Software Engineering

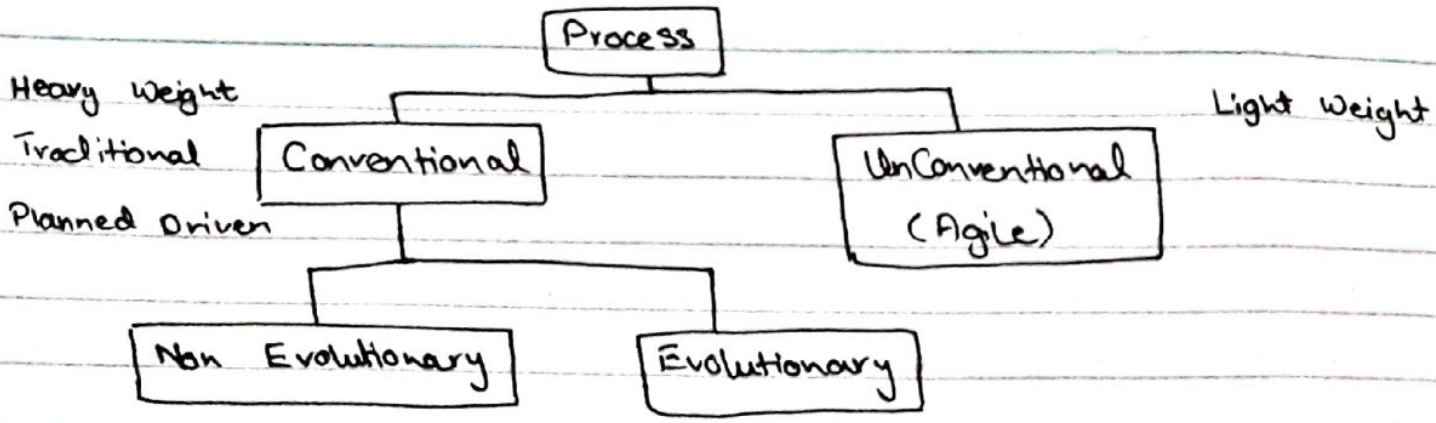
Steps of Software Engineering:

- Inception: find the problem and its solution
- Requirements Engineering: identify the resources needed
- Analysis: analyse the resources and flow of execution

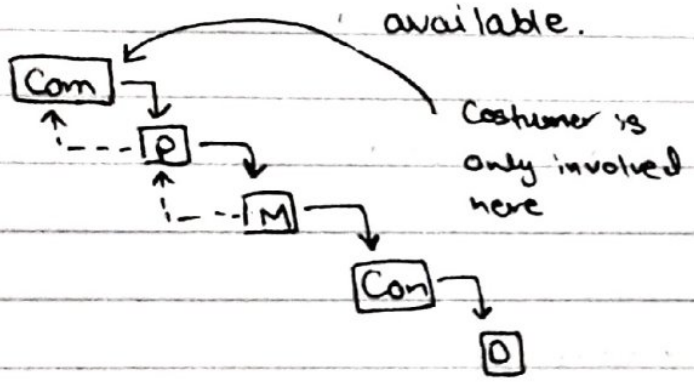
- Design: High level design, Interface, Component level design
- Programming / Implementation / Coding
- Testing: unit, integration and system testing
- Deployment / Delivery
- Operation / Maintenance: character, adaptive maintenance
- Retirement

Process: aproach, framework, road map
a process also tells us the sequence to perform activities in addition to the activities involved. Changing the sequence will result in new process

Generic Activities:-
- Communication: talk with customer & list all requirements.
- Planning: identify the ~~process~~ resources required.
- Modeling: creating architecture of the system
- Construction: coding & testing
- Deployment: delivery, support, feedback

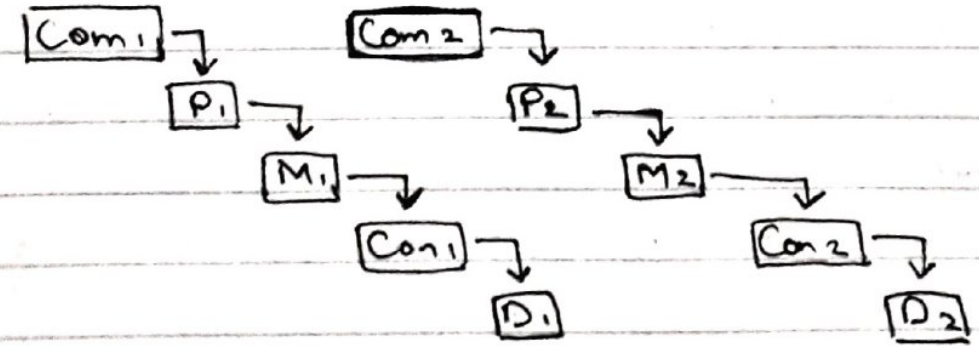


⇒ Waterfall Model: rigid and assumes that requirements are available.

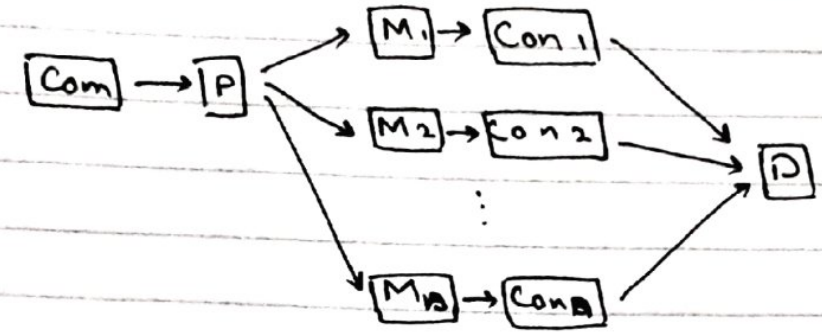


Note: Software Process Model is an abstraction of Software Process

⇒ Increamental Model: unlike waterfall model the software is divided into bits and pieces and each part of the software is developed one by one or may be symultaneously.



⇒ Rapid Application Development Model.



- This model requires alot of man power
- It may only be used if the processes are mutually exclusive

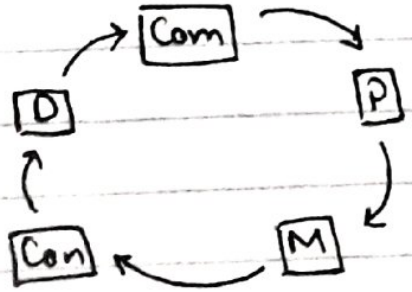
⇒ Reuse Model: using pre developed components / services which may be free or not.

Component Base Paradigme:

- saves time
- less effort
- less man power
- may or may not be cheap.
- good quality

Service Base Paradigme: is expensive than the previous one as you need to pay for service each time.

⇒ Prototyping Process: keep on making prototypes in accordance to the customer needs until the customer is satisfied. (for small scale)



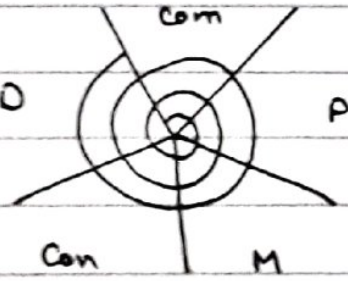
IKIWISI: I'll know it when I see it

the customer may have this syndrome

The problem with prototyping is that in order to deliver it quickly there may be many problems with it. Also the team manager may not be able to plan ahead cause the prototypes are changing very often.

- This model may be embedded with the previous models.

⇒ Spiral Model: meant for large scale products. Risk driven process.



Once a task is complete and the programmers / customer is satisfied you get out of spiral and start working on a new task

⇒ Agile Processes: too much discipline brings rigidity hence the program should have flexibility for change.

- Salient Features of Agile Manifesto:
  - Individuals & Interactions Vs Process & Tools: Individuals are the kings. Processes & Tools are important but not as much as the Individuals.
  - Working Software Vs Comprehensive documentation: The softwar should be given more important than the documentation

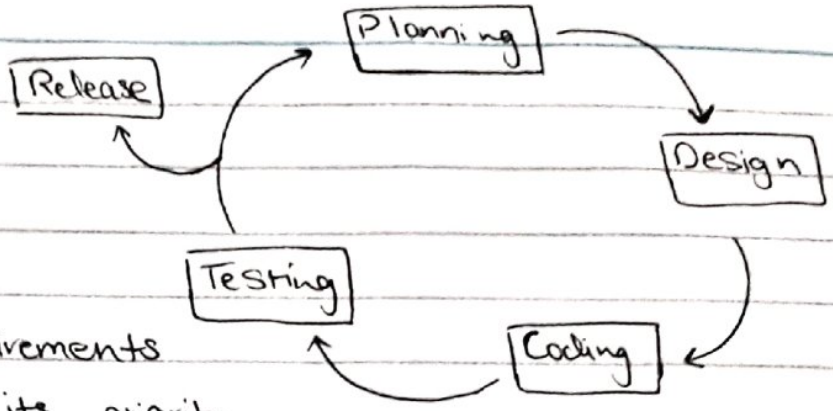
- Customer Collaboration vs Contract Negotiation: Customer & programmers are not two different entities but a single team.

- Responding to change vs following a Plan: The program should embrace change than being very rigid.

- Salient Features of Agile Processes:

  - focus on customer satisfaction
  - customer can demand change any time
  - software is delivered early, continously and frequently.
  - iterative & increamental:
    - iterative is to continously keep on changing same features
    - increamental is to start working on new features
  - small teams of many smart people
  - welcome change
  - tacit knowledge: keeping knowledge (documentation) in mind. so people should be smart
  - Self organizing team (structure, decisions, trust between the team members)
  - not suitable for large and critical processes.
  - only suitable for small programs because of no documentation.
  - In conventional processes you rely on process rigidity but in agile you rely upon team members' intelligence.

⇒ XP (Extreme Process).



Planning:

- customer should be physically present in the office (colocation).
- customer provides requirements on index cards with its priority.
- developer provides cost to the customer stories. The stories may be divided into parts.

Design:

- CRC (class Responsibility Collaboration)
- it is following OOAD principles, make cards for classes with their attributes & methods and identify the other class on which it depends.
- Follows KIS (keep it simple) principle.

Coding:

- Constant Refactoring: change code without changing the behaviour
- Continous Integration: so you can release anytime.
- Pair Programming: one person codes and the other one watches over you.
- Test First Development: before writing the code write the unit testing code.

Testing:

- User Acceptance Testing: user performs the testing

First Release:

- Project Velocity: number of stories implemented in one iteration adjust 2nd iteration on the basis of first.

⇒ Scrum: provides managerial guidance.

- Sprint: an iteration in scrum is called sprint which is 30 days long. There is a backlog of requirements with their priority. Then take out packet to be implemented in next sprint.

* Note: Scrum does not provide guidance on how to implement a scrum.

* Note: If the user demands a change then it enters the backlog instead of the current sprint.

Daily Stand-up Meeting: Every day there is a 15 min meeting wher scrum manager asks following questions from scrum team.

- what did you do?
- What obstacles did you face?
- What are your plans for tomorrow?

- Demo: release in scrum is called a demo

* Note: Scrum is applicable at individual / team / organizational level.

Software Project Management

Every project
- is unique
- is temporary
- has goal(s)
- has constraints

task activity phase project program → heirarchy

Core activities of Software Project Management
- Planning (work unit, resources, schedule, risks
- Organizing
- tracking
- controlling

Management: judicious use of means to accomplish an end.

SPM is based on 4 Ps:

- People: get the right people based on experience, education personality and organize them into teams.

⇒ Closed Team Structure: very strong & rigid hierarchy. Works in situations where the work is repetitive and not much creativity is required.

⇒ Random Team Structure: opposite of closed team structure.

⇒ Open Team Structure: combination of the above two

⇒ synchronous: cross team communication is not allowed but within team communication is performed.

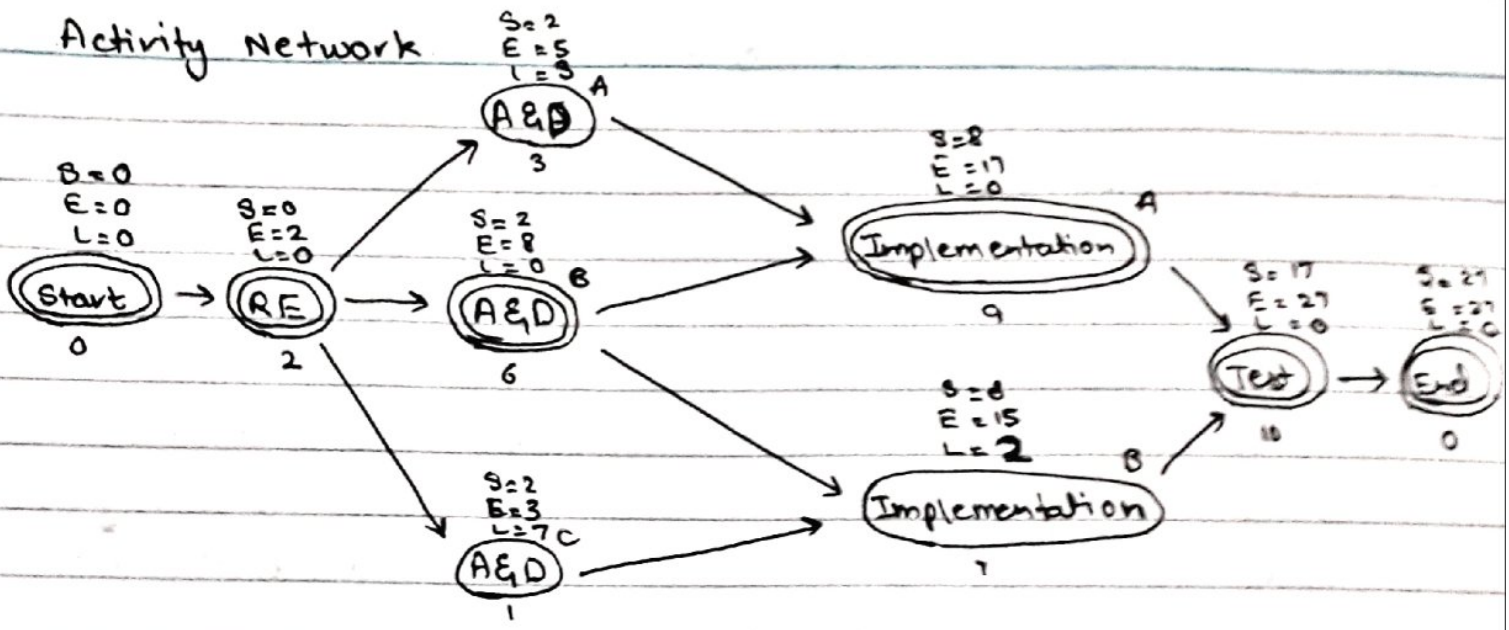
* Note: In Agile Processes Random, Open & Synchronous structure is used.

