
used in a particular time period
(2) Usability - easily used by user
depends upon quality of design. Usability has
(3) Reusability - also performed for better quality. Reusa ⇒ Usabili
feature where we map the content
pre doc matches final doc (4) Correctness - of SRS to the env. (operating system)
(5) Maintenability - low - maintain the design
- add^n of more info should not hamper the original document

Imp ISO 9000 < 9001, 9002, 9003

International Standard Organisation

It's used to recognize the good quality product of an organization.
It carries out 3 versions - 9001, 9002, 9003

ISO 9001 - This certification will be given to the organisations who deals with the design, development (coding), production and servicing of goods.

ISO 9002 - This certification will be given to the organisations which ~~do not design the products only~~ involve in the production & designing

ISO 9003 - This certification will be given to the organisation which is involved in testing of products & installatn

How to get the ISO 9000 certification

1. Application stage - apply to the ISO organisation pre-defined form
2. Pre - assesments - Rough assesments done by the registerr
   - It match the document given in form
3. Document review & adequecy audit
   - Report generated depending upon the document
4. Compile audit - done by going to organisation
5. Registration - If status of compile audit is ok then go for ISO registration
6. Continued survillience
   - Registration will be given for a particular time period.

Quality management models

1. SEI CMM (Software Engineering Institute - Capability Maturity Model)
2. 6-Sigma Model
3. PSP Model (Personal Software Process)

__/__/__

CMM model

Capability evaluation

Software process asesment

CMM model	Focus (ad-hoc)	(KPA) Key process area
1. Initial	competent people	
2. Repeatable	project management	Software project planning
3. Defined	definition of process	process def^{n} (training program)
4. Managed	Product & process quality	quantitative process metrics software quality managem^{nt}
5. Optimizing	continuous process improvement	defect prevention

1. It is characterised by ad-hoc activities where all the software engineers are very competent to complete the task.

2. It is the second level of the CMM model where the old document will be repeated in new document in new environment. For this reason focus will be implemented for project management. It means deciding the no. of modules to be recoded. For this we have to chose a KPA ie software project planning which shows

the path for the method to be reused.

3. This level tells to write the code by the members of the team in which focus will be implemented on the defination of process.
For this we have to assign a KPA Known as process defination by giving a suitable training program for the non-competant team member.

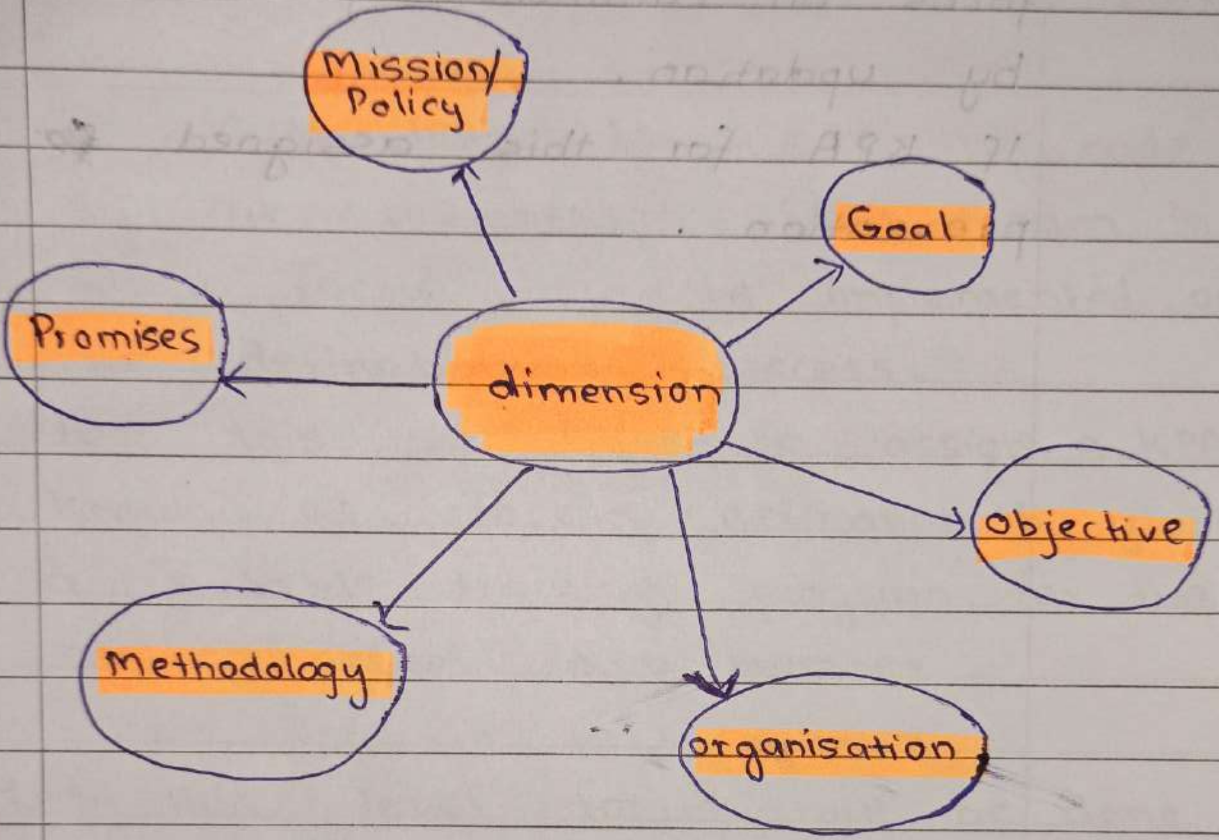
4. In this level focus will be done for product & process quality to measure the no. of LOC present in new module.
For this KPA will be assigned software quality management which will improve the quality of the product in terms of less no. of LOC.

