

Unit - I

Introduction to Software Engineering,

A generic view of process, Process models,

An Agile view of process

→ Software Engineering :-

It is a program/ set of programs containing instructions which provide desired functionality.

It also comprises of data structures that enable the program to manipulate information.

A process of designing & building something that serves a particular purpose

(or) Application of science tools & methods to find cost-effective solution to problem.

SE:- A systematic approach to the development operation & maintenance of desired software.

(or)

It is defined as systematic, quantifiable, disciplined approach for development, operation & maintenance of software.

→ Dual Role / Goal of a Software :-

i) As a product: It delivers the computing potential of a hardware (h/w)

→ enables the h/w to deliver the expected functionality.

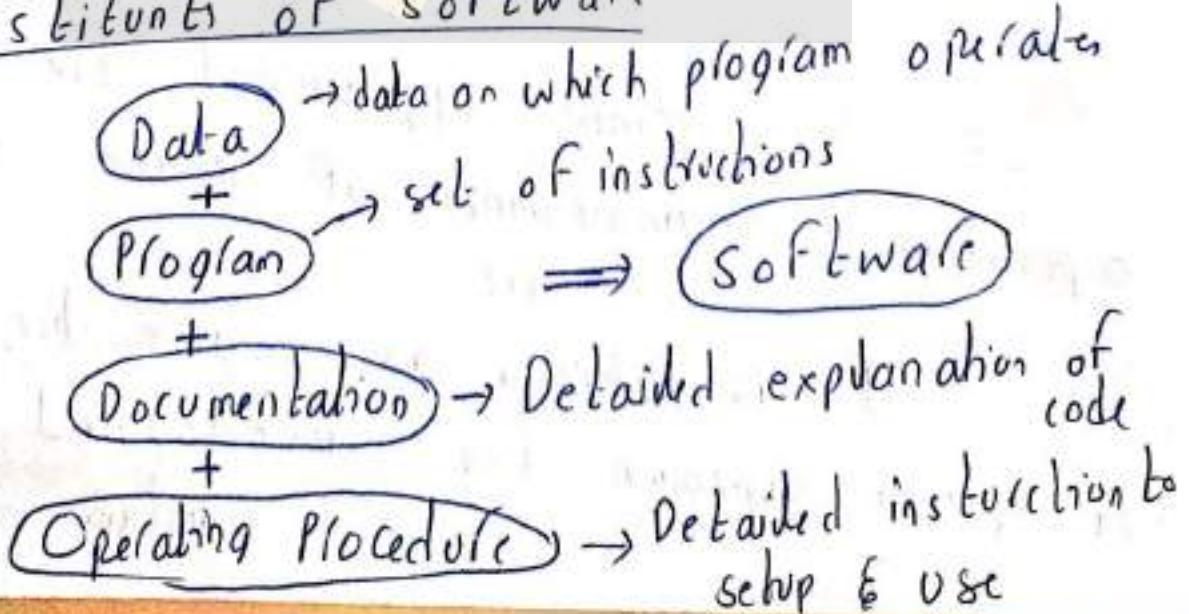
→ Acts as information transformer

Business info Personal info

ii) As a vehicle for delivering a product:

→ helps in creation & control of other programs (to handle other softwares)
ex:- Operating System.

→ Constituents of software



→ Phases of basic Program Development:-

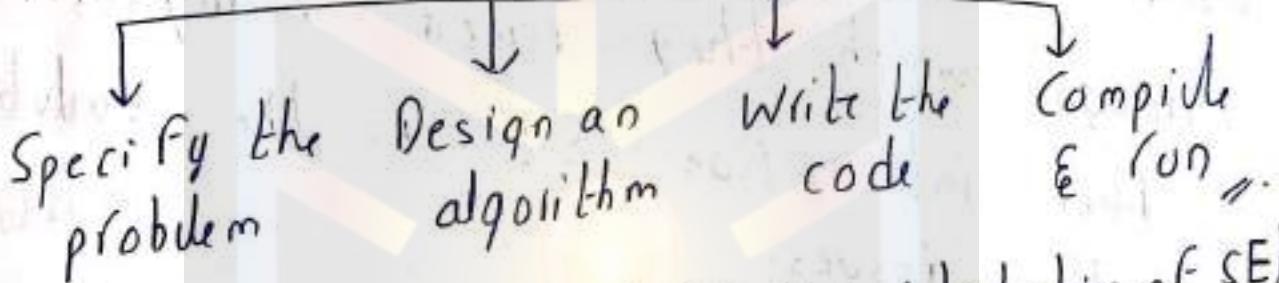
→ Every program we write is to solve a specific problem.

→ Once there is a problem we think how we can solve/use which approach to solve it (algo)

→ Then we write the the code.

→ Then compile & run.

Phases



→ Software Crisis :- (Before introduction of SE)

→ The problem that the developer face while building/developing the software.

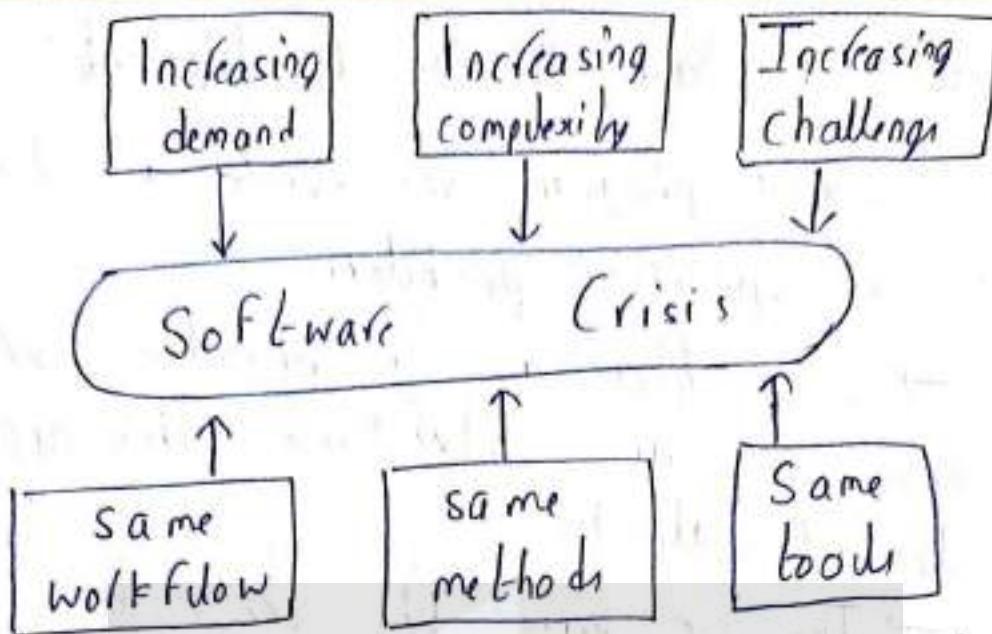
causes of software Crisis:

solution
SE

→ Project running over budget/time.

→ Software was very inefficient/dow Quality.

→ software didn't meet requirement



As previous company or developer's used to use their own methodology through which they faced many issues. Hence introduction SE is the solution for the issues.

~~→ Software engineering A layered technology~~

→ Software Myths:-

→ Users have no idea what thru know, what they want

→ Software Development Comes with a hefty Price tag.

→ You can't assess software quality until the program is running

→ Testing is too expensive

→ Testing is time consuming.

→ Only Fully Developed products are tested

→ A Tested software is Bug-free.

→ Type of Software:

i) System Software:

→ The softwares that provide a platform for other software to run
to run over them
→ they serve/help other programs
→ ex: operating system, compiler.

ii) Application Software:

→ They serve for a particular purpose

→ They can solve business or payroll, etc..

person problem (which we use in PC).

iii) Engineering / Scientific Software:

- Solve a scientific problem
- Complex numerical problems are solved
- or → statistic / calculator applications.

iv) Embedded Software:

- A small code used only for a specific functionality (IOT)

e.g. → micro oven, washing machine etc.

v) Artificial Intelligence Software:

- Acts human like intelligence in machine
- Computer algorithms which are non numeric

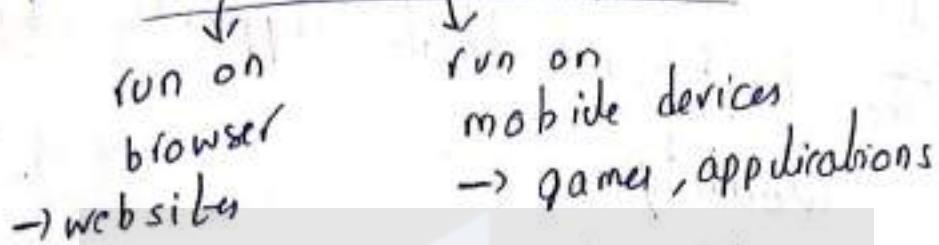
e.g. → Game playing, Robotics.

vi) Legacy Software:

- Very old & traditional software
- To meet business requirements
These are made

- changed from time to time
- Do not have a good quality as the software are made by any one.

vii) Web/mobile application:



viii) Real time software:

- Control monitor, analyse real world events in real time
- Applications which need to run at real time (Satellite launch software etc)

→ Software Engineering :- A layered technology

4 layered approach



↑ importance order

→ Software engineering comprise of a process, a set of methods for managing & developing the software and a collection of tools

→ The backbone that supports software Engg is Quality.

i) Quality :-

- Degree of goodness } as per client requirement
- Correctness
- maintainability (for future)
- Usability (used by users).

ii) Process : (What to do)

→ A framework that must be established for effective delivery of software management & control of software projects.

iii) Methods: (how-to)

→ Provide technical "how-to" for building a software.

→ each method consists of multiple tasks ex requirement analysis, testing & support

iv) Tools:

→ Provide automated/semi automated support for process & methods

→ a set of tools working in sequence
out put as input to 2nd tool (CAS)

Computer Aided Software

→ Software Process:-

→ A software process is a set of activities & associated results that produce a software product.

There are 4 main activities

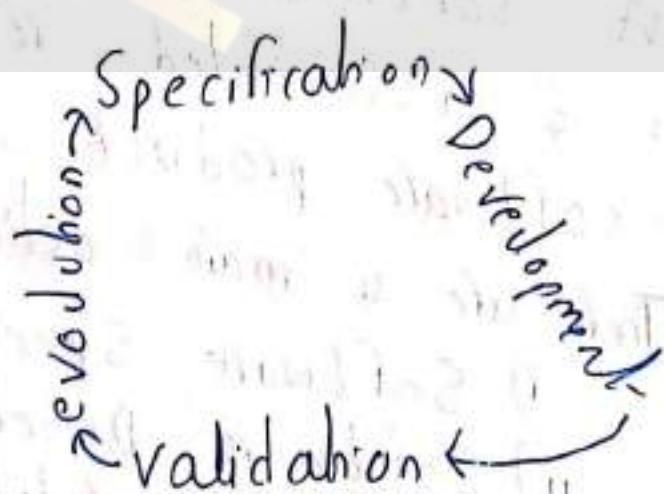
i) Software Specification

ii) Software Development

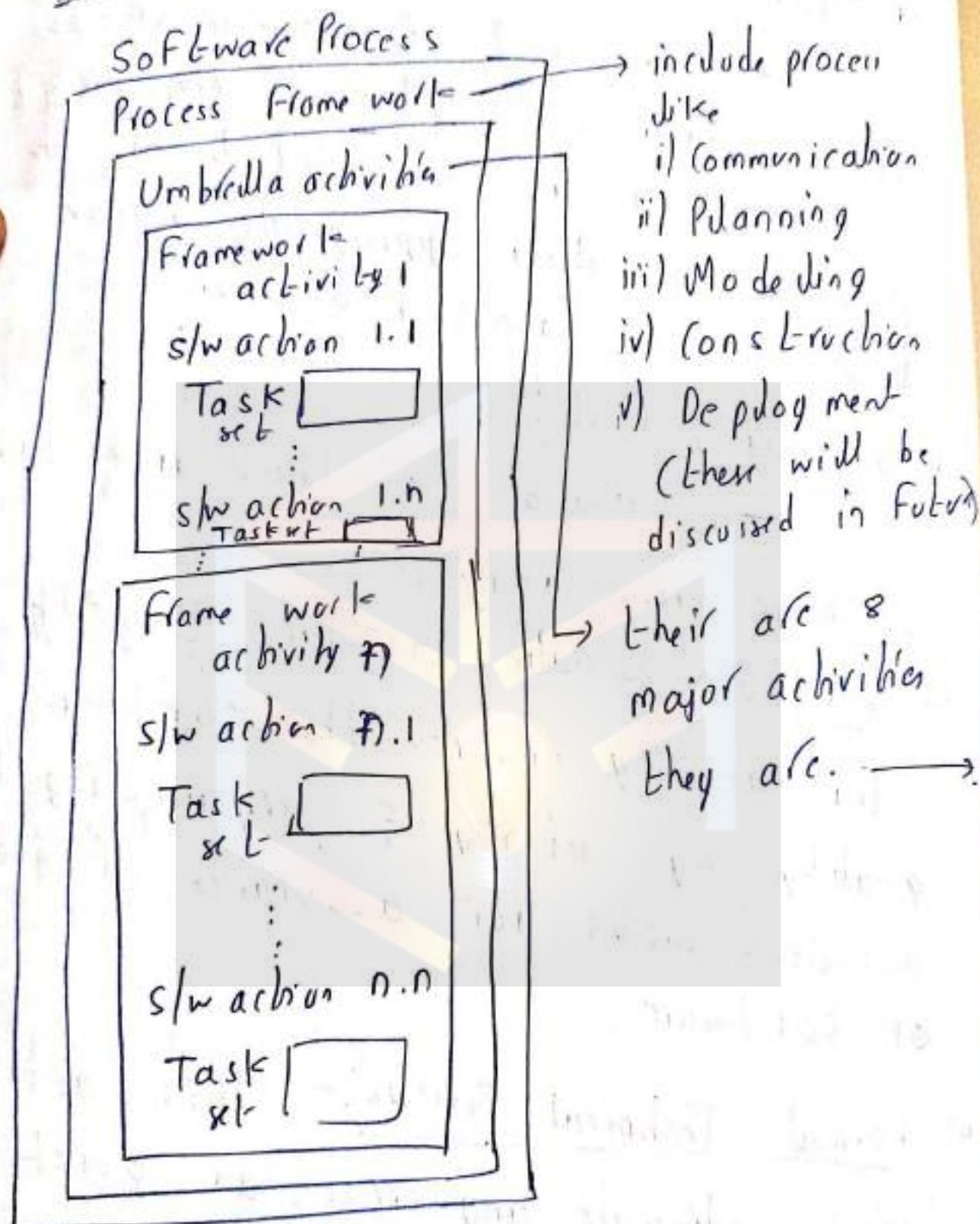
iii) Software Validation

iv) Software Evolution.

- i) Software Specification: where customer & engineers defines the software that is to be produced & the constraints on its operation.
- ii) Software development: where software is designed & programmed.
- iii) Software validation:- checking whether the software is upto the customer/client requirement.
- iv) Software evolution: the software must evolve to meet changing customer needs.
(Further changes should be possible.)



→ A Process Framework:



- include process like
- i) Communication
- ii) Planning
- iii) Modeling
- iv) Construction

v) Deployment
(These will be discussed in future)

There are 8 major activities

They are: →

i) Tracking & Controlling Software Project:-

- Assessment of progress of the project against the plan of the project
- Maintain the schedule of the project by taking appropriate action based on the above assessment.

ii) Managing Risk:-

- evaluation of those risks that can have serious impact.

iii) Software Quality Assurance (SQA):-

This activity ensures the software quality by defining & organizing the activities needed for assurance of quality of software.

iv) Formal Technical Reviews:- This activity tries to eliminate any errors as quick as possible before they spread to other modules.

v) Software Measurement:- The activity describes as well as gathers measures of process, product & project through software team

vi) Software Configuration Management (SCM):-

These are also give controlled changes to the software components can be done here.

vii) Managing Reusability Factor:

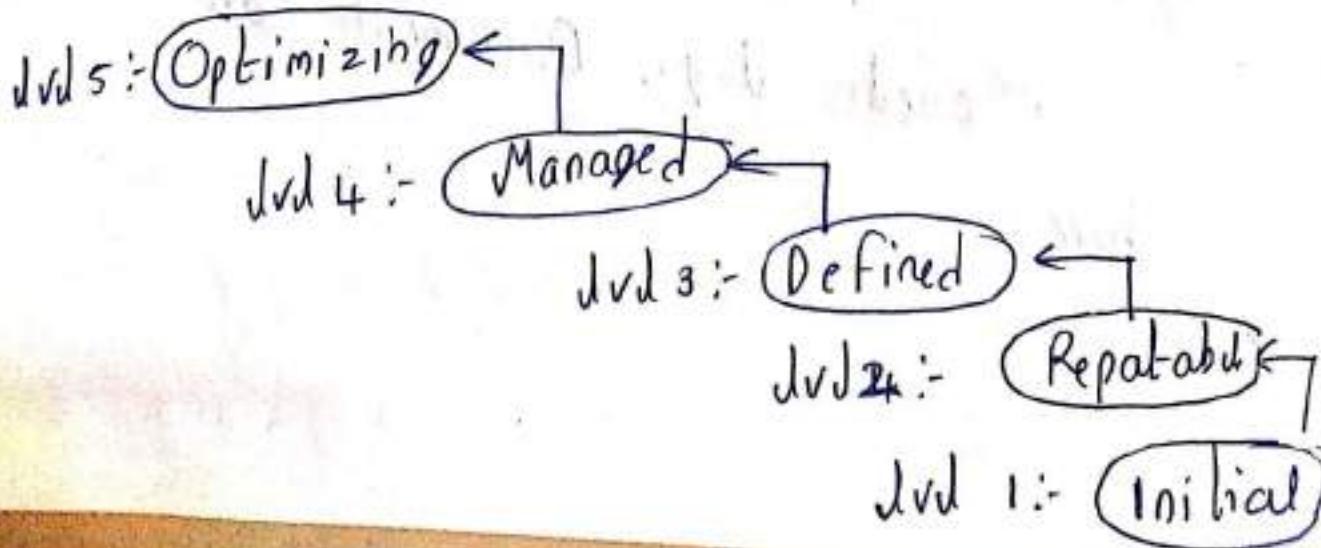
Reusing the previously made project modules for faster delivery.

viii) Preparing & Producing Software ~~Walls~~
Product-1

final activity
Models, Jogs, Documents are maintained here.

→ CMM/CMMI (Capability maturity model Integration)

- CMM was developed & is promoted by the Software engineering Institute (SEI) a research & development cent.
- CMM is not a software process model. CMM is used to a benchmark to measure the maturity of an organization's software process.
- The model describes a five-level evolutionary path of increasingly organized & systematically more mature processes.



→ Level initial :- (work is performed informally)
A software development organization
at this level is characterized by ad
hoc activities (organization is not planned
in advance)

e.g. only project is complete No
documentation or SRC or any. (Group of
freelancers)

→ Level 2 Repeatable :- (work is planned & tracked)
This level of Software Development
Organization has basic & consistent project
management processes to track cost, schedule
& functionality.

The process is in place to repeat
the earlier successes on projects with
similar application.
as we use previous successful
project approach / strategy this method is
repeatable.

→ Jvl 3 Defined: (Work is Well-Defined)

At this Jvl the software process for both management & engineering activities are defined & documented.

→ Jvl 4 Managed: (Work is Quantitatively (time) controlled)

Software Quality management

Management can effectively control the software development effort using precise measurements. At this

Jvl organization set a quantitative quality goals for both dev process & maintenance

Quantitative Process management

At this maturity level the performance of processes is controlled using statistical & other quantitative techniques & is quantitatively predictable

→ Jvds Optimizing: (Work is based upon Continuous Improvement)

→ The key characteristic of this level is focusing on continuously improving process performance

^{main}
^{feature} → Process change management

→ Technology change management

→ Defect Prevention

ex: Jvds → Cognizant, Wipro.

→ Process Pattern:-

→ The software process can be defined as a collection of patterns that define a set of activities, actions, work tasks or work products & similar related behaviour followed in a software development life cycle.

→ In common words while a project is being developed the team do

face multiple problems & they need to resolve it first to continue. So few common problem solutions are prewritten so the issue can be resolved faster. (using process pattern).

→ Using a simple template we will be able to problem & process pattern do suggest for pre written solution if exist.

Fields in template

→ Pattern Name: name with the specific issue (no ambiguity).

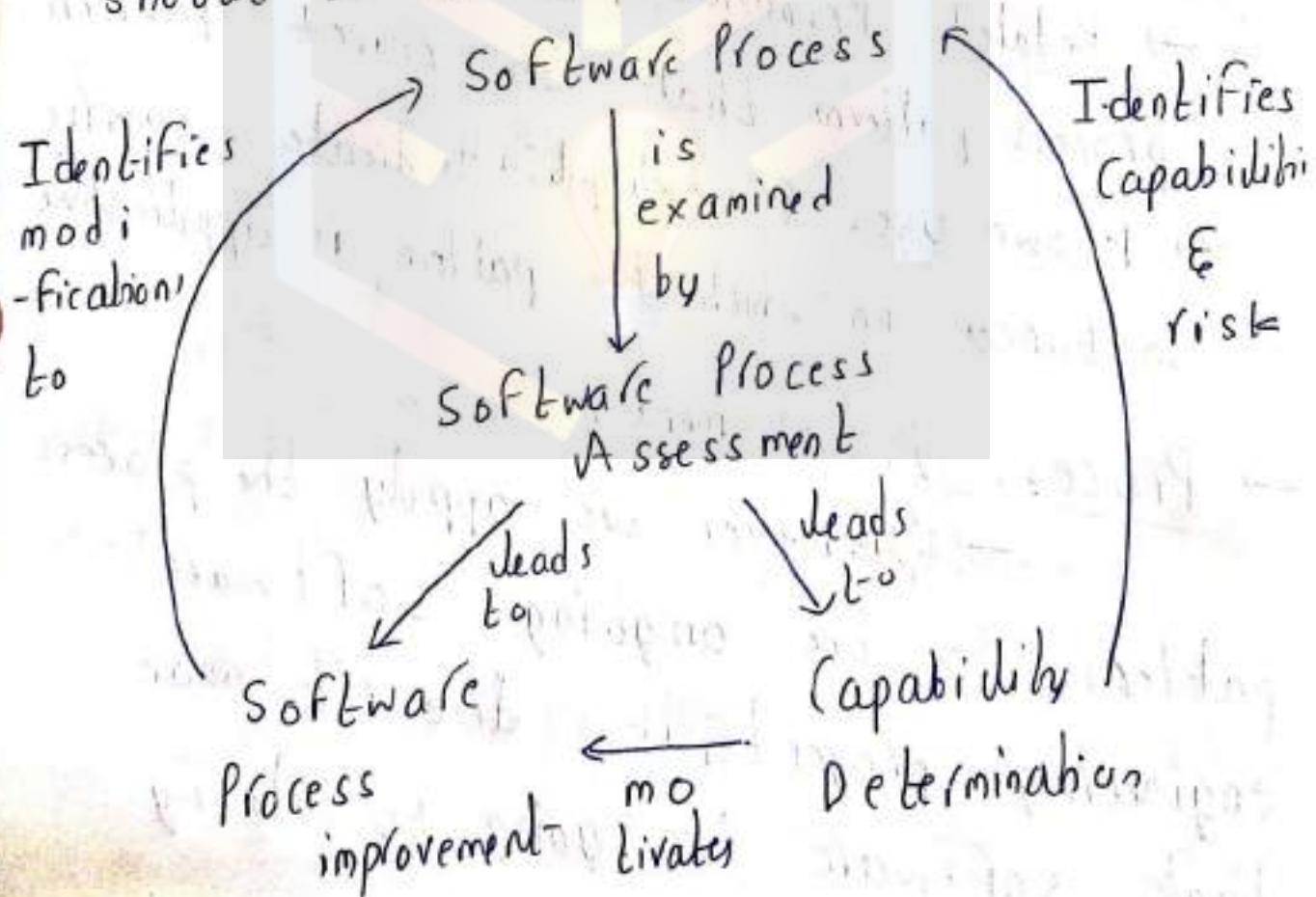
→ Forces: the environment in which the pattern is encountered & the issues the make the problem visible (associated with work basis) ~ in task pattern (from work)

→ Types: Stage pattern, Task pattern (from work) (requirement analysis)
phase pattern (associated with frame work activities) (ext. communication)
(associated with complete process model)
(ex: in spiral model)

- Initial Context: Describes condition under which pattern applies.
- Problem: The specific problem to be solved by the pattern.
- Solution: Describes how to implement process pattern.
- Resulting Context: Describes the condition that will result once pattern has been successfully implemented.
- Related Problems: Provides a list of all process patterns that go current pattern.
- Known Uses & Examples: Indicate specific instances on which the pattern is applicable.
- Process Assessment:
 - Whenever we apply the process patterns to the ongoing software engineering project, it doesn't mean that software is going to satisfy

all the essentials (i.e., on time delivery, customer satisfaction, high quality etc..) which are essential in success of software.

→ Hence in order to achieve this the software patterns should be collaborated with highly valued software engineering practice & the entire process should be sufficiently accessed as required.



i) standard CMMI Assessment method for process improvement (SCAMPI)

→ includes 5 steps

initializing, Diagnosing, Establishing, Acting, Learning.

ii) CMM - Based Appraisal for Internal Process Improvement

→ provides a diagnostic technique for assessing real maturity of the software organization.

iii) SPICE - The SPICE (ISO/IEC 15504)

→ It defines a set of requirement for software process assessment (standard).

iv) ISO 9001:2000:

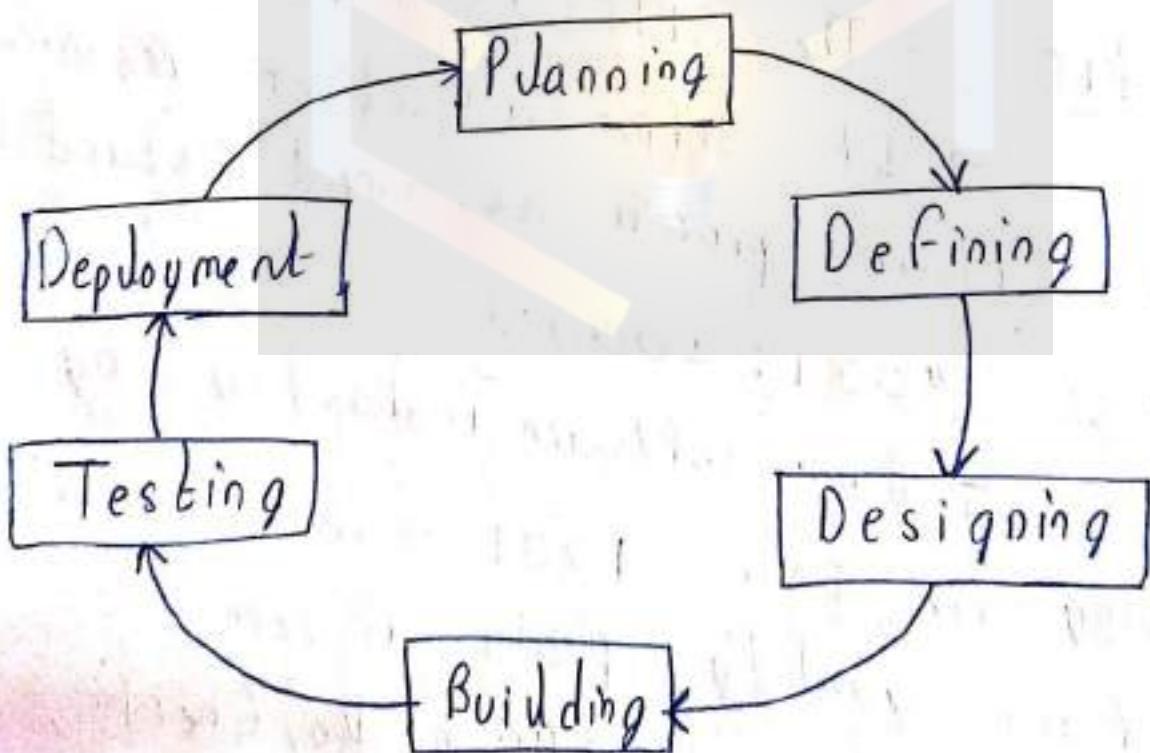
→ Any software industry, by adapting the ISO 9001:2000 standard can reach quality peaks in terms of its products, system (Plan, Do, Check, Act) it follows.

→ Software Development Lifecycle (SDLC)

→ Software development life cycle (SDLC) is a process used by software industry to design, develop & test high quality software.

→ ISO/IEC 12207 is an international standard for software development life cycle

→ Software Development life cycle is also called Software development process



i) Planning & Requirement Analysis:-

- Requirement analysis is the most important & fundamental stage in SDLC.
- It is performed by the senior members of the team with inputs from the customer/client.
- This information is then used to plan the basic project approach & to conduct product feasibility study in the economical operational & technical areas.
- Planning for the quality assurance requirements & identification of the risks associated with the project is also done in the planning stage.
- The outcome of this step is various technical approaches that can be followed to implement the project successfully with minimum risks.

iii) Defining Requirements:

→ Once the requirement analysis is done the next step is to clearly define & document the product requirements & get the approved from customer/client.

→ SRS (Software Requirement Specification) document - consists of all the product requirements to design & develop.

iii) Designing the Product Architecture:

→ Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed & documented in a DDS (Design Document Specification).

→ among which one is finally selected & all the diagrams like data flow representation, (what what are used) every is specified detailedly.

iv) Building or Developing the product:-

→ During this stage DDS is converted to programming code.

→ Developers follow the rule/ software requirement specification and on that they start working & based on requirement they use any specific language (C, C++, Java, Python, PHP etc.).

→ outcome will be the software.

v) Testing the Product :-

→ Once the development is done, testing of the software is necessary to ensure its smooth execution. (Although, initial testing is conducted in every stage of SDLC)

→ here all the flaws are tracked, fixed & retested

→ This ensures that the product confronts the quality requirements of SRS.

vii) Deployment in market & maintenance

- Once the product is tested & ready to be deployed it is released formally in the appropriate market.
- Even few products are released in specific countries before global release & based on the feedback changes are made tested & published global.
- frequent maintenance is also done based on feedback & client requirements.
- There are various software development life cycle models defined & designed which are followed during the software development process.
- These models are also referred as process models.

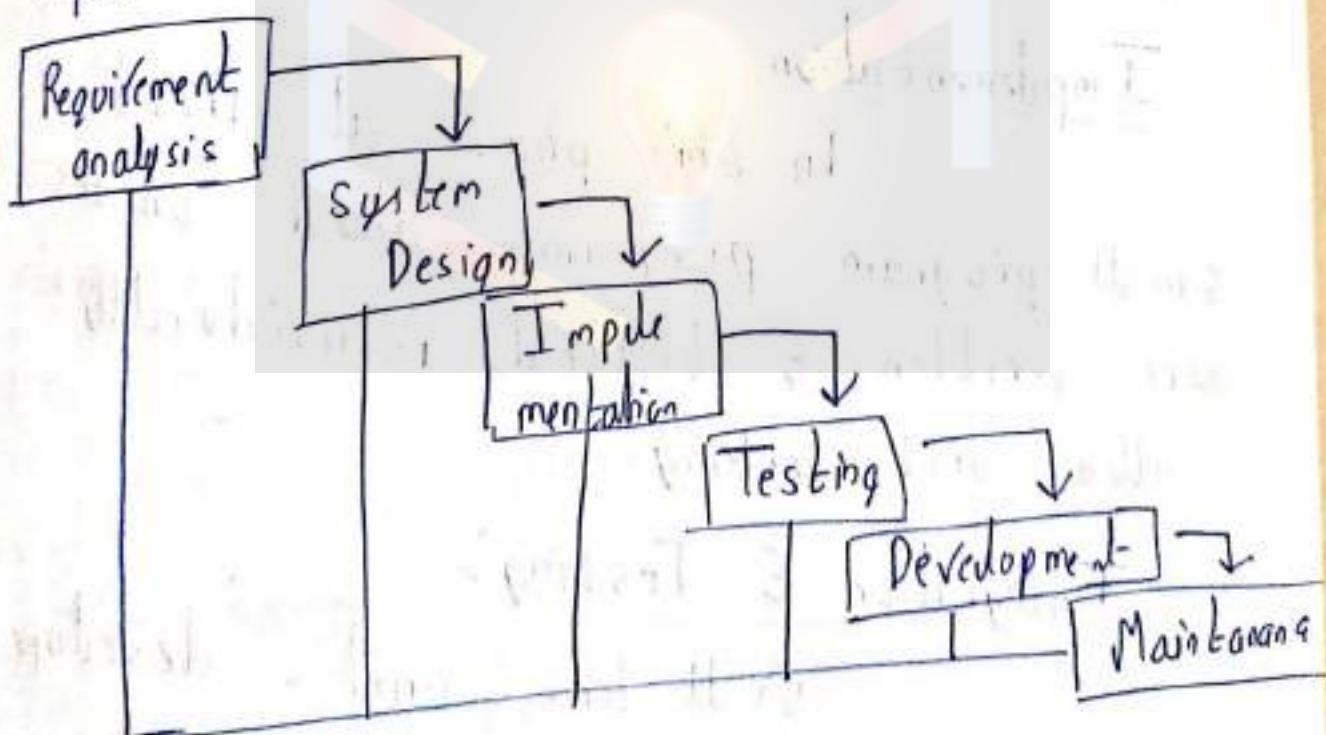
They are.

i) Waterfall model :-

→ Waterfall approach was first SDLC model to be used widely in software engineering. To ensure success of the project.

→ In the waterfall approach the whole process of software development is divided into separate phases.

→ In waterfall model one's, outcome of one phase acts as an input for the next phase sequentially.



Requirement Gathering & analysis

All possible requirements of the system to be developed are captured in this phase & documented.

System Design:

System Design helps in specifying hardware & system requirements & helps in defining the overall system architecture.

Implementation

In this phase at first small program programs called units are written & tested individually called unit testing.

Integration & Testing:

All the units developed are integrated into a system & testing is done.

Deployment of System: Once testing is done the product is ready for deployment.

Maintenance: If there are any issues they will be resolved here.

Adv:

- Simple & easy to understand & use
- phases are started & completed one at a time (Phases do not overlap)
- Requirements are very well understood
- Low cost.

disadv:

→ difficult to acquire of the requirement in the starting.

- high risk, not feasible.
- Not suitable to accommodate

any change.

→ Not good for large size projects.

iii) Incremental Process models

a) Incremental model :-

→ Incremental models divides its software development process into certain number of increments. each increment consist of the same steps (communication, planning, modeling, construction & deployment).