⇒ You need to motivate people to get a well gelledteam. Opposite of such a team is toxic team.

- Product: identify the product by talking to customer and assign to the people
- Process: define the process that people need to follow.
- Project: The end result customer is looking for.

Planning (Core Activity): the risks need to be quantized by impact factor and probability. Minimize risks by providing completion bonus, signing bond and pair programming.

Activity Network



Steps to find Critical Path:

- the start & end are critical
- prerequsite of critical is also critical
- in case of multiple pre reqs the one that ends in last is critical.

Paths: (To calculate lags)

1	S → RE → A&D B → IA → T → E	27
2	S → RE → A&D B → IB → T → E	25
3	S → RE → A&D A → IA → T → E	24
4	S → RE → A&D C → IB → T → E	20

Requirements Engineering

It includes Planning, Communication and part of modeling (analysis)

Software Supplier | Software Consumer

Activities of RE

1 Inception
2 Elicitation
3 Elaboration
4 Negotiation
5 Specification
6 Validation
7 Management

SRS (Software Requirements Specification)
- mostly written in simple / natural languages
- formal language (OCL, Z) [mathematic involved]
- semi-formal (uml)

Stake Holder: anyone who is instrested in achieving goals of a project and see it to completion.

1 Senior Managers
2 Project Managers
3 Practitioners
} SS

4 Customers
5 End users
} SC (mostly foaused on these two)

1 Inception: get stake holders, informal meetings, goals are provided by stake holders.

2 Elicitation: refining the goals and requirements. Some methods are:

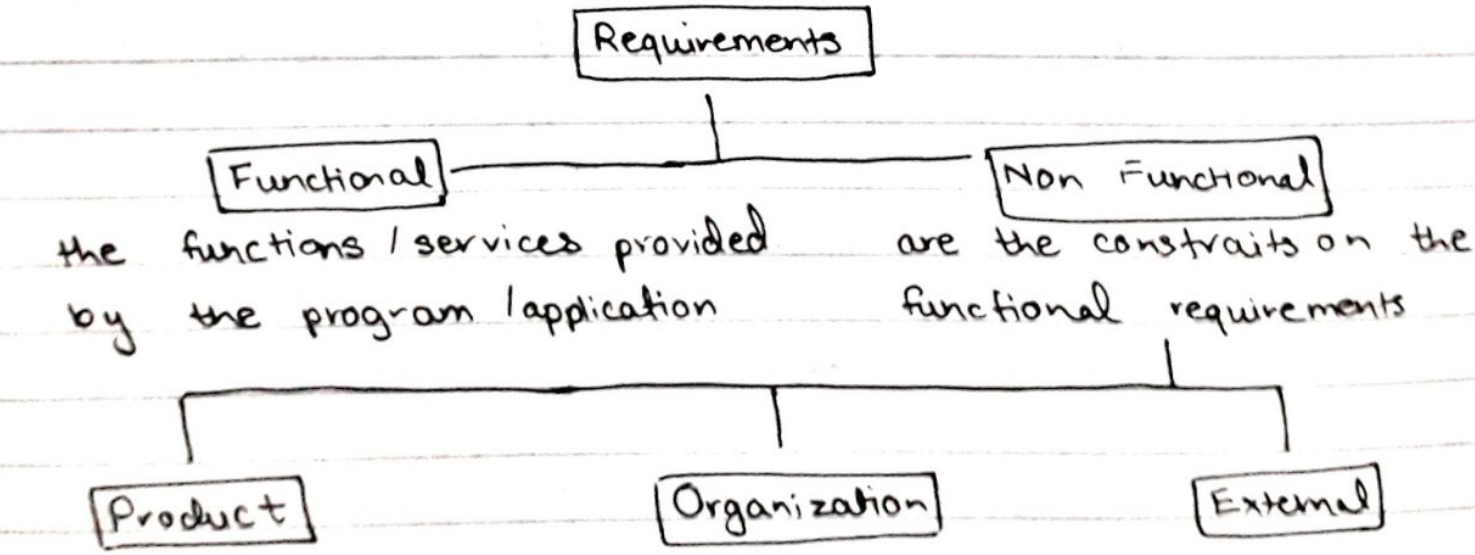
- Introspection: trying to put yourself in the shoes of a customer & trying to identify requirements. (generally not good technique) (except if domain expert)

- **Group Based:** Brain storming sessions, informal meeting take notes, (generally used to determine high level goals)
  Disadvantages :- (shy people, domination by a few people)

- **Interview:** 1 - 1 interaction. Interviewer → R.E,
  Interviewee → end user

	Usage
- Structured: all questions predefined	Convortional
- Unstructured: just go with the flow	Agile Processes
- semi structured: combination of both	Agile Processes

- **Questionaires:** used for surveys.
  - open ended questions
  - close ended questions



3 Elaboration: the analyses part of modeling is performed here. create interaction models

Static Models
- Class Diagram

Dynamic Models
- state Diagram
- Sequence Diagram
- Activity Diagram
- Flow Models

4 Negotiation: different stake holders will have different interests. Hence conflicts occur which R.E must resolve.

- Methods:
  - Stakeholders compromise
  - let a higher authority decide (in case of dead lock)

5 Specification:

[redacted]: We use natural & semiformal languages to make SRS.

- Non functional } natural (Shall Statement Format)
- High level functional } semi formal
- detailed functional } semi formal

6 Validation:

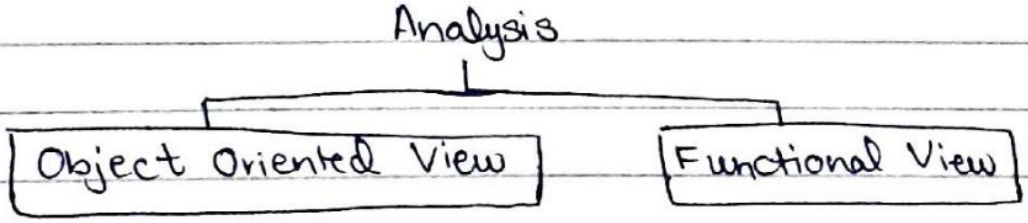
- a requirement can be (incorrect)
- a requirement can be (extra)
- (missing)
- (ambigous)

Note: be very careful with words like every, all, each etc.

Requirements should be: complete
unambigous
readable
correct
tested by Flisch-Kincaid Model

7 Management: manage the requirements and changes in requirements throughout the creation of SRS document.

↓
umbrella activity



Analysis Models:

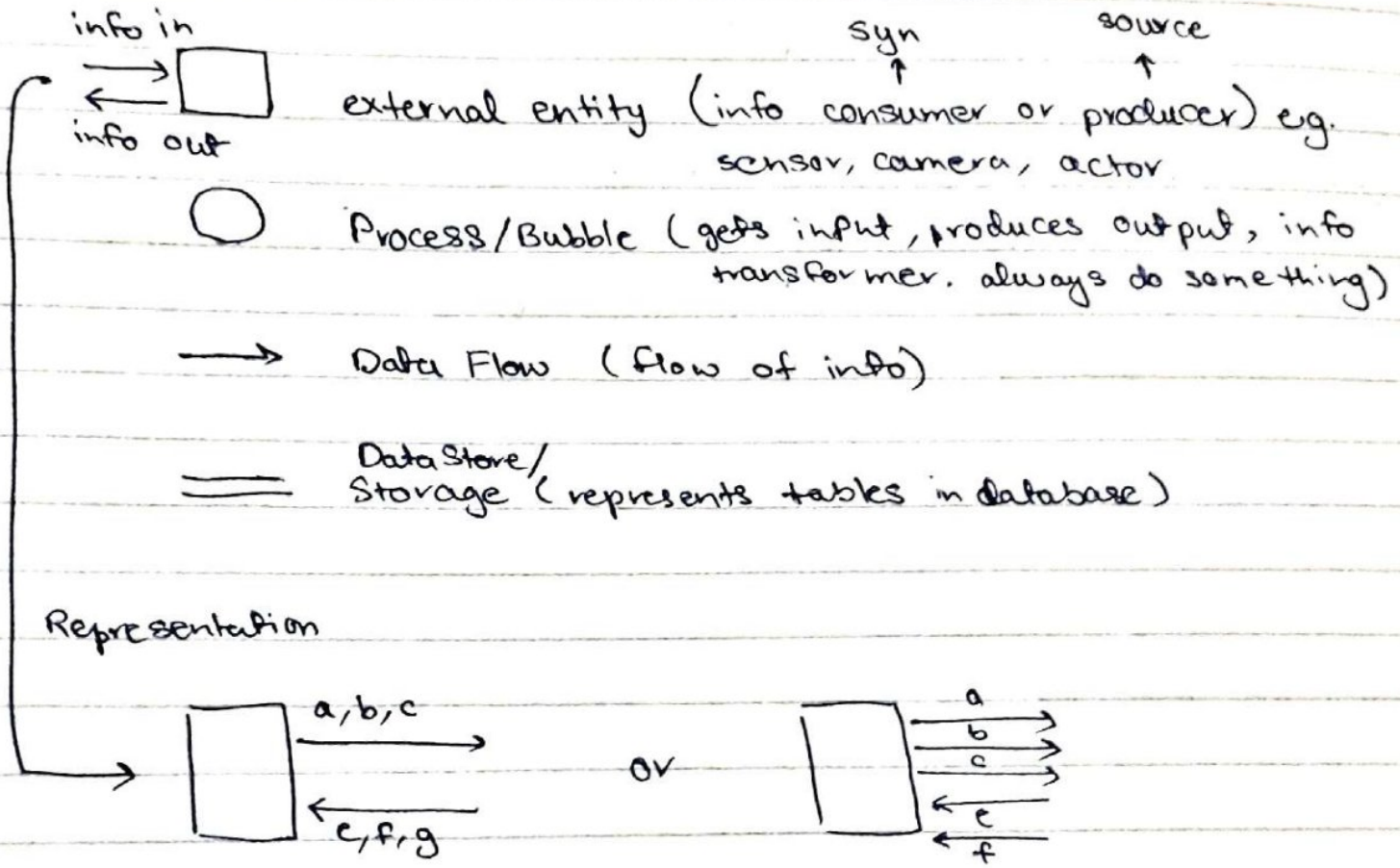
- Use Case Diggram:
  template Based textual use cases

- Interaction Models
  Activity, Swim lane, State Diagram

- Class Based Models
  Class Diogram

Not Part of UML — belongs to structured paradigm not object oriented.

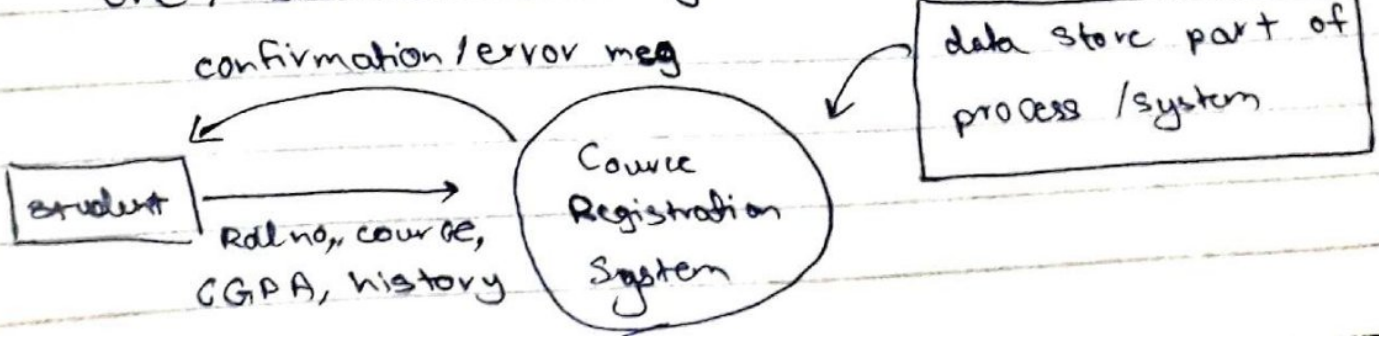
- Data Flow Diagram (DFD)
  how data flows through the system



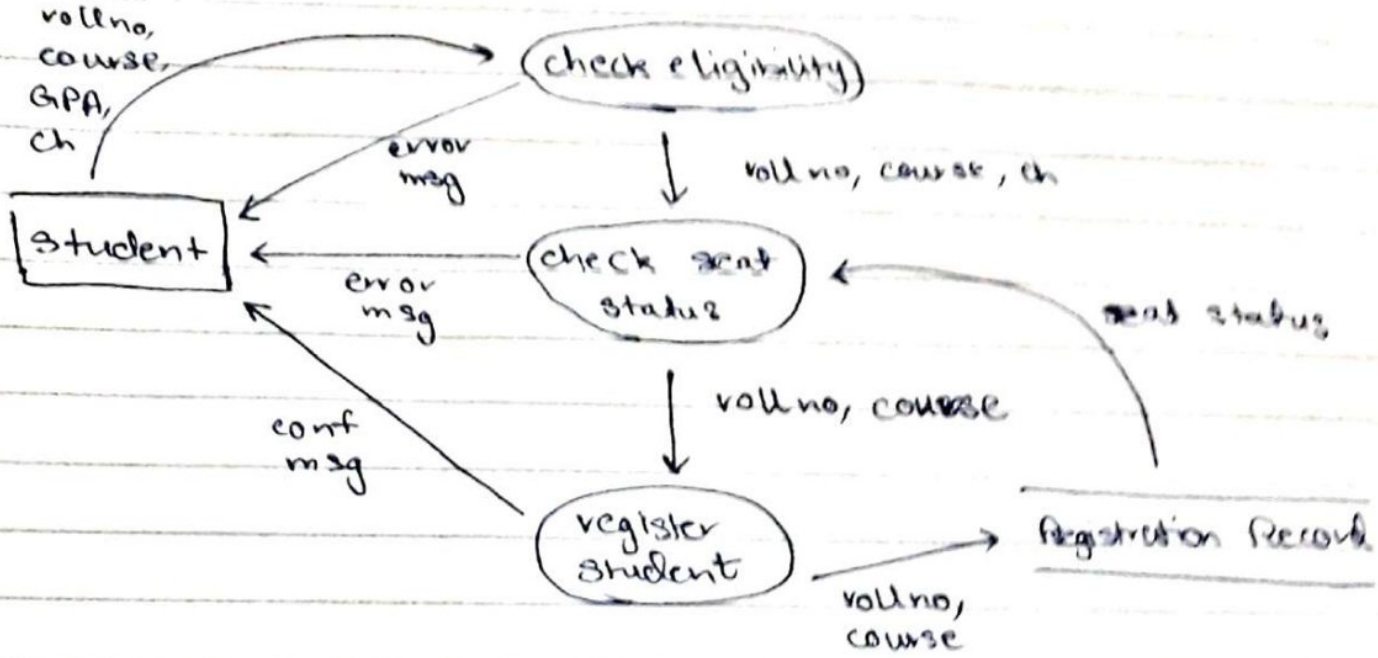
★ Note: no dangling processes & external entities meaning no info going in or out or both.