5. This is the level which will optimize the software product by doing the

focus on continuous process improvement by updation.

If KPA for this assigned is defect prevention .

6-Sigma model



Mission - Aim of implementing the project

Policy - Protocols /rules to process it

Goal - Achievement to be studied to produce good quality product

Objective - Deals with continuous improvement

Organisation - (employee)

Green belt
Black belt
Master black belt
Process owner
Champions

} TOP Quality leader / manager

__/__/__

Team member || Green belt - 10 to 50% effort in project
|| Black belt - 100% effort

Master black belt - for specific area
- for a Specific project in const. time

Process owner - who coordinate the green belt, black belt & master black belt

Champions - Good process owner
complete product in time

Manager - Best champions promoted to manager

Methodology
- DMAIC - Defined Measure Analyse Improve Control
- DMADV - Defined Major Analyse Design Verify

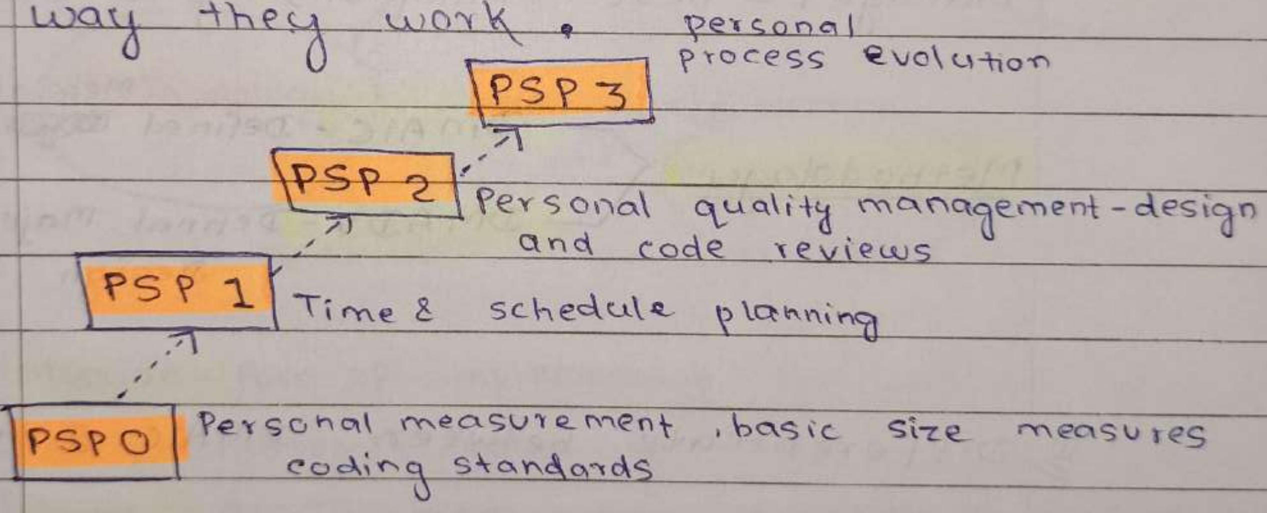
Q Differentiate between DMAIC & DMADV.

Ans - DMADV is the best because it takes feedback of customer

Promises - The method by which the customer will map the initial SRS document to the final software development to prove the quality of the software product.

PSP (Personal Software Process)

PSP is a framework that helps the engineers that measure improve the way they work.



personal process evolution

PSP 3

PSP 2 Personal quality management - design and code reviews

PSP 1 Time & schedule planning

PSP 0 Personal measurement, basic size measures coding standards

PSP 'O' - It is the measurement of a module like the basic size measurement and then the coding standards done by the software developer.

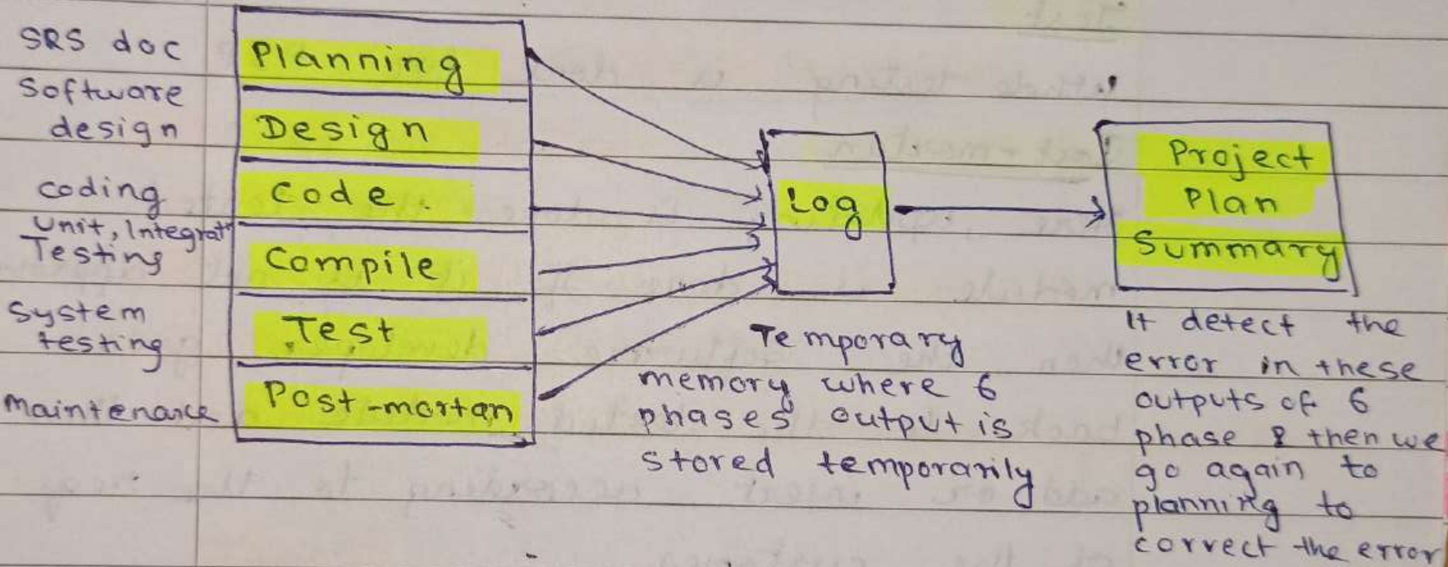
PSP '1" - In this stage the software developer develops the product in a certain time and with schedule planning.