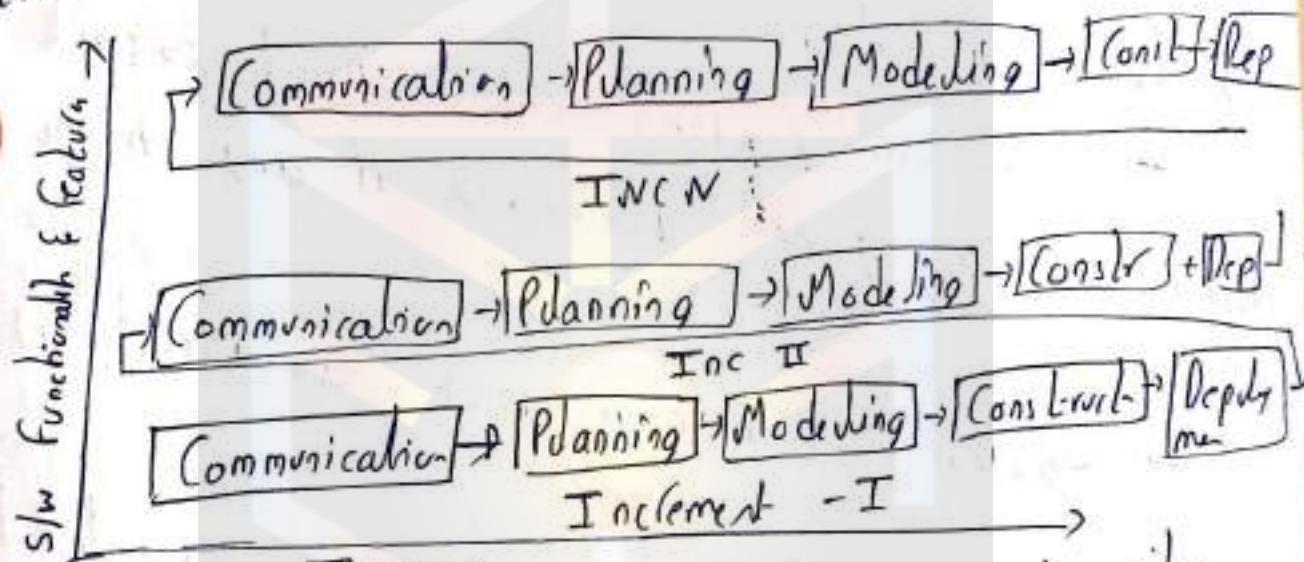
→ Hence to make a big project into multiple iterations / parts. we must be clear with requirements.

→ the process is defined in this way-

"It is based on the idea of developing an initial implementation, exposing this to user feedback, & evolving it through several versions until an accepted system has been developed.

→ important / mandatory Functionalities of the software are done in the 1st iteration.

→ each subsequent release of the software module adds function to the previous release. The process continues till the complete system / software is achieved.



Time →
Note: in examination based on marks describe

the 5 stages

- i) Communication
 - ii) Planning
 - iii) Modeling
 - iv) Construction
 - v) Deployment
- } can be found in previous methods

Adv:

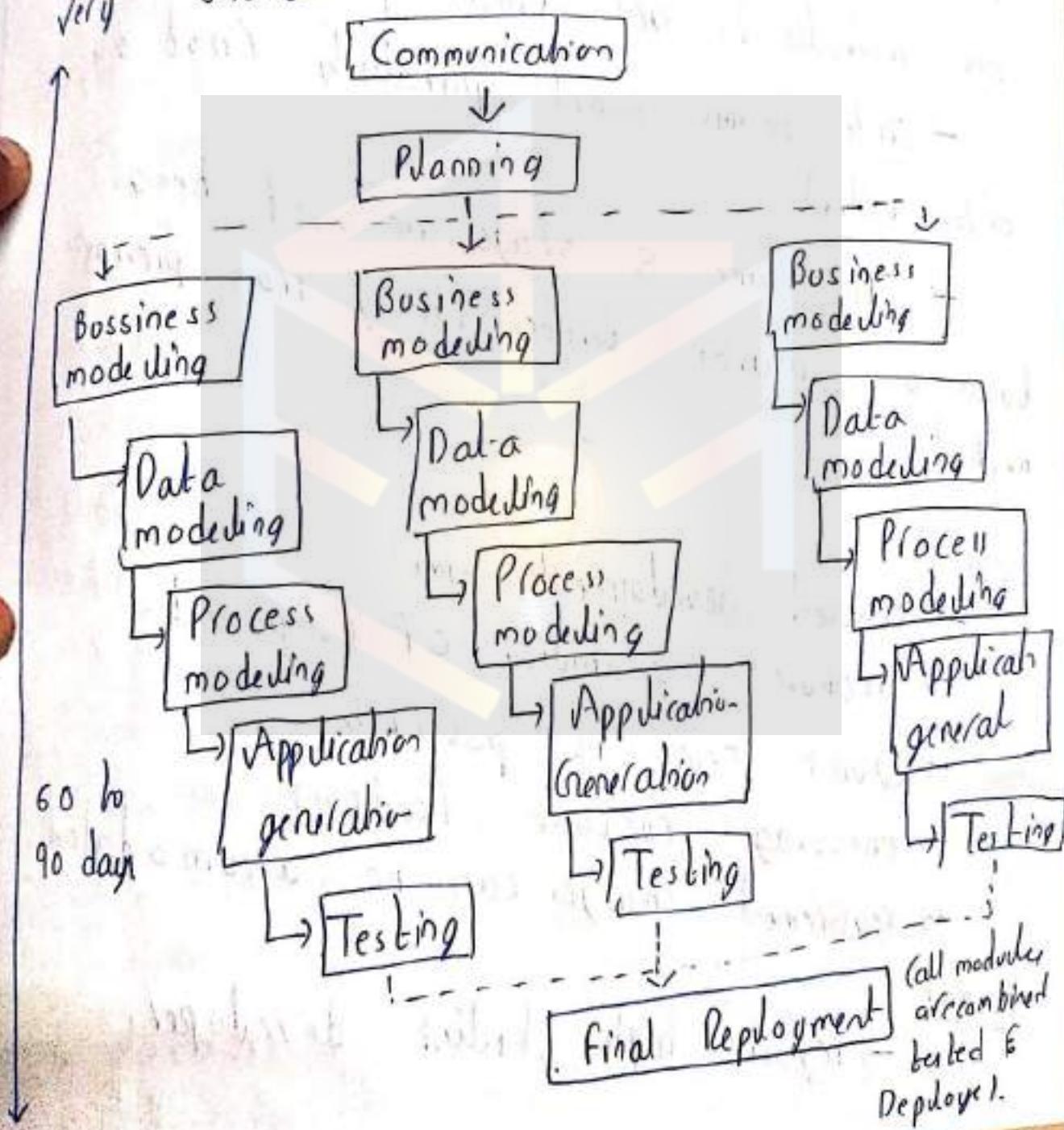
- Generate working software quickly & early during the software life cycle
- This model is more flexible - doesn't cost much to change scope & requirements.
- It is easy to test & debug during a small iteration.
- customer/client can respond to each build & can give feedback if any changes needed.
- easier to manage risk (as iteration)

Disadv:

- Total cost will be higher (or can lead to multiple iterations)
- Needs a clear & complete definition of the whole system before it can be broken down & build incrementally.
- Needs very good planning & design.

b) RAD or Rapid Application Development model

→ Rapid Application Development model proposed by IBM in 1980, is suitable when customer requirement is clearer but the schedule is very short.



→ Here project is divided into different teams & develop simultaneously, over a short period of time & all the independent modules (once available). So, integrate the modules to get final product.

→ Note:- teams work parallelly (not one after other).

→ The same 5 stages are used ^{here} based on waterfall theory from previous methods.

Adv:-

→ Reduced development time

→ Increase reusability of component.

→ Quick review is possible.

→ Encourage customer feedback.

→ Requirement changes can be accommodated.

Disadv:-

→ Require high skilled developer.

- should not be used with small projects
- Only system that can be modularized can be build using RAD

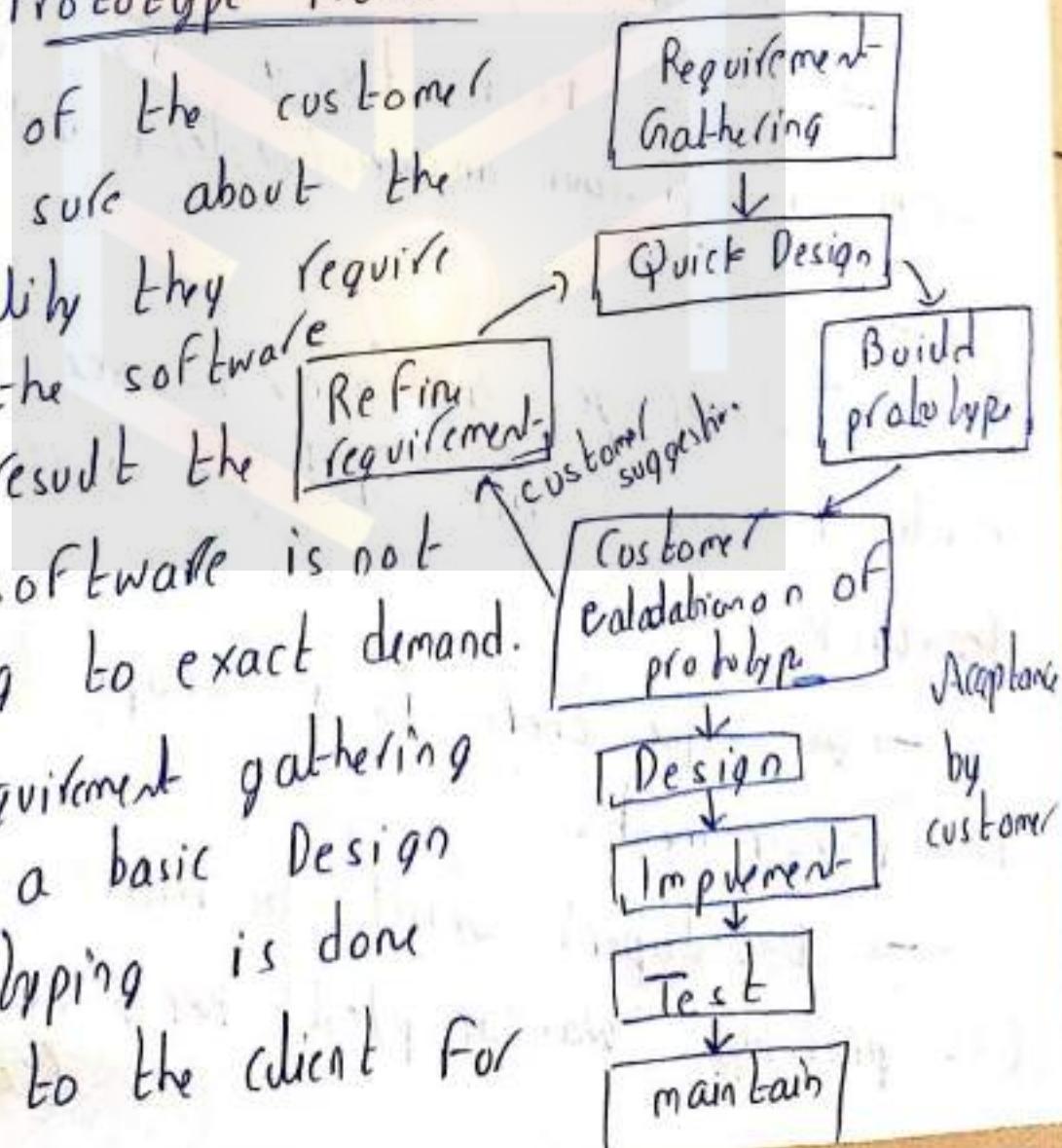
iii) Evolutionary Process models:

→ here we take multiple inputs from the client customer until the adequate system (client requirement)

a) Prototype model

→ Most of the customers are not sure about the functionality they require from the software as a result the final software is not according to exact demand.

→ Once requirement gathering is done a basic Design & prototyping is done & show to the client for



Validation

- if client informs no this was not my requirement & then we collect requirement again & do basic design & prototype & ask ask for validation until he accepts
- Once client/customer accepts it we move on to the common phases.

Design, Implementation, Test & maintenance.

- U can find theory for the above steps in previous answers (Waterfall / SDLC)

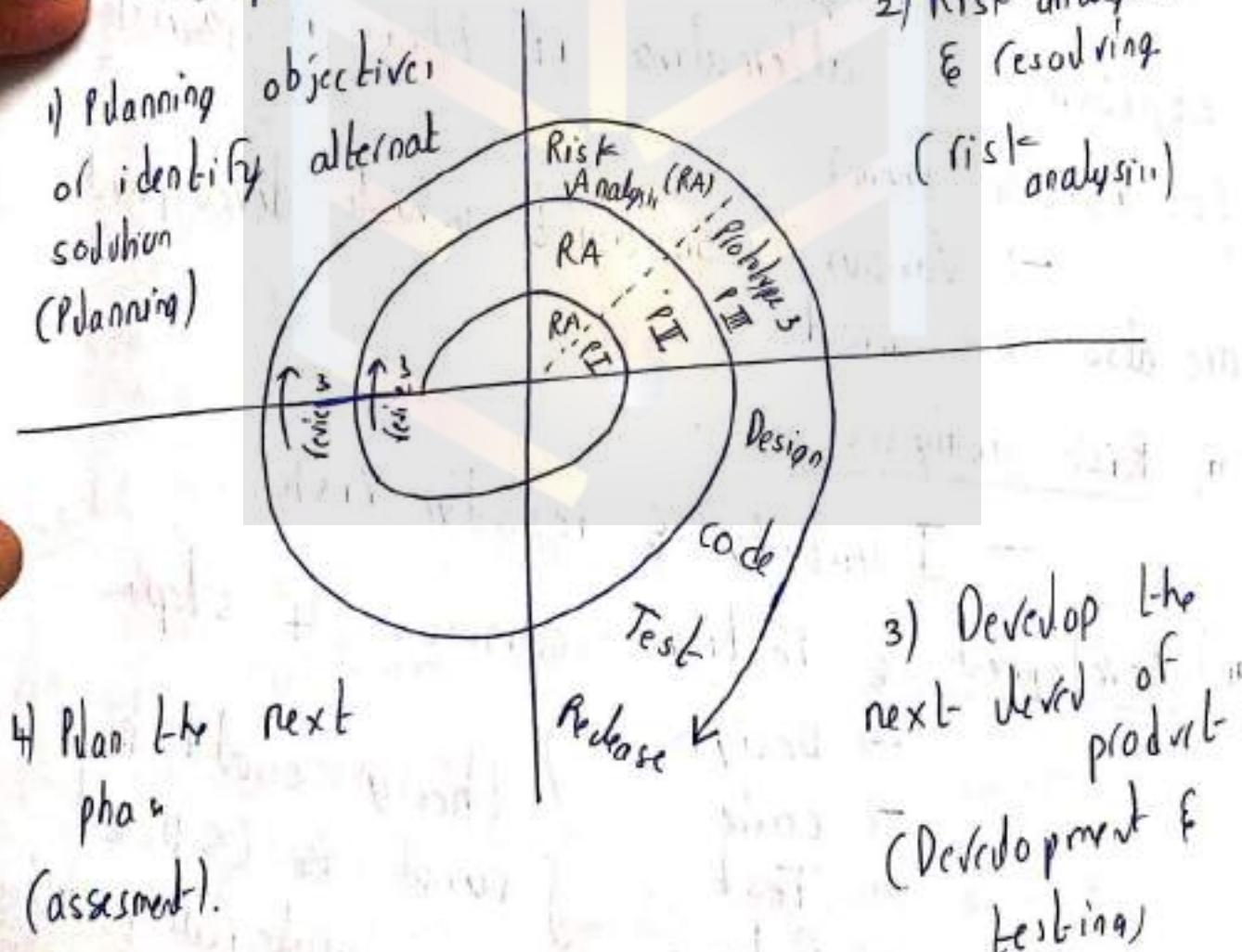
Adv:-

- Customer get a chance to see the product early in the life cycle, & give important feedback.
- At scope there is a scope to accommodate new requirements.
- Developers will be more confident (as prototype was accepted) hence risk is reduced.

DisAdv:

- After seeing the cardy prototype user demand the actual system as per his/her requirements
- If not managed properly the iterative process can run for a long time.
- If client is not interested, he may lose it's interest in the project

b) Spiral model:



→ developed by Barry Boehm in 1986
→ The main objective of this model is to handle risks.

→ each path/spiral has its own cost & keeps increasing.
→ each spiral divided into 4 activities

i) Planning:

→ ~~Definition~~, Here we discuss the key objectives & alternatives if there is a risk (to handle them)

→ Various constraints which development are also considered.

ii) Risk analysis:

→ Identify & resolve risks.

iii) Development & Testing: common 4 steps

→ Design
→ code
→ Test
→ Release

} theory could be found in (SDLC / waterfall model)

iv) Assessment: Feedback from the (customer evaluation) based on that the process continues in spiral format.

in Spiral I: Planning → Risk analysis → Prototype I is build, customer evaluation & feedback is collected.

Spiral 2: based review update prototype II is build (with risk analysis), SRS is written & sent again for validation.

Spiral 3: here based on feedback final prototype II made (avoiding risks) & therefore start the design → code → Test & release or any standard SDLC model.

Adv: → Provides early & frequent feedback of the customer, additional functionality can be added at a later date.

→ resolve all the possible risk involved in the product early in the life cycle.

- Continuous customer involvement, so better customer satisfaction.
- Good for large & mission critical projects.

Disadv:

- not suitable for small size project
- Complex to use.
- Risk analysis requires highly specific expertise.

~~c) The Concurrent Development model:-~~

iv) Specialized Process Model :-

a) Component Based Development :-

- Component based development has many characteristics of spiral model. It is evolutionary in nature & includes iterative approach for software development.
- This model composes the application.

from prepackaged software components
i.e., from existing software modules.

→ In this approach, commercial off-the-shelf (COTS) s/w components, developed by vendors who offer them as products are used in the development of software.

Adv: → Leads to software reuse, which provides number of benefits
→ 70% reduction in development life cycle time

disadv: → Component library must be robust.
→ Performance may degrade.

b) Formal methods model:

→ it encompasses a set of activities that lead to formal mathematical specification of computer software.
→ it consists of specification, development & verification by applying rigorous math notation

Adv: → removes many of the problems that are difficult to remove using SW Eng Paradigms.

→ Ambiguity, Incompleteness can be discovered.

disadv: → Development is time consuming & expensive

→ extensive training is required. (AOSD)

c) Aspect Oriented Software Development

→ When create any application/

software

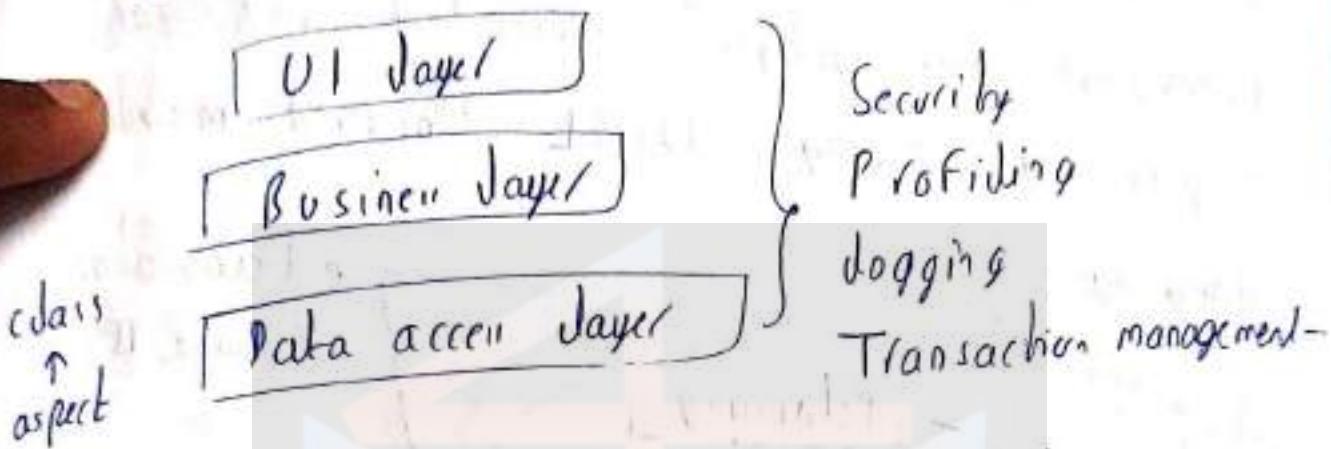
→ Even it has many different modules which has many common snippets of code like (security, profiling, transaction management)

→ Hence these codes are written in

classes & these classes are called Aspects.

→ It involves breaking down a software system into components known as modules.

→ the ^{common} code i.e (security, profiling etc..) are called cross-cutting concerns → code working in different areas of application
ex: for app



Adv:

- Increase modularity, reusability.
- Increase maintainability.

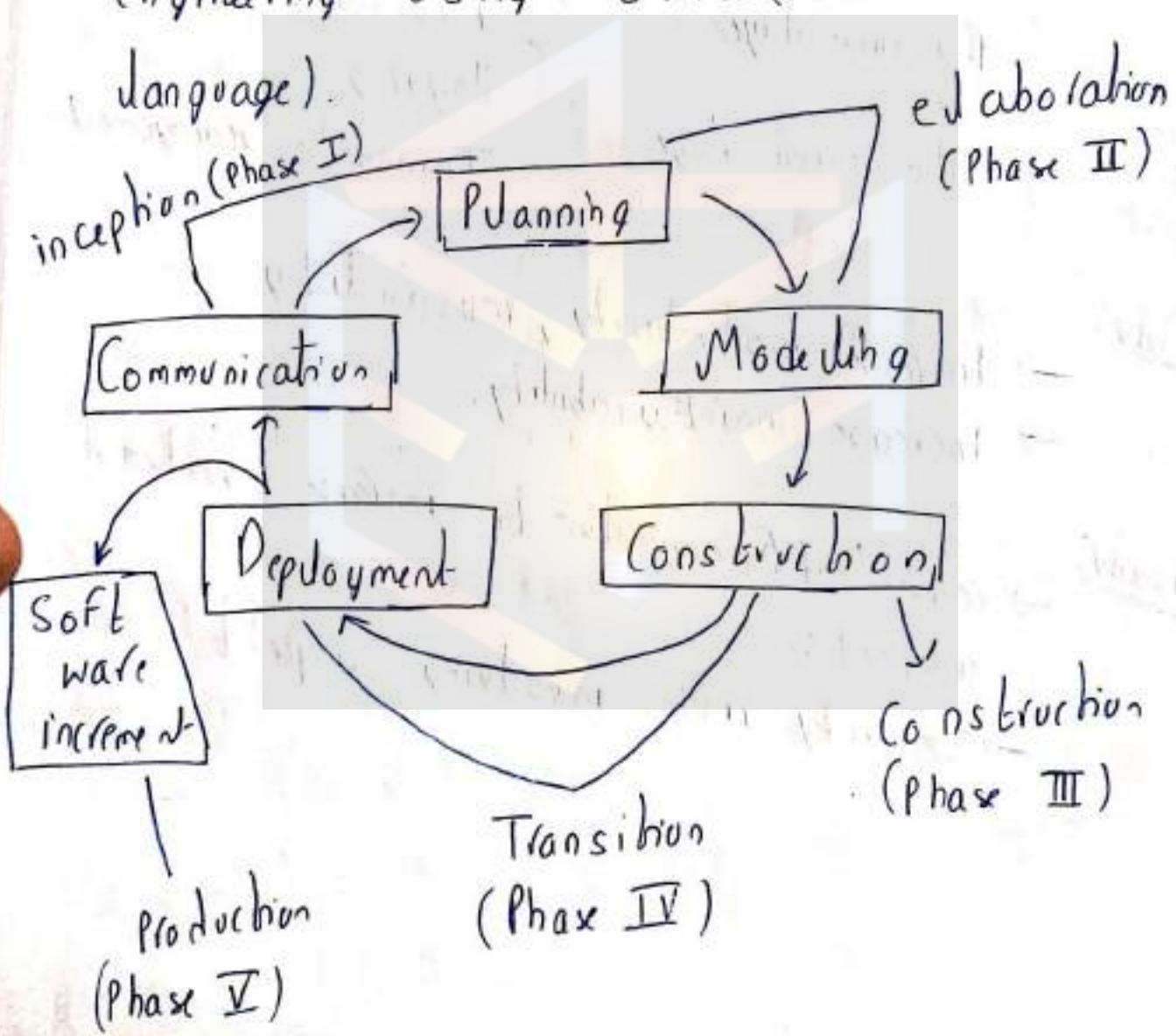
Disadv:

- Reduce efficiency due to increase overhead aspects.
- Security risk involving aspect.

v) Unified process model :

→ The Unified process of

Relational Unified process (RUP) is a framework for object oriented software engineering using UML (Unified modelling language).



I) Inception phase:-

→ The Inception phase of UP include both customer communication & activities by collaborating with the customer, & endeavor to collect business requirements & plans are proposed.

II) Elaboration Phase:-

→ The elaboration phase encompasses the planning & modeling activities of the generic process model.

→ Here elaboration for the usecase to analysis, design, implementation & development model (in UML).

III) Construction Phase:-

→ Using architectural model as input the construction phase develops the software component that will make each usecase operation & further carried out for testing.

IV) Transition Phase:

→ Here the software is given for testing to end user. For β testing user manual & installation manual are made.

→ At the end it is ready to release.

V) Production Phase:

After release any more needs by the user will be added as per requirement.

Adv: → It covers the complete software development life cycle.

→ The best practices for software development are supported by unified process model.

→ It encourages designing in UML

disadv:

- could be complex to implement.
- heavy weight process.
- experts are need for it
- risks cannot be determined.

vi) Personal software Process (PSP)

→ We know that every team member possesses his own personal software process.

→ The Personal software process is a structured software development process that is designed to help software engg to better understand & improve their performance.

It mainly discuss about:

- how to manage their project quality
- make commitments & planning
- improve estimation & planning
- reduce the defects.

it has 5 major framework activities

i) Planning (Plan what you doing)

ii) High-level design (complete project design is made for the help of faster understanding)

iii) High-level design review (review it multiple times to remove errors/mistakes).

iv) Development (Code/develop the components for the project).

v) Post-mortem (efficiency/correctness of each module is done).

vii) Team Software Process (TSP):

→ In combination with personal software-process (PSP), the team software process (TSP) provides a defined operational process framework that is

designed to help teams of manager & engineers organize projects & produce software products.

9)

Few guidelines

- min 3 to 20 engineers per team
- every ^{one} should have a clear target & idea to be achieved.
- every team member will be responsible in delivering his/her work.
- maintaining project up-to-date.
- a proper team leader should be selected & he should main all the tracks & logs of all the team members.

Few activities

- maintaining high - lvl - Design
- Implementation
- Integration (combining code)
- Testing.
- postmortem.

VIII) Agile Process models:

→ Agile: The term agile means "the ability of response to changes from requirements, technology & people".

(or)

Agility is dynamic, context-specific, aggressively change embracing, & growth oriented.

→ When a team is developing a project (the client requirement, technology, staff/employee) working on project can change. Hence adapting to those aspects is done here. (easidly & efficiently)

→ Agile process:

→ It refers to a software development approach based on iterative development

→ Agile process break tasks into smaller iterations or parts don't directly involve long term planning

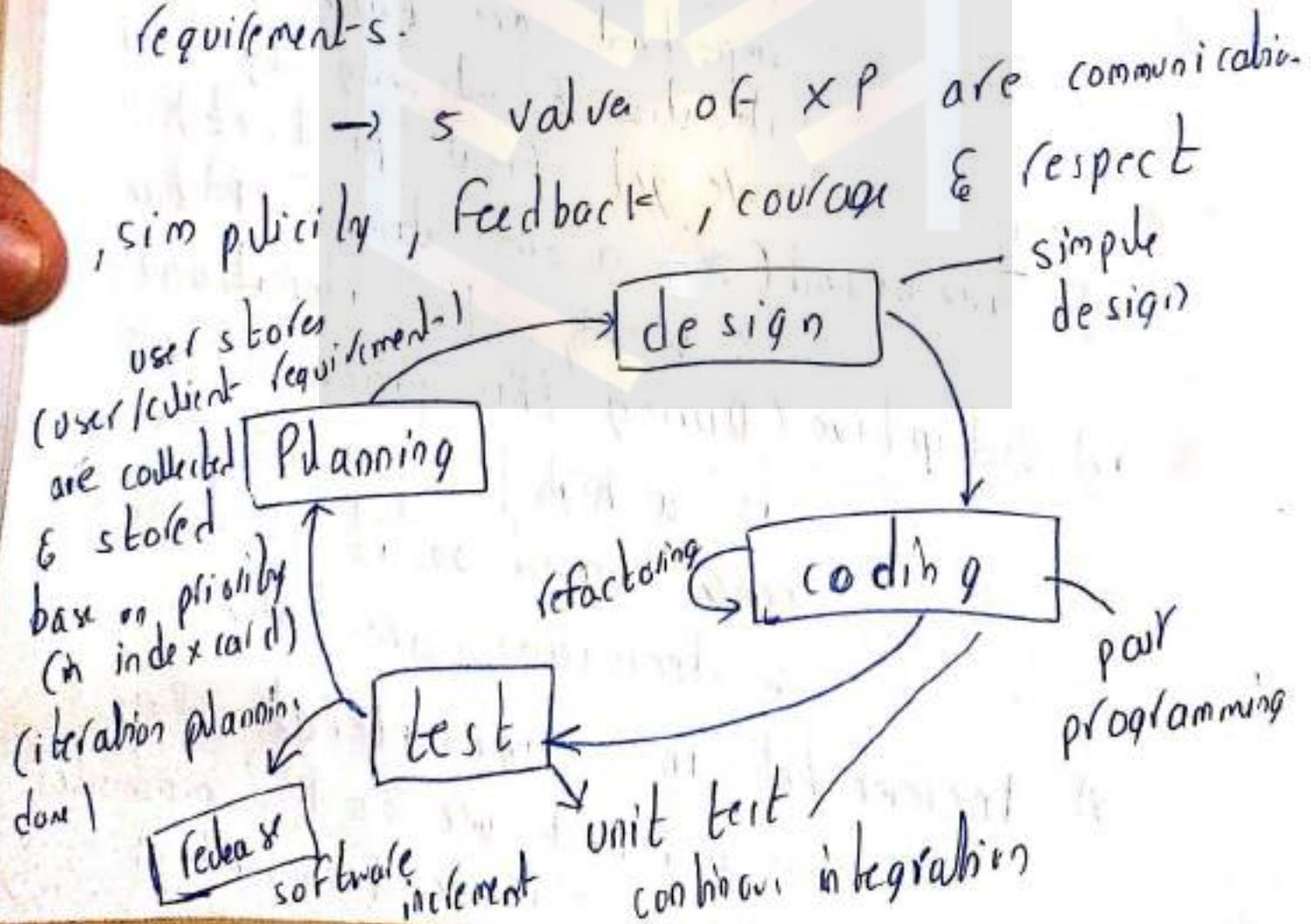
→ Hence if any changes are needed then in next iteration they are applied

→ Characteristics of Agile Process:

- i) modularity (dividing large project to modules / activities).
- ii) iterative (small small modules are taken which are mandatory, important are taken in 1st iteration & slowly by end we get final product).
- iii) Time bound (set a time limit 1 to 6 weeks per iteration)
- iv) Adaptive (During this process there is a high chance of risk hence based on it solution & decisions are taken).
- v) Incremental (in each iteration or we increment we do the modules (not in single increment))

a) Extreme Programming (XP) :

- XP (Extreme Programming) is an agile software development method.
- XP is a lightweight, efficient, low risk, flexible & fun way to develop a software
- Small to medium sized teams that work under vague & rapidly changing requirements.



→ pair programming: instead of 1 guy programming here 2 members are allotted 1 will be programming & other will give feedback on the code.

→ refactoring: improvement improves the internal structure of the code but external behavior not effected.

→ Unit test: each module is tested individually.

→ continuous integration: all the tested code are combined (integrated) & tested again. (system testing).

→ Planning, design, coding detailed, they are already written in waterfall model.

Adv:-

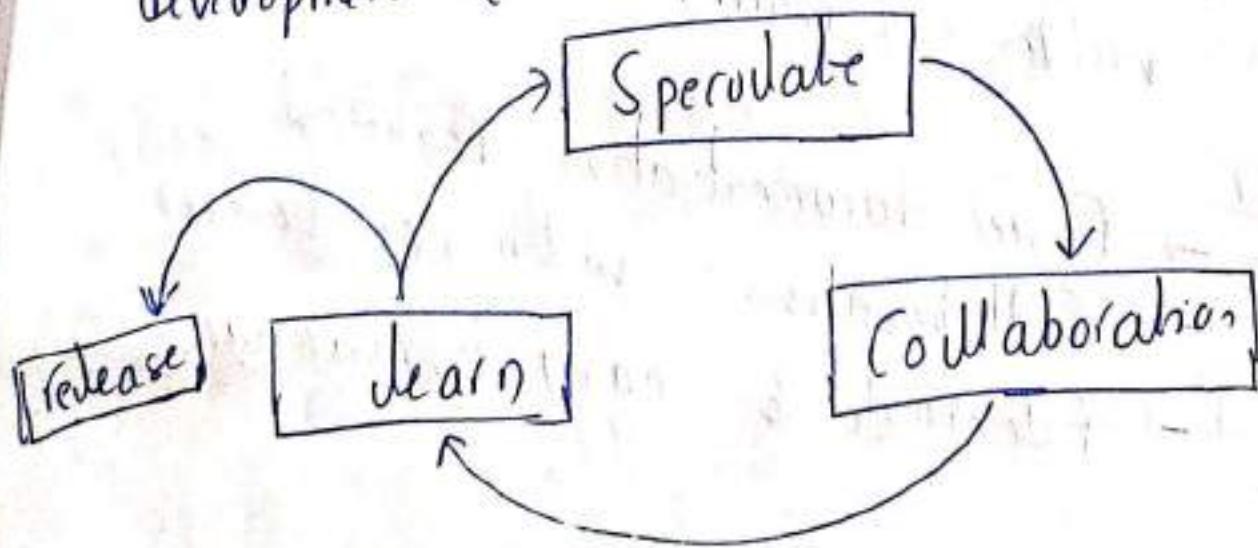
- Fewer documentation required
- Collaboration with customer
- Flexible & easy to manage.

disadv:

- depends heavily on customer interaction
- Transfer of project to new team member is a bit tough as no proper perfect documentation is done.

b) Adaptive Software Development (ASD)

- Adaptive software development (ASD) is a new software development methodology that focuses on rapid creation, application, & evolution of software system.
- It is derived from rapid application development (RAD).