DFD level - 0

- just one bubble
- bubble contains a noun as a process (only in level 0)
- one / more external entity

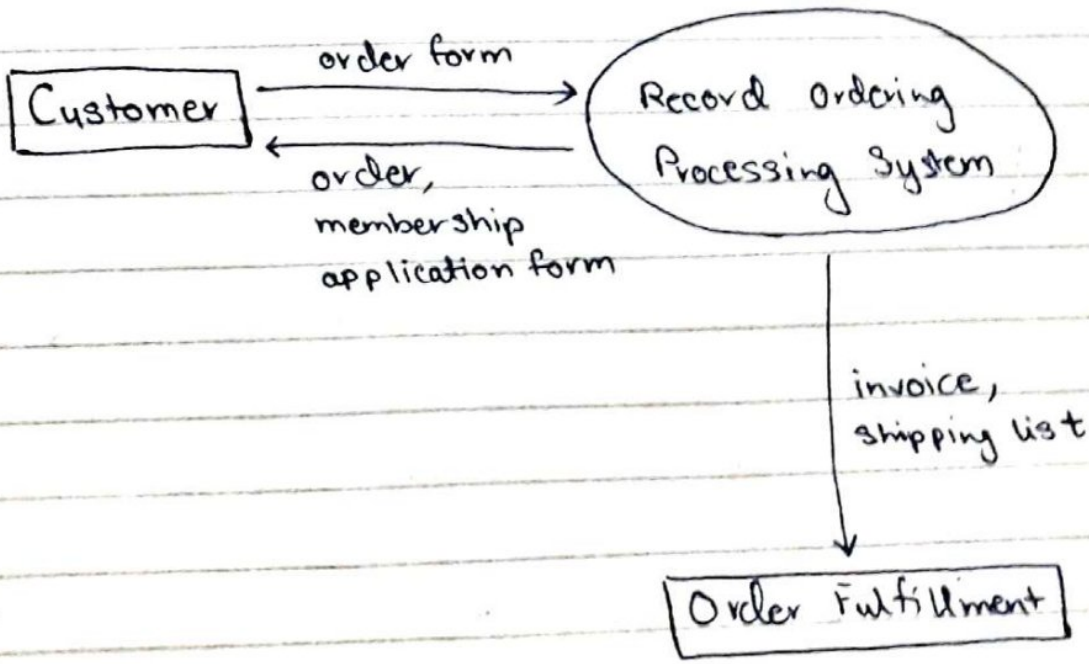


DFD level-1

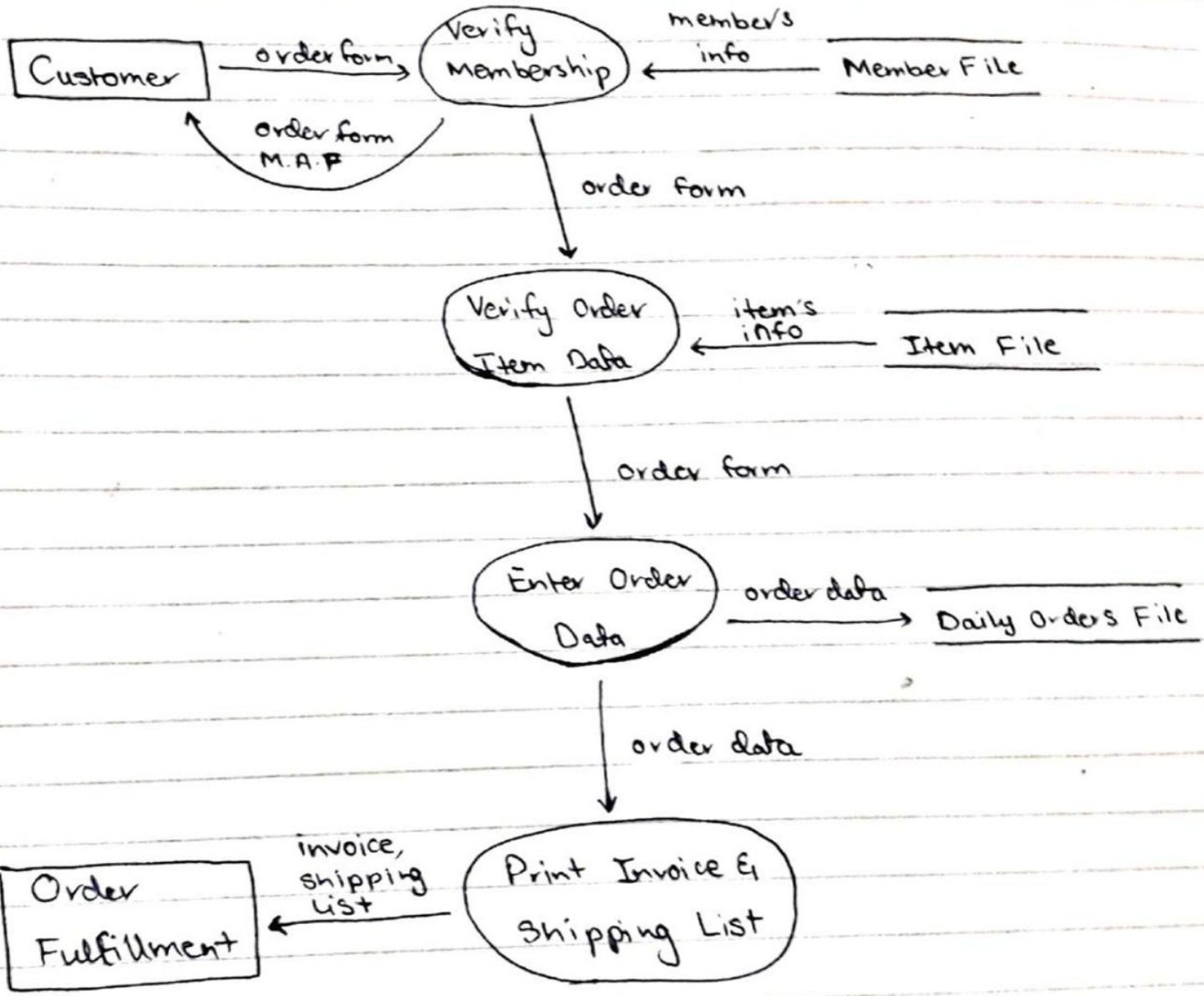


Info flow must be balanced; entities in level-0 should only be present in levels above it.