__/__/__

PSP "2" - In this stage personal quality is being managed by design & code reviews.

PSP "3" - In this stage the software developer checks the whole test & then proceeds for the execution.

SCHEMATIC REPRESENTATON OF PSP



Temporary memory where 6 phases output is stored temporarily

It detect the error in these outputs of 6 phase & then we go again to planning to correct the error

Planning

Here paper work is done or report is done PSP model.

It's for requirement analysis & specification

Design

It is the design process for each module.

__/__/__

Code -

Coding is done for each module.

Compile

It is the testing process for each module independently and thin compilation is done one by one

Test

whole testing is done for PSP

Post-mortan

Here, updating is done the tested module is done. If it is not approved then the software developer goes back to the tested module and then add or insert according to the req. of the customer

LOG - Temporary storage base

Project Plan Summary

It holds all the incorrect document from log in a sequential manner to be corrected by the software developer itself. If needed we can move from Project Plan Summary to Planning

User- interface design

- User interface is a front end application view to which user interacts in order to use the software.
- User interface can be graphical and also text based.

Advantage of user iterface

- Attractive to see
- Simple to use
- Response in small time period
- Clear to understand
- Consistant on all interfacing screens

Types

1. Command - line interface
2. Graphical - User interface

1> CLI - 3 modes - command prompt mode
- cursor (when error comes it blinks line)
- command (syntax - executable)

__/__/__

2) GUI

It provides the user graphical means to interact with the system.