→ Speculation:

→ During this phase, the project is initiated & adaptive cycle planning is conducted.

→ All the requirements are gathered & planning done (like software increments)

→ Collaboration:

→ In this phase, motivated people work together in a way that mobilize their talent & creative o/p.

→ Learning:

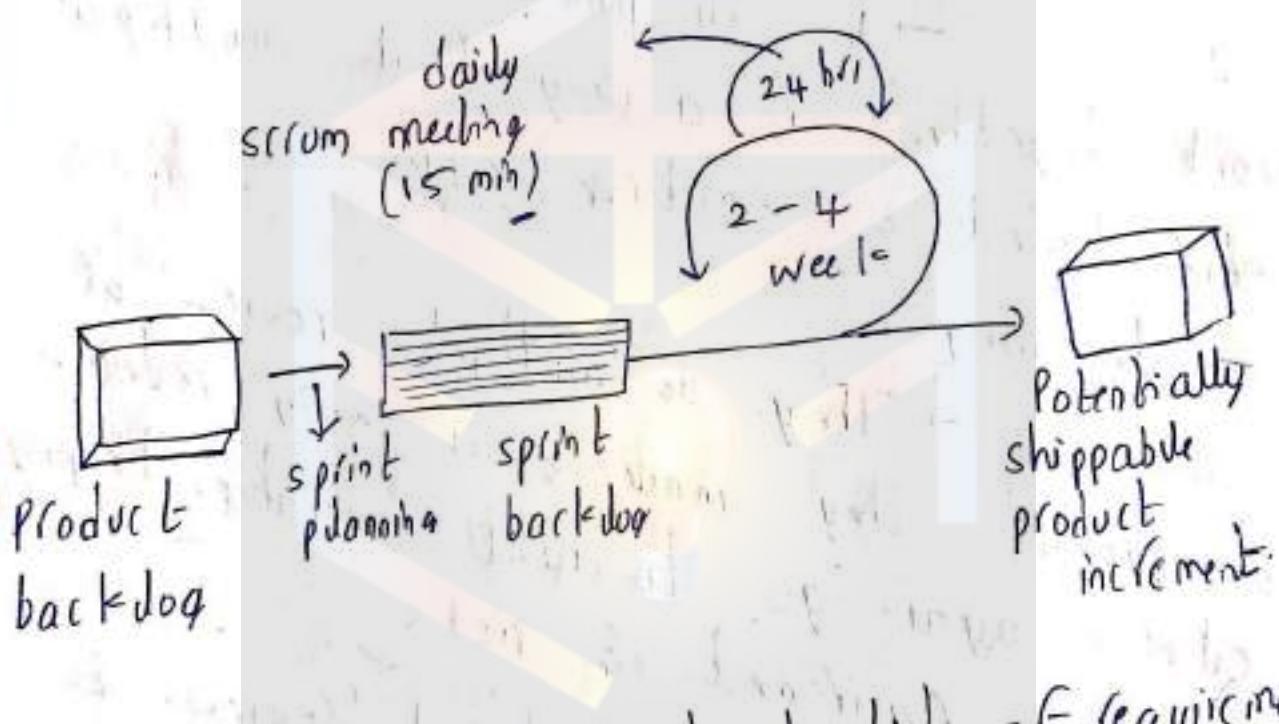
→ They do multiple review of the product they made & if ready review again go to next iteration keeping in the requirements & risks.

→ Formal ^{commonly} technical reviews & post-mortems are done.

c) Scrum Process Model :-

→ Scrum is an iterative, incremental process for developing any product or managing work.

→ Scrum is an agile process to manage & control development work.



Product backlog :- A prioritized list of requirements given by the client/customer.

Sprint planning :- Scrum team (who are working development) select few of the

requirement which can be completed in a limited time of 2 to 4 weeks (30 days) (time bound).

sprint backlog: consist of the activities which team do in the time bound.

Daily scrum meeting: meeting of the current progress & other discussions are done (15 min).

Potentially shippable product increment-1: A final product for that 4 week scrum → on which scrum review is done. (to generate sprint retrospective with all +ve & -ve's of the current sprint).

Unit - II

Software engineering principle & requirement engineering

→ Software engineering principle / practice:

→ The essence of practice:

→ George Polya has presented the concept "Essence of Practice" 4 steps

i) To understand the problem (Communication & Analysis)

The goal is achieved by the basic activities "Communication with customer" & "requirement analysis".

→ requirement analysis is a communication intensive activity. If there is misinterpretation of customer requirements then obviously the requirement analysis will be wrong.

→ Hence wrong design will be formed & finally implementation will be wrong. So customer will not get his expected results.

→ Hence with "customer communication" data, functions, & behavior of the system to be implemented must be stated clearly during requirement analysis activity.

→ Hence it's mandatory to understand problem first.

ii) To plan the solution (Modeling & Software Design)

This goal is achieved by activities, modeling & software design.

→ First checked whether the problem type problems are previously solved or are there any modules which can be used for some functionality. (These components are called reusable components).

→ if there are any large modules they are divided into subsystems / submodules for a simple design & a good practice

iii) To carry out the Pldao (Code generation)

→ The goal is achieved by coding.
→ The design is translated into code by using appropriate programming language.

→ All design details must reflect into coding (actual coding).

iv) To Examine the Result for Accuracy

(Testing & Quality Assurance)

The goal is to achieve "Testing & Assurance".

→ To check for data, functions and behaviour of the system
→ To check whether system is maintaining good quality.

→ Core principle / SE Principles:-

"David Hooker's ^{"proposed"} Seven Principles called core principles"

Principle 1 (The reason it all exists) :- The complete software is one which satisfies all the requirements by customer's point of view (along with client's).

→ Hence before beginning a software project, be sure that the software project has a business purpose.

Principle 2 (Keep Simple but Perfect) :- The software design must be as simple as possible.

→ If there are too many interfaces in the system, it's difficult to construct & implement such type of system.

→ Hence design should be simple (but simple doesn't mean skipping requirements).

Principle 3 (Maintain the Vision) :- A clear vision is required for the success of the project. (should know what you are doing).

Principle 4 (What you produce, Other will consume)

Other members in the team may use the software which is analyzed, designed & constructed by one person in a team.

→ After deployment based on the customer feedback improvements will also be made (maintenance).

Principle 5 (Be Open to the Future) :-

The solution for any problem must be generalized, & not specific.
→ As generalized solution can be reused for other applications too.
(should be ready for any change).

Principle 6 (Plan Ahead For Use) :-

- Reusability is a property of good quality software.
- Reusable software can be used in development of other application.
- Using object-oriented programming languages like C++ are popularly used.

Principle 7 (Think For Better solution)

- When we think about something IF we are having solution & we think for better solution (as better/best ideas will not come in the 1st discussion).

→ Communication principles/Practices :-

- Requirement analysis is communication intensive activity.
- If there is misinterpretation in requirement analysis, the wrong design will be implemented. hence coding will be wrong & customer requirements can't be validated with expected result.
hence following these principle will help.

Principle 1 (Listen Carefully): To understand customers requirements, be a good listener. Concentrate on speaker's words so that there should be ~~any~~^{no} chance of any misinterpretation.

Principle 2 (Prepare before your communication)

Prepare the agenda for the meeting i.e., the points to be discussed in the meeting.

→ Hence prepare for all issues for the meeting to understand the problem statement in detailed

Principle 3 (Be Prepared to Facilitate the meeting)

→ The person, a leader of meeting is called Facilitator.

→ He is responsible to conduct the meeting. (Be prepared to ask your question in it).

Principle 4 (Prefer Face to face communication)

→ explaining anything in a video conference or using ppt is a bit tough than using a pen paper approach
→ similarly face to face interaction is better than video conference of any.

Principle Number 5 (Keep Notes & Proper Documentation of meeting)

→ write down all important points & issues raised in the meeting.
→ keep one person as a recorder. & he takes down (writes) all the points ~~that~~ discussion all through

Principle Number 6 (Try for Collaboration)

→ collaboration in team members is essential. as many ideas can be generated & (if many people/developer contribute a small part) the project progress will be faster.

Principle 7 (keep the Discussion focused & modularize)

→ when many people come together, there is a possibility that participants of meeting will switch from point to point.

→ It is duty of recorder to make conversation focused on important topics under discussion.

Principle 8 (if something is Unclear, Draw the Picture)

→ Verbal communication goes not so far.
→ A proper sketch or drawing can often provide clarity when words fail to do the job.

Principle 9 (Keep the Discussion to "Move on")

→ In the meeting many people require to discuss different issues.

~~There may be~~ s

→ Hence never discuss too long on a single (for getting more clarity).

Principle 10 (Negotiation is successful when both parties Agree)

→ Customer & software engineer must negotiate the functions, priorities & delivery dates.

→ Planning, Practices / Principles:

→ No idea can be implemented without plan. Only good planning leads to successful result.

→ It includes cost estimation, timeline of project development & risk analysis. Hence proper planning is must by follow the below principles

Principle 1 (Plan Understand the Scope of project)

→ Software planning provides the exact goal for software development team.
→ Having perfect planning then the process will be much smoother.

Principle 2 (Involve the customer in the planning activity)

→ Software planning includes complete estimate (ex: cost estimation of project), scheduling (complete timeline chart for project development) & risk analysis.
→ Hence participation of customer is mandatory for cost & deadline & other goals.

Principle 3 (Recognize that planning is iterative).

- A project will never have a constant requirement
- As work begins, it is very likely that things will change. As a consequence, the plan must be adjusted to accommodate the changes.
- Hence, using iterative / incremental process is better idea.

Principle 4 (Estimate based on what you know)

- The main goal of estimation is to provide an indication of effort, cost & basic duration based on the team's current understanding of the work to be done.

Principle 5 (Consider risk as you define the plan)

If the team has defined risks that have high impact & high probability, contingency/optional planning is necessary.

Principle 6 (Be realistic)

→ The schedule or the plan designed for the project should be realistic

→ as any one cannot work for 100% or for a full day hence give some additional days & realistically.

Principle 7 (Adjust granularity as you define the plan)

→ Granularity refers to the level of details that is introduced as a project is developed.

→ Adjust the plan depth based on the team (done over compilations).

Principle 8 (Define how you intend to ensure quality)

→ multiple formal reviews are conducted to assure the quality

→ Discussion are done in meetings etc...

Principle 9 (Plan will be absorb the changes)

→ Even the best planning can be obviated by uncontrollable change.

→ The software team should identify how changes are to be accommodated as software engineering work proceeds, as software engineering make adjustments.

Principle 10 (Track planning & make adjustments)

→ Access the progress of project development on daily basis.

→ If its found that plan is

lagging take actions to recover it.

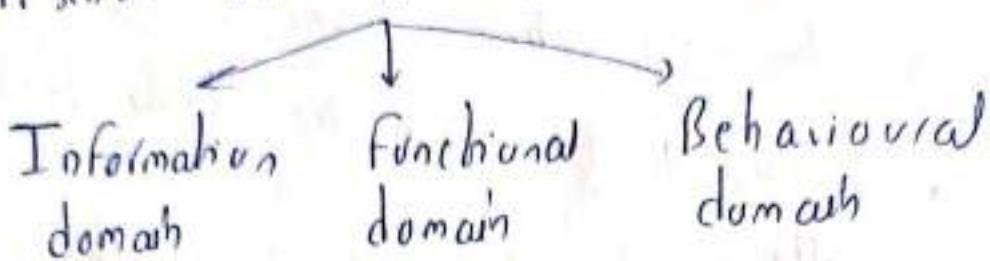
→ Boehm's seven principles for realistic
to develop realistic plan

Boehm has presented five
W-type & two H-type questions
For effective project planning

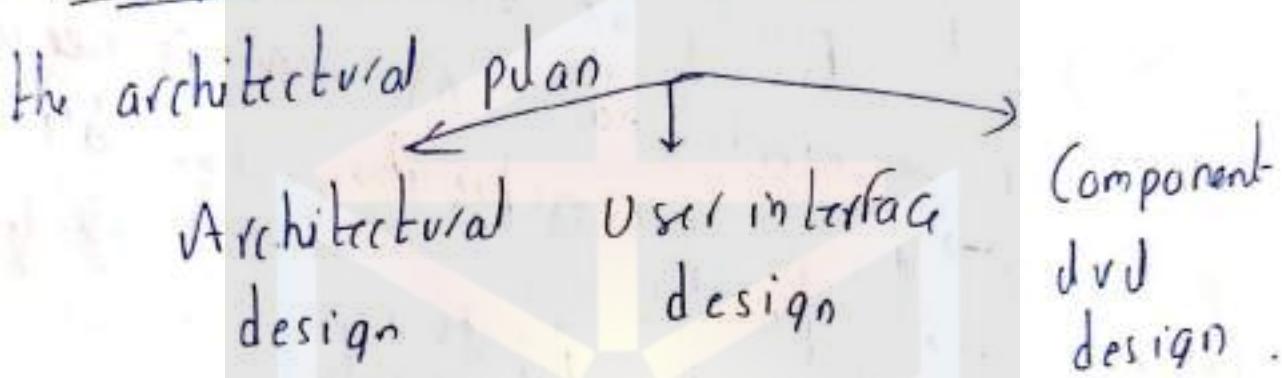
- 1) Why is the system being developed?
→ Focus on the basic aim.
- 2) What will be done?
→ Focus on the functionality of system
- 3) When will it be accomplished?
→ Focus on the time schedule
- 4) Who is responsible for a function?
→ the role & responsibility of each member of software team must be defined

- 5) Where they are organizationally located,
→ focus on role & responsibility
of each team member user & customer!
 - 6) how will the job be done technically
& materially?
→ focus on product technically
& management point of view
 - 7) how much each resource is needed
→ focus on estimation for all
above six points.
- Modeling Practices / Principles!
Modeling helps to understand
the actual entity to be built.
includes 2 models
- i) Analysis model
 - ii) Design model.

i) Analysis model :- Analysis model represents the customer requirements



ii) Design model : It is equivalent to



→ Analysis modeling Principles :-

Principle 1 (The information Domain of problem must be clearly represented)

Here we use data flow diagram (DFD) to show such information domain.

input flow into the system, output from the system & Data collection by data stores to collect data in the system.

Principle 2 (The functions of the software must be defined clearly)

→ Functions are the processes those transform the input flow to the output flow.

→ as the next step is coding here the i/p & o/p ^{& function} should be mentioned properly.

Principle 3 (Behaviour of System must be Defined clearly)

→ here state transition diagram to represent the behavior of the system clearly. (Transition from one state to another is mentioned here).

Principle 4 (The clear hierarchy among information, function & behavior must be shown)

→ The proper hierarchy of analysis model leads to easy design. (hence this representation should be perfect layer by layer).

Principle 5 (Analysis should be clear enough to convert into Design model)

→ The analysis model is converted into design model in next step of modeling.

→ Hence if analysis of requirements is clear & simple then it will be easy for design.

→ Design Modeling Principles

Principle 1 (Design should be tractable for analysis model)

→ Using elements of analysis model, design model is constructed.

→ If previous step was done perfectly

if this step is done perfectly
this step is done clearly coding
can be written efficiently.

Principle 2 (Consider the architecture of the system to be built)

→ architecture styles

i) Data centered "

ii) Data flow "

iii) Main / Sub Program "

Principle 3 (Design of Data is as important as Design of Function)

→ Design of data is important

factor.

→ The data design shows the relationship b/w different data objects.

→ The design model includes "entity relationship diagram" (ER diag) for relationship b/w data objects.

→ Thus data modeling is as important as "Functional modeling".

Principle 4 (Internal as well as external interfaces must be designed)

→ In a given system, data may flow from one component to another component.

→ Also may flow from external environment to the system.

→ These internal & external interface must be designed properly.

Principle 5 (User Interface Design must satisfy all needs of end user)

→ The design must be as per the ~~end user~~ convenience of the end user of the system.

→ hence user interface should be as simple as possible.

Principle 6 (Component level design should be functionally independent)

→ functions should be designed such that all tasks of single software component (i.e. modules) should be related.

Principle 7 (Components should be loosely coupled to one another & to the external environment)

→ For good design, the interaction b/w components should be minimum. (it is called coupling).

→ Hence for good design the coupling should be low.

Principle 8 (Design module should be easy to understand)

→ The design is implemented using appropriate programming language.

→ Hence the design should be clear & as simple as possible

Principle 9 (Accept that design modeling is iterative)

→ In case of incremental model of software process, customer may suggest modifications after each delivery of the increment.

→ According to that design model is required to change.

→ Construction Principles:

- Coding: Designed details are implemented using appropriate programming language.
- Using suitable programming language to generate the source code e.g. C, COBOL, Visual Basic etc.,
 - Using automated tools like Microsoft Front page where code is automatically generated
 - Using 4th generation programming language like VC++ to generate executable code.

Testing:- To check whether flow of coding is correct

- To check out the errors of program
- To check whether program is giving expected output as per input specifications.

include 4 testing steps.

i) Unit Testing :- This test focuses on units as individual. each module of project is checked out individually for expected o/p.

ii) Integration Testing :- This test focuses on design. The units are integrated (combined) as per design specifications.

iii) Validation Testing :- The test focuses on requirements. In this test the out coming results are checked with customer's actual requirements.

iv) Acceptance Testing :- When software is completed , the series of testing are conducted to enable to validate his requirement.

→ Coding principles:

Preparation principles: before starting the code.

→ Understand the exact problem for which you are trying to solve.

→ Understand the basic design & the concepts for the same.

→ Select appropriate programming language to implement the solution

→ Select programming environment that will be suitable for writing the code.

→ Create set of unit tests to be applied after completion of unit code.

Actual Coding Principles: where we write code.

→ algorithm must follow structured programming principles

→ Use suitable data structures those will fit for coding

- Keep minimum nested conditions & loops.
- Variable names should be meaningful & as per standards.
- Code must be self documented (with comments).

Validation Principles after completing first code pass -

- Conduct unit test
- Refactor the code if necessary
↓
Optimize.

→ Testing Principles :-

Principle 1 (Test must be conducted to validate customer requirement)

- The basic aim of testing is to find defects from the developed modules
- based on the customer's IP
- Hence testing must validate customer's requirement

Principle 2 (Tests should be well planned before starting Testing work)

Testing plan can be started as soon as analysis model is completed. (testcases can be planned before starting actual coding).

Principle 3 (The Pareto Principle is applicable to software Testing)

According to this principle 80% of all errors found in testing will likely be traceable to 20% of all programming modules. → hence, these suspected components must be isolated & tested.

Principle 4 (Testing should start from individual Unit & finally go towards testing of complete system as whole).

Principle 5 (Accept Testing of every combination of path is not possible)

→ as checking each combination is not possible (like testing for each & every input possible).

→ Deployment Principles

This stage of software development includes software delivery, support & feedback of customer.

Principle 1 (Manage Customer Expectations)

→ It always happens that customer wants more than he has stated earlier as his requirements.

→ It may be a cause for his disappointment, even the software is as per requirements.

→ Hence at the time of software delivery, developer must have skills to manage the customer expectations.

Principle 2 (Assemble & Test Complete Delivery Package)

→ not only software is delivered we include installation procedure, manuals, system requirements etc..

Principle 3 (Record - keeping mechanism must be established for customer support)

→ customer support is most important & to be recorded hence support should be based on the satisfaction the future is designed

Principle 4 (Provide essential instructions, Documentation & manuals).

Principle 5 (Don't Deliver any defective/buggy software)

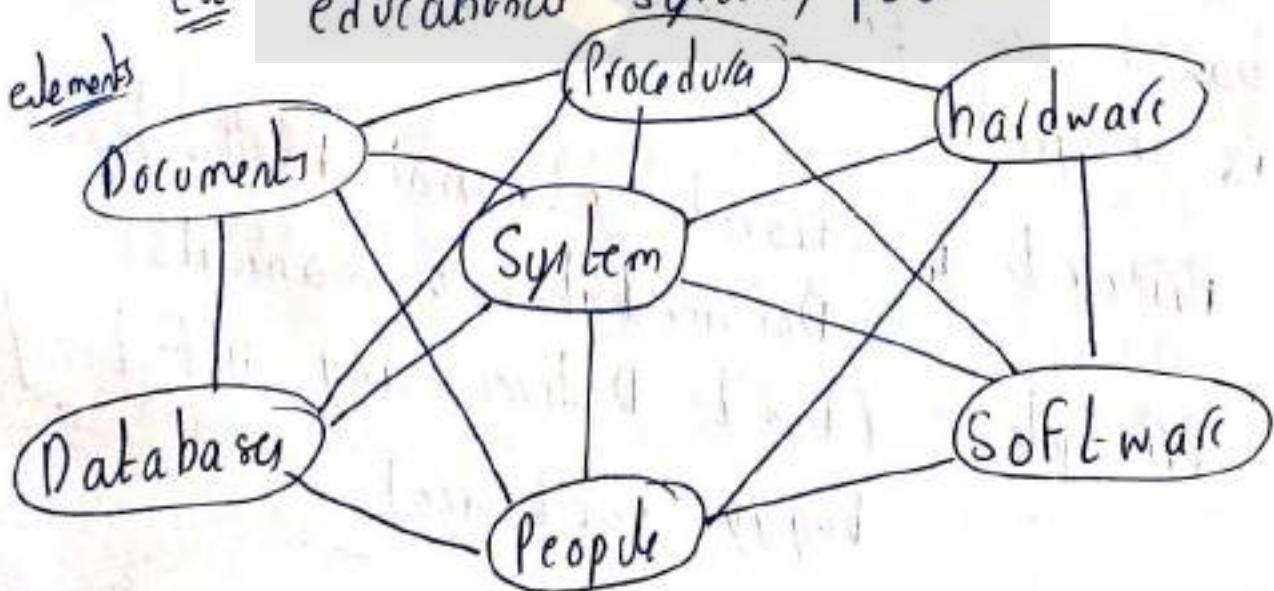
→ System Engineering :-

It is an interdisciplinary field of engineering & engineering management that focuses on how design, integrate & manage complex system over their life cycle.

(include hardware, software, database)

→ Computer based System :-

A set of arrangement of elements that are organized to accomplish some predefined goal by processing information
e.g. educational system, political system.



software: it is a collection of programs, data structure & documents

hardware: hardware components & devices communicate with software
e.g. sensors, pumps etc..

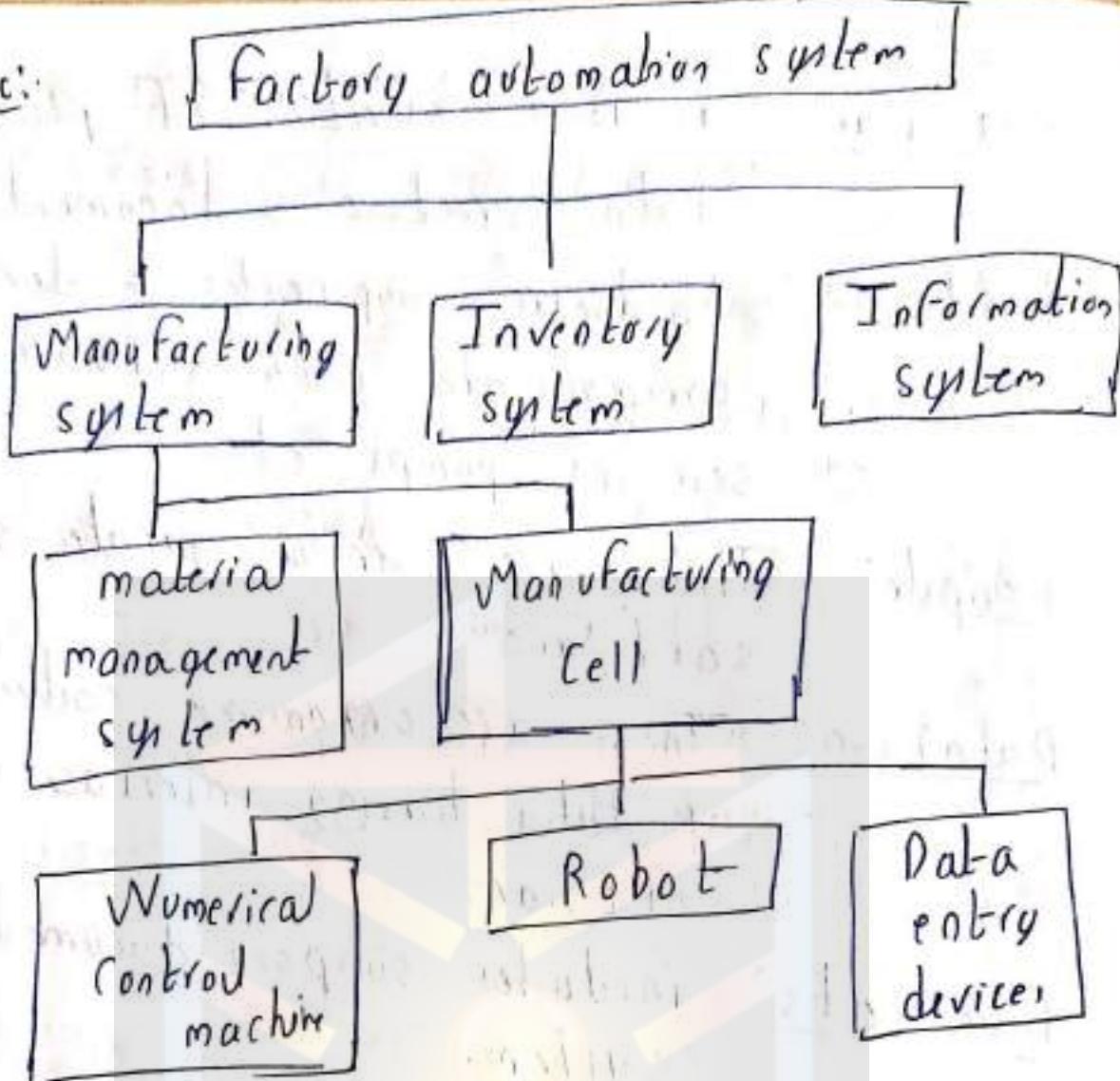
people: These are user & operator of software

Databases: These are organized collection of data having interface with software.

Documents: includes support document of system.

Procedures: The steps those define the use of each element of the system is called procedure.

e.g:-



→ macro element is a computer based system that is one part of large system

→ System engineering hierarchy :-

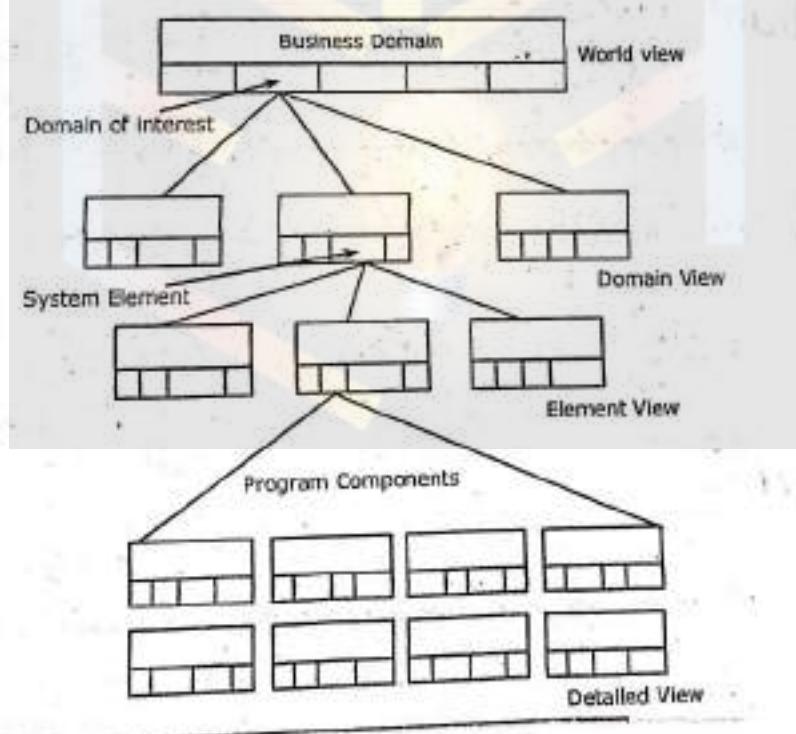
→ System engineering process includes either top down approach or bottom up approach.

→ In top down approach the entire application called as world view

→ The main application is then refined to different domain at Domain view.

→ The specific domain is then refined to different elements at element view.

→ Finally detailed view of each element is considered.

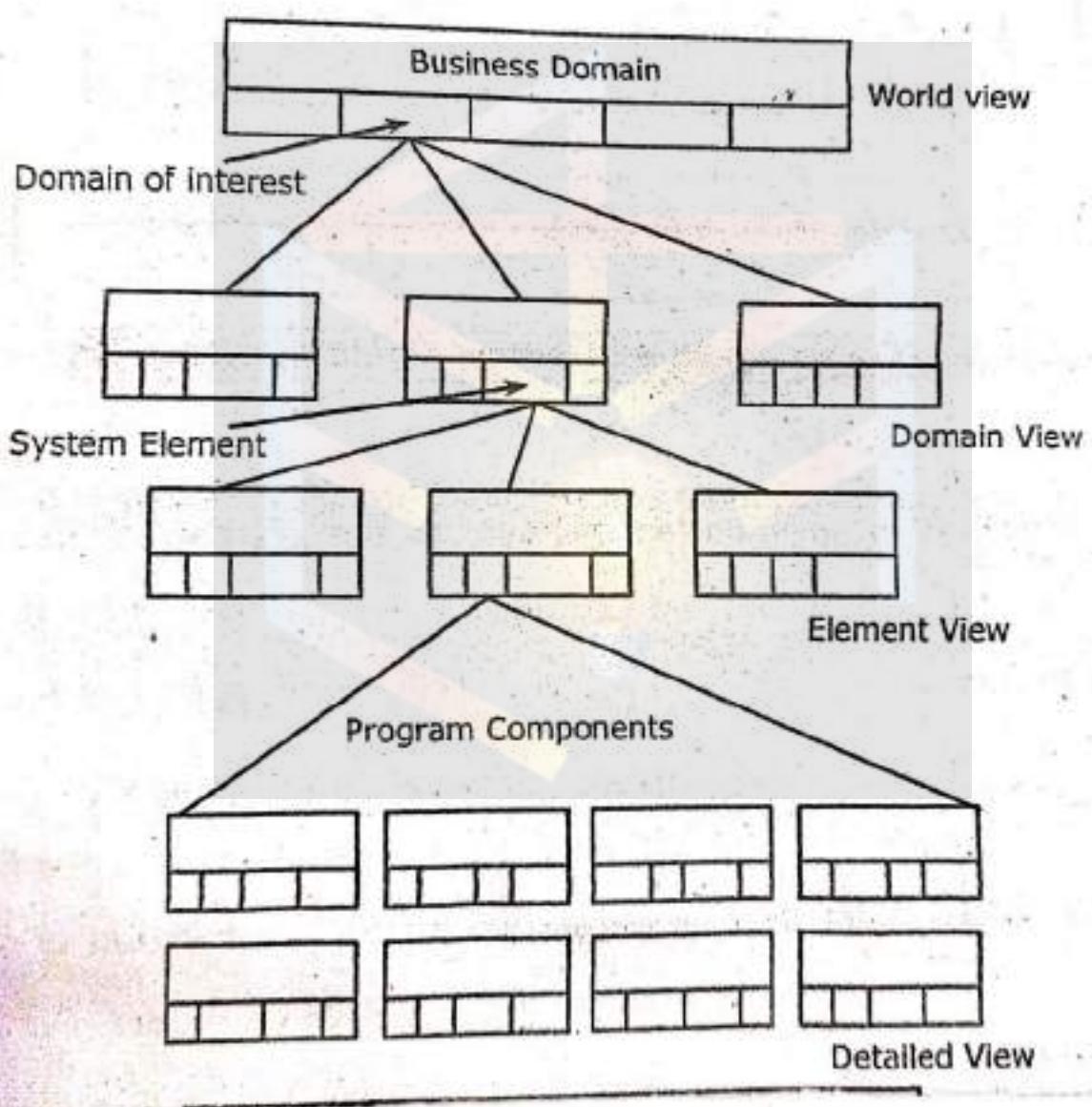


top
down

ations



The specific domain is the different elements at element level
→ Finally detailed view of element is considered.



→ The main application i.e. world view (WV) is represented as

$$WV = \{ D_1, D_2, D_3, \dots, D_n \}$$

where, each domain itself is a system or a subset

The domain view D_i is represented as

$$D_i = \{ E_1, E_2, E_3, \dots, E_n \}$$

where, each element may be software, database, hardware etc..

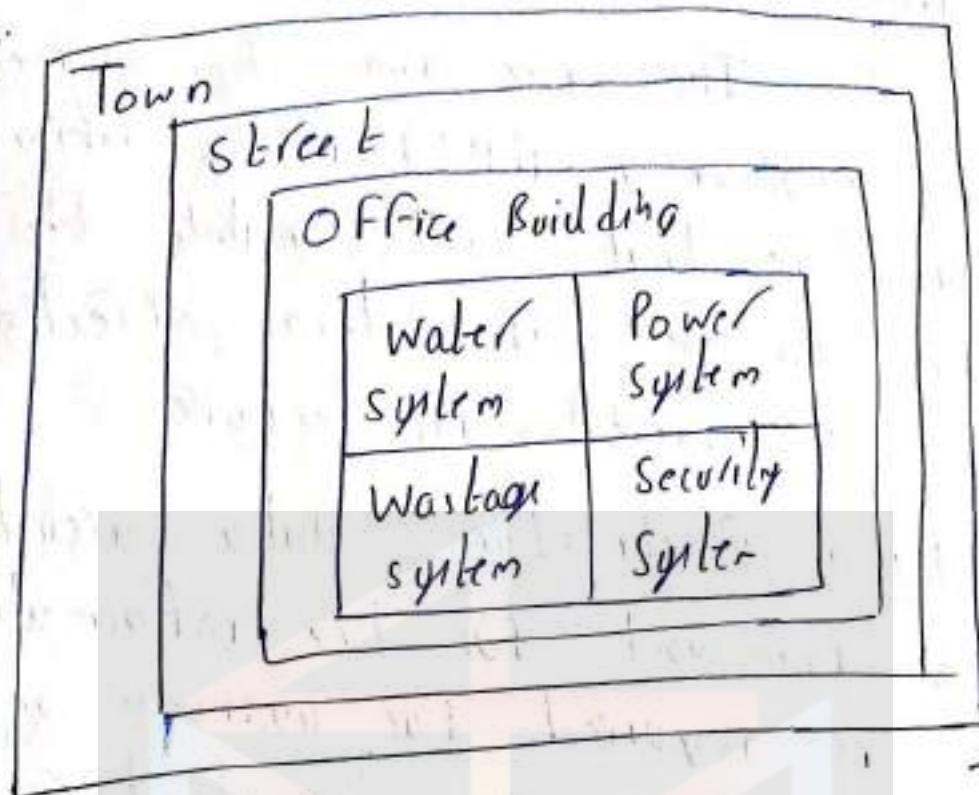
The element view E_i can be represented as

$$E_i = \{ C_1, C_2, C_3, \dots, C_k \}$$

where component may be the software module constituting the software element.