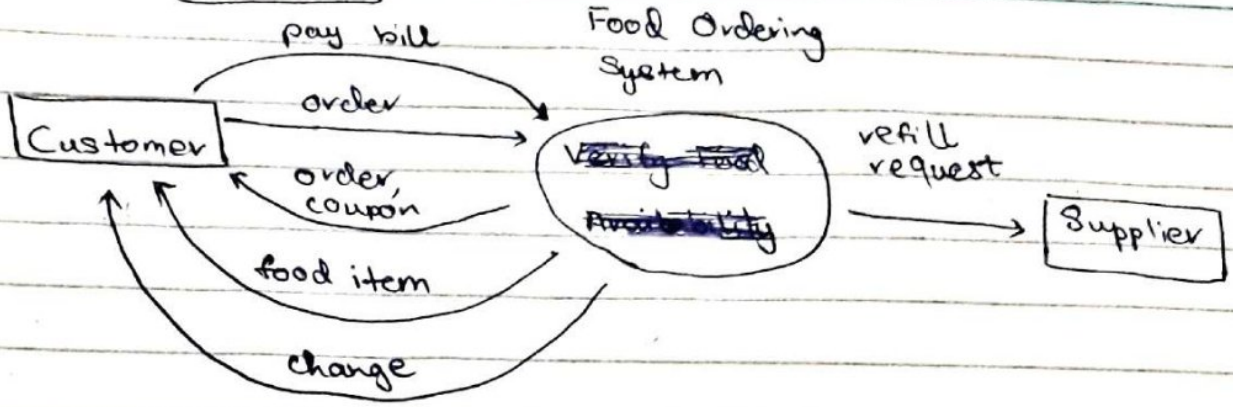
Example: Level zero



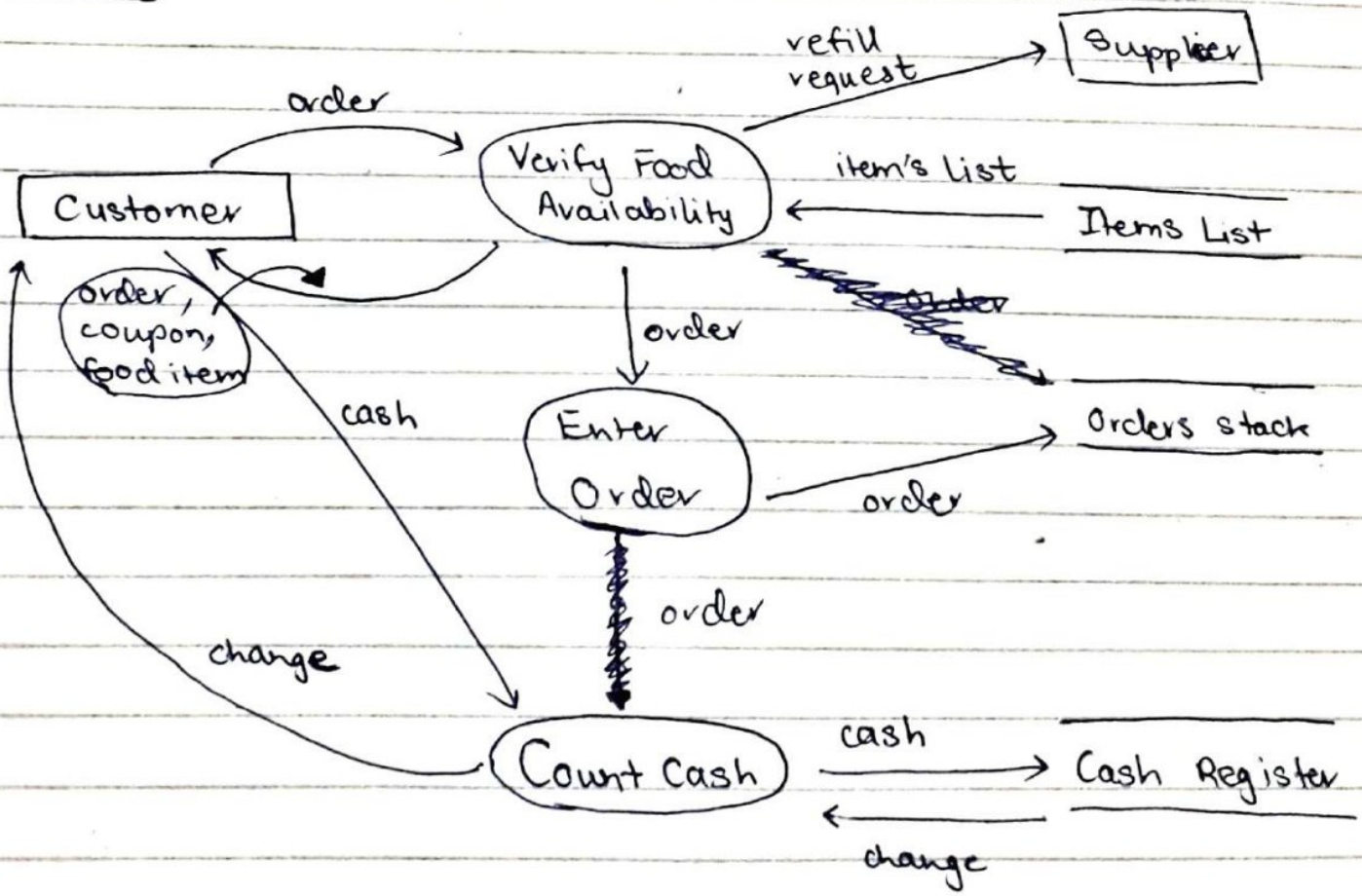
Level 1

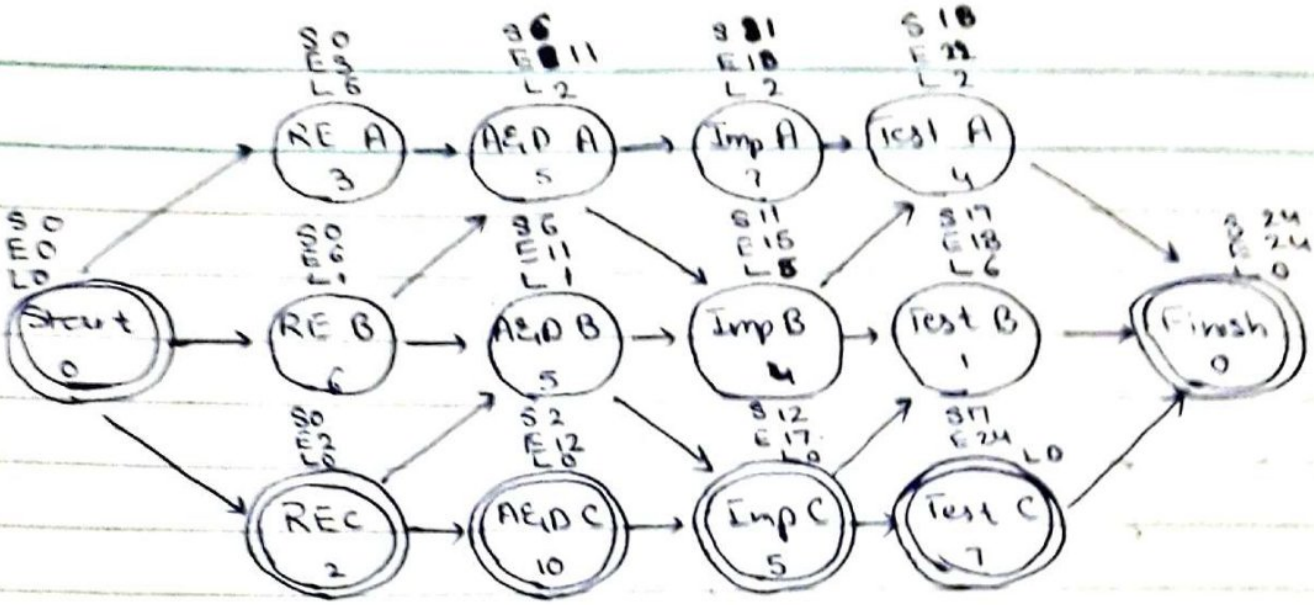


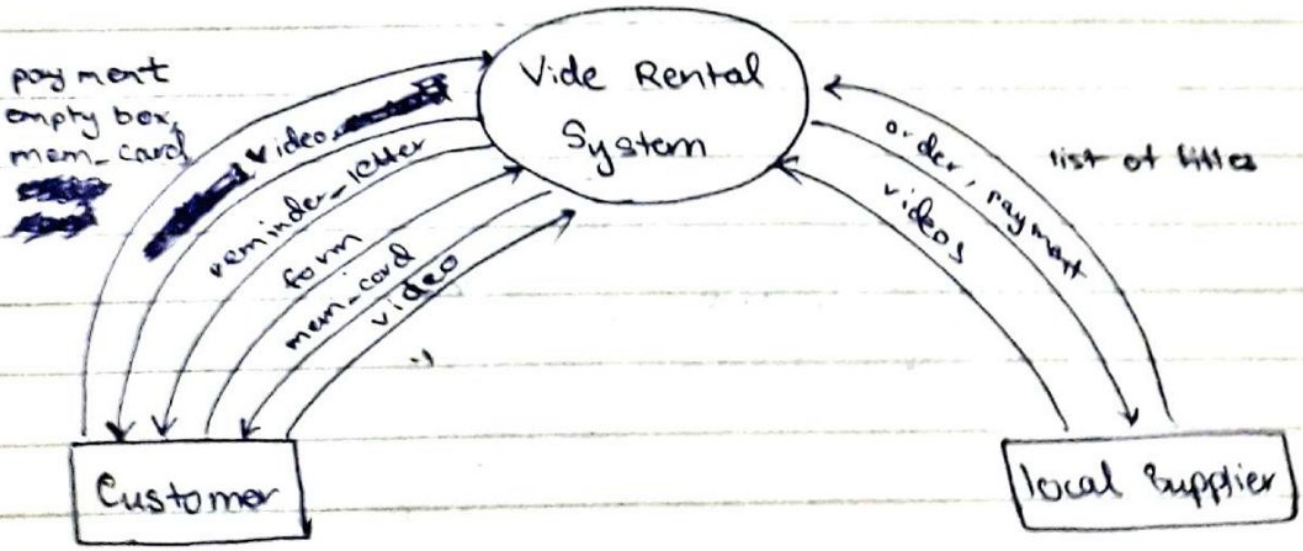
Example 2: Level 0

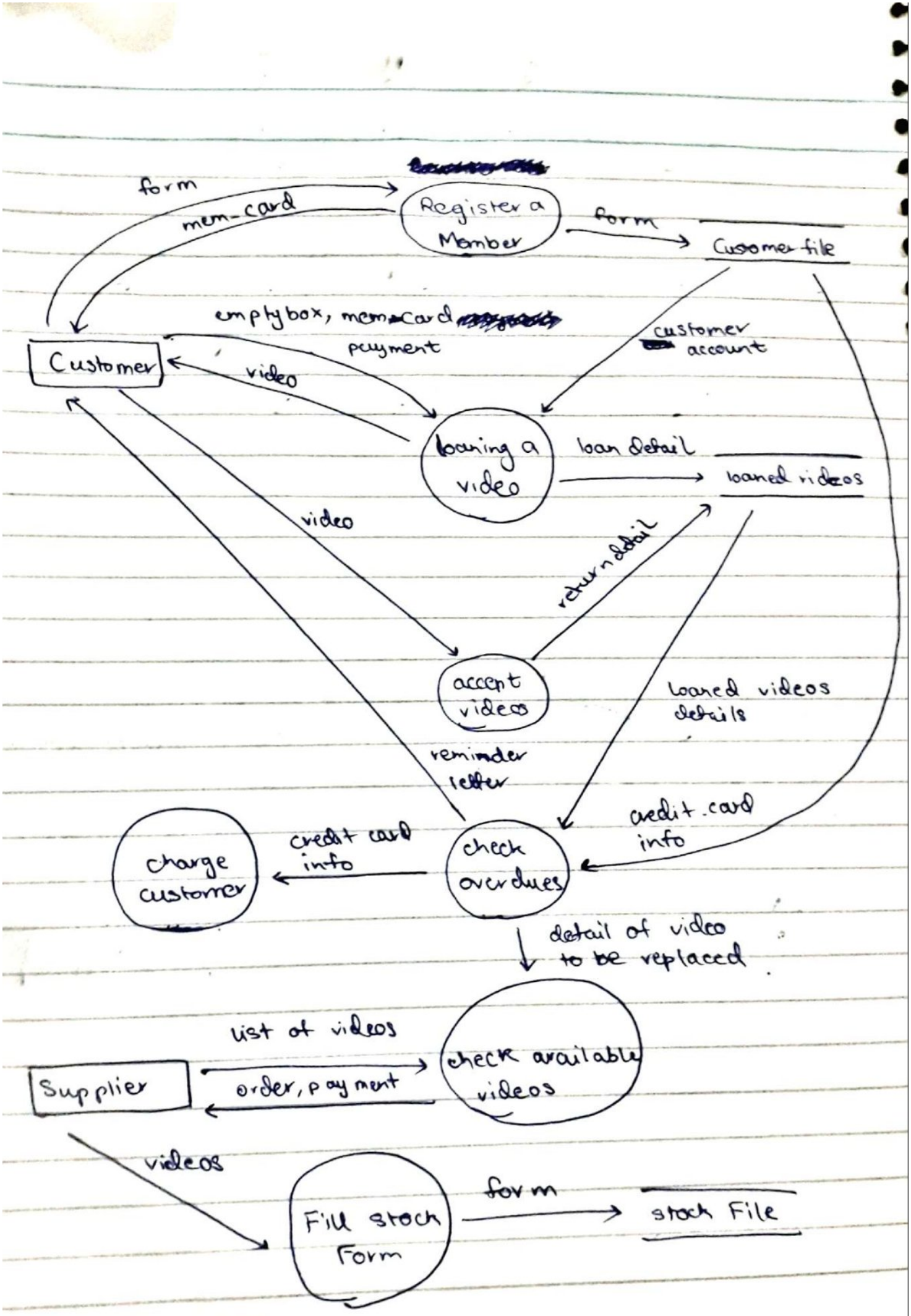


Level 1









Design

Communication, Planning and Analysis part of Modeling has been covered now we'll start with Design part of Modeling.

- Design is the first step in solution space
- It is concerned with how instead of what
- Simplified model of the code.

Data ERD is pure data design as compared to OOA which consists of attributes & methods.

Architecture: deals with the overall structure. What are components and how they are controlled and connected.

Interface:

Internal: how components communicate to each other.

External: how to communicate with human & external non human entities.

Component level: deals with the design of each component or module in the system.

Deployment level:

Design Pricipte:

→ Abstraction: in structured paradigm → procedural abs
is object oriented paradigm → procedural & data abs
ignoring details

→ Refinement: add details step by step
adding details

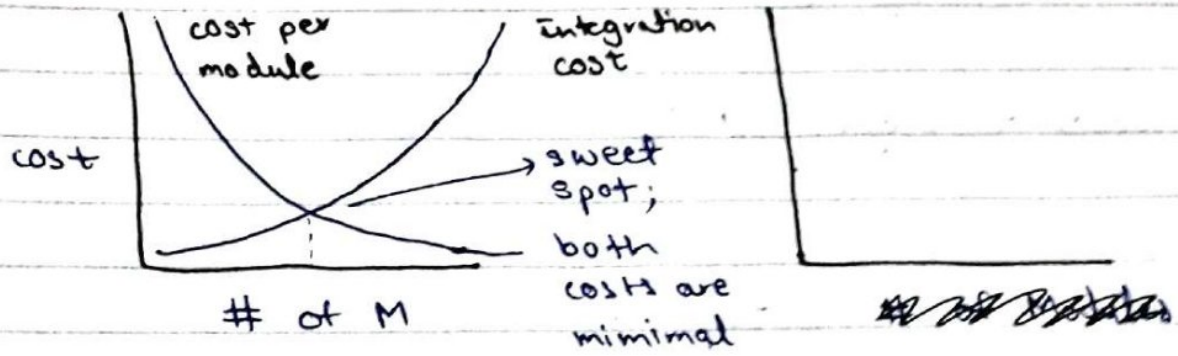
* Hence Abs & Ref are opposite to each other. But not mutually exclusive.

Information Hiding (Encapsulation):

- Encapsulation is combining methods & attributes
- Info hiding is making methods & attributes privete

Modularity: divid & conquer / compartmentalize.

↳ cost per module
↳ integration cost



Functional Independence:

↳ Cohesion: single mindedness
↳ Coupling: connectedness

★ coupling should be minimum & cohesion should be higher; coupling <<< cohesion

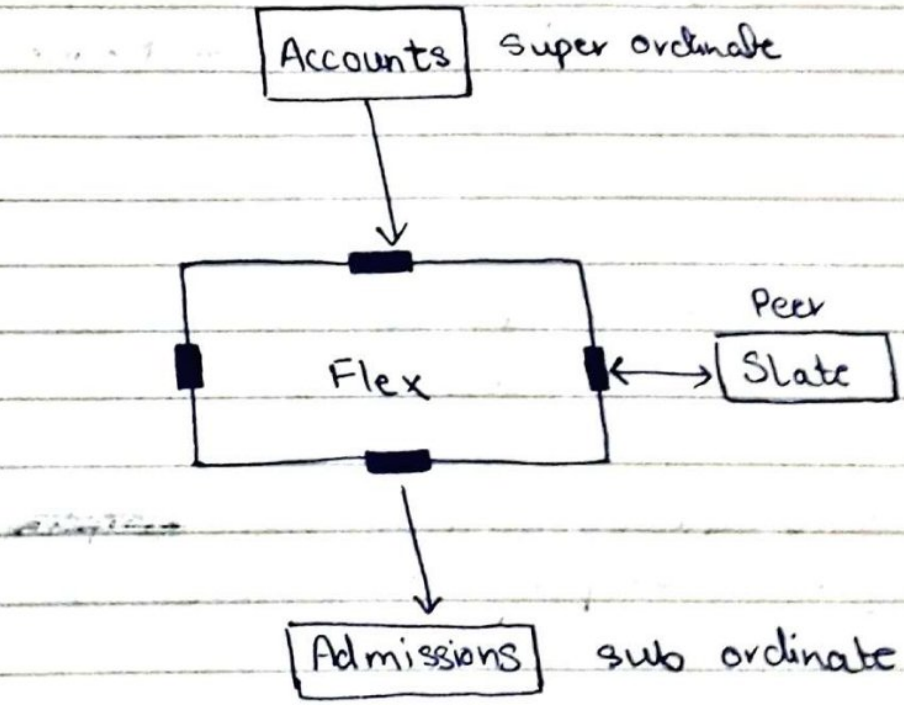
$$\text{cohesion} \propto \frac{1}{\text{coupling}}$$

Refactoring: change internal structure without changing external behaviour.