GUI elements

① window - specific area of application by the user

② tabs - sub window

③ menu - button which contains a large no. of parameters

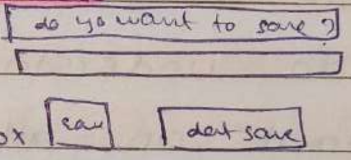
④ icons - pictures

⑤ cursor - selecting - way to run the icon

Application specific elements of GUI

① Application window - the application being run

② Dialoge box -



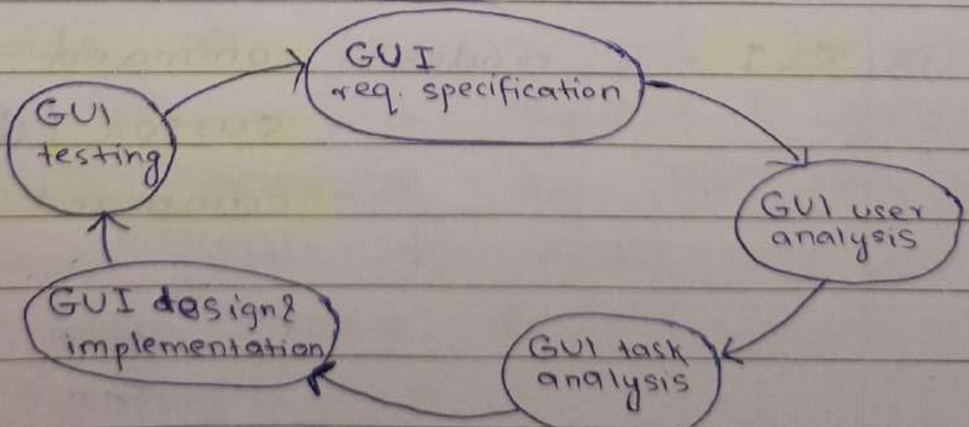
③ Text box Part of dialoge box

④ Radio button ⦿M ⦿F

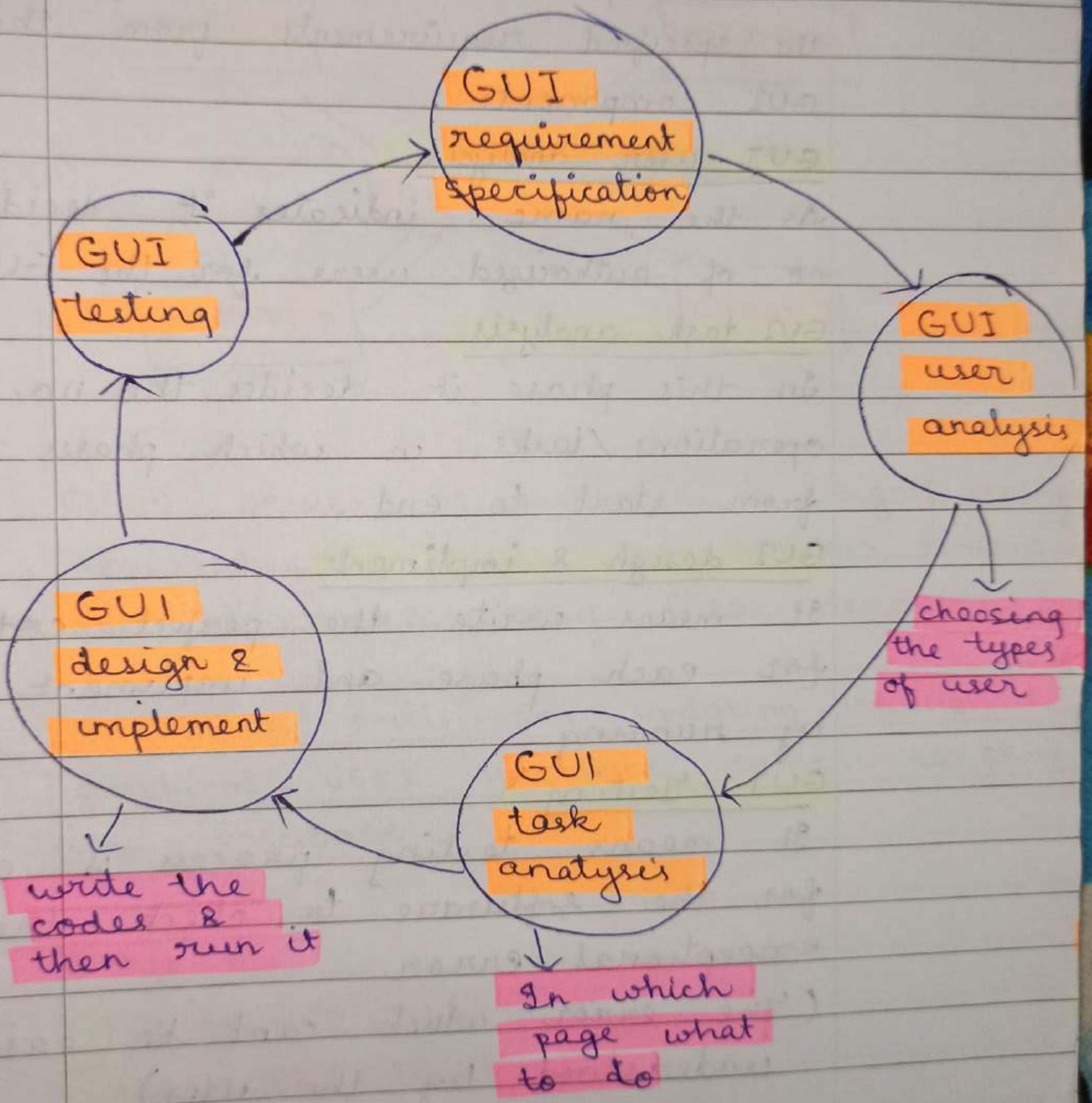
⑤ Check box - MCQ type □ □ □ □

⑥ List box - DD mm YY

select



GUI :


GUI requirement specification
GUI user analysis
choosing the types of user
GUI task analysis
In which page what to do
GUI design & implement
write the codes & then run it
GUI testing

GUI Requirement Specification

In this phase the customer gathers all the specified requirements from the GUI components.

GUI user analysis

As the name indicates it decides the no. of authorized users for the GUI system

GUI task analysis

In this phase, it decides the no. of operations / tasks in which phases to link from start to end.

GUI design & impliment

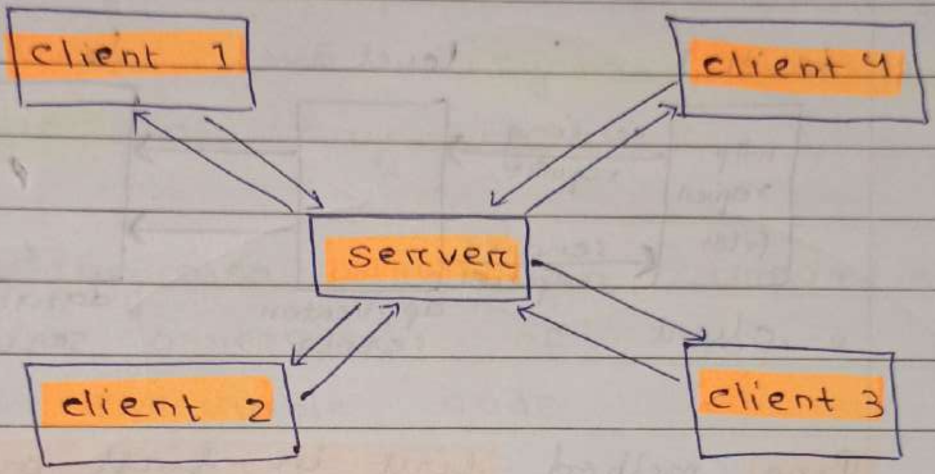
It means write the perfect coding for each phase and impliment it by running.