→ Thus, system engineer narrows the focus of work as he moves down

e.g:-



→ System modelling :-

System modelling is an important element of the system engineering process.

- Input to the model
- Output of the model.
- Output of the model
are considered.

→ System simulation:-

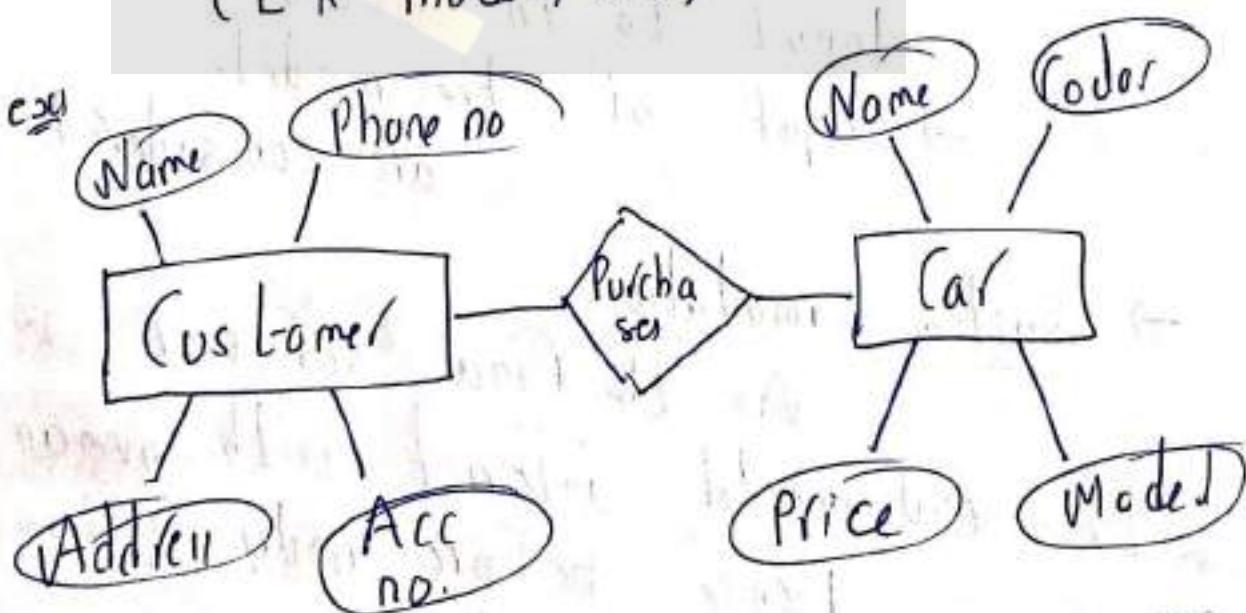
As the final software works with real world interact with human etc..
Hence before only simulation
practices should be done.

→ Business Process engineering:

The basic aim of business process engineering (BPE) is to define architecture that will enable the business to use information effectively.

3 different- architecture.

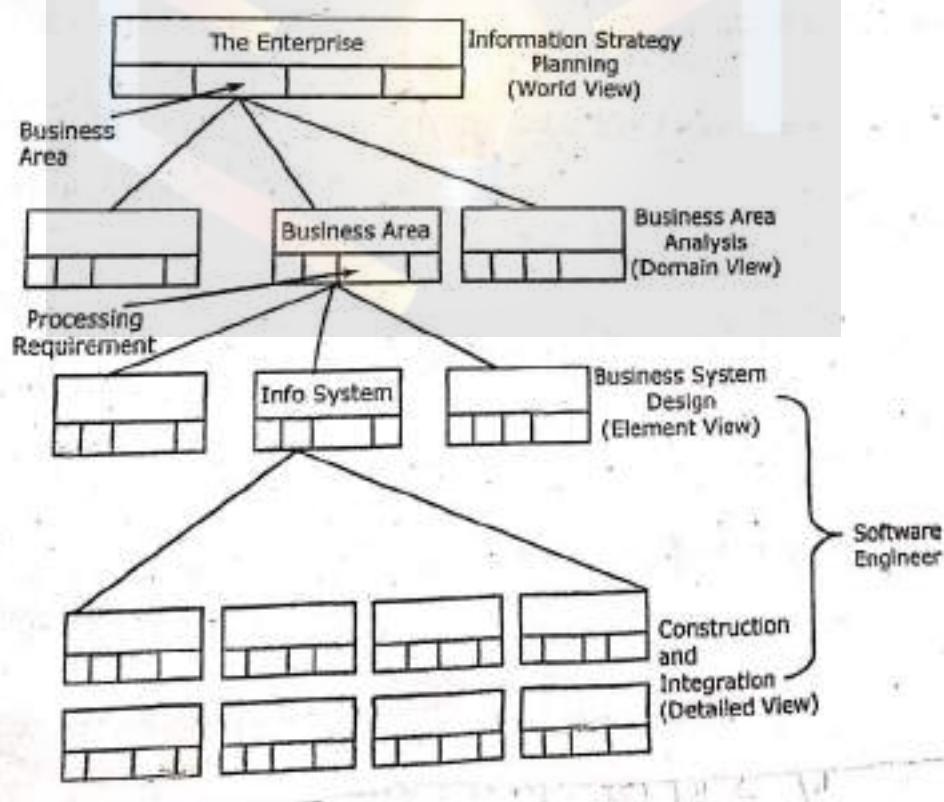
i) Data Architecture: Data architecture provides framework for the information which is required for business or business include objects each object has set of attributes
(ER model diagram)



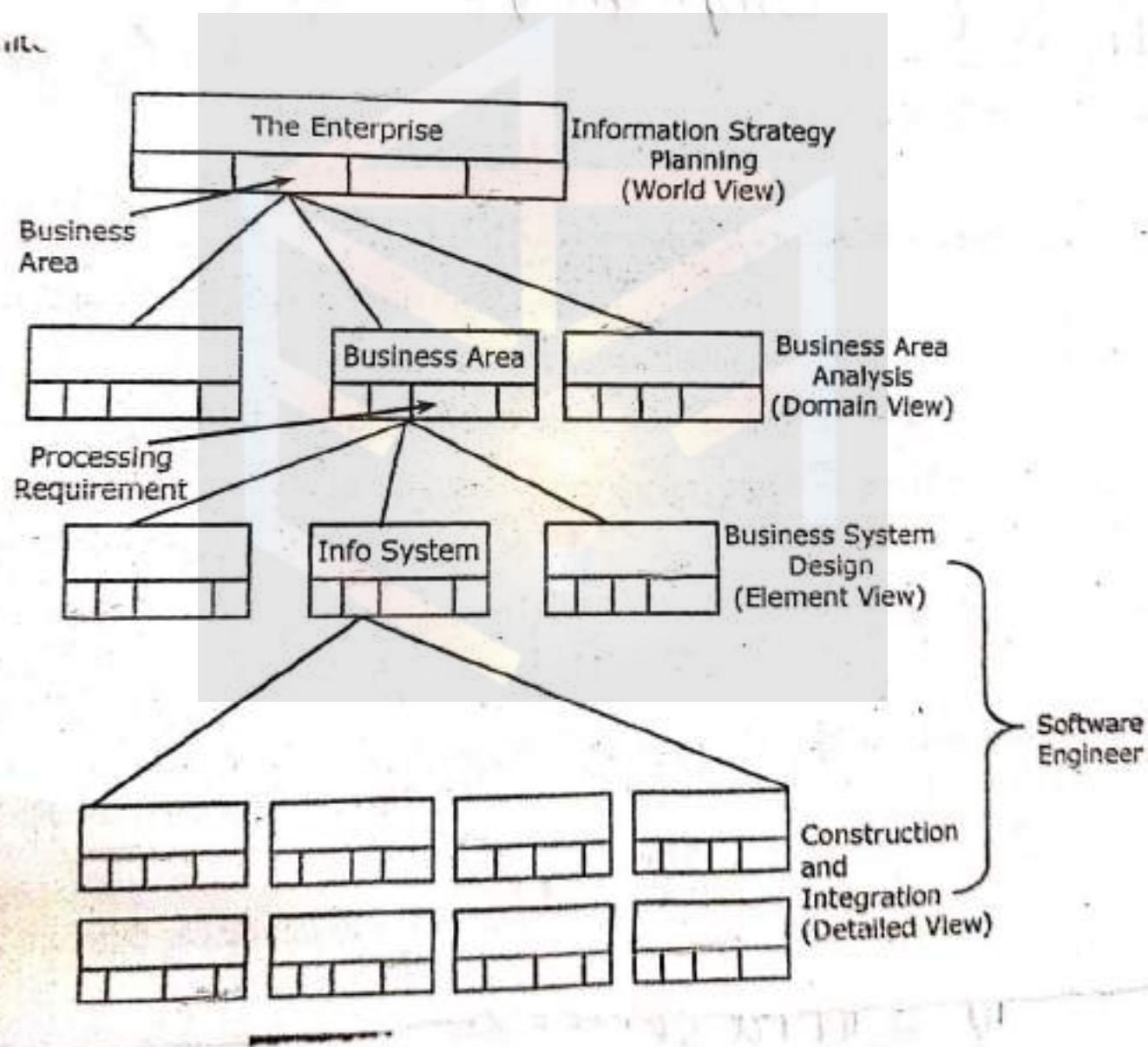
The data objects flow b/w business functions are organized within the database & are transformed to provide information for business.

iii) Application Architecture:

It includes those elements of system that transform objects within the data architecture for some business purpose.



It includes those elements system that transform objects with data architecture for some business purpose.



iii) Technology infrastructure: It provides the basis of data & application architecture.

→ In other words technology infrastructure provides supporting software & hardware for data & application.

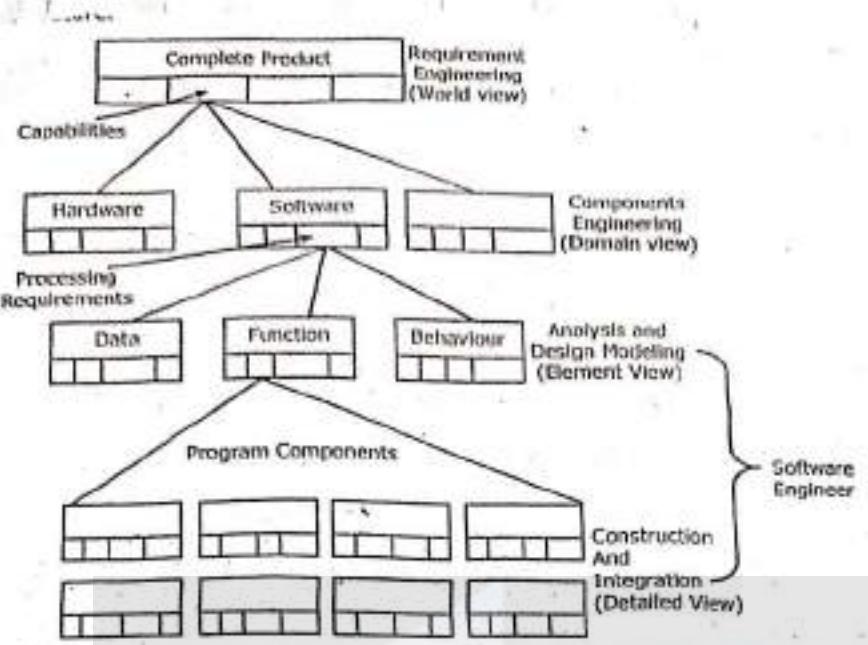
ex: computers, OS, client server model etc...

→ Product Engineering:

The basic aim of product engineering is to translate customer's desire regarding the product into actual working product.

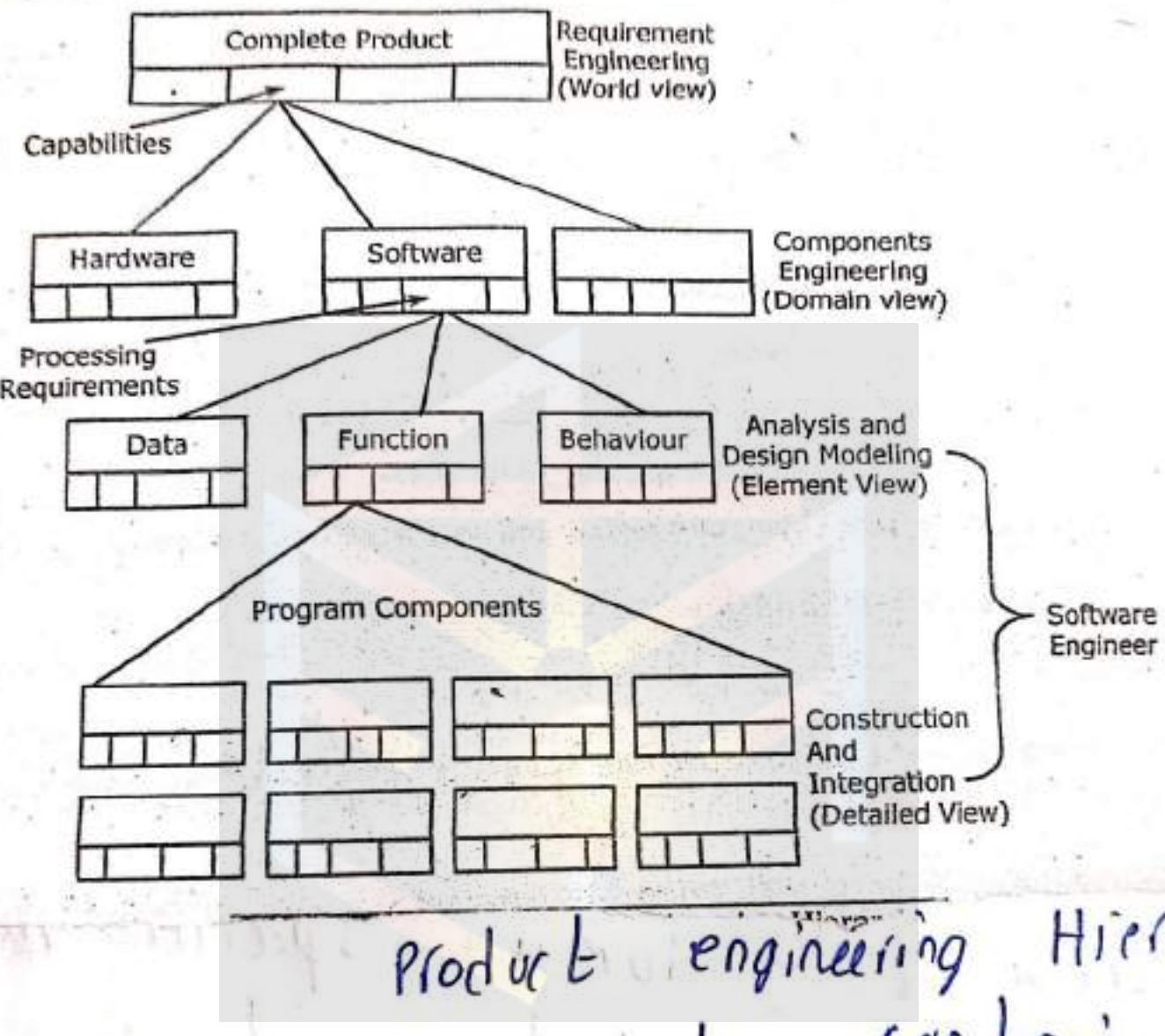
Hence to achieve this goal product engineering uses

- Architecture
- Infrastructure



Product engineering Hierarchy

- i) Architecture: mainly contains 4 different components (software, hardware, Data or database, people)
- ii) Infra structure:
 - it includes the above components with addition few components like (video, CD ROM, other documents) to support these components
 - first all requirements are collected from the customer. Once all requirements are clear, the system engineer allocates these requirements to particular component



Architecture: mainly contains 4 components (software, hardware, Data database, people)

Infra structure: all above com...

→ After allocating specific functions to components, all activities are considered individually

software engineering

hardware engineering

Database engineering

human engineering

→ All these run parallelly / concurrently each of them take domain specific view
The element view includes analysis & design modeling. program modules are then constructed & integrated.

→ System modeling:

system modeling helps in providing hierarchical structure.

→ It starts with top view i.e. "world view" showing complete system.

→ Then it is refined into different "Domain view" (collection of subsystems)

→ Then it is refined to different "element view" (like software, hardware, database, people).

→ Finally detailed view of each element is considered (different modules).

2 ways to model.

i) hatley Pishhai modeling:

→ Here a system modul template is used. It helps the analyst to create a hierarchy of details.

It has 5 regions

i) User interface region

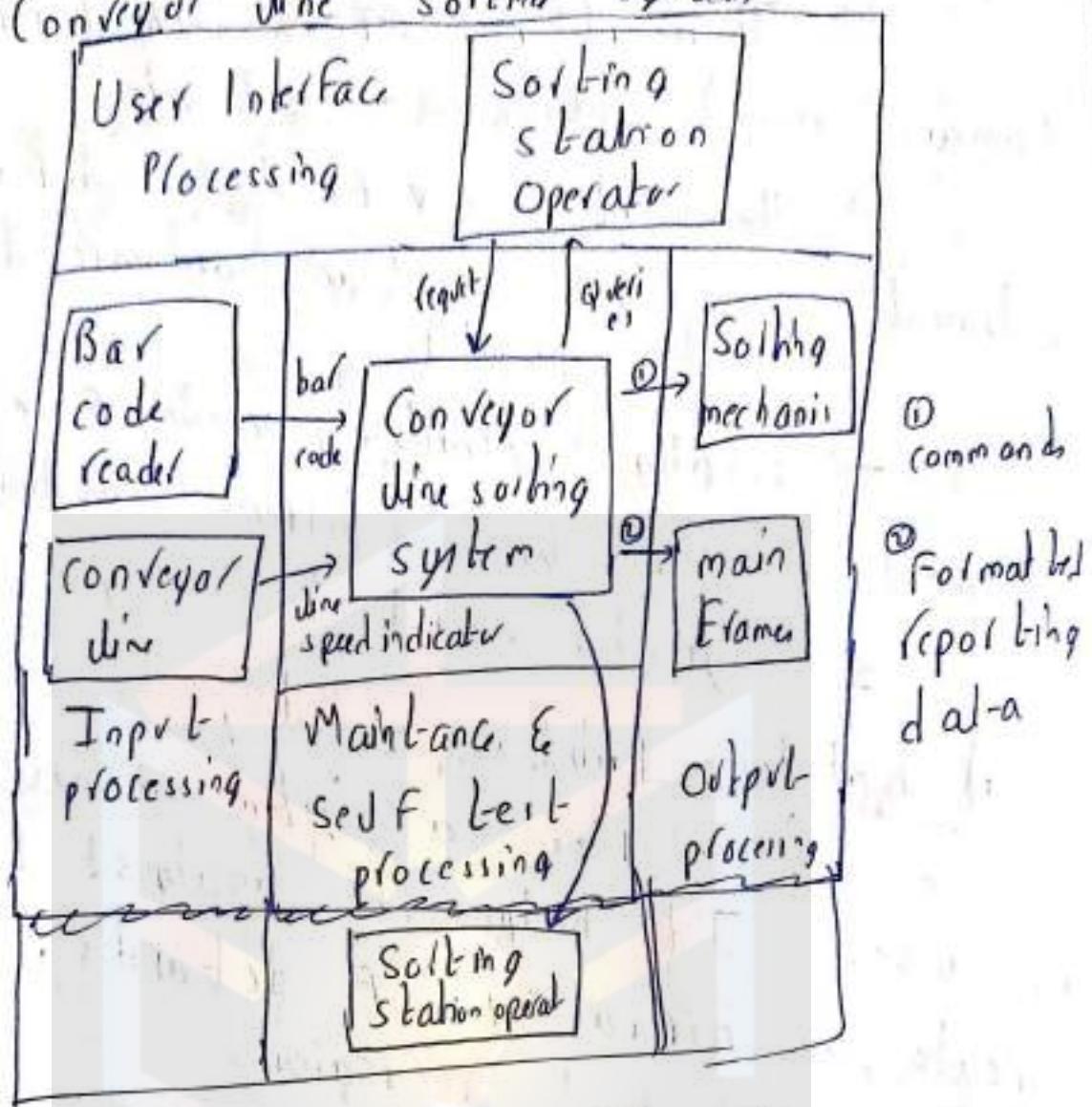
ii) Input processing region

iii) System function & control region

iv) Output region

v) Maintenance & self test region

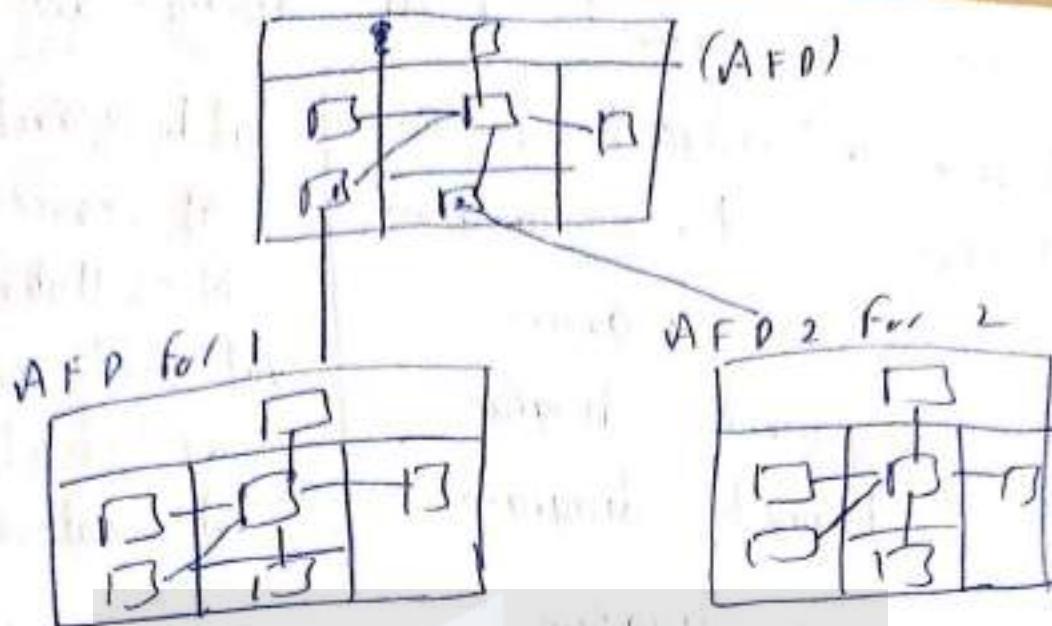
ex) Conveyor line sorting system



→ The above system context diagram

becomes top node of hierarchy of system flow diagram.

→ From the top/root major subsystems are identified & expanded into another (Architectural flow diagram) AFD



ii) System modeling with UML:

→ UML (Unified modeling language) provides the set of diagrams those can be used for analysis & design at system as well as software level.

→ UML standard language for writing software blueprints.
→ UML is a modeling language which builds on understanding of the system.

The different UML diagram are

- i) ~~use case~~ object diagram
- ii) ~~use case~~ class diagram ✓
- iii) Use case diagram ✓
- iv) Component diagram
- v) Deployment diagram
- vi) Interaction dig.
 - a) Sequence dig. ✓
 - b) Collaboration dig. ✓
- vii) State machine dig.
 - a) Activity dig. ✓
 - b) Statechart dig. ✓

I) Use case diagram:

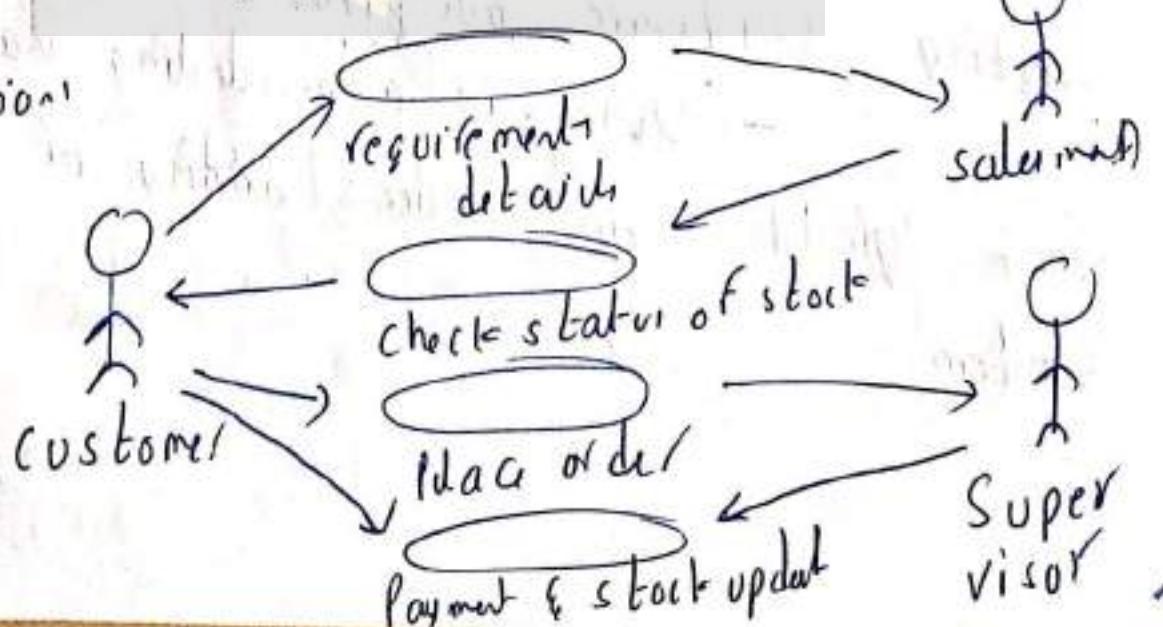
→ Use case diagram contains

actors & use cases

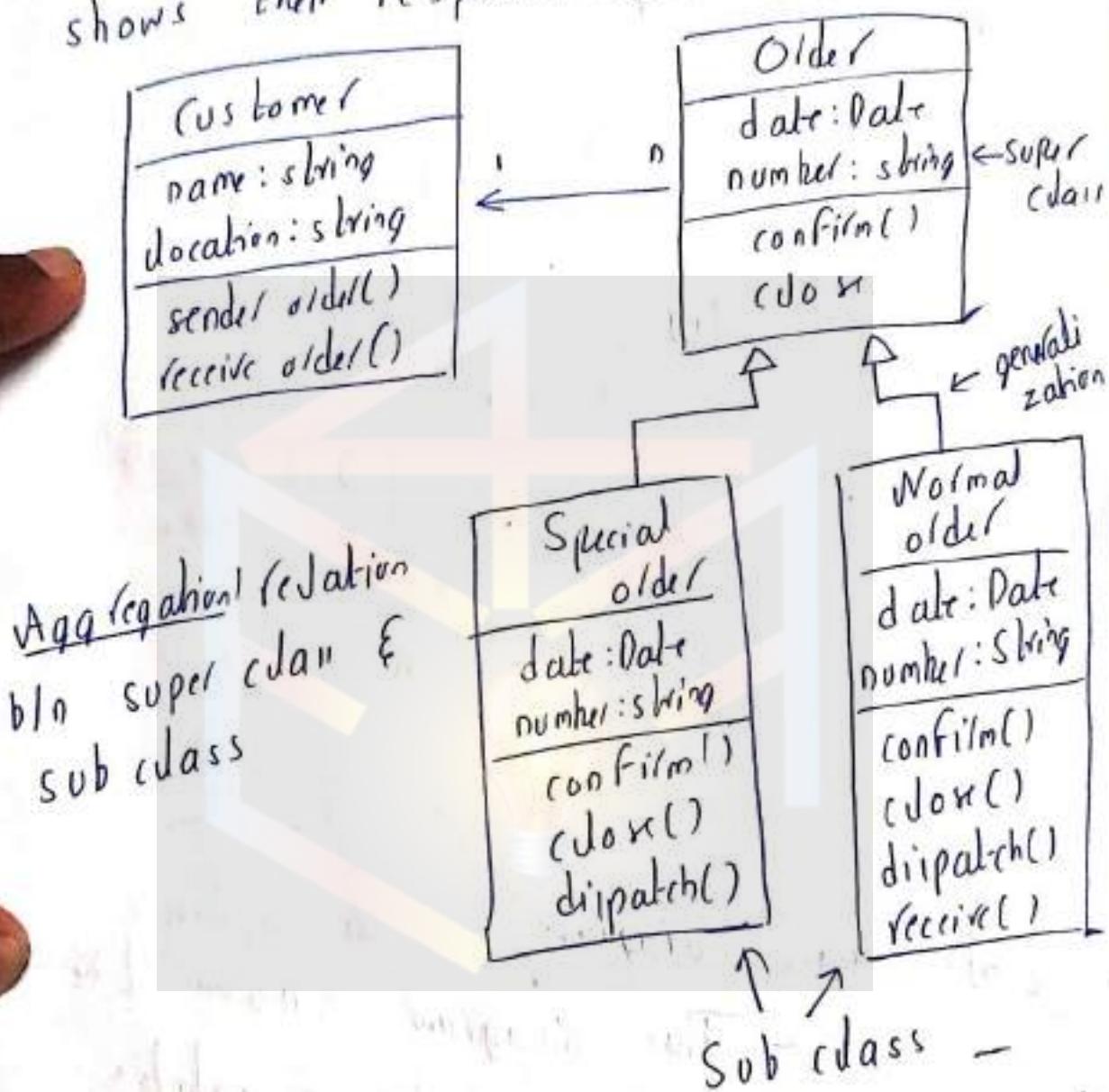
→ actors are entities that interact with the system. actors may be people or other machine or system those have interface to software.

→ use cases denotes the

function



iii) Java Diagram: Class Diagram include identification of candidate classes & it shows their responsibilities & collaborations.



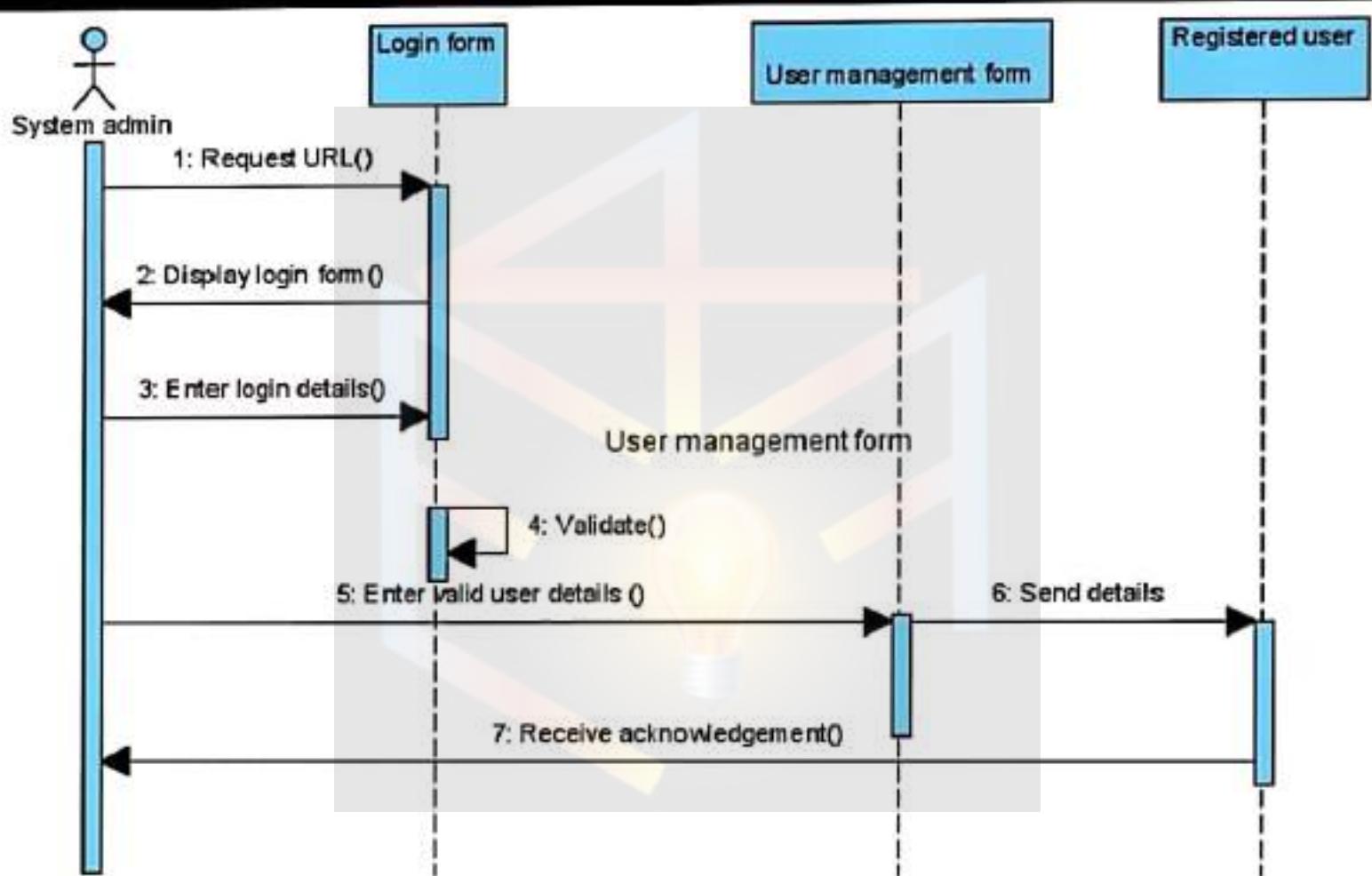
iii) Sequence Diagram: A sequence diagram is a kind of interaction diagram that focuses on sequential ordering of the message.