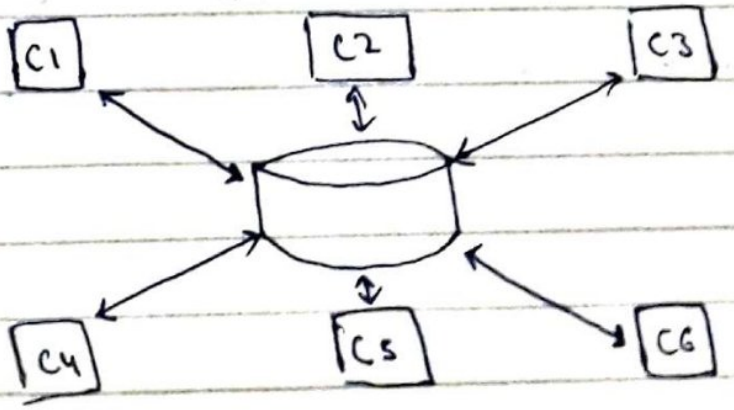
Architectural Design:

- ↳ high level design
- ↳ big picture
- ↳ no details
- ↳ intellectually grospable

1) Architectural Context Diagram: (ACD)



2) Data Centered Architectural Design:

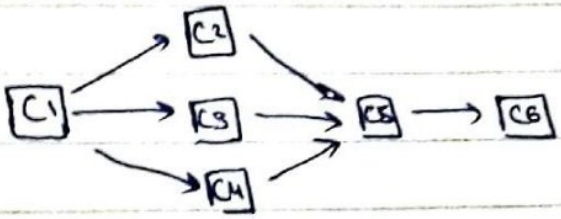


Variations:

Pasive: when a component changes data, other components not notified.

Active: when a component changes data, other components are notified.

3) Data Flow Arch Design:



 Variation: Batch Sequential

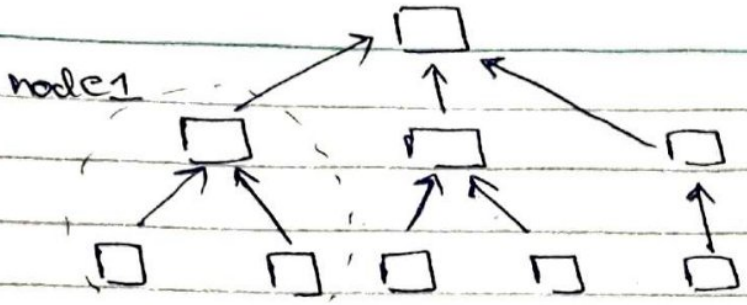
Every component is information processor meaning it recieves info in a certain format & produc output in a certain format.

4) Call & Return Arch Design:

↳ it is a tree

↳ only used in structured paradigme

↳ nested heirarchy of function calls.

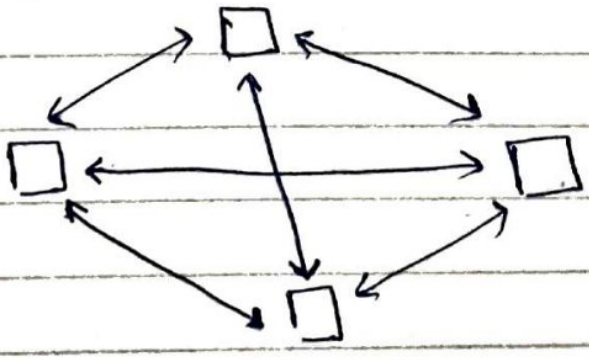


* If a group of componenhts is inplemented in a node then you need **Remote Procedure Call**

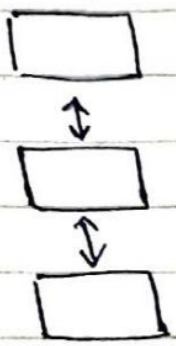
5) Client Server Arch Design

replace data with component in Data Centered Arch Design.

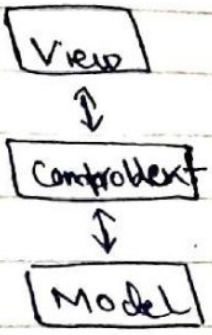
6) Object oriented Arch Design



7) layered / Tiered Arch Design



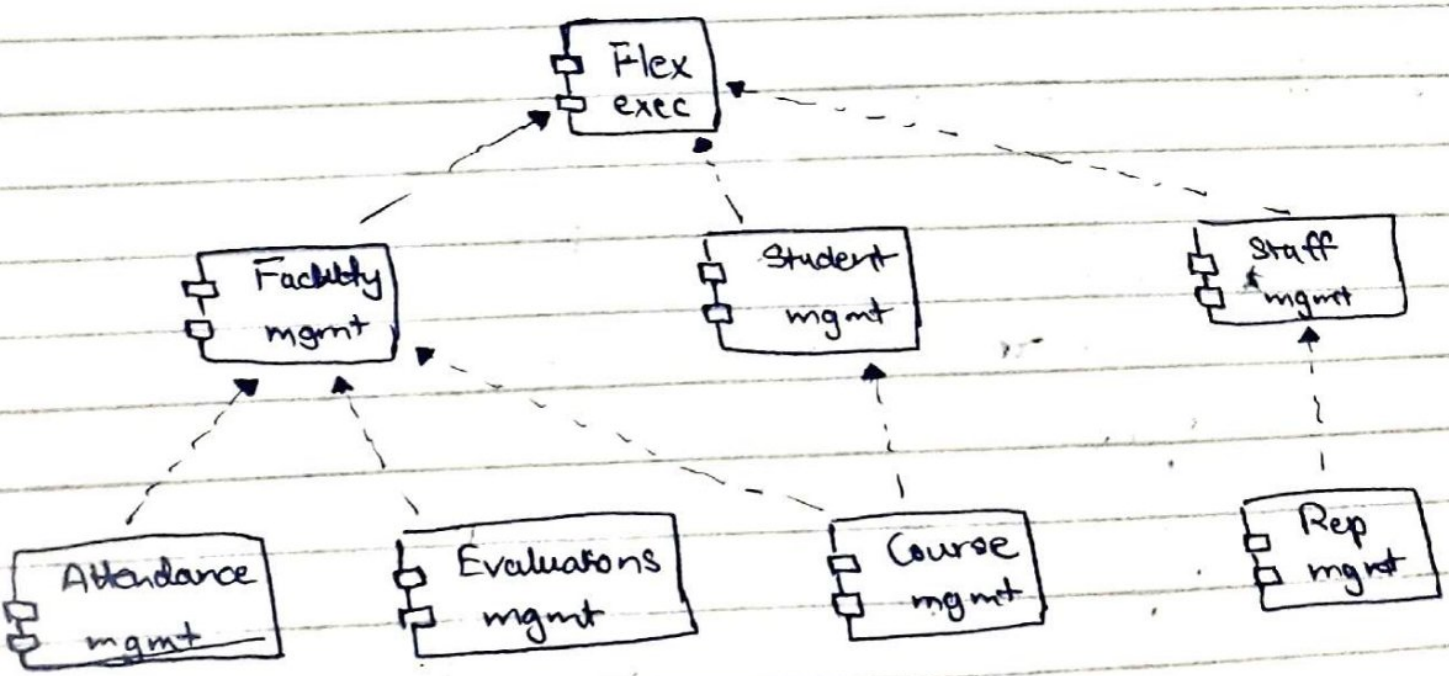
Example : MVC



↳ layers can not by pass
↳ n tier architecture — in this case n = 3

8) UML Component Diagram:

↳ follows Call & return Arch Design



Structured Design: derive call & return arch design from lowest level DFD.

Flow types:

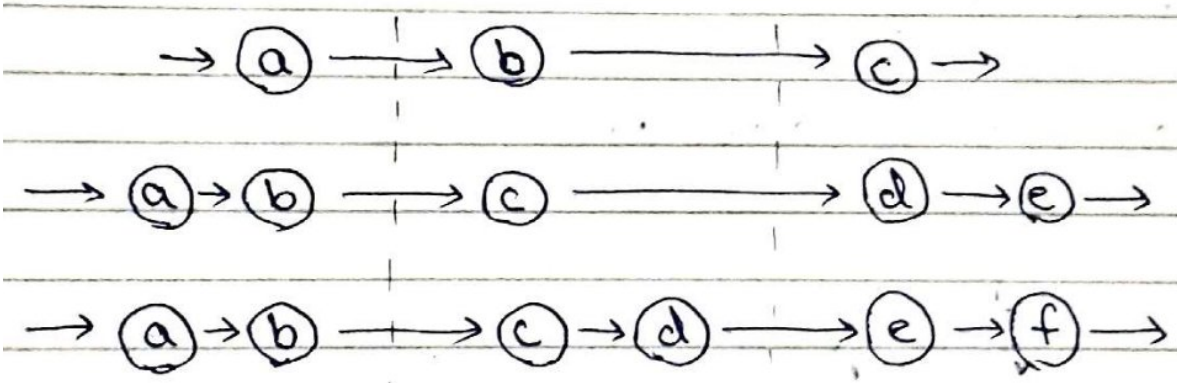
- Transform flow
- Transaction flow

Transform flow: has 3 parts:

↳ input
↳ process
↳ output

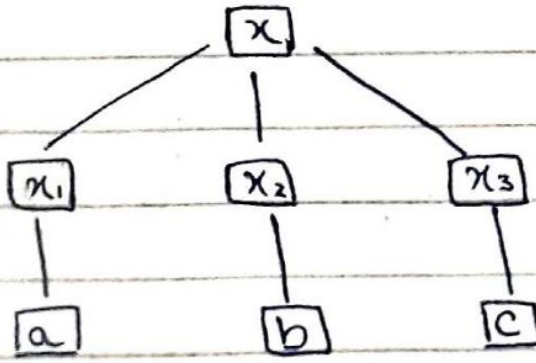
} There may be multiple processes in each part

Examples:

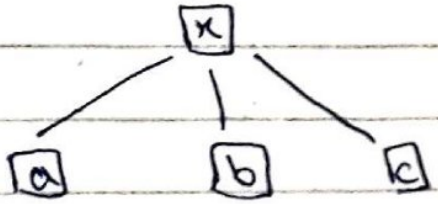


input process output

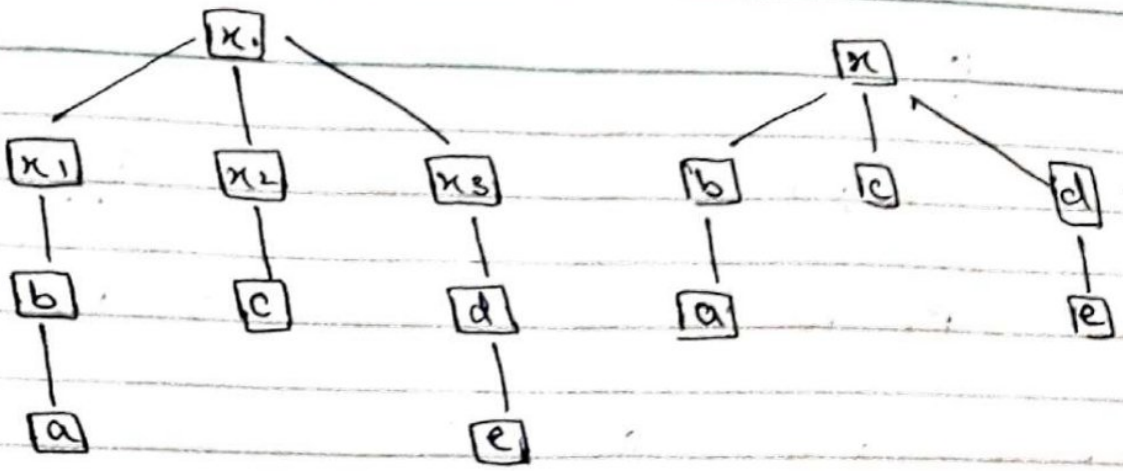
★ (known as transformation centers)



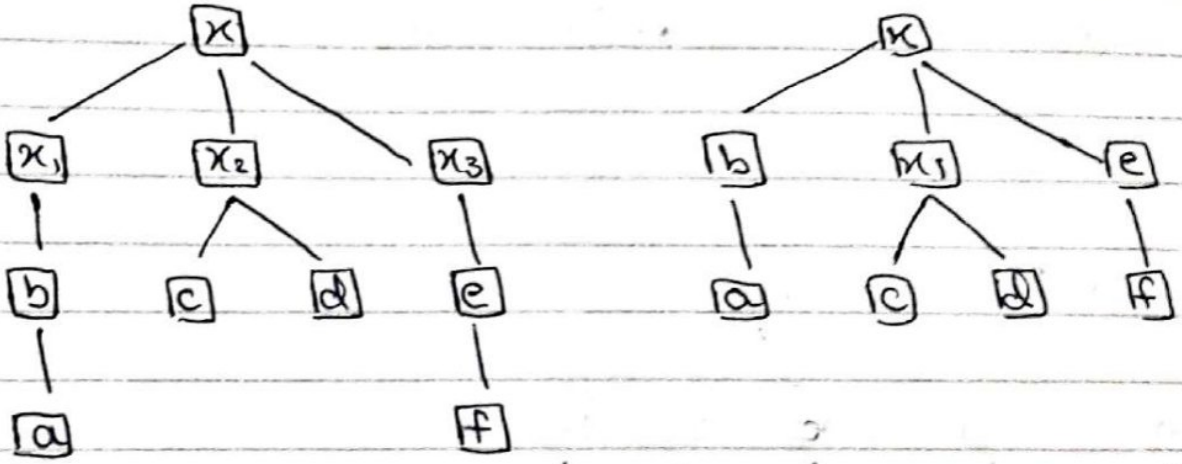
Optimised



②



③

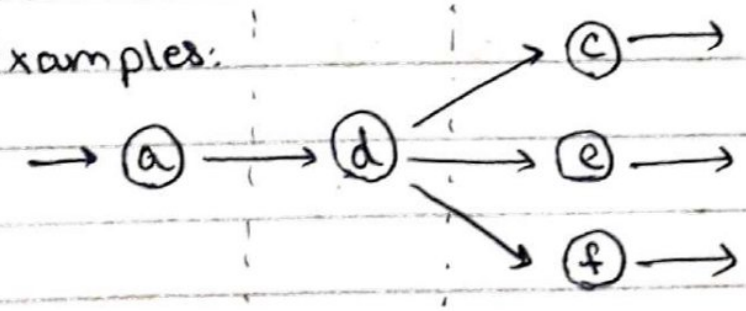


Transaction Flow: has 2 parts:

↳

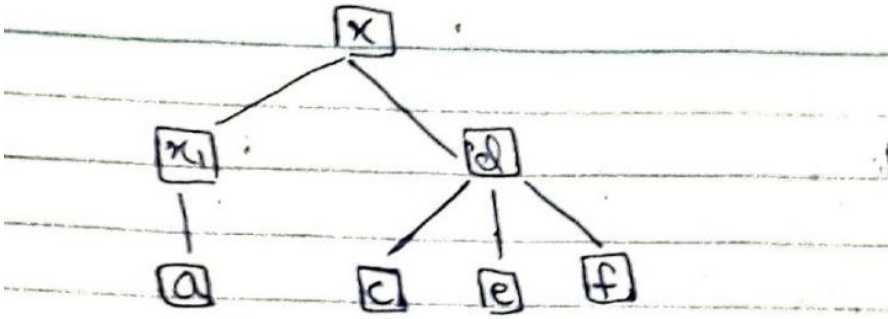
↳

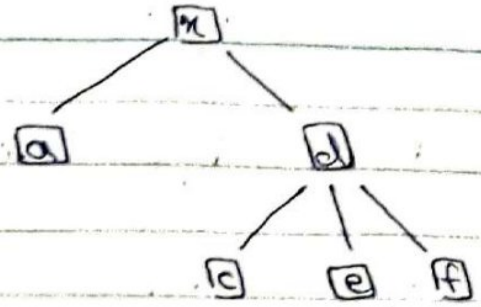
Examples:

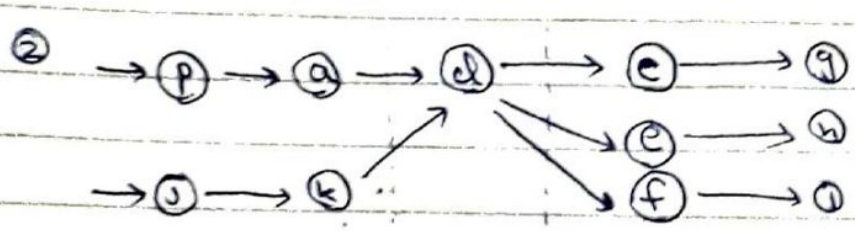
① 

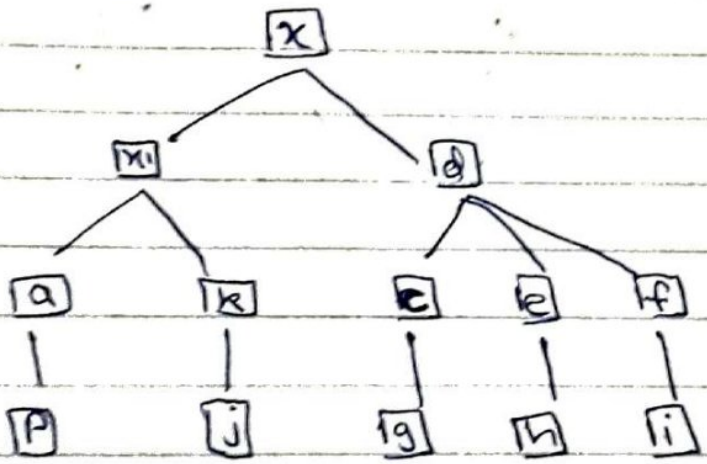
input transaction action paths

(know as transaction center)

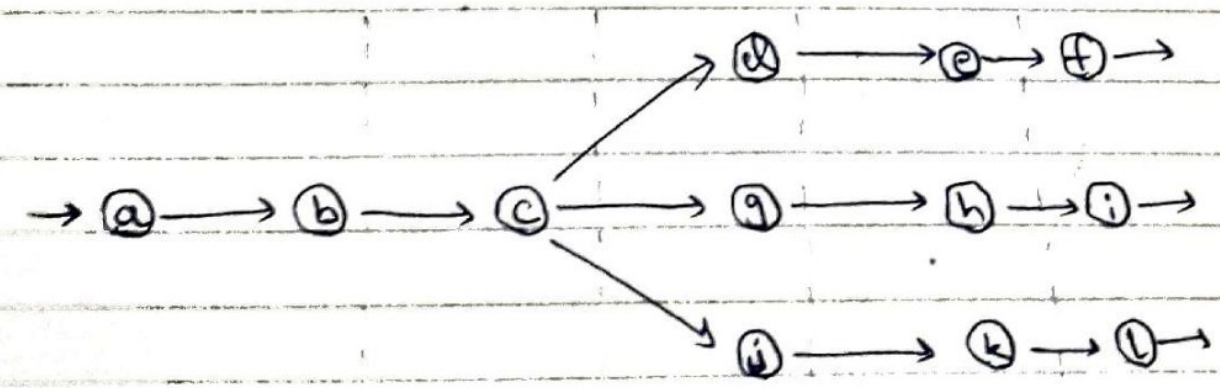






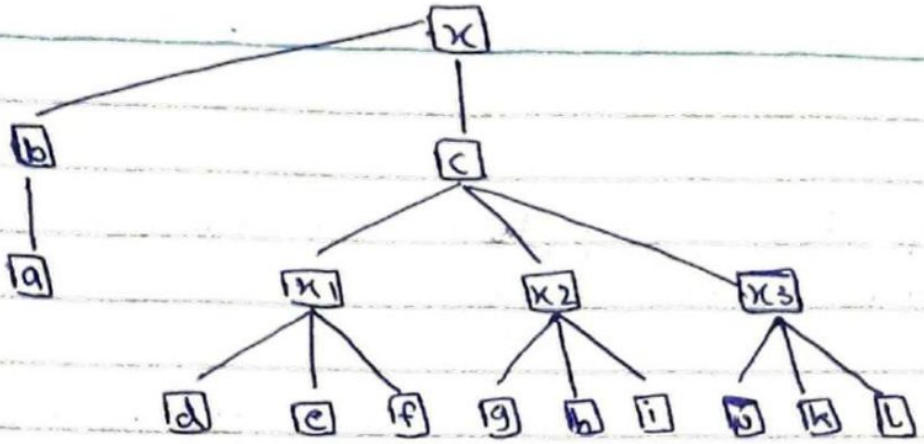


Example:



input dispatcher action paths

transaction center input processor transform center output



Component level Design:
deals with algorithms and data structures and determines which to use.

Objective: make cohesion as high as possible.

Cohesion: → only used for functions

- functional: if a function just performs one computation and then returns, it is highly cohesive. Also known as perfect cohesion.

- Layered: can be used for class level. An example can be class heirarchy like an Abstraction

- Communicational: operations that manipulate the same data are encapsulated in a class that has already encapsulated that data. Also known as Informational.

Note: Functional, layered & Communicational cohesiveness are high level of cohesions.

- Procedural: operates like pipline; one function executed after another and so on.

- Sequential: one the functions pass data to the function after them in a sequence.

- Temporal: operations that are invoked at a certain time. e.g operations called when turning on PC or turning off PC.

Note: Procedural & Temporal are low level of Cohesions. Also Utility.

- Utility: also known as Coincidental Cohesion. functions(helper) that perform tasks related to a certain domain grouped together.

Coupling: keep as low as possible. It is a necessary evil. Find compromised ground with cohesion.

- Content: one class manipulating data of another class. Violation of Information Hiding.

- Common: 2 or more classess accessing same shared data. like global variable.

It is dangerous because if A changes data & B not modified then B will access wrong data.

- Control: Funct 1 provides an argument to Funct 2 which is a flag & based on that flag the sequence in funct 2 is determined.

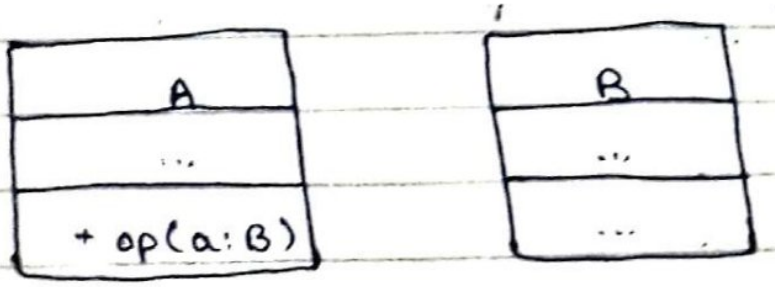
A type of Data Coupling.
Hence one funct controls another function.

- Data: function 1 calls another function and passing data.

It may lead to data hazard if the interpretation of the data by the two functions is different.

- Routine Call: one funct calling another funct. (data not passed)

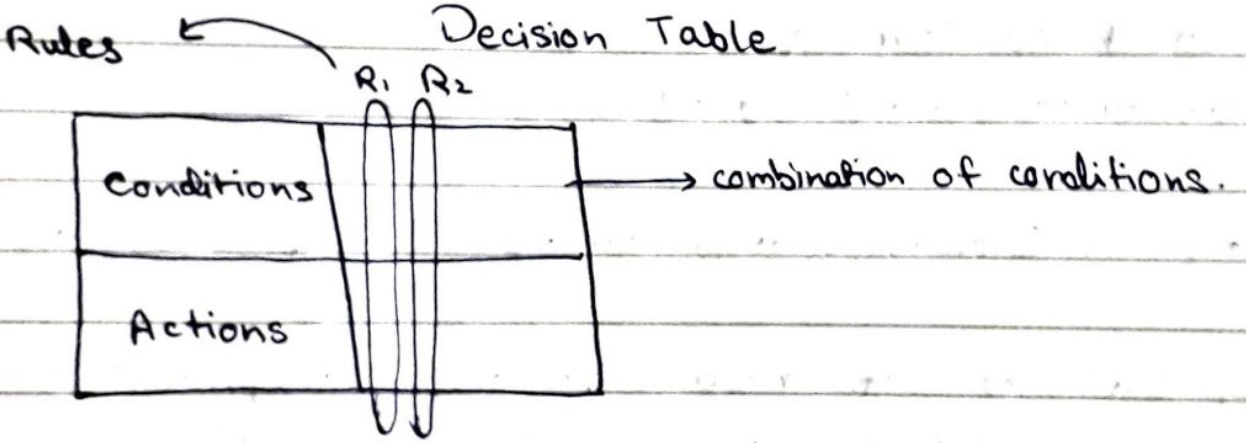
- Stamp: Datatypes of one of the arguments of one of the operations of class A is class B



type use: if the attribute of class A is of type class B.

Import: class A imports another class B

external: class calls some external component. like APIs, functions related to DBMS, OS.



actions should be atomic not compound. Same goes for conditions.

Example:

conditions	R1	R2	R3	R4	R5
Fee defaulter	T				F
On warning		T			F
enough CH			F		T
Prerequisites				F	T
Actions					
Registration					✓
Conf msg					✓
error msg	✓	✓	✓	✓	

This decision table is used to show the algorithm of the function in a consise way. Its an alternate to pseudocode. It is a component-level design of a task.

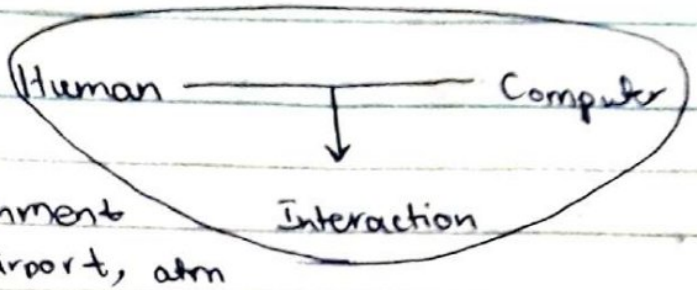
Example:

Conditions	R_1	R_2	R_3	R_4	R_5
student	T		F	F	F
senior		T	F	F	F
week day			T	F	F
saturday			F	T	F
sunday			F	F	T
Actions					
no tee	✓	✓			
25 % discount					✓
50 % discount				✓	
75 % discount			✓		

User Interface Design:

A software product should be:

- useful:
  should do what it is supposed to do
- usable:
  user should be easily able to use the
- used:
  it should end up being used



Environment
e.g airport, atm booth

HCI is a multi disciplary field [redacted] UI designer should know computers & human behavior

★ 3 Golden Rules

1 Place the user in control:

Examples:
- editing while searching
- undo button / command
- provide facility to user to manipulate data which resembles real world analogies.
- short-cuts
- configurable interface

2 Reduce the user's memory load:

go for recognition instead of recall

Examples:
- menus
- using first letter of the task in shortcuts, e.g Ctrl+S, Ctrl+B
- provide trail to the user (bread crumbs)
- shopping cart

3 Make the interface consistant:

- keeping interface consistant helps in better learning.
- use expectations like ctrl+Z in your software otherwise users get frustrated.

Examples:
- Microsoft Office

1 Interface Analysis

- user
  - age small children should be provided with a interface with more graphics & large buttons
  - literacy use graphics for illeterate people
  - gender use different color themes based on gender
  - culture take into account different languages
  - religion
  - disabilities interface for visually impared, autism, paralysed individuals
- tasks

  tasks & flow of work identified using use cases
  - automated doors help people carying goods, physically disables, children
- content

  consists of text, videos etc
  - do not use all caps, readability suffers
  - whitespace helps in readability and highlighting
  - colors, formatting, alignment of numbers ie right alignment of itegers, alignment on decimal in floats.
- environment
  - posture of users
  - noise, lighting

2 Interface Design

3 Interface Implementation

4 Interface Evaluation

*) Interface analysis

*) Interface design
- layout ([illegible] table, button, etc [illegible])
- menu — more frequently used at top ([illegible] less = satisfied)
  — natural workflow (new, open, delete in MS word)
- navigation

In more frequence approach, when you place consequently opposite menu (save and delete) consequtively then error can occur e.g. hibernate and shut down

Error Messages

User has to keep each pop up messages in mind.