GUI Testing

It means testing process is done for the software to check the exceptional error
(The error which can't be easily understood by the user)

CLIENT- SERVER ARCHITECTURE

Client-server software engineering



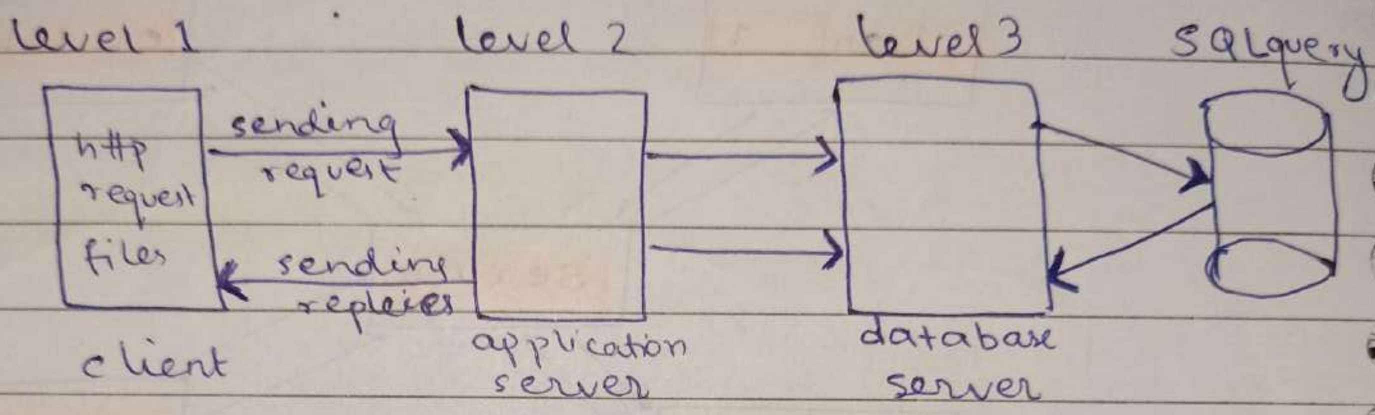
Client server architechture is significantly designed to reduce the network traffic and to make easy files transfer.

It allows multi-user updating through a graphical user interface content to share the database.

2 types

1. 2 tier / 2 level
2. 3 tier / 3 level

Client-server software engineering



Pipo method - first in - first out

Application server

Database server stores the data

_/_/_

Coding

Coding is a process by which we can maintain the syntax, semantic and logic of any code.

Coding also plays an important role in conversion of source code to executable code.

Coding standards

1) Rules for limiting the use of global variables
2) Standard headers to preceed the code of different modules
3) Naming conventions

(2) Header Package contains - name of the module
- date on which module was created
- authors name
- modification history
- synopsis of the module
- different function in module with its all inputs/outputs
- global variables accessed/modified by the module

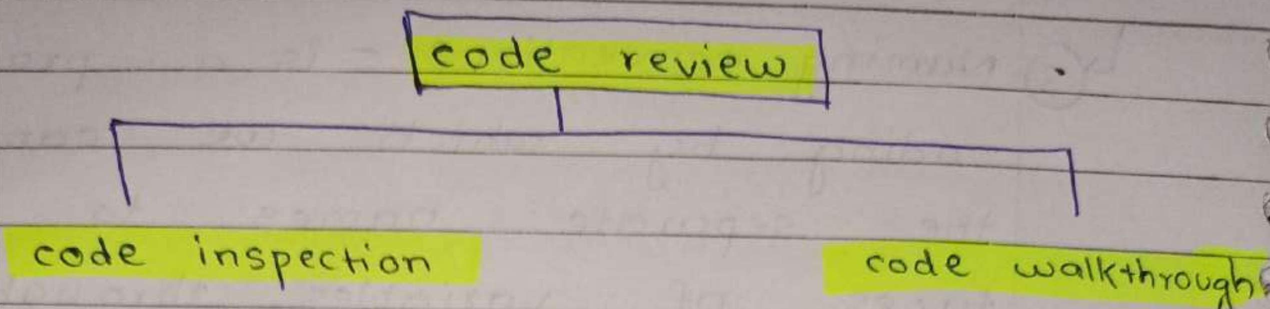
_/_/_

↳ (3) Naming conventions - is a process in coding by which we can assign the separate names to the diff. types of variables throughout the program.

local variable - int a = 5

const. variable - const - int a = 10

global variable - int



Code review for a module is done after the module successfully compiles ie all the syntax error having eliminated from the module.

Review directly detects the error which can be easily corrected by testing.

- Code inspection - is a process which is used to find out the some common programming error by the set of friendly set of S/w dev.
- Code walkthrough - is a process which will be done after the successful completion of code inspection. It is a primary process of testing where all the primary test cases will be mapped to each line of code so that it will give the correct output to the user.

__/__/__

Q What types of error to be detected by the friendly set of S/w developer.