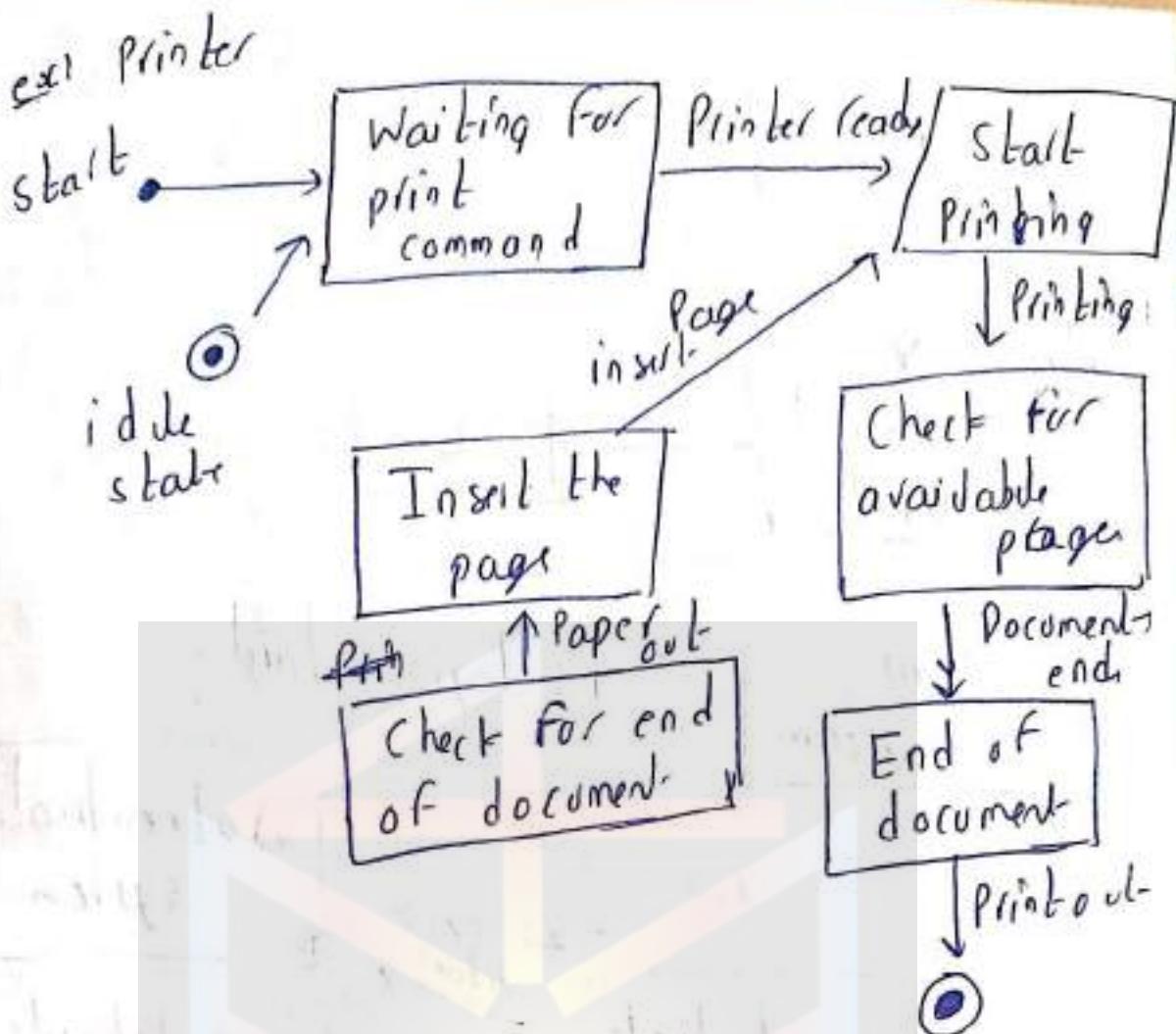
(cont)
It shows the sequence of steps during implementation of some procedure.



iv) state chart Diagram:

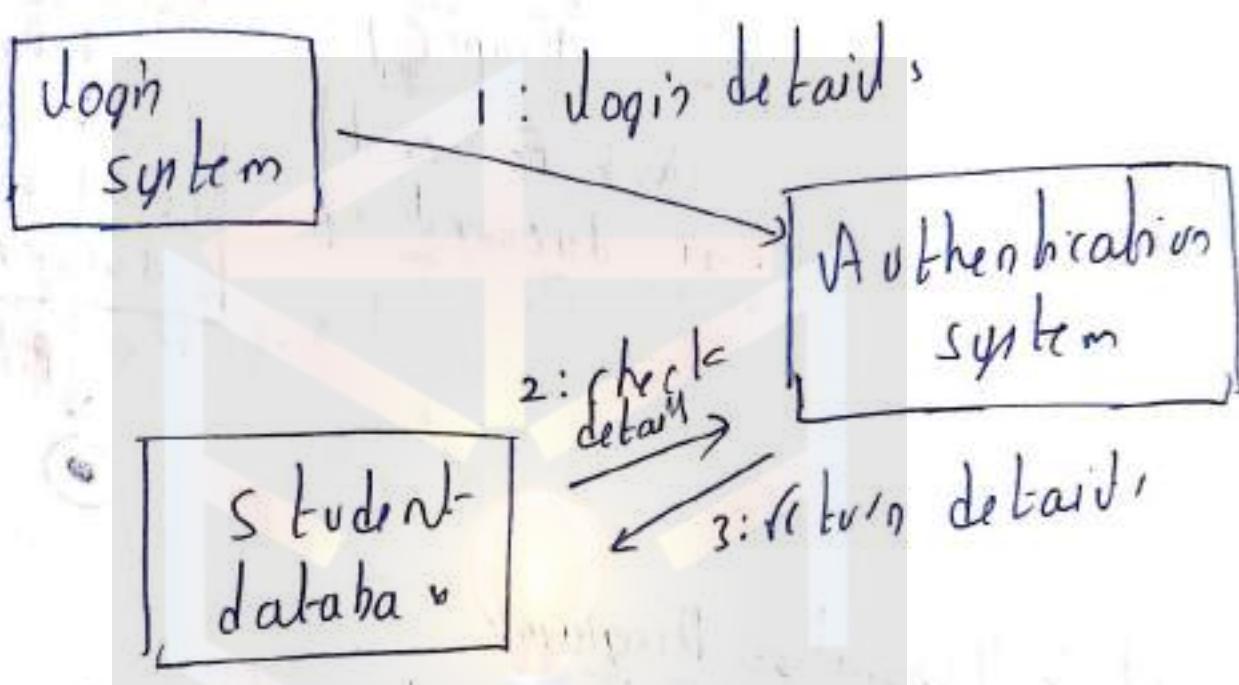
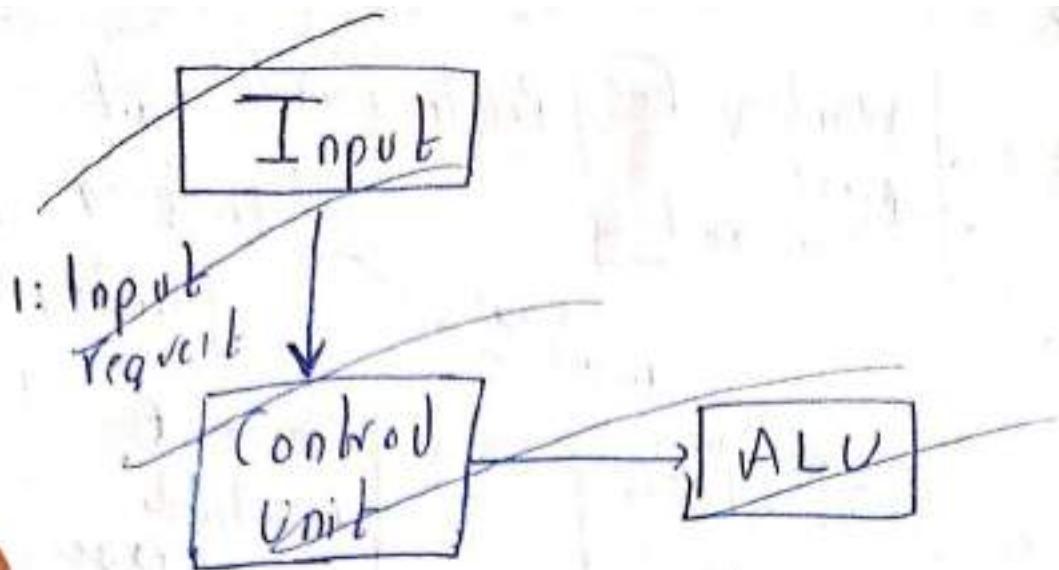
→ This diagram shows the states, transitions, events & exhibits like a flow chart but for different states.



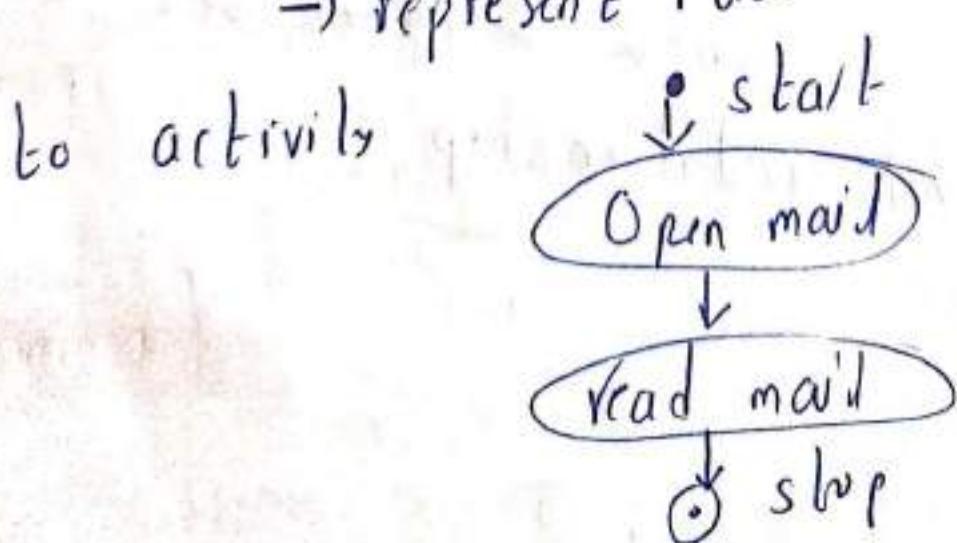


v) Collaboration Diagram:

Collaboration diagrams are kind of interaction diagrams. They show federation set of objects & their relationship.

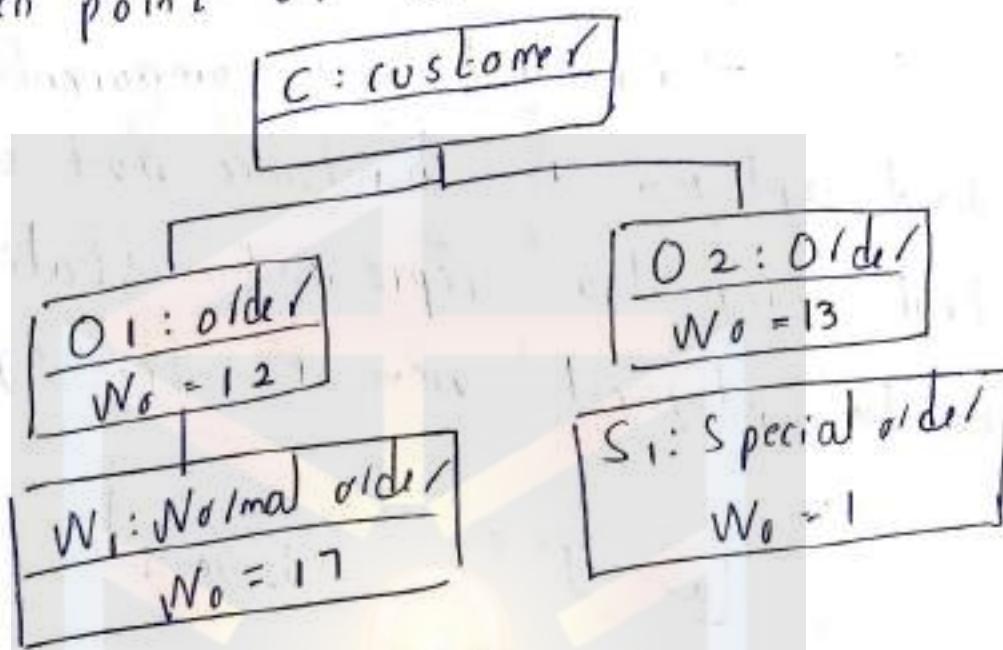


vi) Activity Diagram
 → represent flow for activity



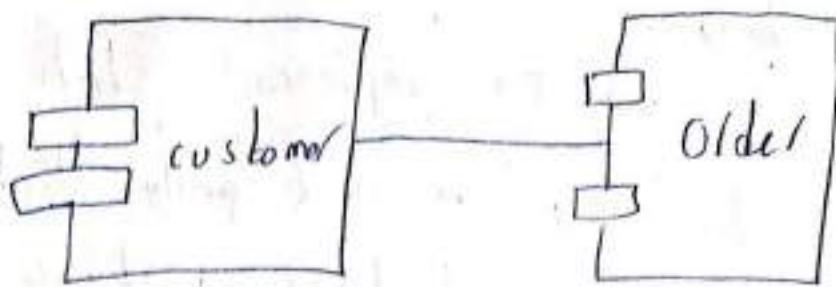
vii) Object diagram:

→ It represents static design view of the system. not only that they also have a snapshot of instance at any given point of time.



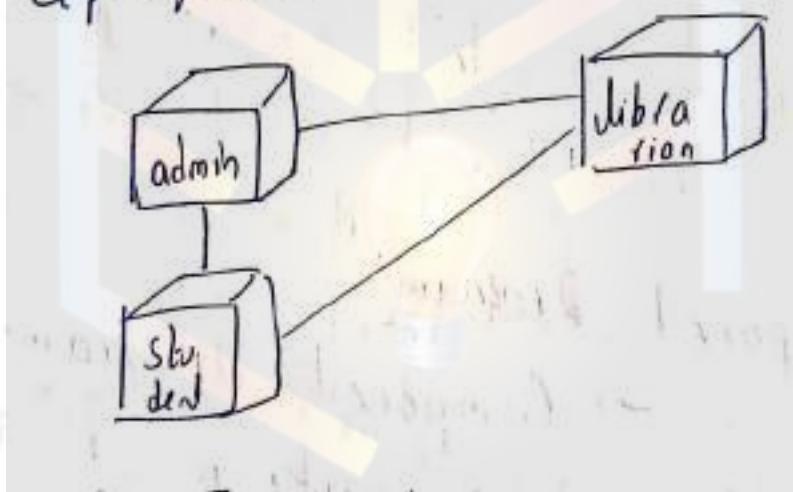
viii) Component Diagram:

→ Component Diagrams are used to describe working of system components → They represent static implementation view of the system.



viii) Deployment Diagram:

→ represent how components are deployed in hardware not only that they also represent static deployment view of the system.



→ Requirement Engineering:-

→ it was previously called requirement analysis

→ requirement engineering is the process of defining documenting &

maintaining the requirements

→ Requirement engineering process consists
of the following main activities

→ requirement elicitation

→ requirement specification

→ requirement verification & validation

→ requirement management

→ A bridge to design & construction:-

→ requirement engineering helps software
engineers to better understand the problem
that they are working to solve.

→ software engineer (system engineer or
analyst) & other project stakeholders
(managers, customer/client & users) all
participate in requirement engineering.

→ based on the requirements next
steps like planning & design is done.

→ & completely based on design

coding / Construction is done.

→ Hence design will 100% reflect
the construction part
→ similarly all steps are related
(requirement gathering → planning → design - construct
deployment ← testing)

→ Requirements engineering tasks:

The requirements engineering
is carried out through the execution of
seven functions.

- i) Inception
- ii) Elicitation
- iii) Elaboration
- iv) Negotiation
- v) Specification
- vi) Validation
- vii) Requirements management

i) Inception:

↳ means beginning.

- The requirement engineering itself is a "communication intensive" activity.
- The customer & developer meet & they decide the overall scope & nature of the problem statement.

The aim is

- To have the basic understanding of the problem.
- To know the people who will use the software.
- To know exact nature of problem, that is expected from customer's side.
- To have collaboration b/w customer & developer.
- The requirement engineering itself is a "communication intensive activity" as it is an initial step for design;

ii) Elicitation:

→ elicitation means "to draw out the truth." helps customer to define what is requirement.

→ To know objectives of the system or the project to be developed is a critical job

They face problems like
(end up with unnecessary info) \leftrightarrow Problem of scope
 \rightarrow Problem of Understanding
(gap of understanding) \leftrightarrow Problem of volatility.
 \downarrow
 (change in requirement).

iii) Elaboration:

→ explanation in detailed.
→ information obtained during inception & elicitation is expanded (explained in detail).

→ This phase is an analysis, modeling phase.

→ Functions, features & constraints are explained detailed (discussed)

iii) Negotiation:

→ here discussion on financial & other commercial issues.

or → to rank the requirements (Priority)

→ to decide priority

→ to decide risks

→ to decide the project code

→ to finalize the project cost & delivery date

v) Specification:

→ The specification is the final work product produced by requirement engineering.

→ written document (SRS)

→ graphical & mathematical model.

→ any prototype if any-

vii) Validation: (100% mandatory)

→ All previous work done/completed will be just useless & meaning less if it is not validated against the customer expectations.

viii) Requirements management:

Requirements management is a set of activities that help the project team identify, control & track requirements & changes to requirements at any time of the project proceeds.

Multiples tables are maintained & updated to keep track.

Requirement	→ Aspects of the system/environment				
	A ₀₁	A ₀₂	A ₀₃	A ₀₄	A ₀₅
r ₁	✓				✓
r ₂		✓	✓		✓
r ₃				✓	
⋮					
r _n		✓		✓	✓

→ Initiating the requirements engineering process

There could be many issues to even start requirement engineering.

→ Customer may be located at-

different cities or country.

→ Customer does not have clear idea on requirement.

→ Language problem & limited

technical knowledge (for customer).

hence by following below steps we can start a good requirement engineering.

i) Identify the stakeholders:

↳ is anyone who has direct or indirect way of benefits from the system

a) software engineer

b) business, product manager

c) customer

d) end user

e) support & maintenance engineer

etc...

ii) Recognizing multiple viewpoints:-

- As many stakeholders exist, they all have different views regarding the system regarding the system, that is to be developed
- It's a duty of software engineer to consider all the viewpoints of all stakeholders & select based on the requirement

iii) Working toward Collaboration:-

- From all the views now business / senior manager should discuss for requirements & approve the most appropriate one.

iv) Asking Questions:-

- Once customer & every discussion is done. if there are still any questions in you make sure you clear them (else it will effect other steps).

→ Eliciting requirements:

eliciting is a task that helps the customer to define what is required.

steps

i) Collaborative requirements gathering -

→ Gathering software requirements is team oriented activity. In this activity software team members, software engineering manager, members of marketing & product engineering representatives work together.

aim → To identify the problem.

→ To suggest solution

→ To negotiate about approach

→ To specify the preliminary

set of solution requirements.

ii) Interview / meetings:

→ Objective of conducting an interview is to understand the customer expectation from the software.

→ It is impossible to interview every stakeholder hence representative from groups are selected on their expertise & credibility

→ Interview may be open ended or structured
↳ (acc to plan/agenda) (question on spot)

iii) Brainstorming Sessions:-

- It is a group technique
- It is intended to generate lots of new ideas, hence providing a platform to share views,
- every is documented so that everyone can see it
- finally a document is prepared which consists of the list of requirement & their priority if possible.

iv) Facilitated application Specification (FAS) Technique

→ Its objective is to 'bridge' the expectation gap b/w the developer & customer requirements.

think & customer prepare
→ each participant prepares his/her list, different lists are then combined & planned carefully.

v) Quality Function Deployment (QFD)

→ In this technique, customer satisfaction is of prime concern (hence requirements are most valuable for software development).

divided into 3 types of requirements.

a) Normal requirements: (stated by customer)

Requirements mentioned/stated by customer & they are considered into Normal requirement for customer satisfaction.
ex: Graphic display, Specific system func

b) expected requirements:- These requirements are implicit type of requirements. Their requirements are not clearly stated by the customer but even then the customer expects them.

e.g. easy to install, user friendly, help etc...

c) Exciting requirements:- These requirements are neither stated by the customer nor expected, but to make the customer more satisfied, the developer may include some unexpected requirements

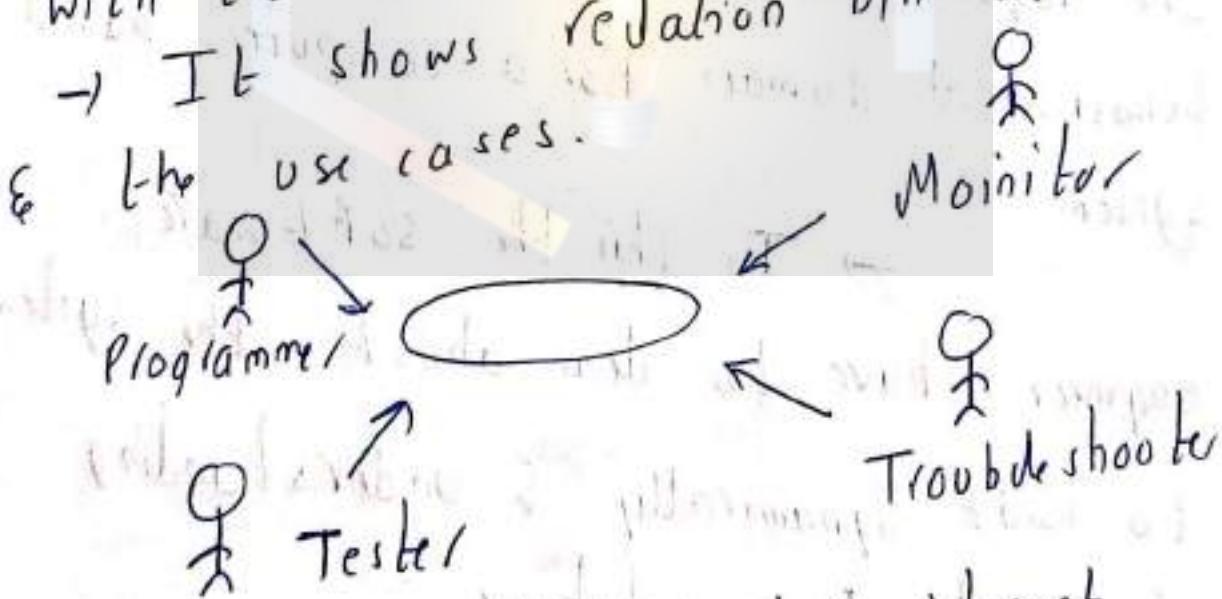
e.g. proper UI & interaction/interface

vii) Use Case Approach:

This technique combines text & pictures to provide a better understanding of the requirements. (as pictures help a lot for fast & easy understanding)

→ Developing use case:

- A use case describes the system behavior under various conditions as the system responds to a request from one of the stakeholders (user/customer).
- In other words, use case shows how the end users will interact with the system under specific circumstances.
- Use cases are defined from actor view. An actor is a role that refers the user or device that interacts with the software.
- It shows relation b/w actors & the use cases.



- i) Primary actor: These actors interact with the system to achieve required system

functions & derive intended benefit from the system. The work directly with the software.

iii) Secondary Actors: These actors support the system so that primary actors can do their work.

→ Building the analysis model:

→ The attention of the analysis model is to provide a description of the required information, functional & behavioural domain for a computer based system.

→ In this the software engineer have to learn about the system to build dynamically & understanding requirements is mandatory.

It has 4 elements

- i) Scenario - based elements (user point of view)
(like use - Text, User case diagram, activity diagram etc..)
- ii) Flow oriented elements (data objects are transformed by processing the function)
(Data Flow diagram, Control Flow diagram, Processing narratives)
- iii) Class based elements (objects, attribute & relationship)
(class diagrams, analysis diagram, CRC models, collaboration diagram)
- iv) Behavioural elements (how the act)
(state diagram & sequence diagram)

→ Negotiating requirements:

Negotiations are based on
need ranking & risk ranking.
includes i) Business

iii) Technology iii) economics

iv) Time Frame v) Social Impact

→ Validating requirements:

→ All previously done / completed

will be just useless & meaning less
if it is not validated against

the customer requirements / satisfaction.

→ It's 100% mandatory to
validate requirements before going further
as every next step (planning → design → constr
deployn[↑])

depends on requirements.

Unit - III

Building analysis

model & Design engineering

→ Analysis model:

→ Analysis model uses a combination of text & diagrammatic forms to depict requirements of data, functions & behaviour so that it will be easy to understand overall requirements of the software to be built.

→ The 'software engineer' also called analyst builds the model using requirements stated by the customer.

→ Analysis modeling approaches:

- i) Structured approach (Data modelling)
- ii) Object oriented approach.

I) Structured approaches:

Analysis is made on data & processes & it transforms the data as a

separate entity

→ Data is modeled in terms of only attributes & relationship (not operation)
ex: ER diagrams. (in Data modeling)

→ Data modeling concept:-

In data modeling data objects are examined independent of process.
ex: ER diag.

i) Data objects :-

→ Data object is representation of any composite information (many attributes/properties).

ex: Car

Name (company)	Color	Price
Nexus	Silver	x lac
BMW	blue	y lac
Ford	black	z lac

Here car is a data object & name, color & price are properties/attributes

iii) Data Attributes:

each data object is having
of properties those properties are the
attribute.

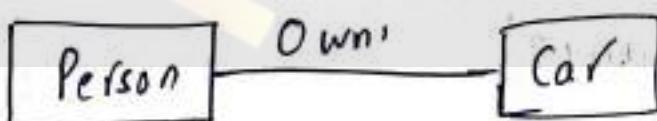
e.g. in previous table name, color
& price are attributes.

(we also have multiple types
of attribute & their representation - check
in DBMS Unit 1). (For complete ER model).

iii) Relationships:

relationship indicates how
data objects are related to one another.

e.g. customer purchases the car.



iv) Cardinality & Modelling:

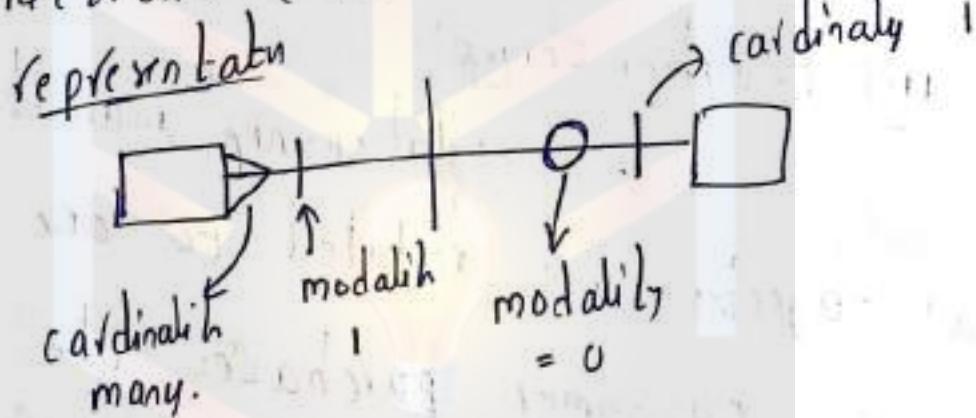
Cardinality specifies number
of occurrences of one object related
to number of occurrences of another
object.

(or)

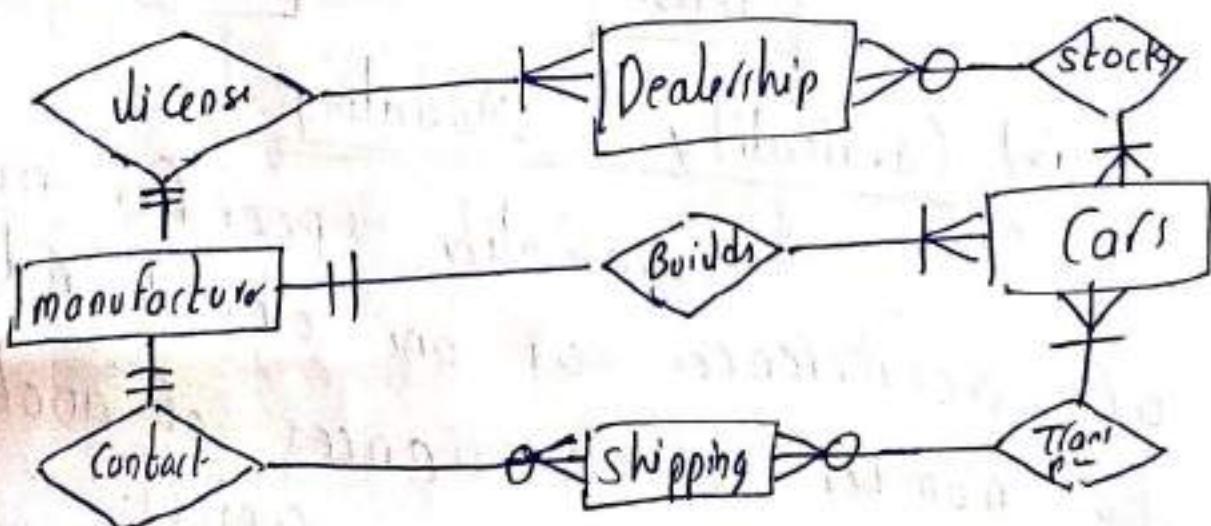
The maximum number of objects relationship is represented by cardinality

- One-to-one (1:1)
- one-to-many (1:n)
- many-to-one (n:1)
- many-to-many (n:m)

modality :- is the least number of row connections (can be 1 or 0)



E-R Diagram:



II) Object Oriented analysis:

→ In object oriented approach, analysis is made on the classes & interaction among them in order to meet customer requirements.

→ Object oriented approach views information - system as collection of interacting objects that work together to accomplish tasks.

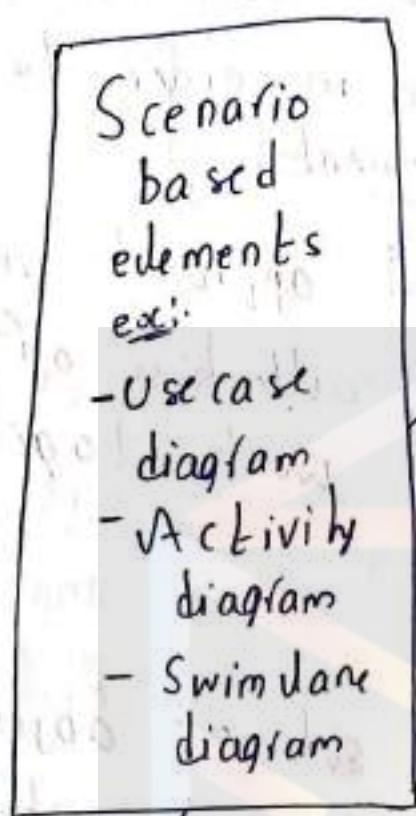
2 steps

i) Definition to the set of objects that will make up the system & describing the interaction or communication among various objects. (interaction ^{means takes} in the form of messages ^(respond / message is nothing but calling))

ii) Describing the internal processes that go on within each object to respond to messages from other objects & to initiate messages to other objects.

→ elements of the analysis model :-

@ with UML

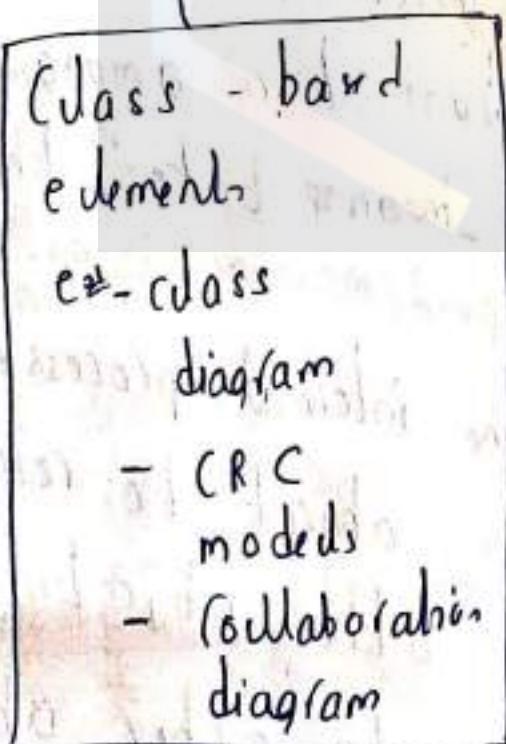


Analysis model

Flow-oriented elements

e.g:- Data Flow diagram

- Control Flow diagram



Behavioural elements

e.g:-

- State diagram
- Sequence diagrams

→ Scenario based modeling :-

→ Analysis modeling with UML begins with the creation of scenarios (we have multiple software for designing UML diagrams, rational rose enterprise edition, liferay, ArgoUML etc.) SBM includes:

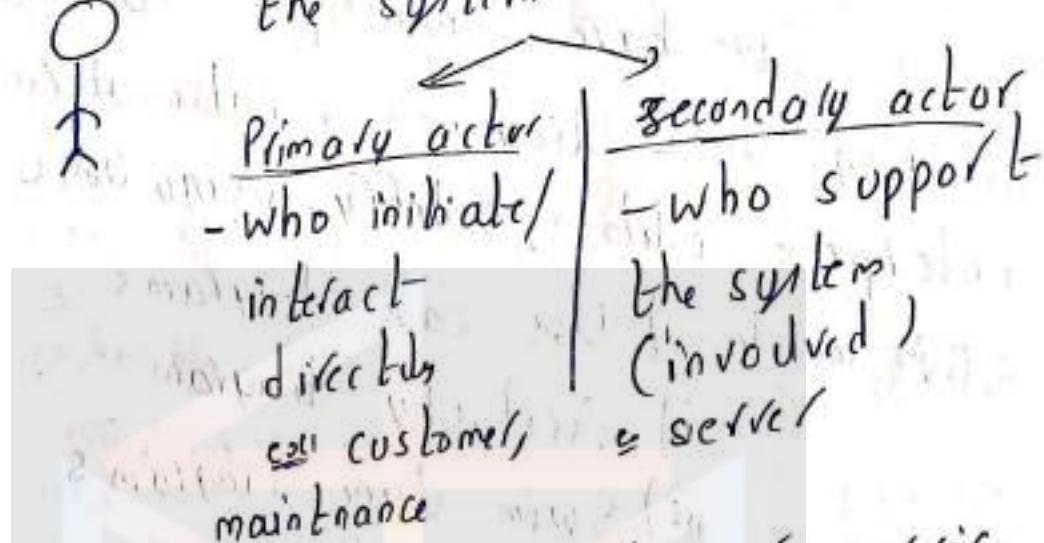
- i) Use case diagrams
- ii) Activity diagrams
- iii) Swim lane diagrams

I) Use case diagrams:-

Use case model, use to identify processes & define all elementary business processes that system must support.
use case: usecase is a scenario by which user can use the system & pictorially we represent these scenarios by usecase diagram.
(single system can have multiple use cases).