Error messages need to be user friendly (not programmer friendly)

Error messages should be non-judgemental

Response time

Time b/w user action and system response

Mobile / Video games ([illegible] affect it)

UX part is affected by it

Scanned by TapScanner

Variability

If response time is variable then user will frustration

It lead to break user redhemn
(ردھم)

Use progress bar to reduce frustation by it because, yes functionalities have different response time.

Disable click button when clicked once
or
Take to another screen

In money involved or Safety sensitive (X ray dose) applications

Universal/Inclusive Design

You do not want to exclude user

e.g., Flags on top of website (Different languages)

Vitual weak people → Variable fonts

Red vs Green Color blindness
↳ background and foreground

cyclemon.com toyota.com/prius 2019
↳ 360 view

Cultural interpretation of color

In West, danger in representation in red

Solution

Multiple communication ways

e.g.,

Beep also with color

Check vs Cross on Signals

In Eastern culture, Red means bravery

universal/ inclusive

accessibility internationalization

Interface implemantation

Translating application in Html, CSS

Interface evaluation

Quatitatum Company

↳ Usability testing

Human test them → Heuristic testing

Tools test them

Use Human pysiology → Monitor through pulse, sweat

Designing interface is Science as well as Arts

↓

Students of NCA

Implementation

Construction includes implementation & testing.

Implementation: is translation of component level design into code

Issues:

- which programming language to use

Programming language Generations

- 1st zeros & ones
- 2nd assembly language : 1 to 1 mapping
- 3rd C++, Java : 1 to (5-10) mapping } mostly deal with these
- 4th SQL : 1 to (30 - 50) mapping

Compiler vs Interpreter.

How to choose language:

- Platform: desktop / android / ios
- Type / Nature : web / communication with OS / more potability
- (Tool) support : forums / stack overflow
- Libraries: reuse / sometimes need to pay
- Past Experience: which language do programmers know

Note: Sometimes the client may specify the languages to be used.

Good Programming Practices:

- Variable names should be meaningful and consistant
  - ↳ use terms of the problem space
  - ↳ min CGPA, max CGPA, avg CGPA ⟶ consistancy
- function and class names should be meaningful to make the code self explainitory.
- Comments
  - ↳ prologue comments: written above a function / class etc
    - brief description
    - programmer's name
    - any known problems

Note: some managers ask programmers to write comments first and then code

- Don't Hard Code values
  - ↳ takes time to update value everywhere it is written
  - ↳ if one not updated then error occurs
  - ↳ read value from database or config file

- Code Beautifiers:
  - to indent the code
  - different color for types (reserved words / functions ...)
  - spacing

Testing

Software Quality Assurance

Quality: conformance to the requirements

Software Quality: conformance to all the requirements mentioned in SRS.

- Implicit / Explicit
- Functional / Non Functional
- Technical / Non Technical

- Technical: Implicit / Explicit & Functional / Non Functional
- Non Technical / Managerial: complete & delivered in time

Quality Assurance: is a bigger umbrella. Testing is one of the Quality assurance activity. Testing is most important QA activity & takes most resources

Software Quality Assurance: includes all activities that make the software reliable.

Activities:

- read / review RE, A, D, I (Non testing activity)
- test code (execute) (Testing is only concerned with coding)

Note: all activities before testing are psychologically constructive in nature but testing is destructive in nature.
That is why it is difficult to find errors in your own work therefore testing is done by other people.

Goals / Objectives of Testing:

- uncover max no of problems (effectiveness) using least no of resources (efficiency).

Note: Before deployment software is with Software Supplier (SS) but afterwards the software is with Software Consumer (SC).

Reasons of failure:

1)

error (mistake made by a human)

↓ may

fault (fault is the manifestation of the error in software state, so fault deals with software state)

↓ may

failure (maps to an event. the presence of a fault may lead to an error)

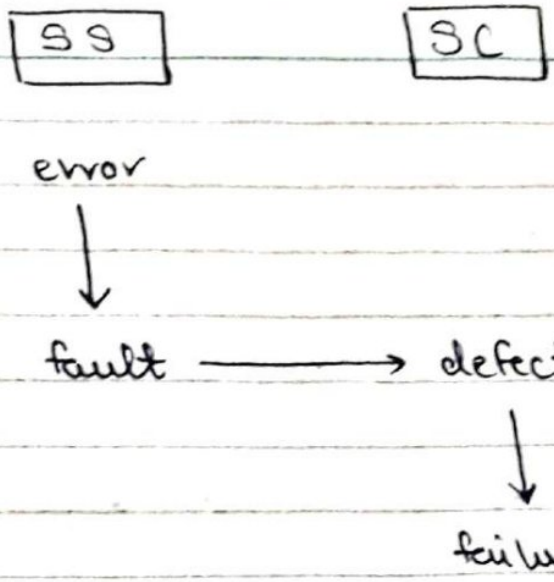
If error is resolved by some other person, fault does not occur

when a fault is activated only then a failure occurs

Note: error & fault occurs on SS side and failure on SC side

2) Another reason of failure may be mis-communicati between supplier & consumer.

- Defect: If there is is a fault in the software and was not fixed before deployment then fault becomes failure.
  It occurs on SE side.



Verification: check whatever has been done is correct or not

Validation: check whatever was asked by the customer has been implemented or not.

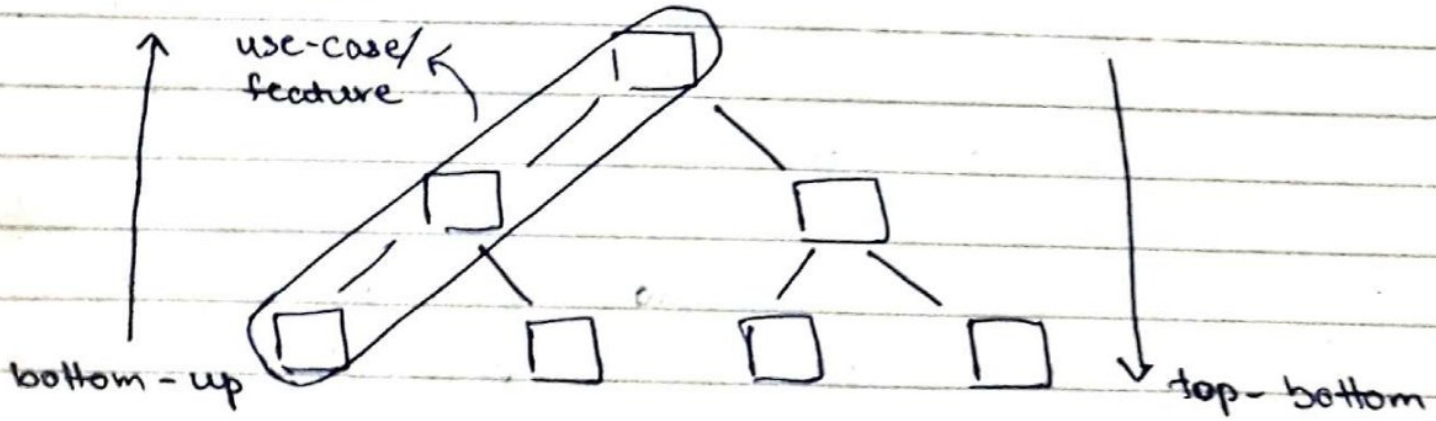
Testing Strategy: concerned with what, who and when. It deals with the overall view, plan of action.

Testing Tactic: concerned with how. It deals with how the testing is to be performed.

=> Testing is done in levels:

1) • Unit Testing : focuses on individual component. focus on sequential statements, if-else conditions, error handling, loop conditions etc
   • generally, this testing is done by programmer himself.
   • In XP it is done during the programming of the code.
   • Data structures are also looked at in this level.
   • We have apps to test and techniques like Test First.

2) Integration Testing:
   • focus on multiple units at a time. more focused on interfaces



• In bottom up approach all the unit testing is perform first and then integrated modules. vice versa in top-bottom.

- Bottom-up:
  - need drivers to call units. These drivers are dummy models which may pass dummy data.
  - drivers are an overhead, as they need to be make for testing but not delivered.

- Top-bottom:
  - need stubs to replicate lower level modules.

- Generally drivers are less in number but stubs are more in number. However, drivers are decision making modules so they should be tested first.

- Testing can be done depth first or breadth first.

- In depth first, top-bottom approach you can develop a prototype.

- Big Bang Testing: integrate all units and then test.

- Incremental Testing: incrementally test units one by one as you integrate them.

- Object Oriented Paradigme:
  Identify classes with lowest coupling and then integrate them and so on.

- Regression Testing:
  When a new module is brought into an integrated module, change occurs in the system so to be sure that no faults have been introduced we perform a subset of tests that we did on the original integrated module.

- Smoke Testing:
  does not focus on nitty gritty, just focuses on the major aspects, without going into details. It can be used in itegration testing to look at major aspects.

3) Validation Testing
   - make sure what you have implemented matches the requirements of the customer.

   - In this level, system as a whole is present.

   - End-user is frequently present.

- User Acceptance Testing:
  If only one user involved then include that customer. (May also work for few customers)

- Alpha Testing:
  Testing done in controlled environment. With a lab, premises.

- Beta Testing:
  software is testing in real world environment.

- The above testings are done because it is hard to imitate user / end user.

- AlphaTesting is expensive and may suffer hawthorne effect, that is if a person is being observed, they may change their behaviour.

- Beta Testing is pretty comanon but it is difficult to observe. feedback may not be good and accurate.

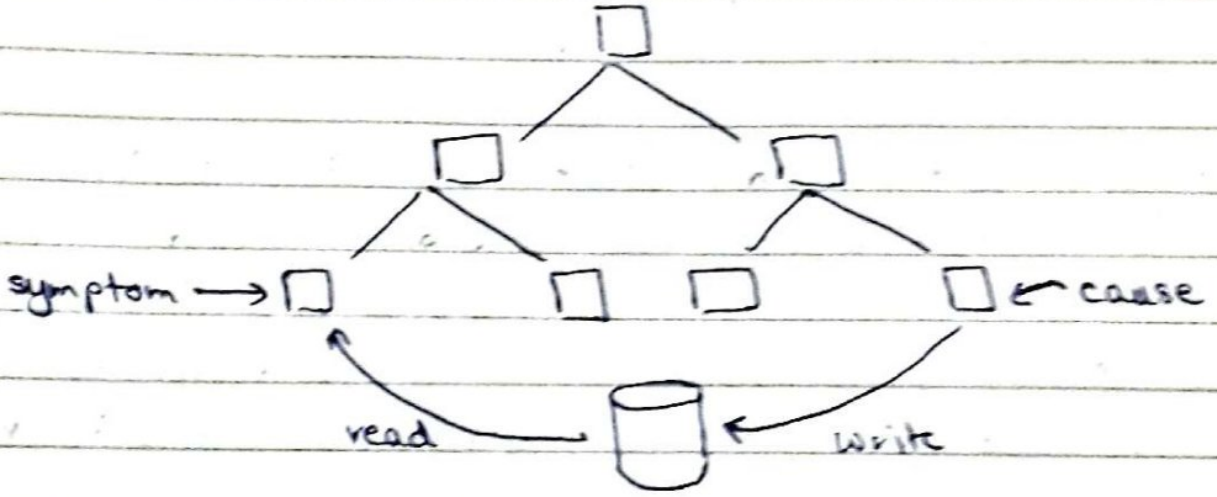
4) System Testing: performance can not be tested untill software deployed. Software & Hardware tested togethe

- Stress Testing:
  overwhelm the system. extreme physical environment.

- Security Testing:
  make the software so hard to penetrate that the cost of penetration is a lot more than the value of info penetrated.

Debugging: a consequence of testing, it is associated with testing.

- It is a diagnostic process. Identify symptoms & causes
- It involves intuition & Luck and may not be very easy.



- one module causes error and another suffers from symptom.
- a fault may be caused by a combination of non faults. for example rounded off number recieved by a module may not produce correct result.
- symptoms may come and go.

Hence debugging is difficult

Debugging is a consequence of testing & regression testing is a consequence of debugging.

Debugging Techniques

1) Cause Elimination: apply break points. Identify possible causes one by one and eliminate them.

2) Brute Force: alot of print statements to identify the source of problem

3) Back Tracking: problem was identified in module A so go back to the module that called it and so on

Good Practices

- take a break because you may be involved so much in debugging that you may not look at bigger picture. Tunnel Vision
- involve others.

Testing Techniques
- White Box
- Black Box

White Box: look at the internal program structure, loops, conditions, statements etc.

Black Box: only look at Interfaces. Data going in and comming out

Grey Box: Combination of Black & White BOX

White Box
- Control Structure
  - Loop
    - Simple
    - Nested
    - Concatenated

Simple Loop Testing: test the loop for 0 to N

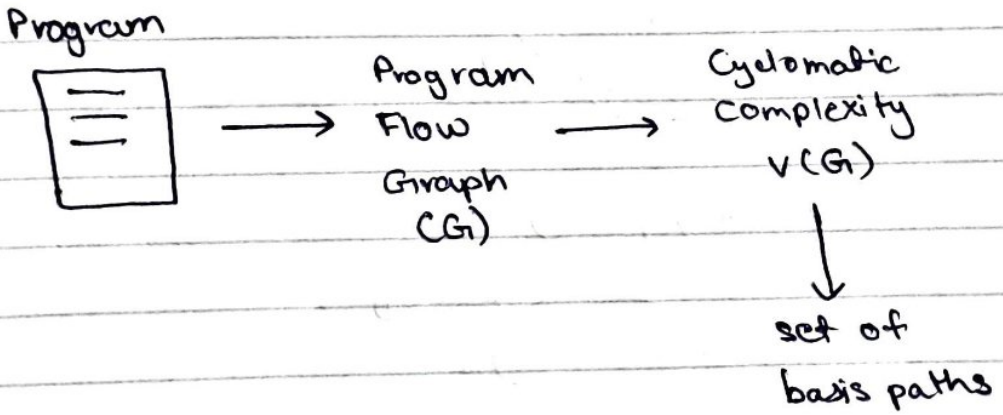
Nested Loop Testing: test inner loop then 2nd inner loop and so on

Concatenated
- Simple → independant loops
- Dependant → dependant loops

Simple Concated Loop Testing : same as simple Loop testing

Depedent Concatenated Loop Testing : same as nested loop testing.