Ans- 1. use of uninitialized variables

2. Jumps into loops.

3. Non-terminating loops

4. Array indices out of bounds

5. Improper storage allocation & deallocation (to make memory utilization high).

6. Mismatch bet^n actual & formal parameter in procedure call

7. Use of incorrect logical operators.

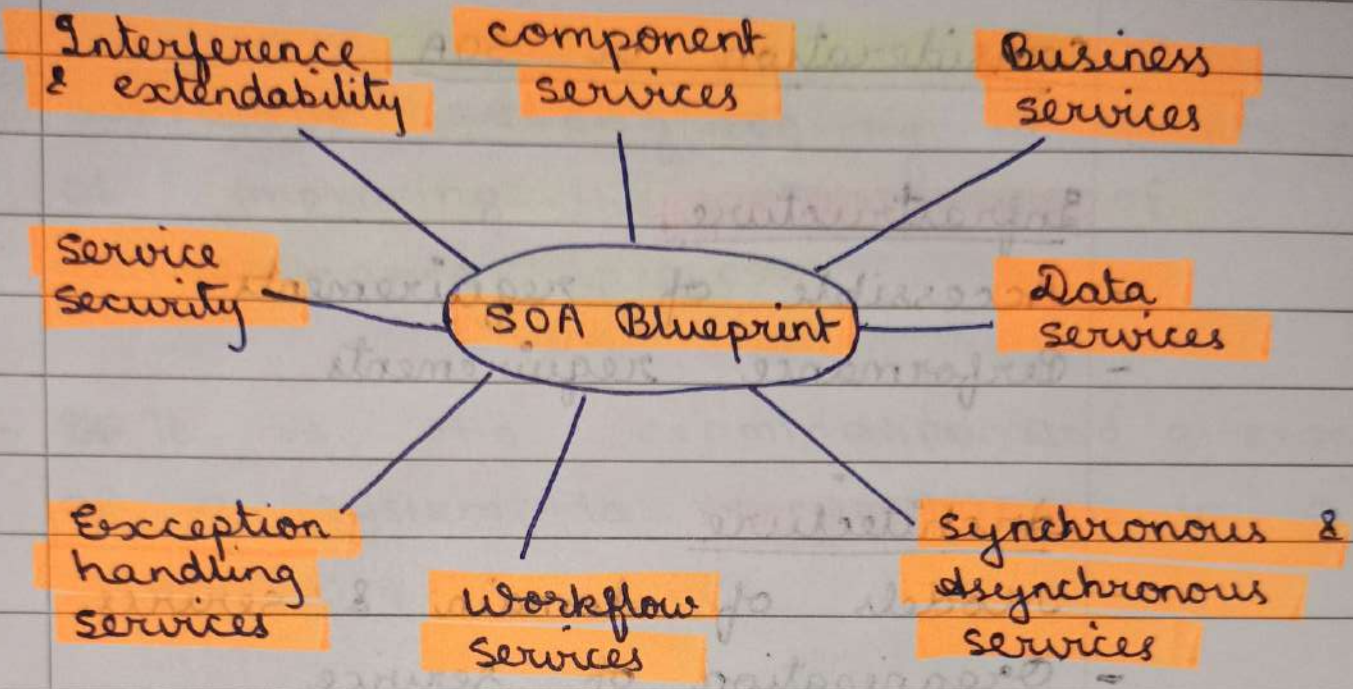
SOA (Service oriented Architecture)

- The service oriented Architecture is an architectural design which includes collection of services in a network which communicate with each other.
- SOA uses interfaces which solves the difficult integration problems in large systems
- As it reuses the service, there will be lower software development and management costs.

SOA-blueprint

It contains following goals

- Requirements of design principles
- Specific tasks of design principles
- Interaction of services
- Details of Integration scenario
- Templates for the specific tasks


Interference & extendability
component services
Business services
Service security
SOA Blueprint
Data services
Exception handling services
Workflow Services
synchronous & Asynchronous services

Consideration in SOA

Infrastructure

- Accessible of requirements
- Performance requirements

Architecture

- Models of domain & service
- Organization of service
- Quality of the service
- Message exchange patterns

Development

- Design guidelines for project development
- Required tools for project
- Validation and modification required things
- Handling errors
- Security for service access

Administration

- Managing & building
- Testing & deploying the project
- Location of data stored & registering the application.

SOFTWARE RE ENGINEERING

- Software re-engineering is a process of improving the efficiency of maintainance process.

OR

- It is the examination and alteration of a system to re-consitute it in a new form.

- It is done where the system(modules) are very large.

Why do you need Software Re-Engineering

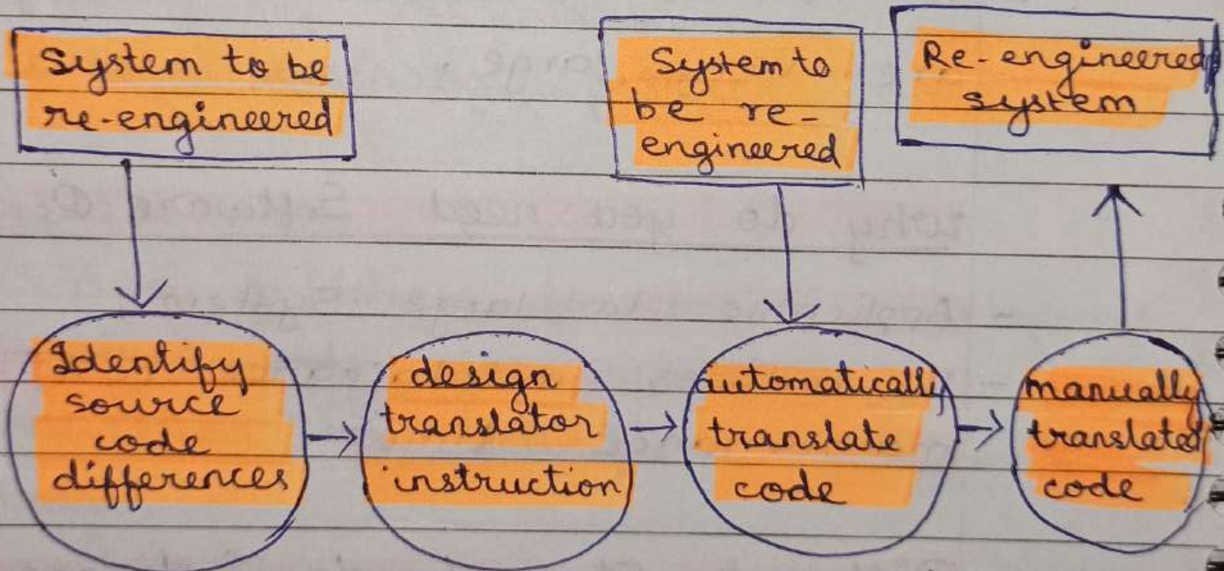
- Applicable to large System
- It reduces maintainance cost and maintainance time