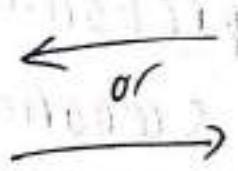
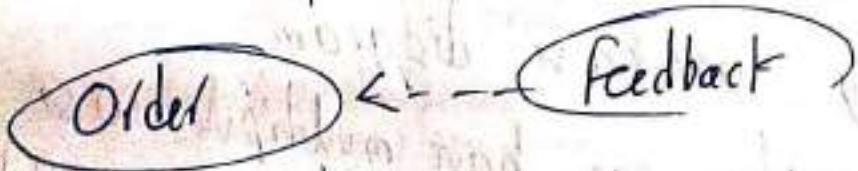
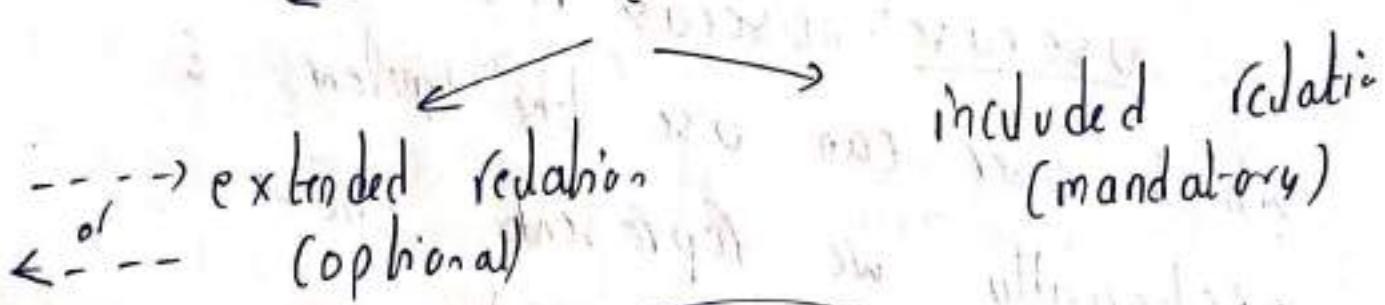
→ Components of use case diagram (3)

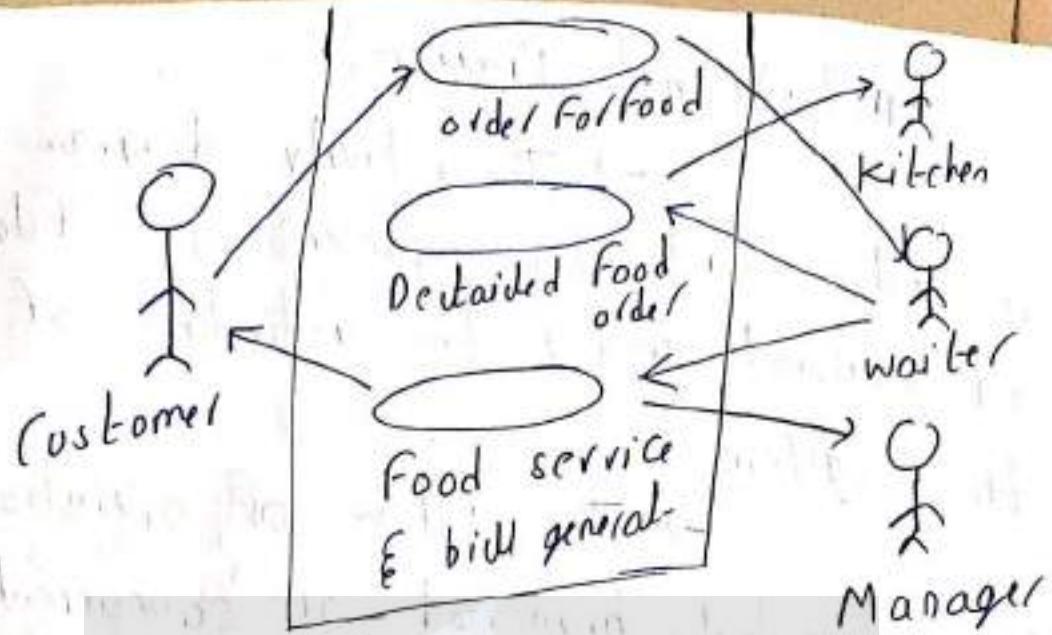
i) Actor: user who interact with the system



ii) Use case: functionality or service provided by the system.

iii) Relation: It shows relation b/w actor & the system.





Path / Flow:

- Customer orders for the food
- The waiter receives this order
- The waiter then forms detailed order
- The detailed order is submitted to kitchen & service
- To the customer
- Finally bill is produced.
- Finally bill is submitted to customer
- The bill is sent to manager.

II) Activity Diagram:-

→ The activity diagram is a flow chart to represent the flow of control among the activities of a system.

→ The flow of operation can be sequential, branched or concurrent. The activity diagram is sometimes considered as the flow chart. Although the diagrams look like a flowchart, they are not.

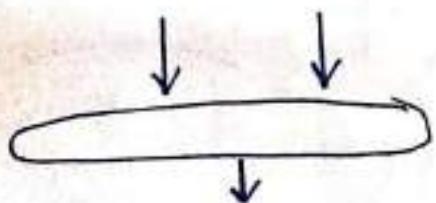
Components / symbols

• start

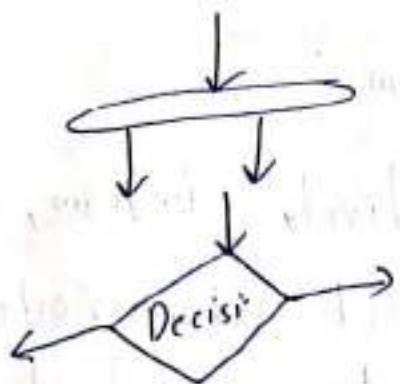
○ final activity node.

Activity Activity (with short description)

→ Control flow / state transition



Join (2 to 1) conversion
joint

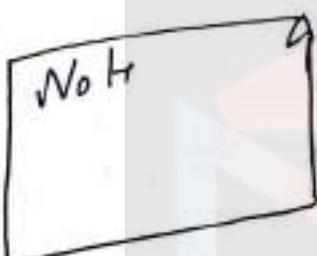


Fork (1 to 2) (divide)

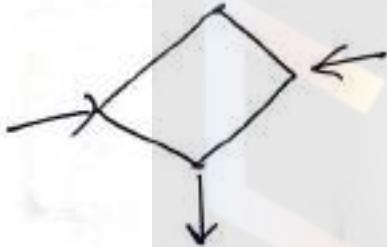
Decision (condition)



final flow node (end of
only that flow.)

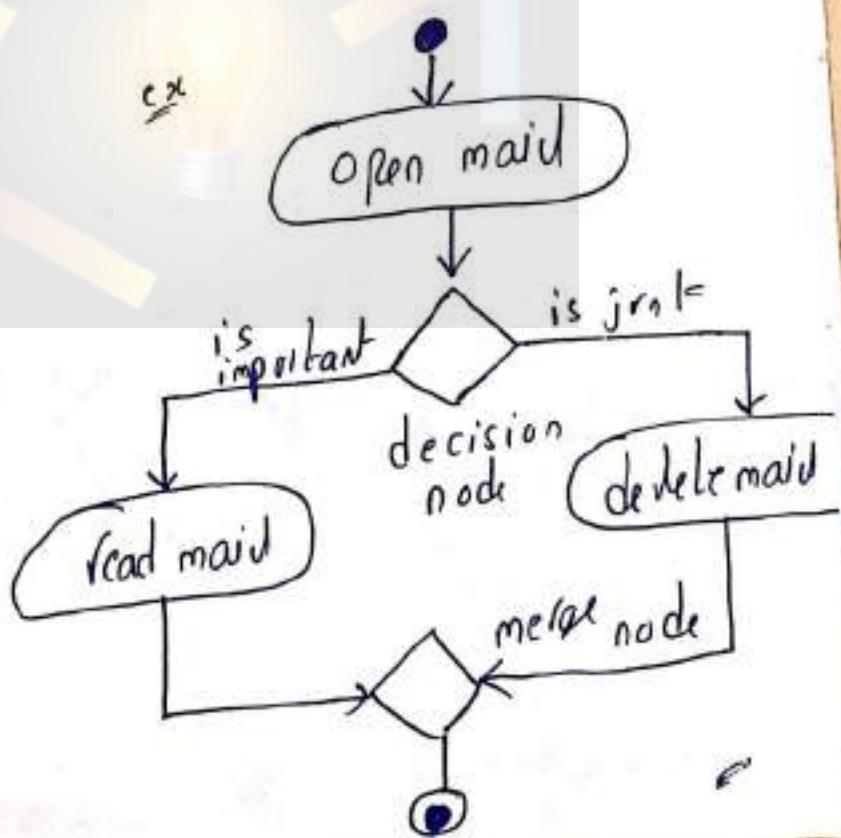


Note (for comment).



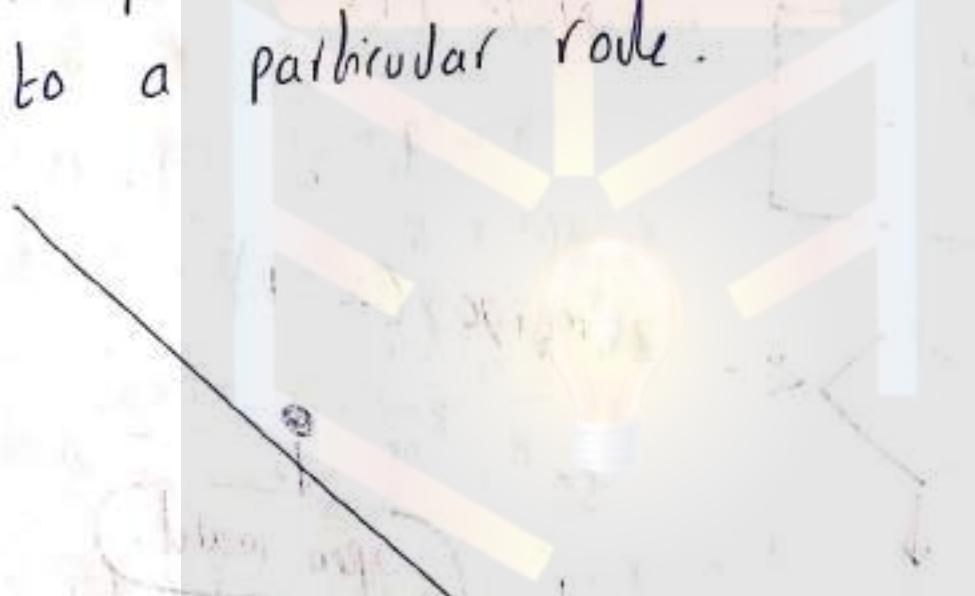
merge (2 : 1)

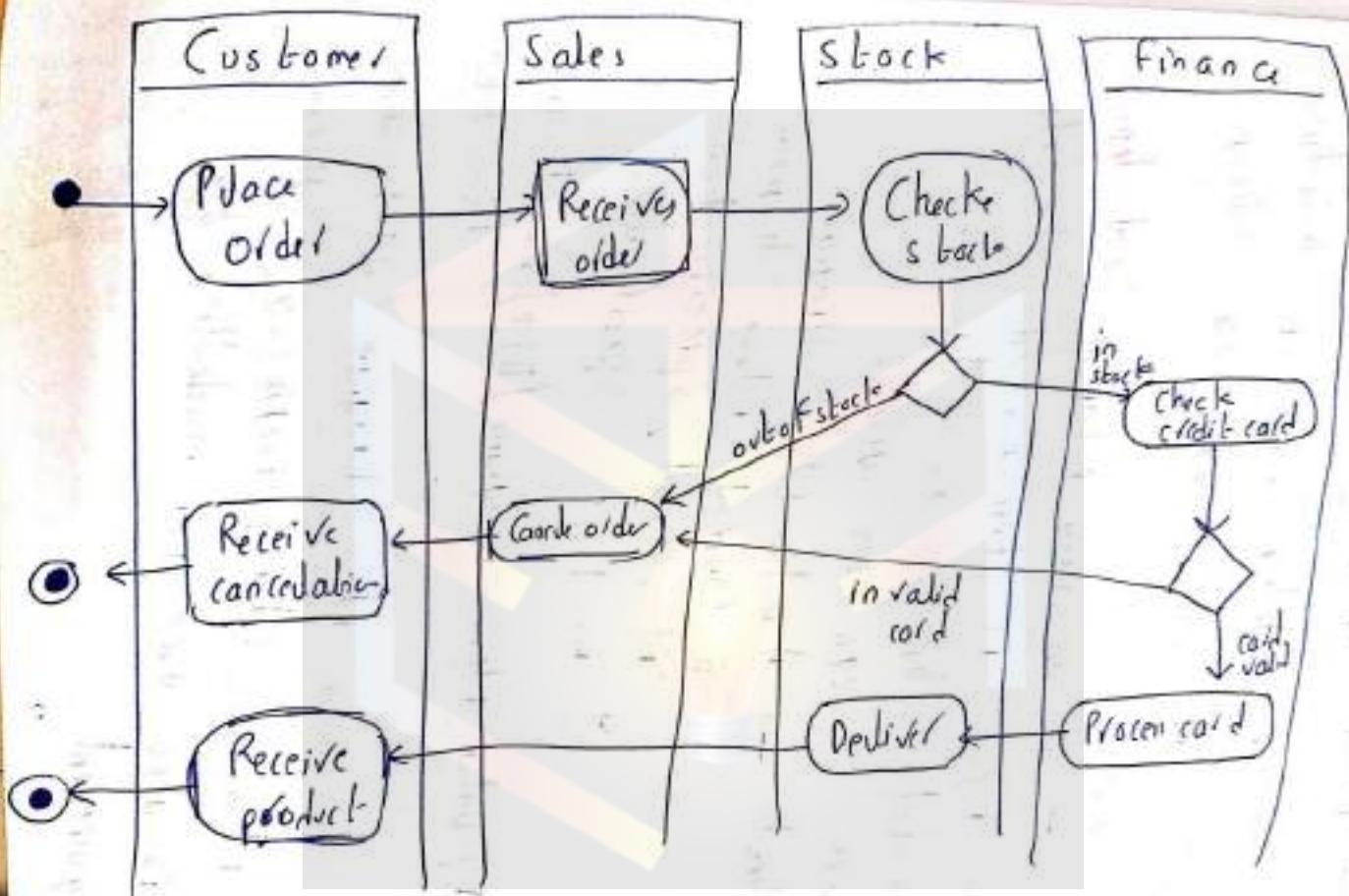
ex



III) Swimlane Diagrams:

- A flavor of activity diagram, which also give information about which role is performing the underlying activity.
- Activity diagram is divided into multiple columns & each column is dedicated to a particular role.





→ Flow Oriented modelling:

- This modelling represents how data objects are transformed/moved through the system
- A modelling notation that depicts how input is transformed into output as data objects as data objects move through the systems

- elements
- i) Data flow Diagram
 - ii) Control Flow diagram
 - iii) Control Specification
 - iv) Process specification

I) Data flow Diagram / DFD Bubble chart / Flow graph

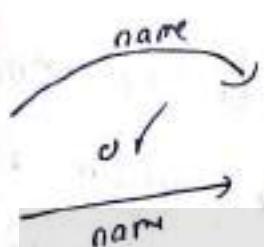
A data flow Diagram (DFD) shows the flow of data through the system & is also used for modelling the requirement.

Components / Symbols:



Process

(takes i/p &
produces output)



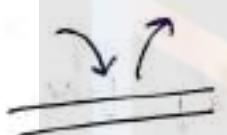
Data
Flow

(shows flow of
data into or
out of a process/
data store/source).



source
or
sink

(external entity
that acts as an
i/p to system (source)
or o/p to system (sink)).



data
store

(Data repository).

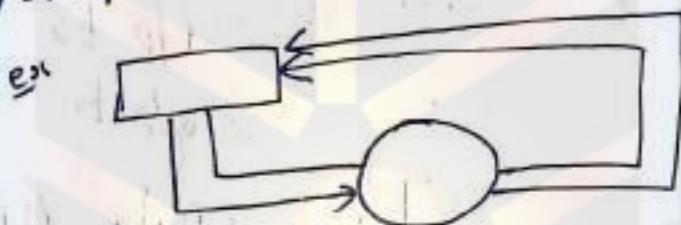
Note:

- unique names are important
- DFD's depict flow of data & not order
of events like a flow chart.
- avoid representing of decision nodes
(diamond nodes).

→ Leveling in DFD:

→ DFD's can be drawn to represent the system at different levels of abstraction.

DVL-0 DFD: represents the entire system as a single bubble with input & o/p data indicated by incoming & outgoing arrows.

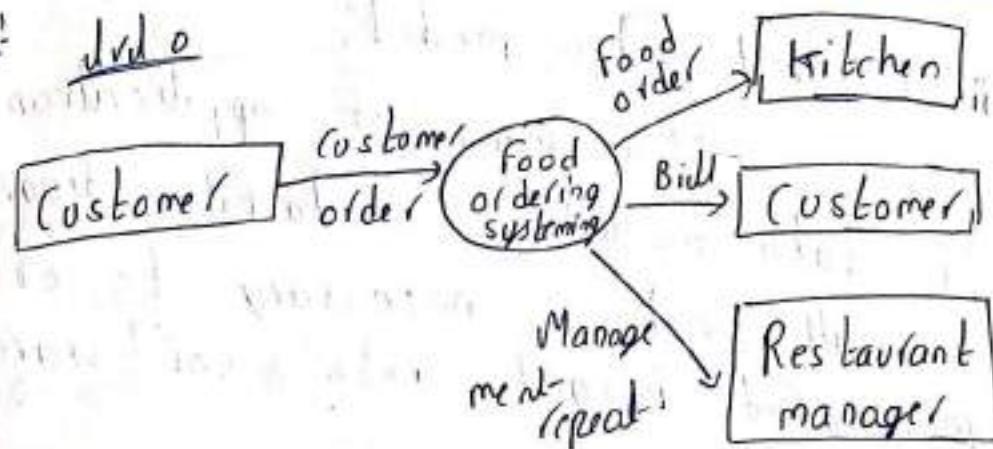


DVL-1 DFD: expanding the above DVL-0, or a bit depth representation is done

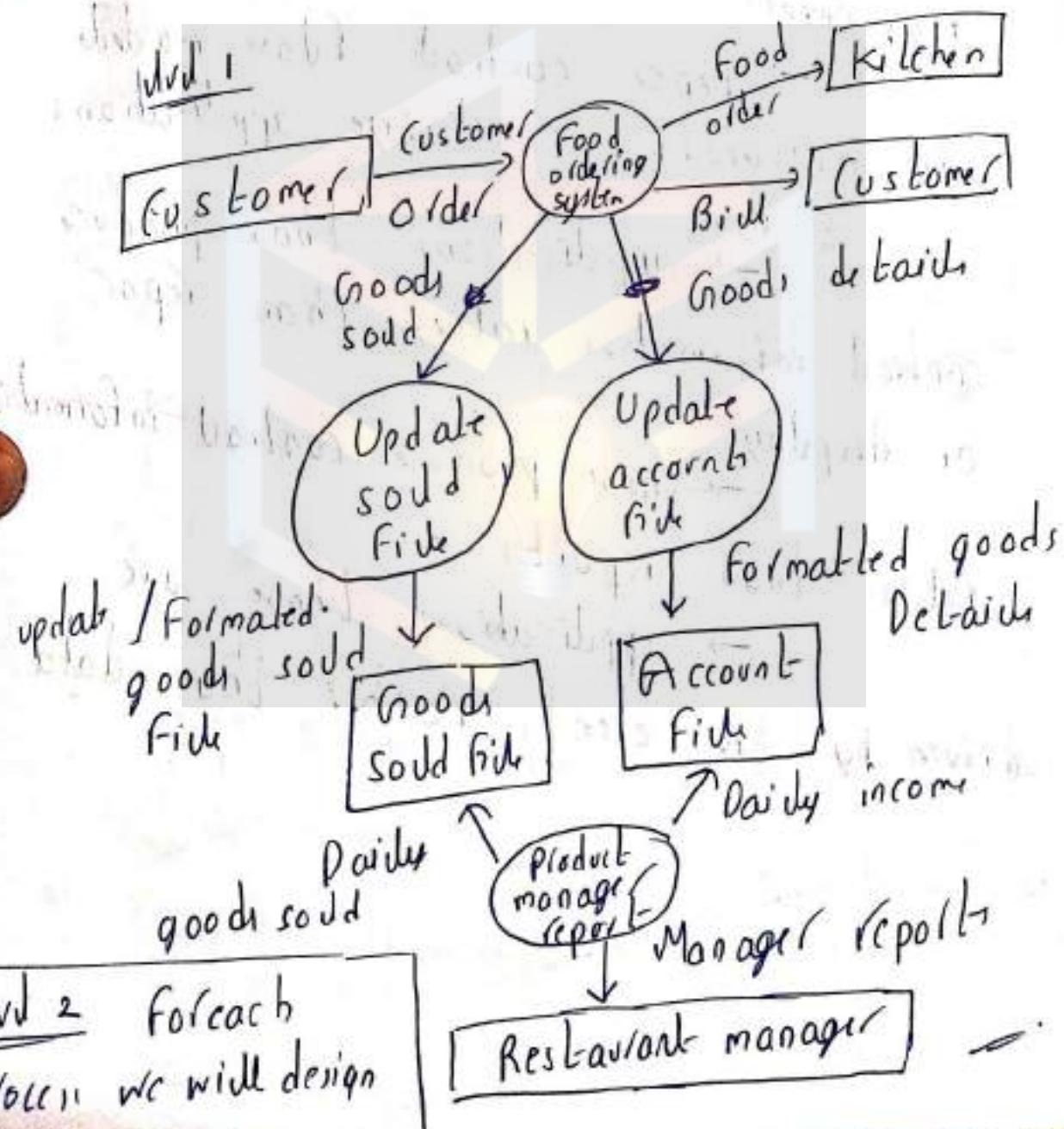
Note: that the no. of i/p / o/p should be kept constant

DVL-2 DFD: each individual module will have its own DFD (full depth & 100% detail).

ex: Jrd 0



Jrd 1



Jrd 2 for each process we will design

III Control flow model:

→ for many of applications the data model & data flow diagram are all that is necessary to obtain meaningful insight into software requirements

→ hence control flow models are designed for large applications

& for → applications those produce control information rather than reports or displays

→ those produce control information rather than reports

→ applications those are driven by the events rather than data.

III) The control Specification:

The control specification represents the behaviour of the system. The behavioural of the system can be represented using state transition diagram or state chart diagram.

state chart diagram: A state chart diagram shows the state machine that consists of states, transitions, events & activities.

state. (is a situation/
condition/activity)

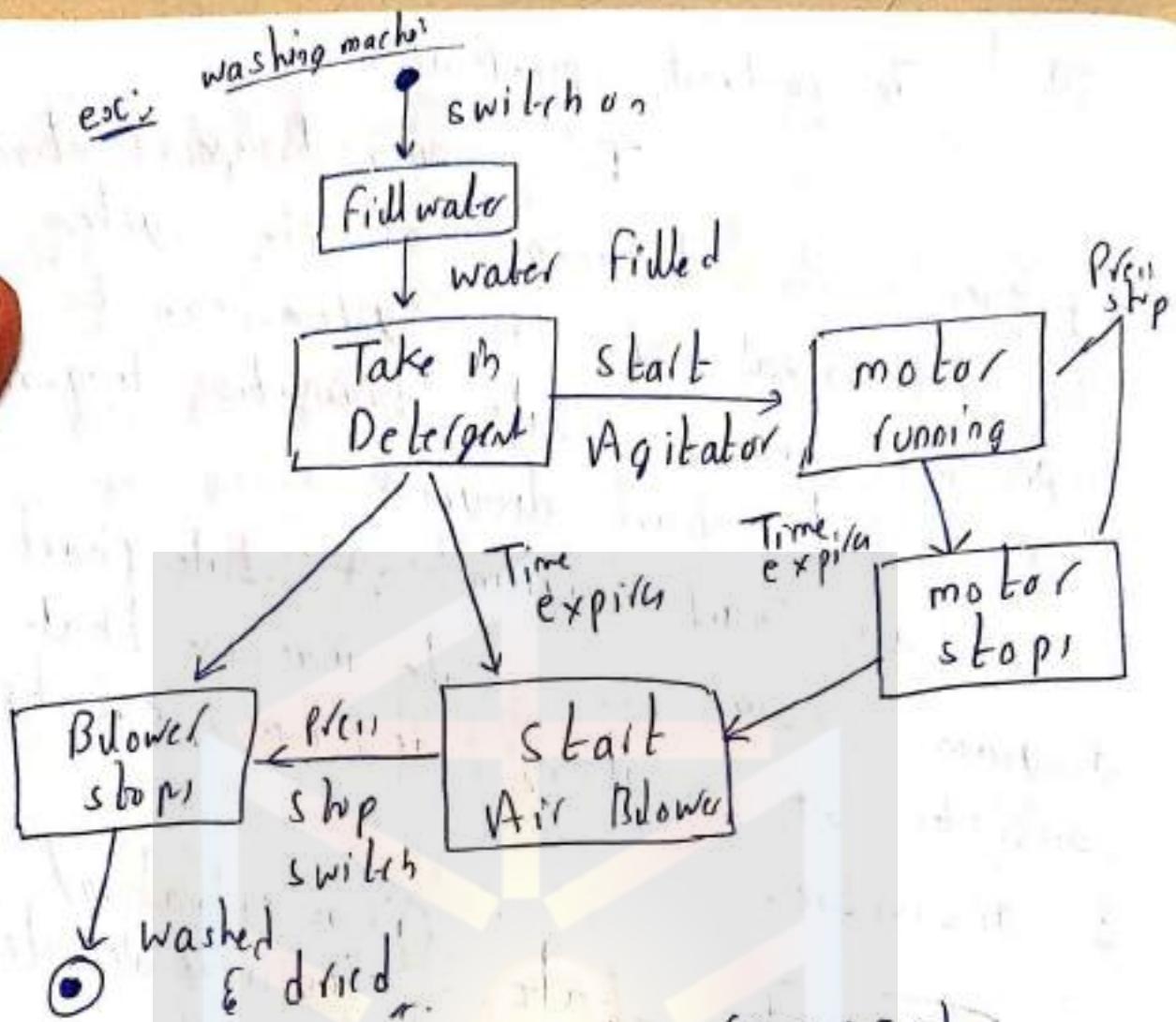
start state

End state

state transition

transition to itself





IV) Process Specification (PISPEC)

The process specification is used to describe all flow modeled processes. The content of process specification includes Program design language (PDL). It's nothing but Pseudo code.



Pseudo code is written to analyze the triangle.

→ Class Based modelling:

→ Class diagram include identification of candidate classes and it shows their responsibilities & collaboration

↓
attribute &
operations

relation b/w one obj
to another.

→ The main idea / it describes the static structure of the system.

Class is a template or formal-

→ represent entities with common characteristics or features

→ attribute & operations

form class

→ in object

attributes get the visibility

values & operations are used by object
object with underline

(Class Name)

+ Attributes

→ Data

+ Operations()

→ method

Animal

Initiation of

Dog

obj name

relation b/w classes

Association

peer classes

→ Association can have a
name & direction, or be bidirectional

(Class B)

(Class A)

+ Association Name

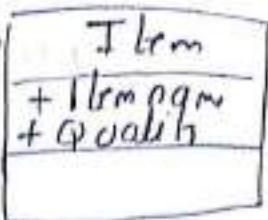
+ Role A

+ Role B

cst



+ contains
+ made-
up-
of
+ included
in



order made up of item

item is included in order

Multiplicity / Cardinality:

(Jai)

Unspecified
exactly one

(Jai)

Many
(zero or more)

(Jai)

Optional
(zero or one)

same above example with

→ one instance of order is * (0 or more)
instance of item.

→ one instance of item is connected
with 1 instance of order

Aggregation :-

A class has an attribute which is an object of another class.

(Part)



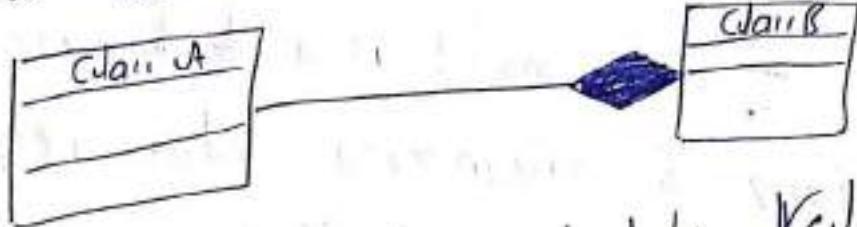
→ here class A is an object of class B (in attribute)
 → This is not strong relation
 (hence ~~deleting~~ deleting, will not effect other)

Composition :-

A class has an attribute which is an object of another class

→ here also class A is an object of class B (in attribute)

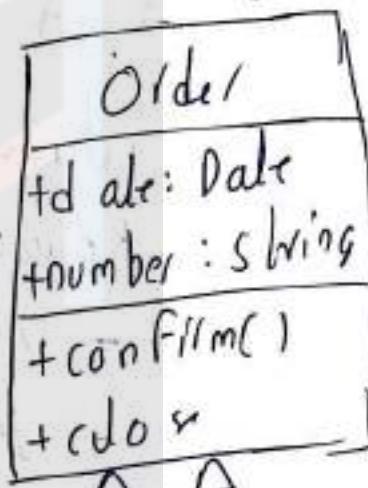
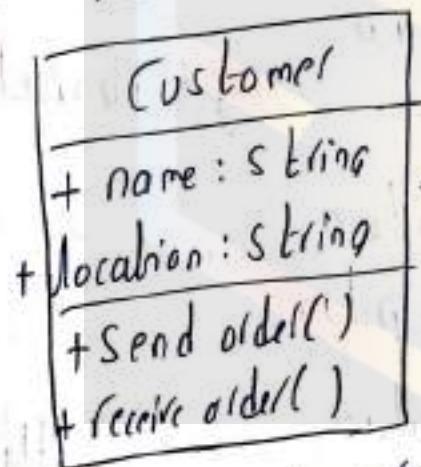
→ here it's strong (hence destroying
one will delete other)



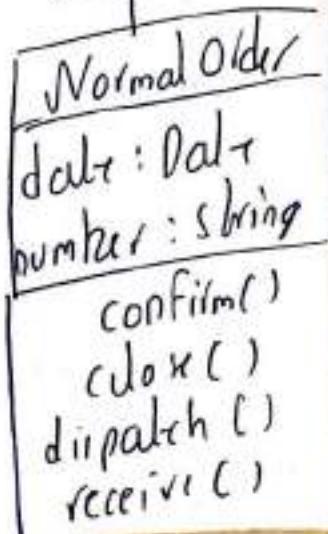
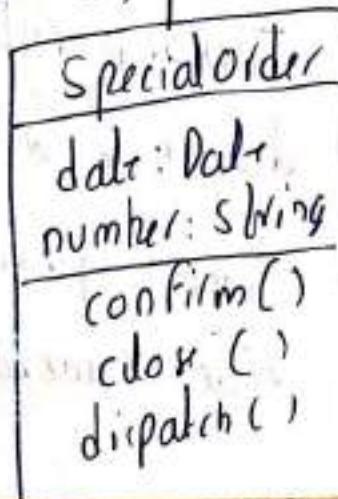
Dependency to establish Relation
depending on another



ext class diagram



generalization →



Sub
class →

Class responsibility Collaboration (CRC)

→ CRC model is a technique for identifying & organizing class responsibilities

→ It is a collection of (3 x 5) that represent classes

→ CRC card is a useful technique for identifying class responsibilities.

→ A simple CRC card

Class : Floor Plan

Responsibilities:

Collaborations

- Define floor plan

- Manage floor plan

- Create four plan

- Incorporate walls

- doors & windows

- Show position of cameras

walls

walls

cameras

RC)

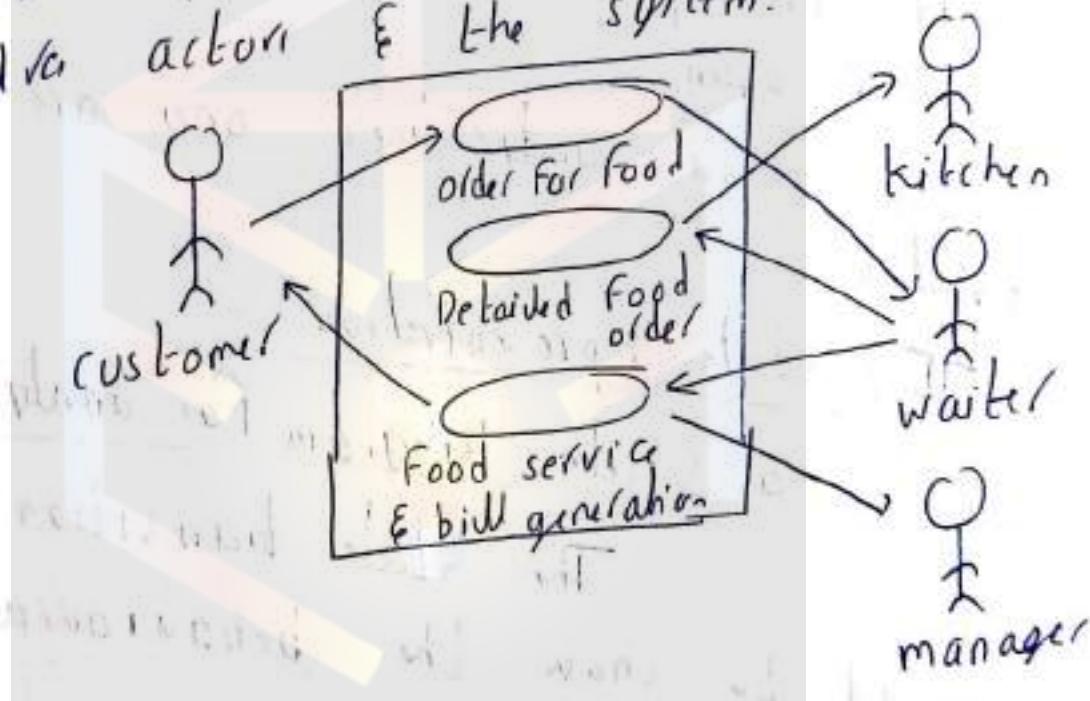
bitics

→ Behavioural model:

The behavioural model indicates how software will respond on occurrence of external events.