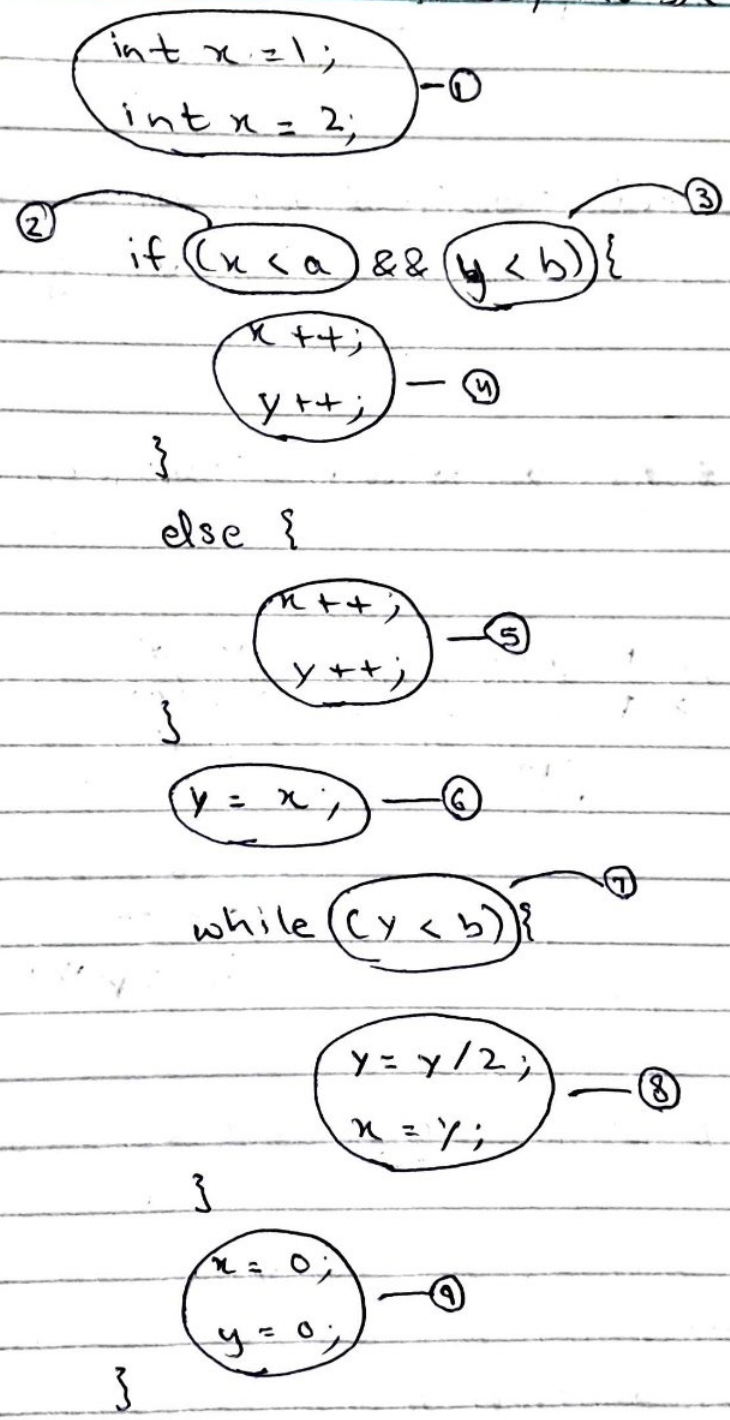
Basis Path Testing : a White Box technique



Line Coverage / Statement Coverage : every line is executed atleast once.

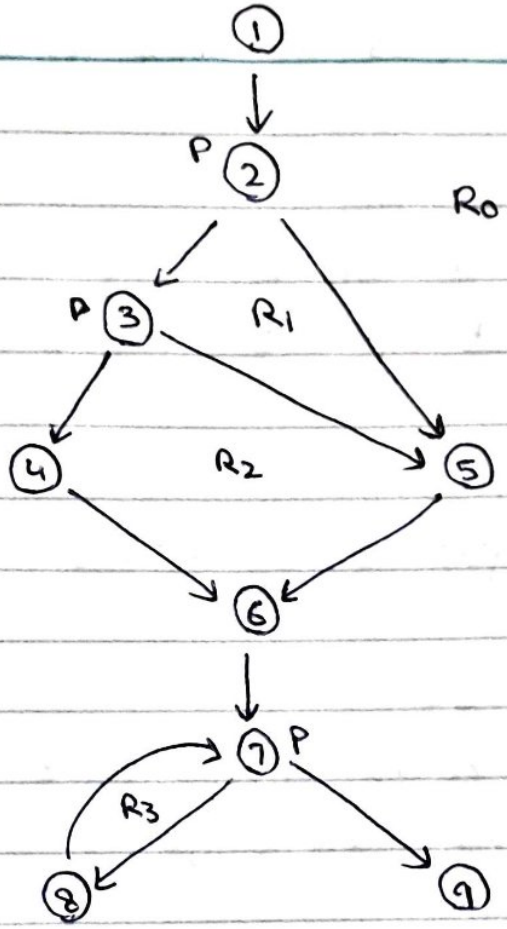
- Basis Path Testing , in addition to Line coverage also provides Branch coverage

Example:



- We will use these nodes to make a program flow graph. Note left is for true

Program Flow Graph



V(G) = E - N + 2
= 11 - 9 + 2
= 4

E = edges
N = nodes

V(G) = P + 1
= 3 + 1

P = predicate node; nodes which have 2 or more outgoing nodes / edges

V(G) = R or ER + R0
= 4

R = region enclosed by nodes or edges

The V(G) metric gives us some intuition of how complex a function is.

focus on top 10 which have highest cyclomatic complexity as they are more error prone

A path is a sequence of nodes and edges from start to finish

~~Basis Path Set~~ Linearly Independant path is such a path which adds a new ~~node~~ edge

Basis Path Set:

path 1: 1 → 2 → 5 → 6 → 7 → 9

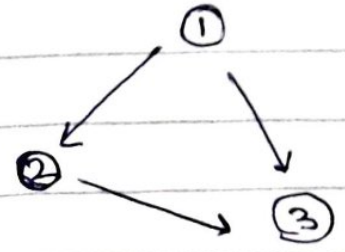
path 2: 1 → 2 → 3 → 4 → 6 → 7 → 9

path 3: 1 → 2 → 3 → 5 → 6 → 7 → 9

path 4: 1 → 2 → 5 → 6 → 7 → 8 → 7 ··· → 9

using this set gaurantees 100% branch coverage

Note: if you get branch coverage then line coverage is garanteed.

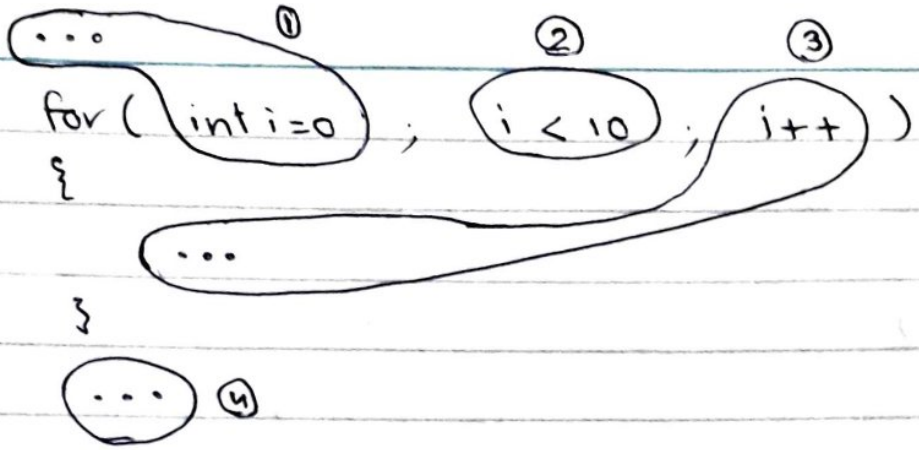


line coverage: 1 → 2 → 3

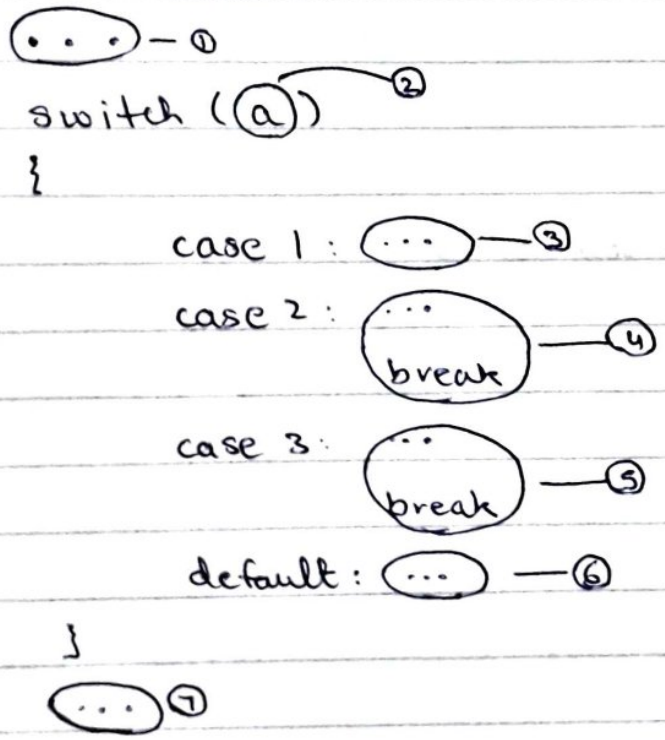
branch coverage: 1 → 2 → 3

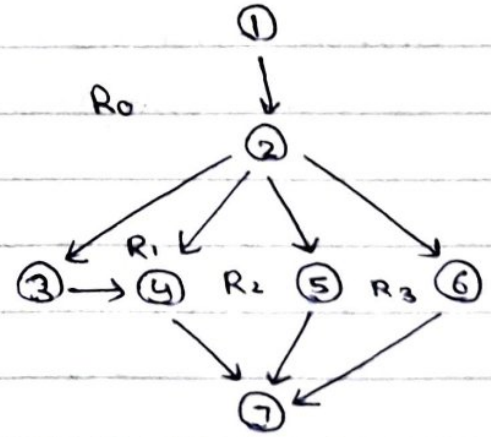
1 → 3

Loop



Switch Statement



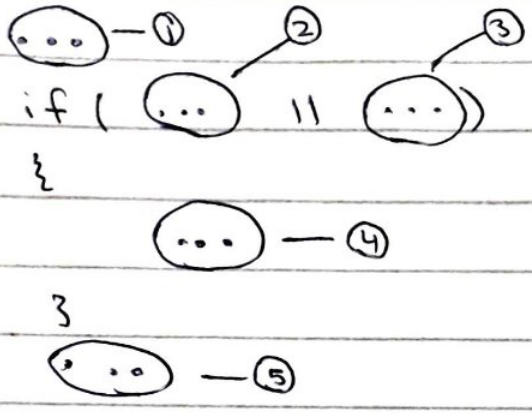


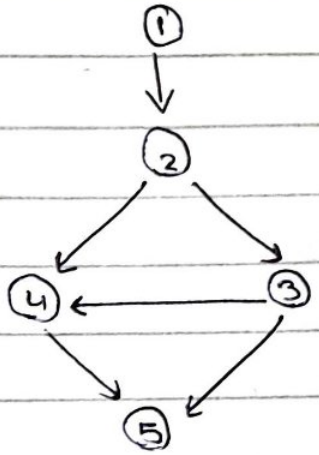
$V(G) = 9 - 7 + 2 = 4$

$= P + 1 = 1 + 1 = 2$

$= R = 4$

OR condition .





Advantages of White Box

- branch coverage
- checking compliance to standards

Advantages of Black Box

- less time
- less resources

Only use White Box for modules that are more error prone.

You can check the complexity from cyclomatic complexity graph of a module.

Software Engineering

Black Box Testing
- ↳ Equivalance Class Partitioning (ECP)
- ↳ Boundary Value Analysis (BVA)

- Bugs lerk at corners and congregate at boundaries
- provide invalid values one by one in different test cases.

Deployment

Software Configuration:
is a set of software configuration items with different versions

idea is to prepare release for shipment

- Deployment:
is an activity that makes software available for the customer to use

Operations

- You may need to provide help desk (support)
  - ↳ through email
  - ↳ phone
  - ↳ at client's location

support may be part of the contract.

Types of contracts:

- Fixed Price:
  total cost is known before development. It may be paid in segments

- Cost-Plus-Profit:
  some softwares may require R&D so consumer pays cost that was spent on development plus some profit.

Contract: a comprehensive document which is legal. It may have termination clauses, penalty clauses.

Software house to satisfy customer instead of just following the contract.

Causes of Termination:

- ↳ competitor launches the product and it captures the market
- ↳ technical issues
- ↳ key developer resigns
- ↳ can not be developed in time.

What to do on termination:

- Post Performance Analysis / Project Post Mortem:
  identify the bad and good things likes
  - mistakes
  - good practices

Maintainance

Software Evolution is a better word than maintainance as ~~in maintainee~~ we are introducing or replacing things / modules in the software.

Corrective Maintainance:
fix bugs after deployment. (defects)

⇒ Testing never stops:
End users may find bugs or suggestions and report them

Functionalty Improving Enhancement/ Maintainance:
customer demands new functionality as customer requirements (CR)

You should have a Change Control Board (CCB) to look at the CR and identify whether it should be implemented and if so ask for payment.

Perfective Maintainance:
improve the performance of already developed functionalities without changing them.

Adaptive Maintainance:
configure the software for new users with changing or introducing new functionalities