Different Steps to do Software Re-Engineering

1. Source code translation
2. Reverse engineering
3. Program Structure Improvement
4. Program modularization
5. Data Re-engineering

1. Source code translation

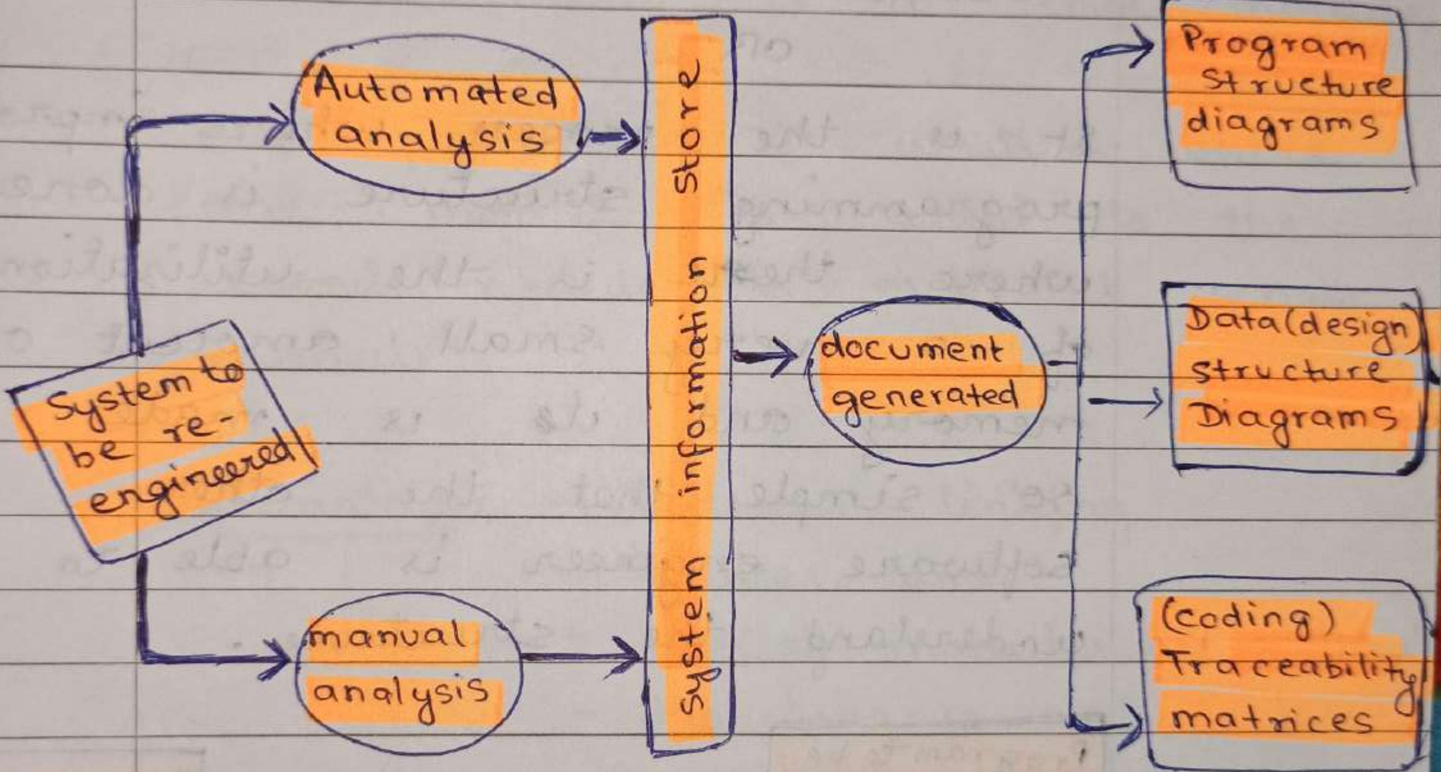
- It is the process of converting the source code from one programming platform to another programming platform for each module.
- It is used to improve Skill, Efficiency


System to be re-engineered
System to be re-engineered
Re-engineered system
Identify source code differences
design translator instruction
automatically translate code
manually translated code

2. Reverse Engineering

- As the coding changes, the whole documentation process changes and finally the whole module will be changed.

This states the reverse engineering as it states to go back and do the improved module.


System to be re-engineered
Automated analysis
manual analysis
system information store
document generated
Program Structure diagrams
Data(design) structure Diagrams
(coding) Traceability matrices