I) Identifying events with the usecase:

At sequence of activities can be represented by a usecase that involves actors & the system.



Whenever the system & an actor exchange information an event occurs
ex: The homeowner uses the key pad to enter his four digit password

The password is compared with the valid password stored in the system

If the password is incorrect, the control panel will keep once & reset itself for additional i/p.

If the password is correct, the door opens.

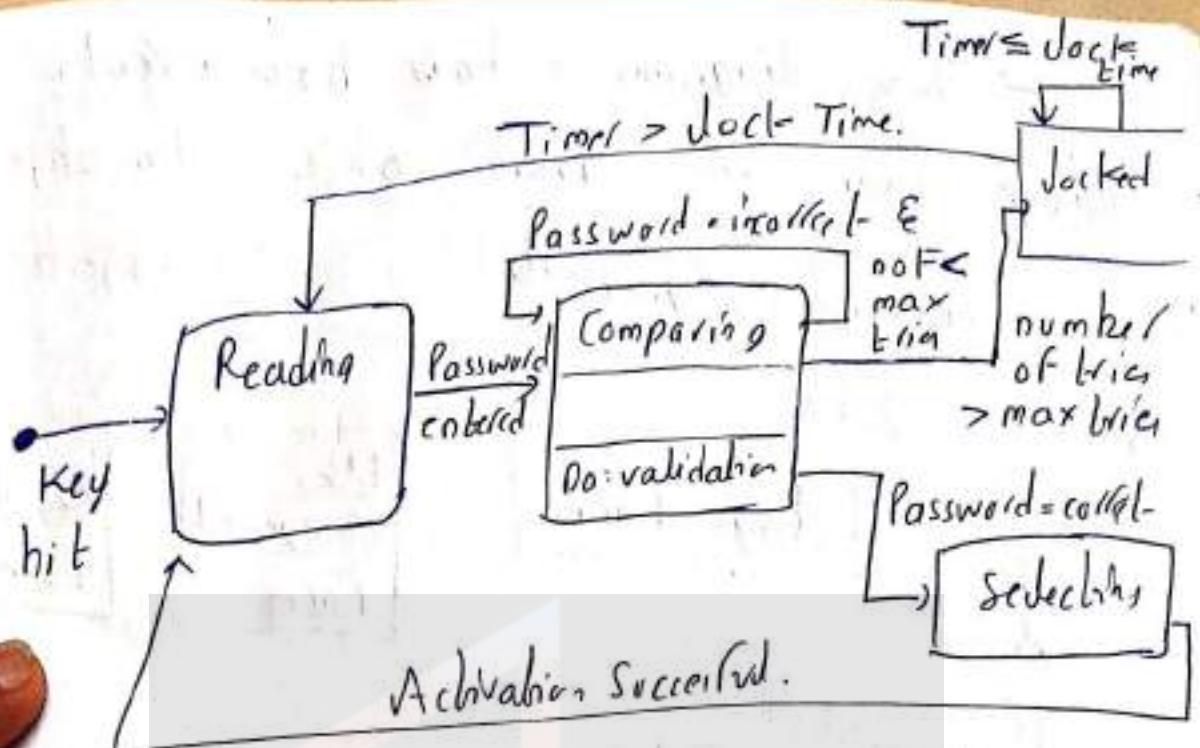
here underlined ones are the events

II) State representation :-

a) State Diagram for analysis class

The state transition diagram are used to show the behavioural of the system. They show transitions from one state to another.

see same above security's state Diagram



$\Rightarrow \text{no } l = < \text{max tries}$

$\Rightarrow \text{no. of tries} < \text{max tries}$

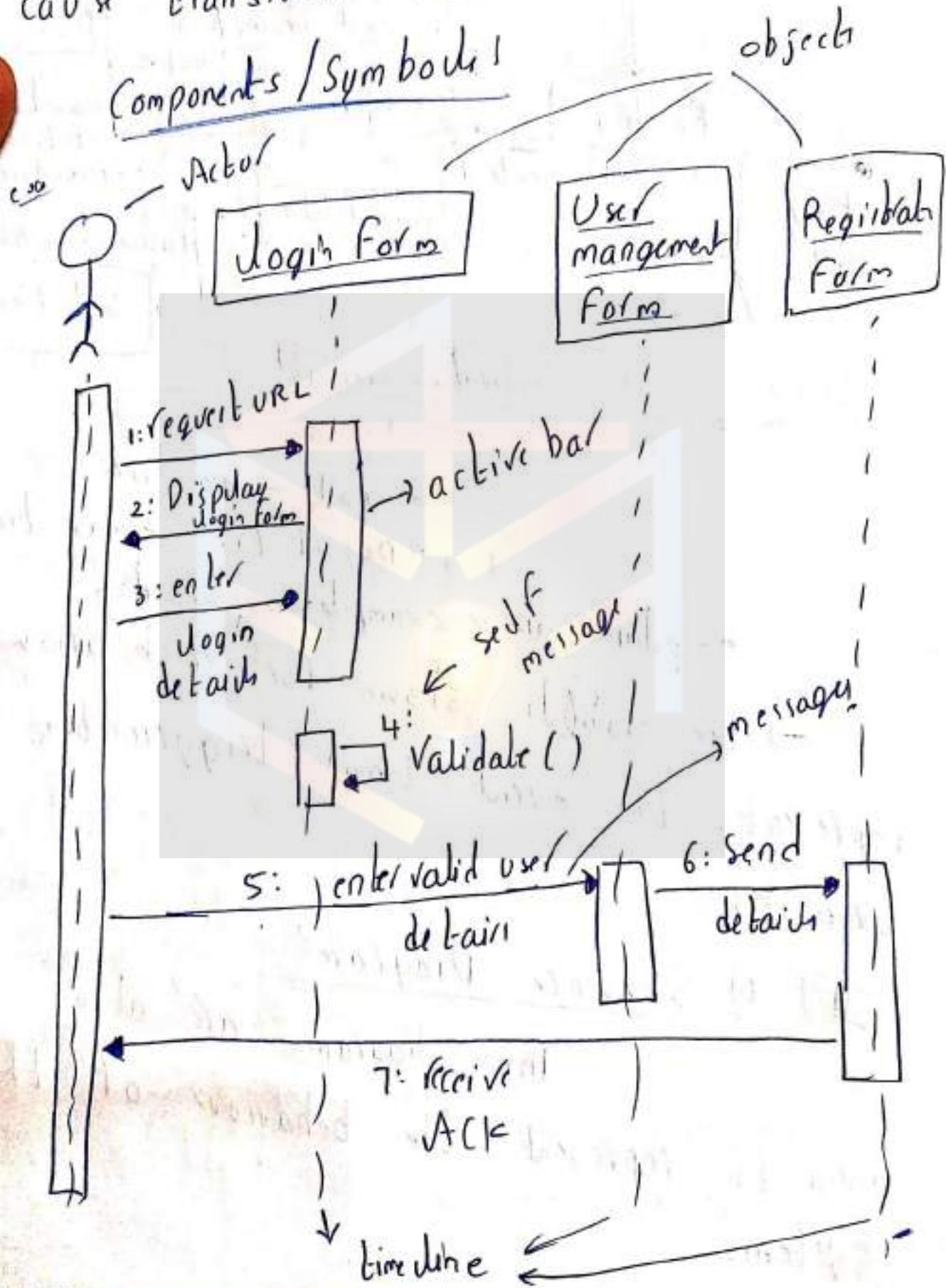
explain above example in words.

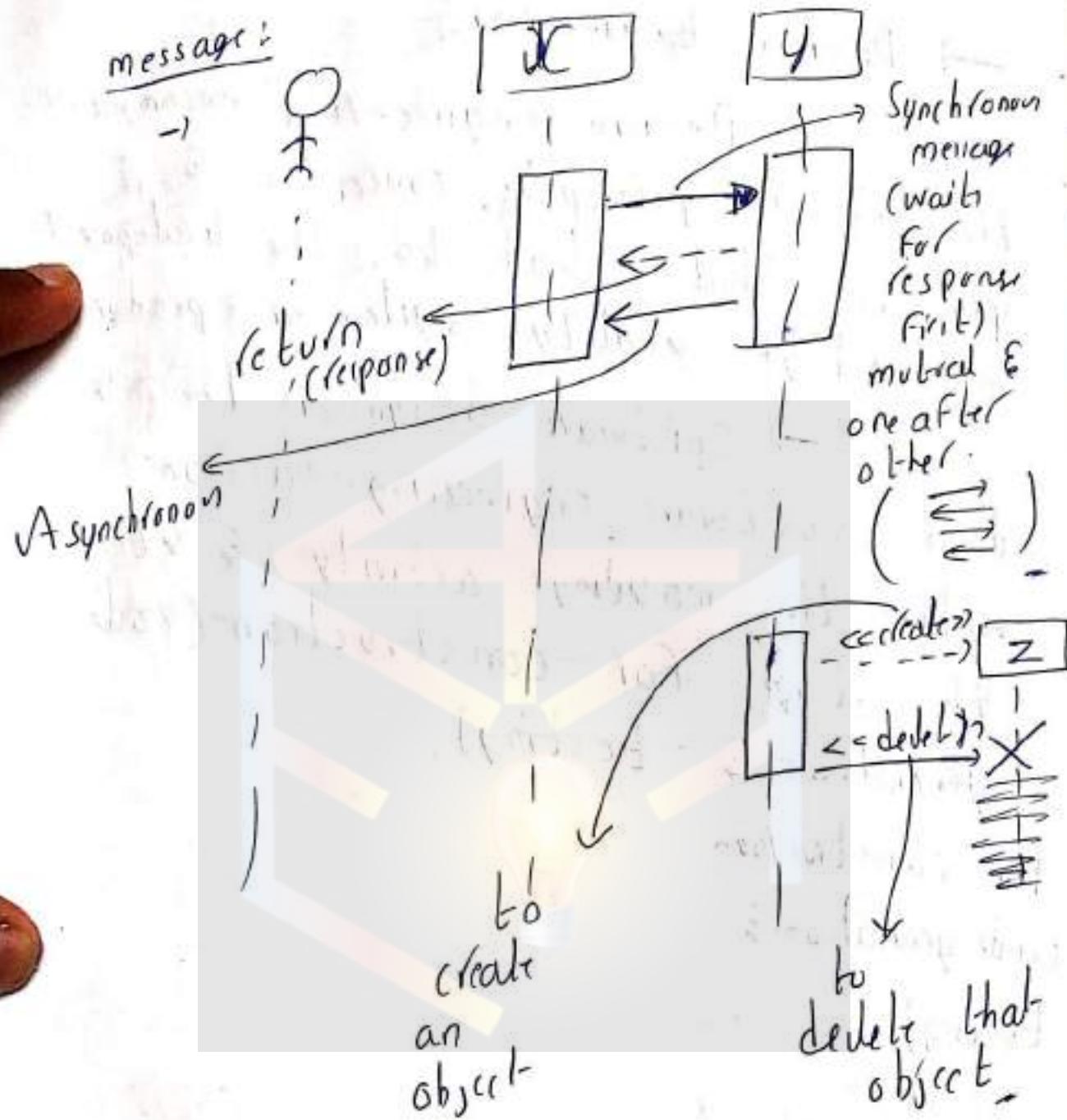
→ the labels shown for each arrow represents the event that triggers the transition.

iii) b) Sequence Diagrams:

These diagrams are also used to represent the behaviour of the system.

→ These diagrams show how events cause transition from object to object



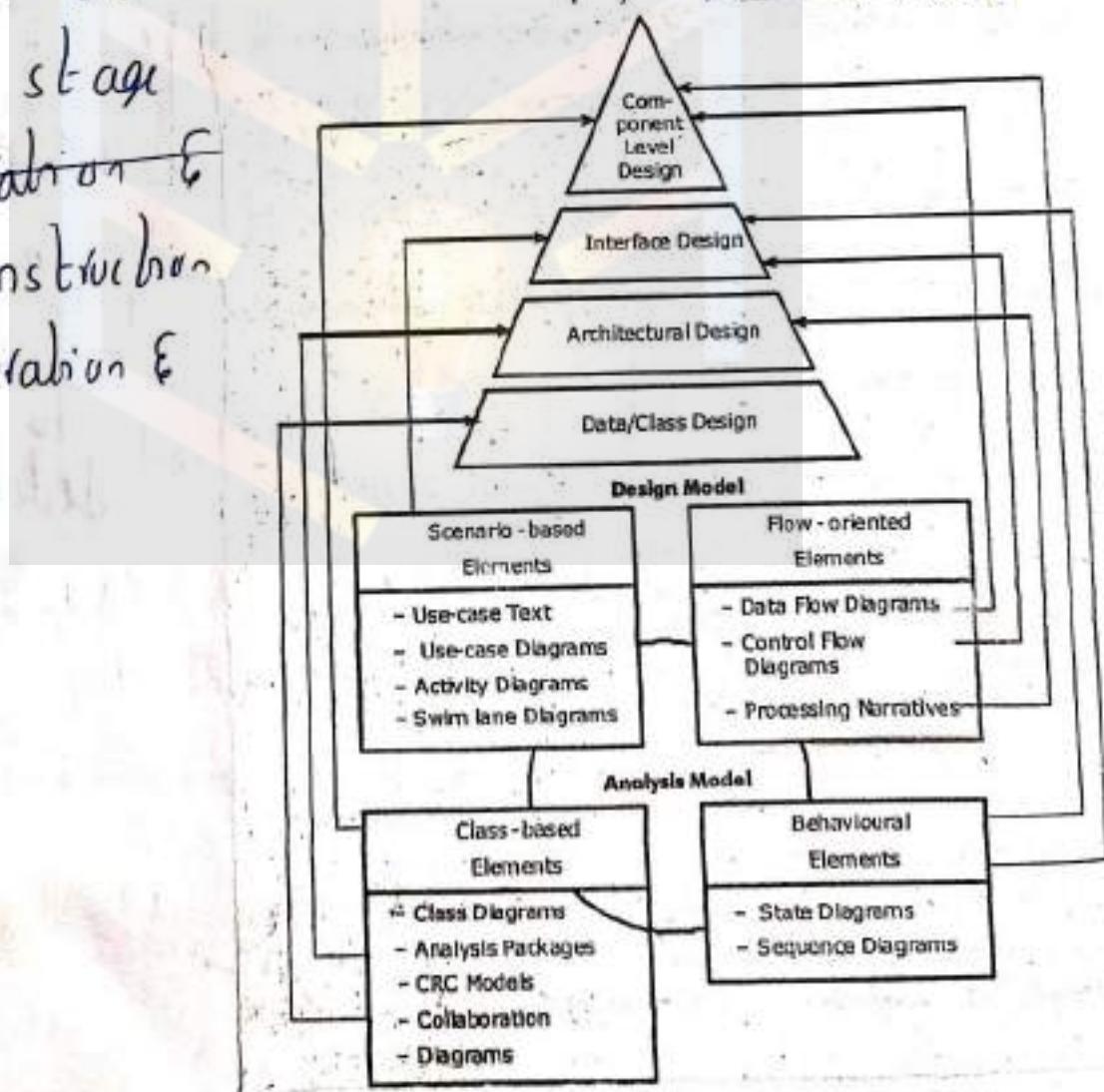


→ Design Engineering :-

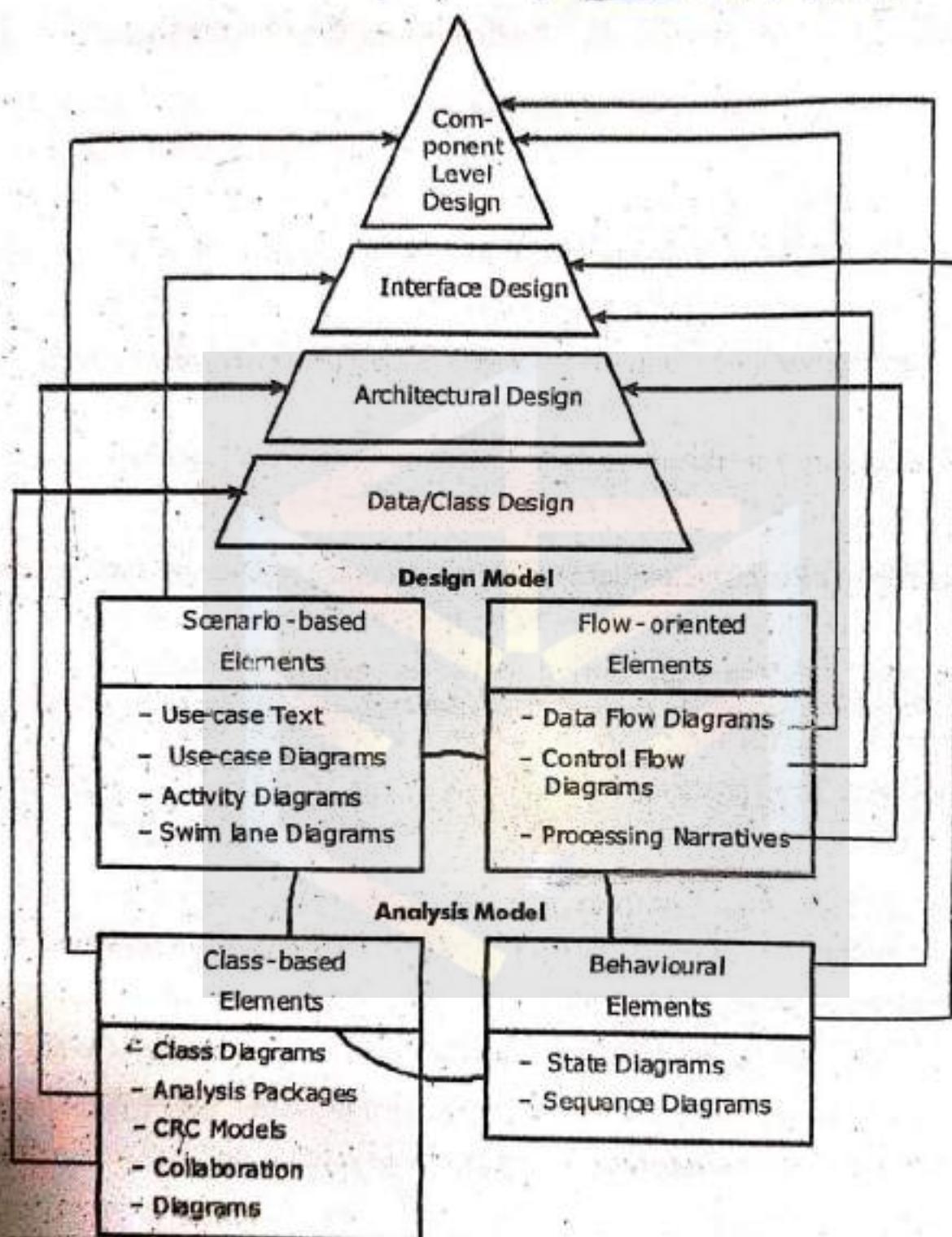
Design engineering encompasses the set of principles, concepts & practices that lead to the development of a high quality system or product.

→ Software design is the last software engineering action within the modeling activity & starts from

the stage generation &
for construction
(code generation &
Testing)



modelling activity & its



→ Design Process & Design Quality:

→ Design Process:

→ Design process is an step by step & repetitive procedure where software is simulated according the requirements.

→ The simulation/will be in detailed (high level of abstraction)

→ The design represented by this level can be directly traced to the specific system objective & requirements such as data, functional & behavioural requirements.

→ As said earlier, design process is required for better quality software. (This quality is achieved by a series of formal technical reviews.

The 3 main characteristics are

- i) Design must consider & fulfill all the customer's requirement that are implicit & must implement explicit requirements contained in the analysis model.
 - ii) The design must be readable, understandable & as a guide for those test who generate code & for those whose test & subsequent support the software.
 - iii) Design should provide a computer picture of the software, addressing data, functional & behavioral domains.
- Design quality :-
- These are the few guidelines which are the characteristics of a good design.

- 1) A design should exhibit an architecture that has
 - a) Creation by recognizable architectural styles or patterns.
 - b) Components leading to good quality design characteristics & evolutionary fashion which facilitates implementation & testing.
- 2) A design should be module based (partition of elements or subsystems).
- 3) A design should have different representation of data, architecture, interfaces & components.
- 4) Design should explore the data structures useful for the classes to be implemented.
- 5) A design should consist of components that have independent functional characteristics.

- 6) A design should lead to interface that reduce the complexity
- 7) design should be achieve from iterative method.

The above are the mandatory guidelines to be followed while designing —

→ Characteristics of Good Design Process

- 1) The notion of design should reflect its meaning.
- 2) Design should be implemented based on the information obtained from the requirement analysis phase of the software development.
- 3) The design should clearly represent the interface application in the system.

- 4) The design should consist of all the independent modules.
- 5) The design should exhibit clearly all the modules of the software
- 6) Various elements such as data architecture, interface & components, should be distinguished clearly.
- 7) Using the design, a software engineer should directly build data structure & start implementation.
- 8) A design should describe several architecture, should encompass several components & have dynamic behaviour.

→ Quality attributes:

Hewlett Packard (HP) developed a set of quality attributes given by acronym FURPS

i) Functionality :-

ii) Architecture:- It is a key property of software system (It describes the internal structure of software system). A good, simple structure makes the system easy to understand, change, test & maintain.

iii) Correctness:- A design should be created as per the client requirement (collected & analyzed).

iv) Performance:- It is measured by processing speed, response time, resource consumption, throughput & efficiency.

v) Functionality:- It is assessed by evaluating feature set & capabilities of the program, generality of functions & security of overall system.

→ Design Concepts:

Following is a set of fundamental software design concepts

- 1) Abstraction
- 2) Architecture
- 3) Patterns
- 4) Modularity
- 5) Information hiding

- 6) Functional independence
- 7) Refinement
- 8) Re Factoring
- 9) Design classes.

1) Abstraction :- (hiding data/code).

→ Abstraction simply means to hide the details to reduce complexity & increase efficiency or quality.

→ Different levels of abstraction are necessary & must be applied at each stage of design process so errors that are present can be removed efficiently.

Applied for → Functional abstraction (in functional methods)

→ Data abstraction (for data structures & data representation)

→ Control abstraction

(to the flow).

2) Architecture :- (design a structure)

It is the structure of organization of program components(modules), their interaction & structure of data that are used by components.

↑
used to represent major system elements & their interactions.

3) Pattern :- (a repeated form)

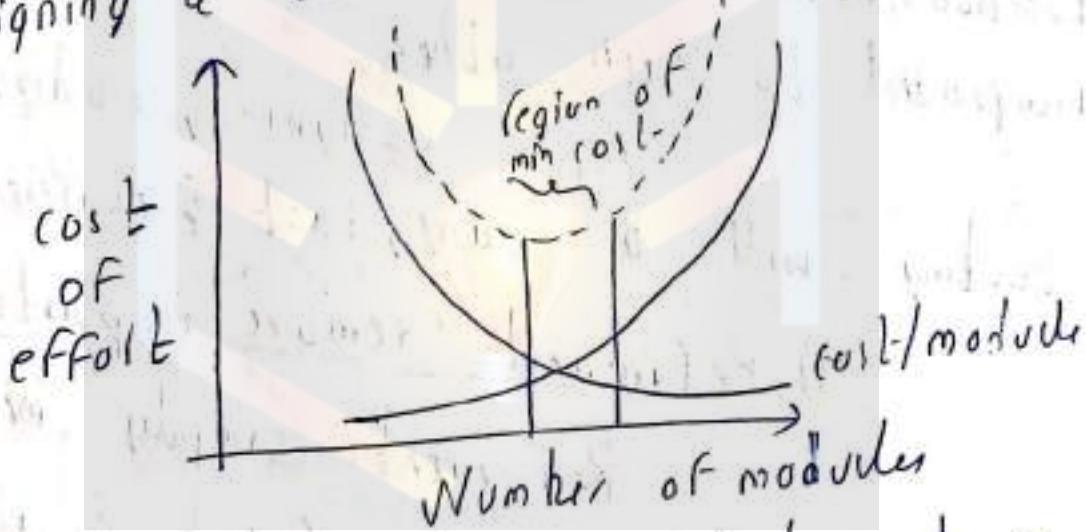
The pattern simply means a repeated form or design in which the shape is repeated several times to form a pattern.

The pattern in the design process means the repetition of a solution to a common recurring problem with a certain context.

4) Modularity (Subdivided the system)

→ modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project.

→ In the same way, modularity in design means to subdivided a system into smaller parts so that best parts can be created independently & then designing & construction will easily.



→ Hence always select not a medium module based on project cost & complexity

5) Information hiding (hide the info)

Information hiding simply means to hide the information so that it cannot be accessed by unwanted party

6) Functional Independence (independent methods)

As a project has multiple functions/methods they all should be independent to each other.

hence designing, coding, testing will be easy, fast & efficient.

7) Refinement (remove impurities)

Refinement simply means to find impurities if present & increase the quality.

Refinement is very necessary to find out any bugs/ errors if present optimisation is also done here.

8) Refactoring i: (reconstruct something)

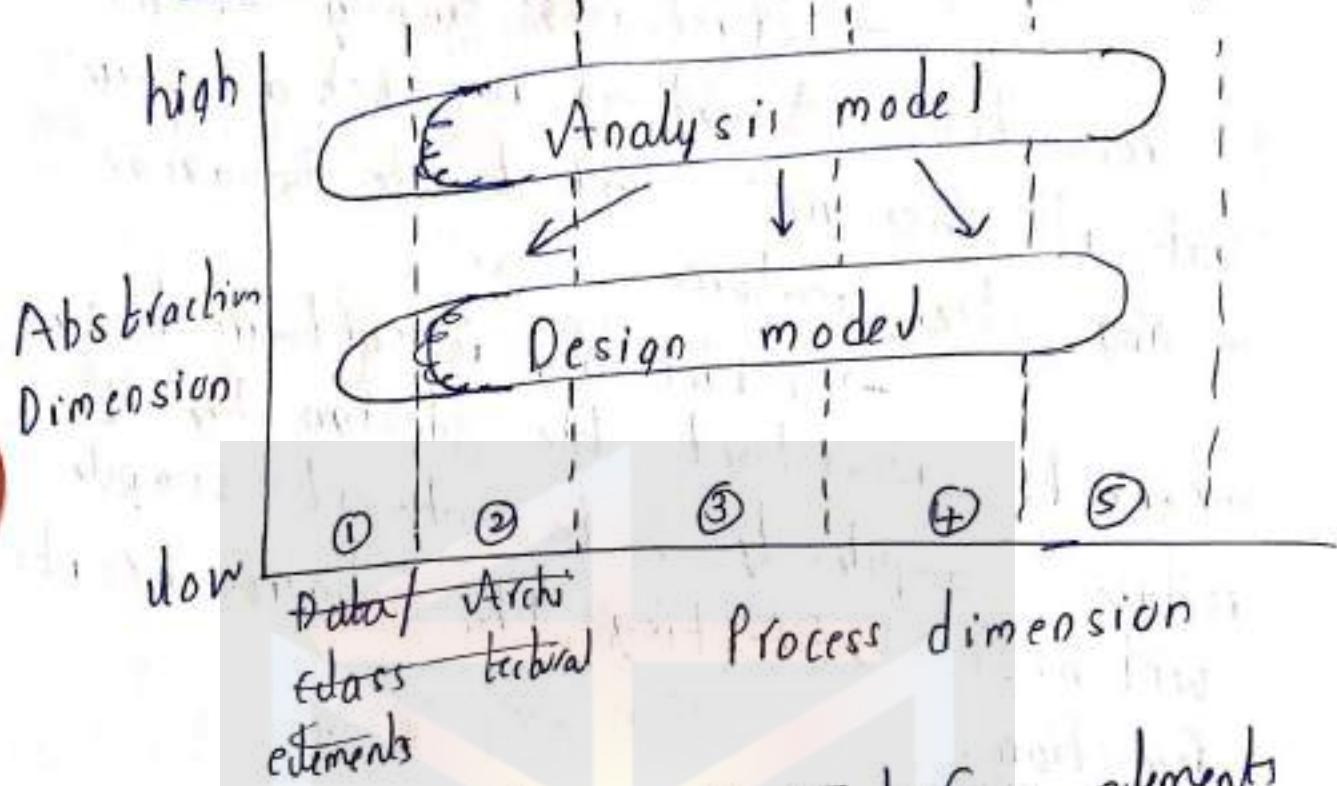
→ Refactoring simply means to reconstruct something in such a way that it does not affect the behavior or any other feature.

→ Refactoring in software design means to reconstruct the design to reduce complexity & to make it simple without affecting the behavior or its functions.

9) Design classes

→ System classes, process class, Persistent class, user interface & business domain classes are referred as design classes.

→ The Design model:



- ① Data / class elements
- ② Architectural elements
- ③ Interface elements
- ④ Component-level elements
- ⑤ Deployment-level elements

→ The design model can be viewed in two different dimensions.

- (horizontally) The process dimension indicates the evolution of the parts of the design model each design basic executed.

- (vertically). The abstraction dimension represent the level of details as each element of the analysis model is transferred into design model & then iteratively refined.

→ elements of the design model use many of the same UML diagrams used in the analysis model

→ They are only refined & elaborated.

→ more implementation specific is provided.

→ Data design elements: - It is also referred as "Data architecturing". Data design creates a mode of data and/or information that is represented at a high level of abstraction.

→ As in any project Data plays a key role hence designing it properly further (coding, testing) helps a lot in the steps.

→ Architectural Design elements:

→ architectural design is nothing but a photocopy of the end software to archive that follow

- i) knowledge on the application domain
- ii) relationship & uml diagrams are drawn
- iii) proper architecture & styles.

→ Interface design elements:

→ Tell how information flow into and out of the system
→ includes user interface, external interfaces & internal interface.

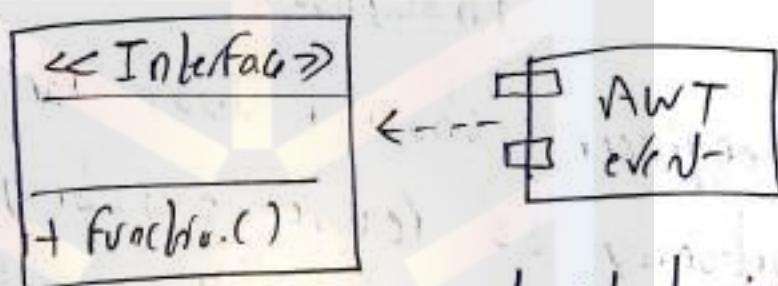
→ Component-level design

As the name suggest component-level design provides all details of a given software component.

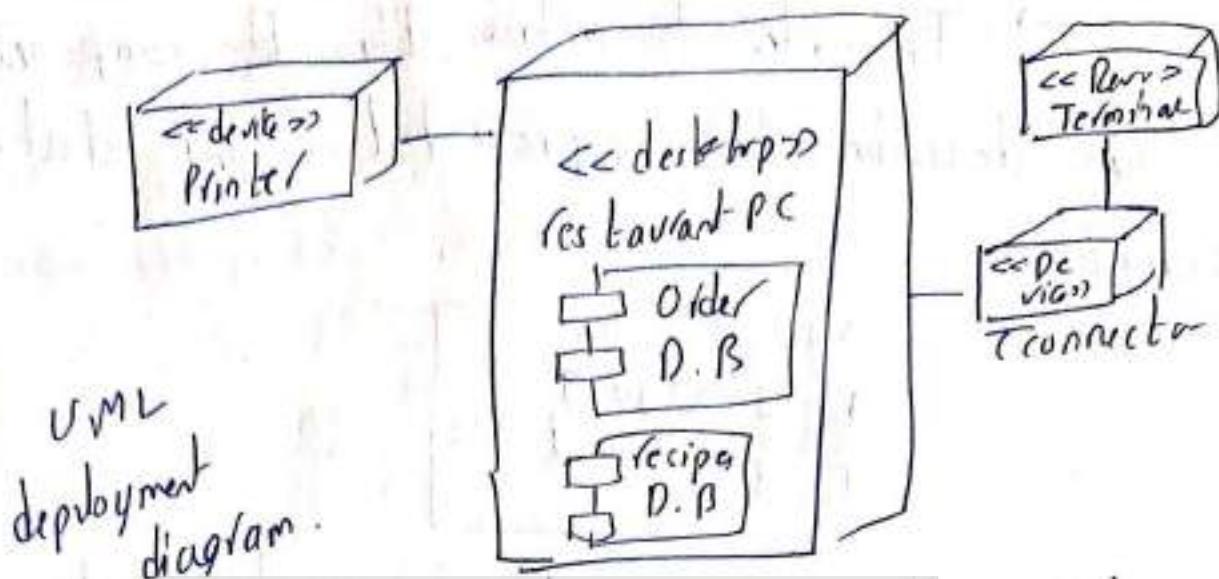
→ In order to achieve this the component design describes the representation of data structure.



The component is inscribed into the interface (along with).



→ Deployment - derived design
These indicate how software functionality & subsystems will be allocated within the physical computing environment that will support the software.



→ Pattern Based Software Design:

Throughout the design process s/w engineer should look for every opportunity to reuse existing design patterns (when they meet need of the design) rather than creating new one.

→ mature engineering disciplines

make use of thousands of design patterns

→ Design Pattern Template:

Pattern name: Describes the importance of the pattern in short (short & sweet name)

Intent: Describes the pattern & what it does (its uses)

Also known as: Lists any synonyms for the pattern.

Motivation: Provides an example of the problem

Application: Notes specific design situations in which the pattern is used

~~Participant structure~~: Describes the responsibilities of the classes that are required to implement the pattern

~~Participants~~: Describes the responsibilities of the classes that are required to implement the pattern

Related patterns: Cross references related to design patterns.

→ Using Patterns in design: Design pattern can be used throughout s/w design. The problem ~~described~~ description is examined at various levels of abstraction to determine if it related to one or more types of pattern.

i) Architectural patterns: These patterns

- define overall structure of software
- indicate relationship among subsystem & components.
- Define rules for specifying relationships among the elements (class, components, packages, subsystems) of the architecture

ii) Design patterns: These patterns address a specific element of the design such as an aggregation of components to solve some design problem.

iii) Coding patterns: They are also called idioms, these language specific patterns generally implements an algorithm element of a component, a specific interface protocol, or a mechanism for communication among components.

→ Framework: A Framework is not an architectural pattern, but rather a skeleton with a collection of plug points (also called hooks & slots) that enable it to be adapted to a specific problem domain.