3. Program Structure Improvement

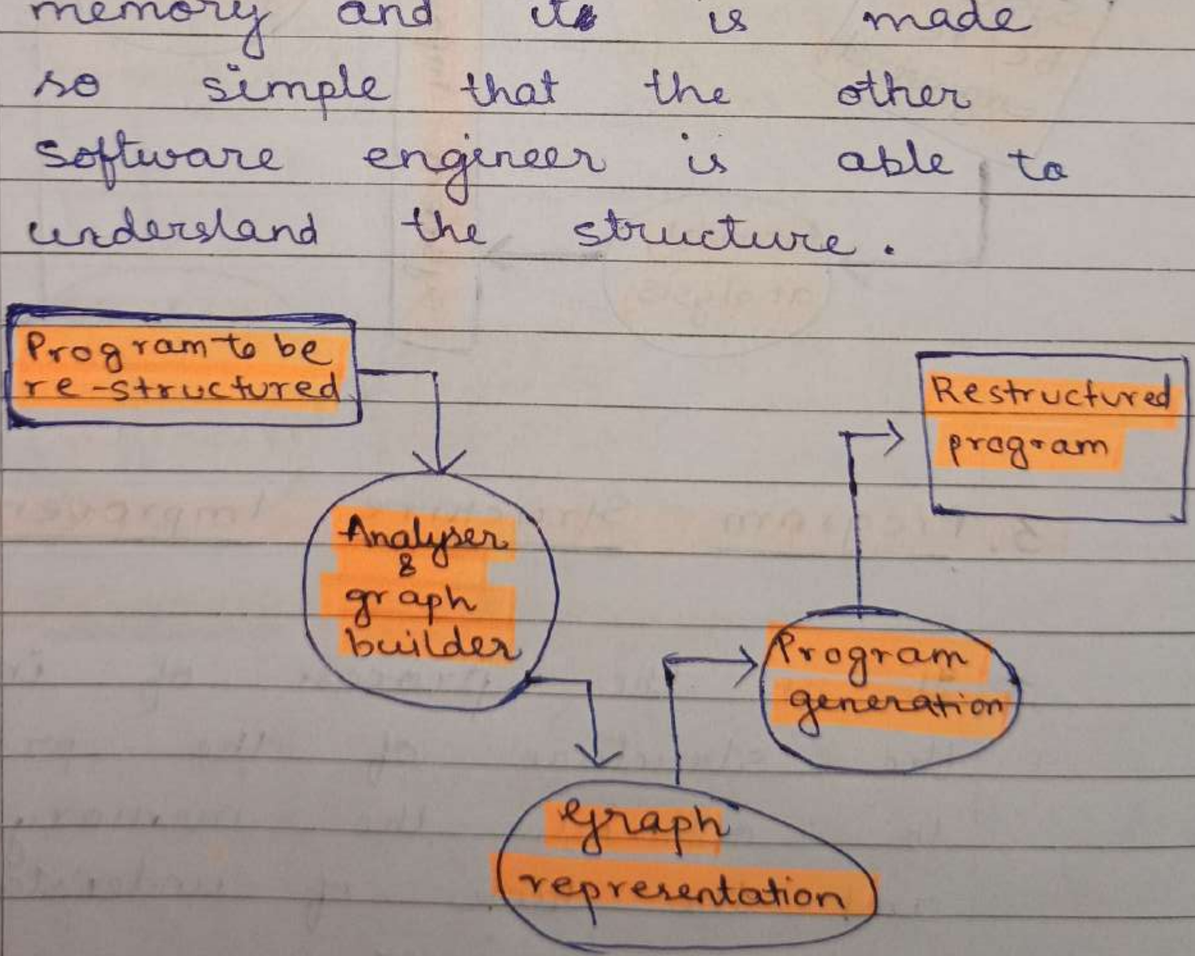
- It is the process of improving the structure of the program to optimize the memory use and the lack of understanding of software engineer.

OR

[Improving the usage of small size of memory and the other software engineer is able to understand the structure]

OR

It is the process where improved programming structure is done where there is the utilization of a very small amount of memory and its is made so simple that the other software engineer is able to understand the structure.

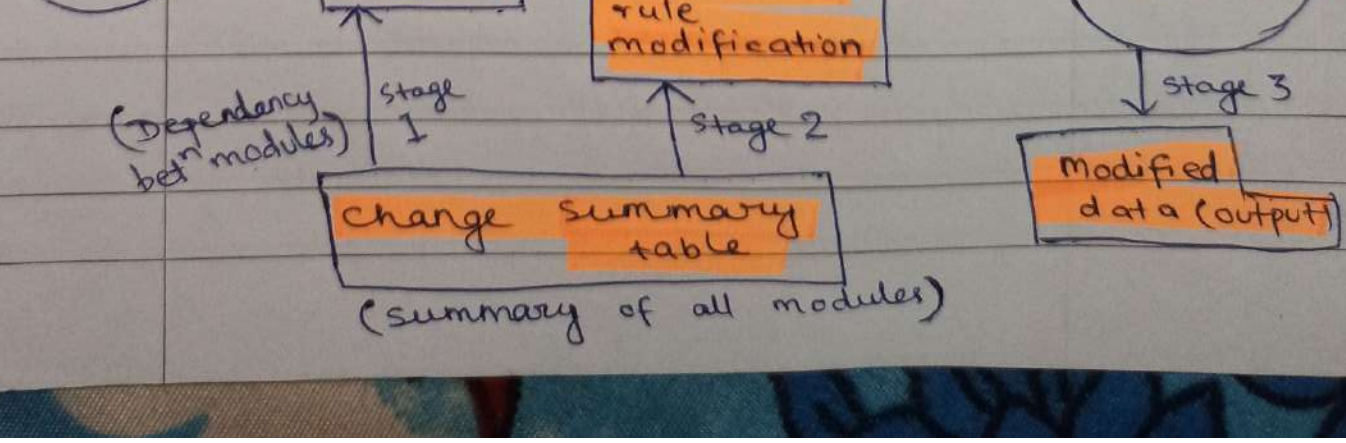


4. Program Modularisation

It is the process of re-organising a program so that all the related program parts are collected together and considered as a single module.

(The program is written where one module is added to the second module to make it single module)

Advantages - Reduces linking
- Reduces time
- No. of modules are reduced
- Maintenance cost is less
- Increase efficiency
- Decrease risk



5. Data Re-engineering

(Under process of develop)

It is the process of analysing and re-organising data-structure in a system to make it more understandable.

("Totally based upon data structure and it is internally moving part)
[Inside the system]