Unit 4

Architectural Design, modeling component-JDL design, Performing user interface design

→ Architectural Design / ^{software}Architectural design

→ Software architectural design
represents the structure of data & program components that are required to build a computer based system

→ It is not an operational software (code or any) but its just the representation

→ software architecture provides a uniform, high level view of the system to be built.

→ It depicts (show) :-

- Structure & organization of the software components
- Properties of the component.
- The relationship (connection) among components.

→ Importance of Software Design:-

→ A software architecture describes a scenario, through which various components interact & communicate with each other.

→ It provides an overview of each software development aspect which remains important for overall success of the software development.

→ Representation of software architecture are an enabler for communication between all stakeholders interested in the development of a computer-based system.

→ Data Design:

Data Design actions translate data objects ~~as part of the analysis~~ into data structure, further into a database architecture

→ Data Design at architectural lvl :-

In today's software business environment, where a lot of data & Database are used to, the challenge is to extract useful information,

→ To solve this challenge, business IT community has developed data mining techniques, also called knowledge Discovery in Database (KDD), to extract important info from database.

→ An alternate solution called a Data warehouse is a large, independent DB that has access to data that are stored in DB.

→ Data Design at the component lvl:

Data design at the component lvl, focuses on representation of Data structures that are directly accessed by one or more software components.

→ Principles:

- A software design & programming language should support the specification & realization of abstract data types.
- A library of useful Data structures & operations that may be applied to them should be developed.
- The representation of a data structure should be known only to those modules that must make direct use of data contained within the structure.
- All data structures & the operations to be performed on each should be identified.

→ The systematic analysis principles applied to function & behaviours should be applied to data.

→ Architectural styles & Patterns:

→ Architectural style

An architectural style is a transformation that is imposed on the design of an entire system.

(When a builder builds a house he uses an architectural style as a distinctive mechanism to differentiate other styles)

→ The software that is built for computer based system also exhibits one of many architectural style

→ each style describes a system category that encompasses (has)

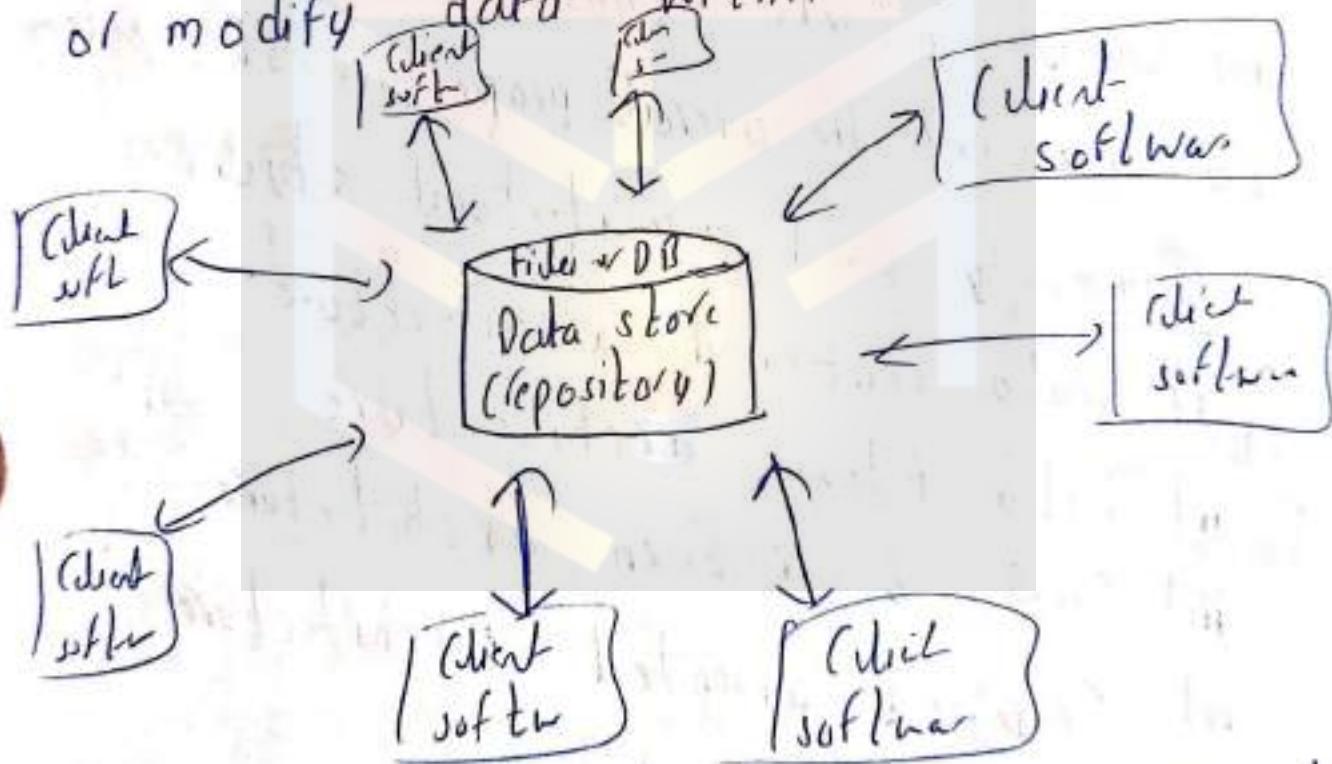
- i) A set of components: (e.g. database, computation module) that perform a function required by a system.
- ii) A set of connectors: that enables "communication, coordination, & cooperation" among components.
- iii) constraints: that defines how components can be integrated to form the system.
- iv) semantic models: that enable a designer to understand the overall properties of system

Commonly used architectural styles

- i) Data centered architecture
- ii) Data flow architecture
- iii) Call & return architecture
- iv) Object oriented architecture.
- v) Layered architecture.

II Data-Centred-architecture :-

→ (from the name it is clear that data is located in center)
→ A data store (file or DB) resides at the center of its architecture & is accessed frequently by other components that update, add, delete or modify data within the store(DB).



→ Client s/w access a central repository
It accesses data independent of any changes to data/action or other client s/w

→ Data centered architecture promote integrability, i.e existing components can be changed & new client components added to the arch without concern about other clients (scalability exist).

→ Client components are independently execute process.

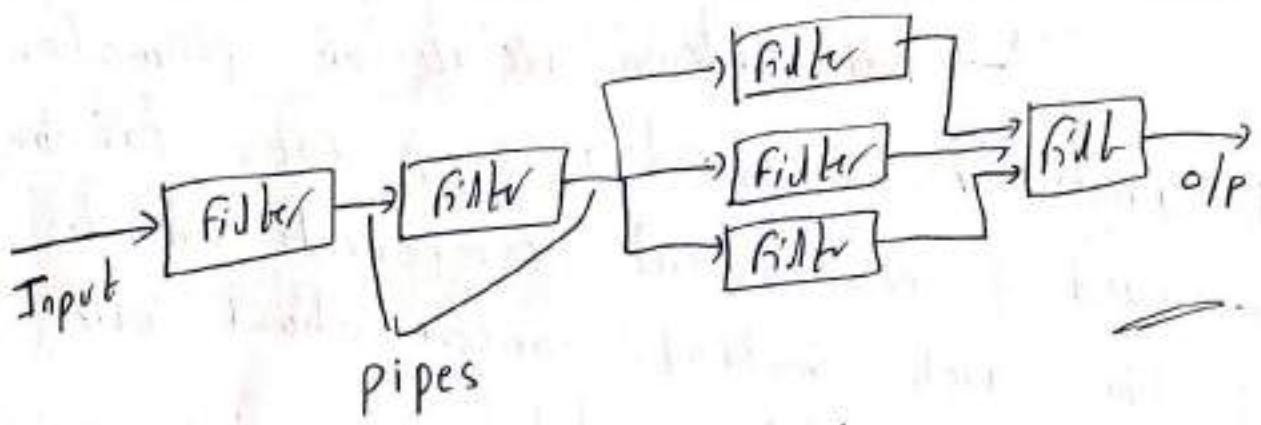
II) Data flow architecture :-

→ This architecture is applied when the input data is converted into a series of manipulative components into output.

e.g → Pipe & Filter structure has a set of components called filters, connected by pipes that transmit the data from one component to the next.

→ each filter is independent & does its job

→ Pipe is just to transfer data



III) Call & return architecture

→ This architecture style enables a software designer to achieve a program structure that is relative easy to modify & scale.

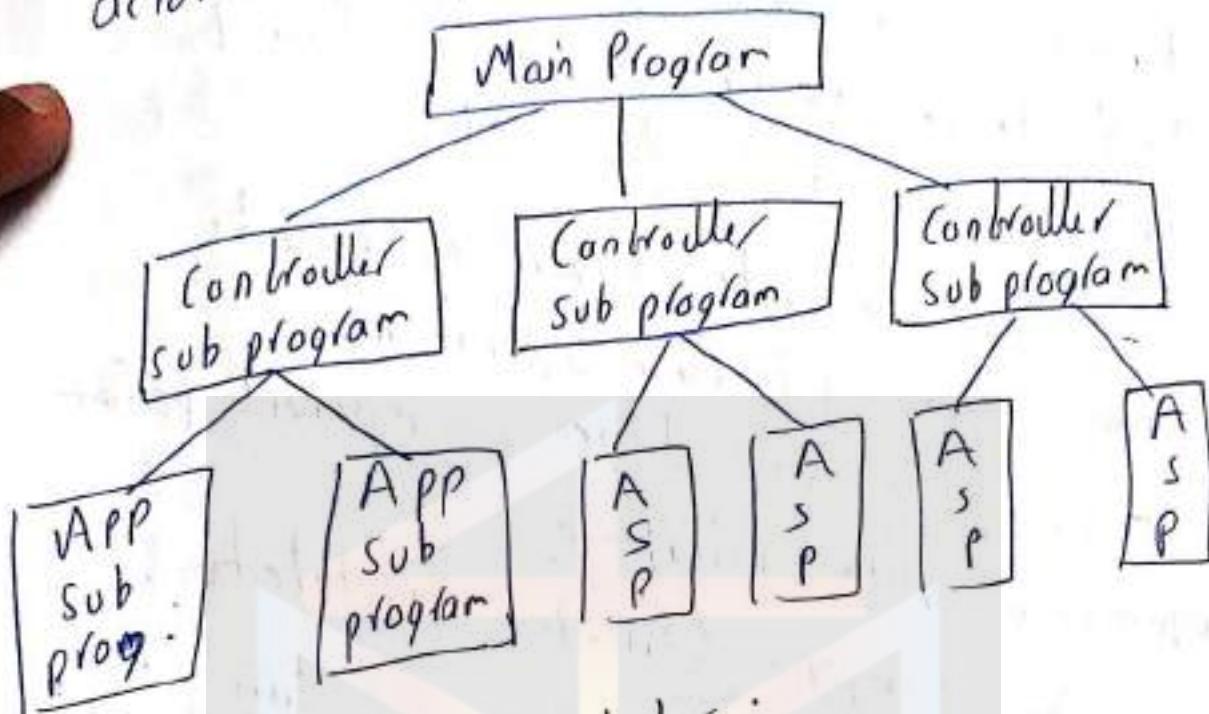
i) main program/sub program archi

The classic program structure decomposes function into a control hierarchy where a main program invokes a number of program components which in turn may invoke more components.

ii) Remote procedure call architecture

The components of the main program

/ sub program arch. are distributed across multiple computers of a network.

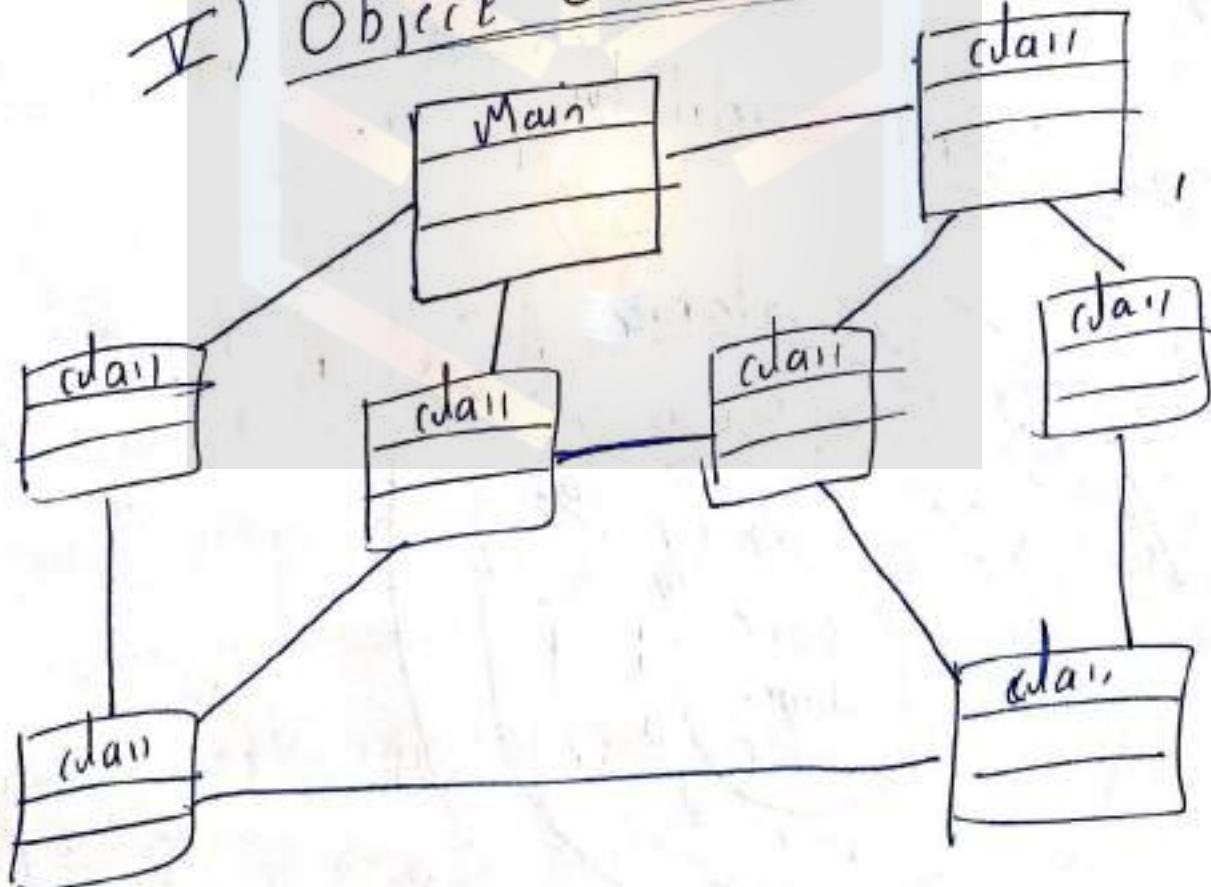


IV Layered Architecture :-



- A no. of different layers are defined, each accomplishing operations that progressively becomes closer to the machine instruction set
- At the outer layer components service user interface operations.
- At the inner layer, component perform operating system interface

II) Object Oriented architecture



- The components of a system encapsulate data & the operations that must be applied to manipulate the data
- The communication & co-ordination b/w components is accomplished via message passing.

→ Architectural Patterns:

→ Architectural Patterns

i) Concurrency: Now a days, almost every software application is suitable crafted to achieve parallelism/concurrency.

eg a) Task Scheduler Pattern: Here there will be existence of multiple active objects (each having an operation Tick(✓))

→ The tasks scheduler by executing tick(✓) activates the object to perform their task once done it returns.

↳ System Process management

b) Operating System Process management
pattern :- built in OS feature for
allow components to execute concurrently.

ii) Persistence:- Any data is said to be persistent if it gets remained even after the completion of execution of process has created.

→ In many organizations the immediate way to store the persistent data is either in data base or in the formal way (system files).
includes → Database management system

→ Application and Persistence pattern

iii) Distribution:

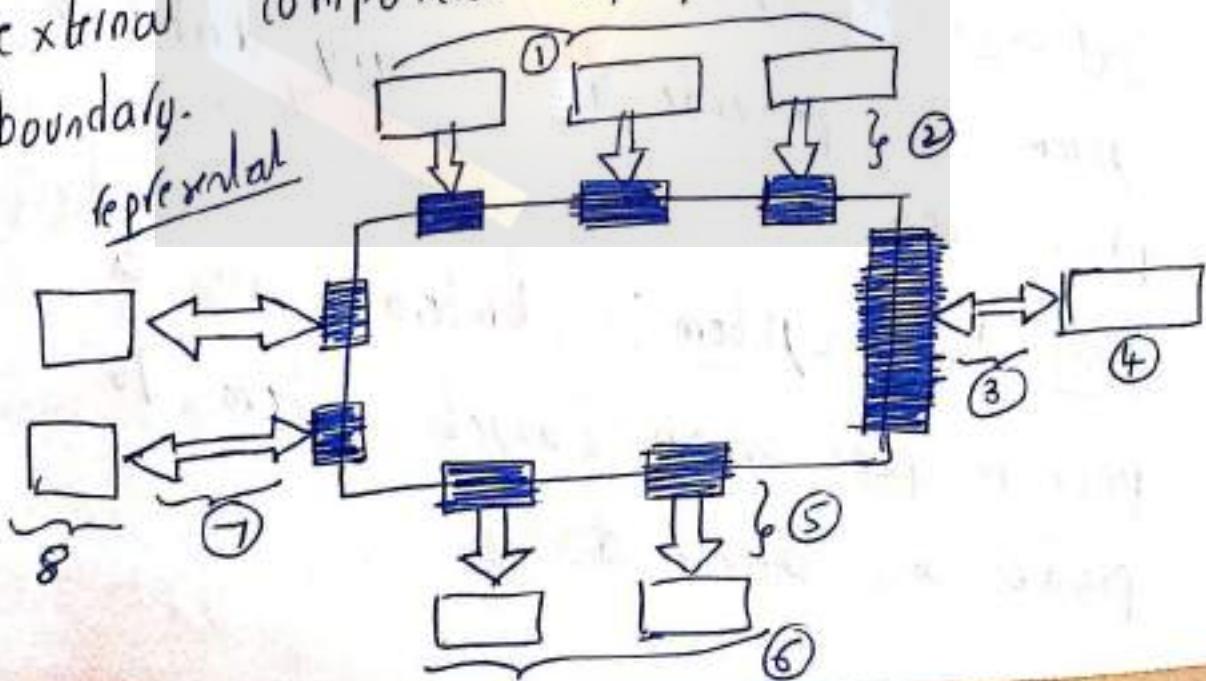
In this case, the process of communication implemented b/w various components of a given system (when they are far apart).
(Hence we use a broker pattern where broker is like intermediate component).

→ Architectural design:

- i) Architectural design using context design
- ii) Defining Archetype
- iii) Refining Architecture into components
- iv) Defining initialisation of the system

I) Architectural Design Using Context Design

We usually refer architectural context diagrams in order to diagrammatically represent the scenario in which a given software performs interaction with various external components lying away from its boundary.



1) Super-ordinate system

2) Used by

3) Uses

4) Peers

5) Depends on

6) Sub ordinate system

7) Uses

8) Actors

9) Target

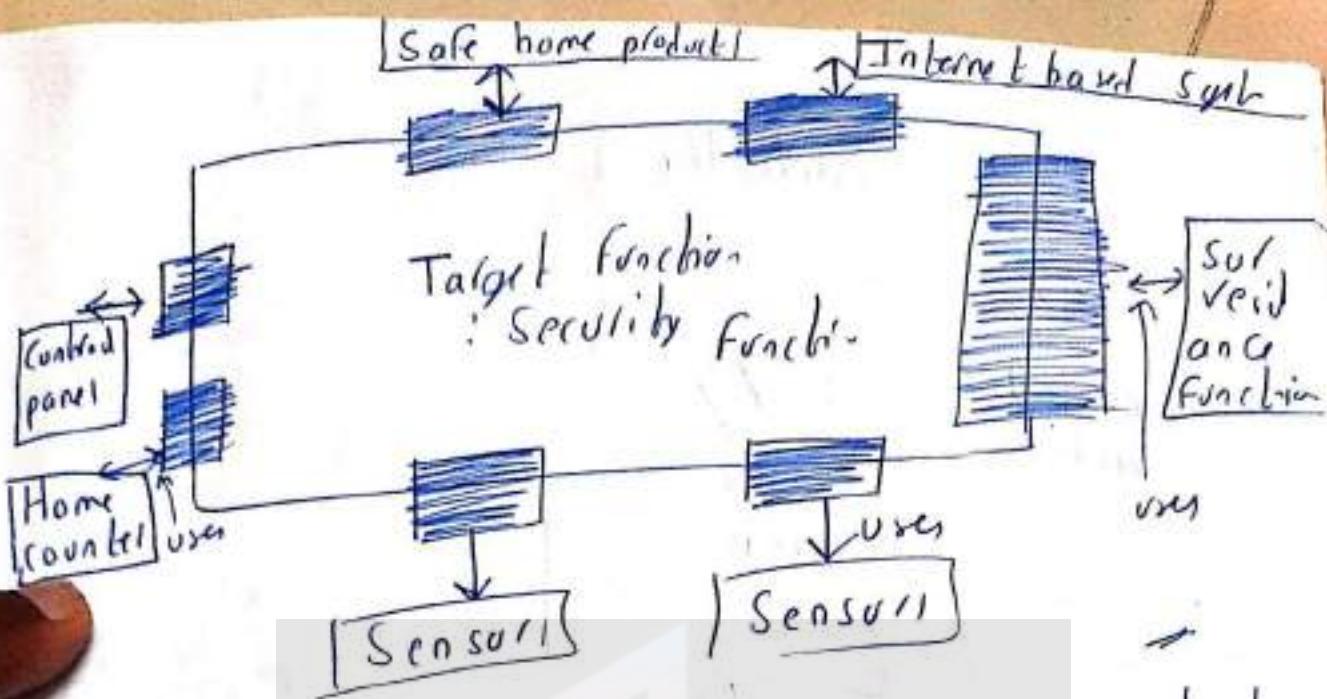
system.

~~Subordinate systems: other all system which function along with target systems.~~

Superordinate systems: systems that use the target system as a part of some higher level processing scheme.

Subordinate systems: used by target system & provide necessary data or processing.

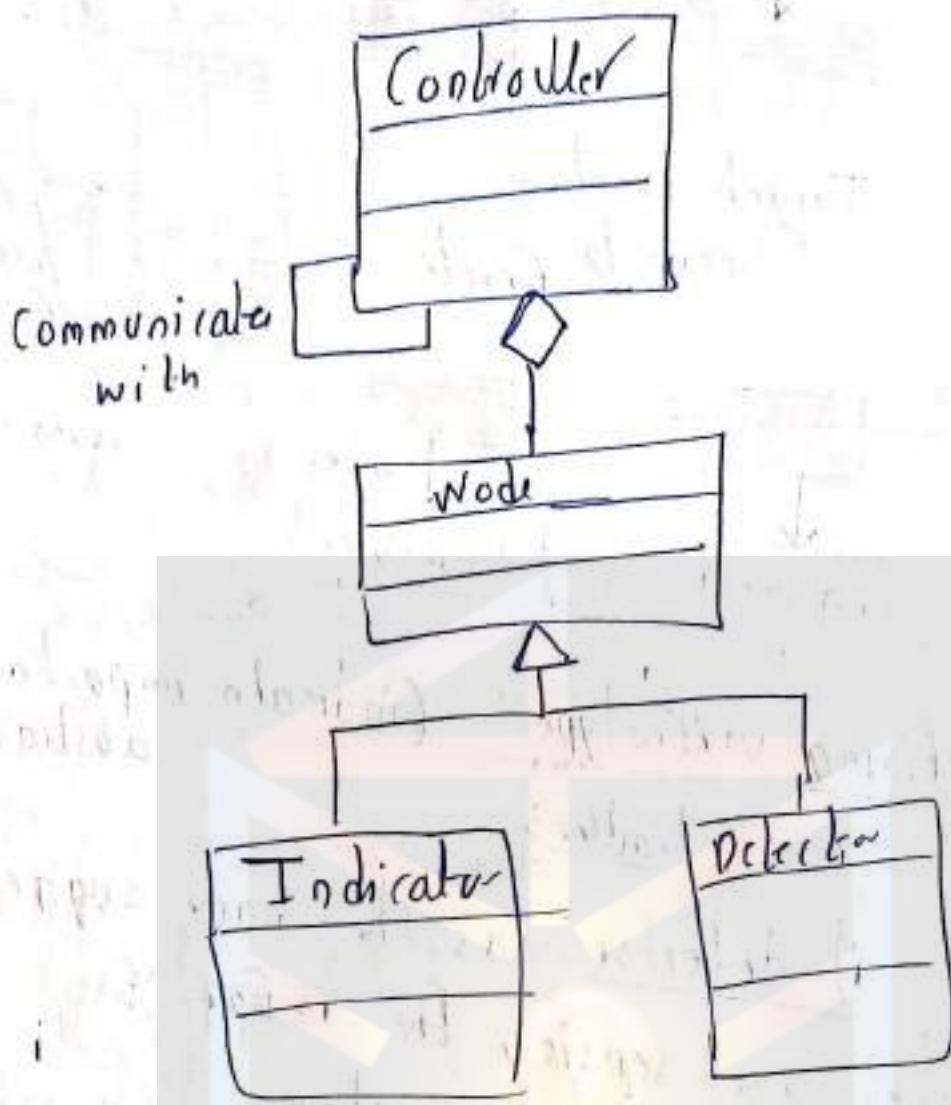
Peer-JvJ system: interact on a peer to peer with target system to produce or consume data.



II) Defining Archetypes: (indicate important abstraction)

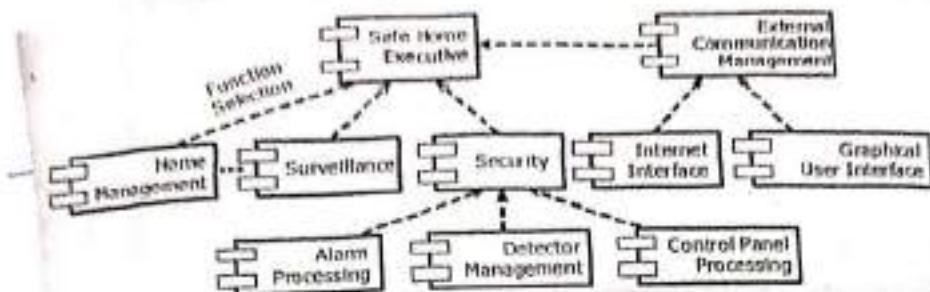
~~A) Controller~~

- i) detector: as the name suggests, it refers to sensing the information & delivering to any end system.
- ii) Indicator: it indicates the issue / status
- iii) Note: it collects several input & output
- iv) Controller: based on i/p decision all made & controlled using logic

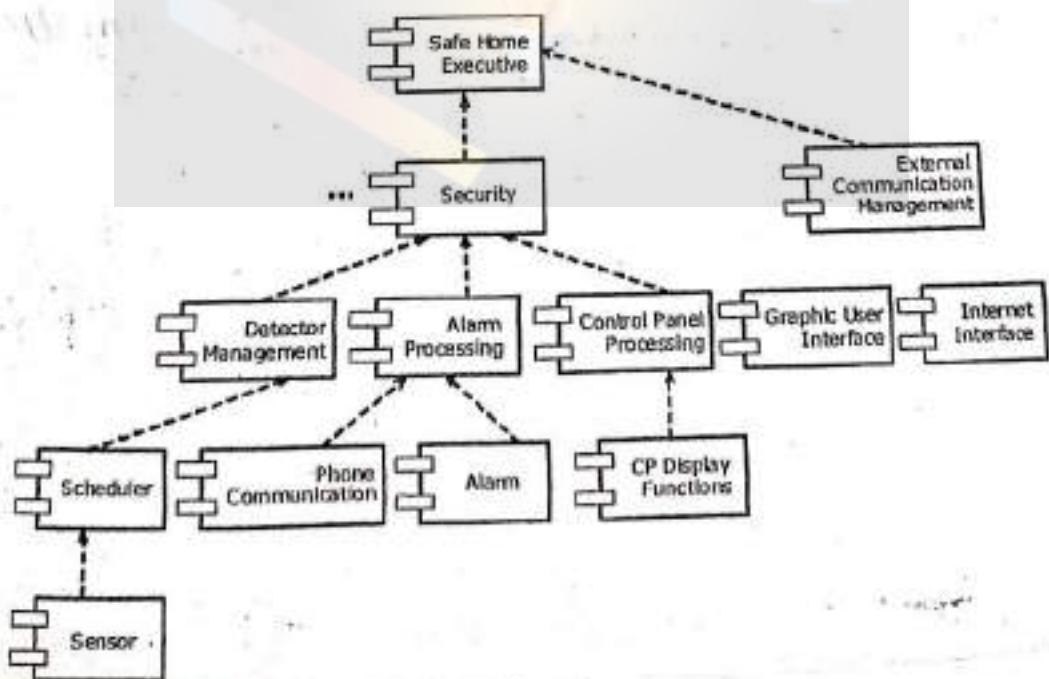


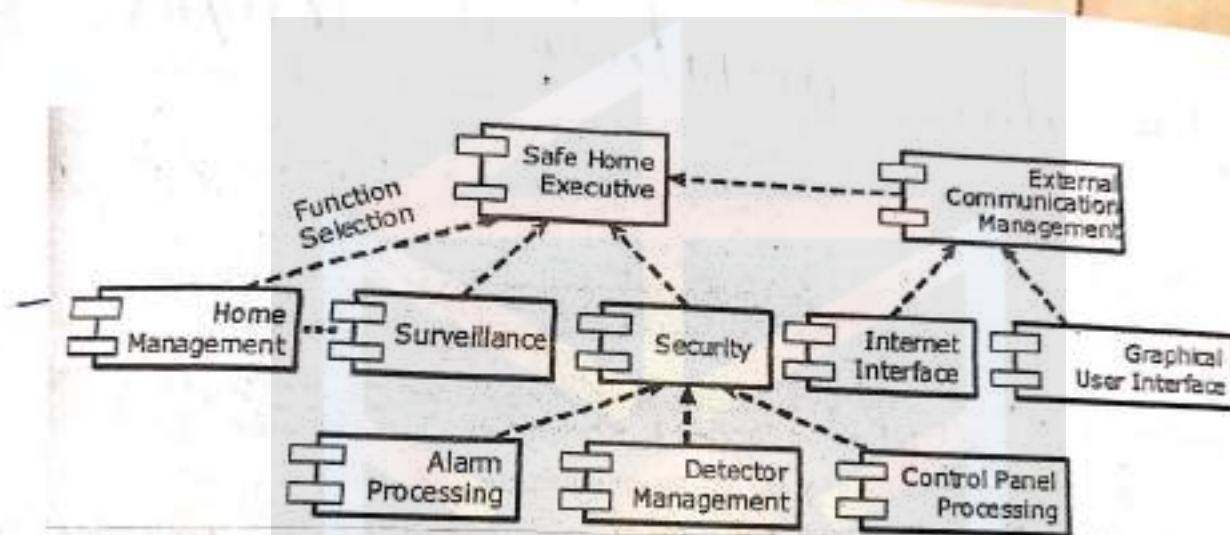
III) Refining Architecture into components

Once we begin with the process of refining the architecture, into its constituent components, the process itself marks the beginning of structure of the end system.



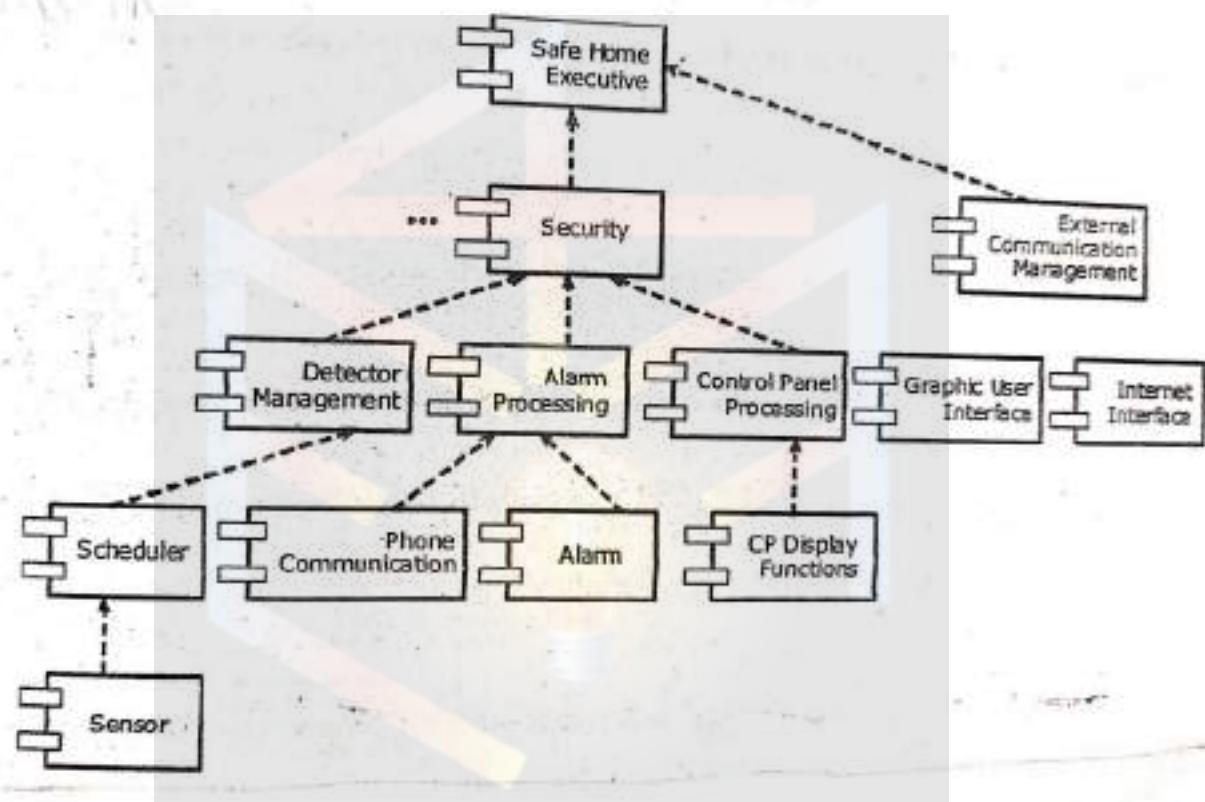
IV) Defining instantiation of the system:
 once context design, Archetype
 & overall system architecture done
 Here we actual instantiation of
 the system architecture is developed.
 (Further refinement is done iteratively).





IV) Defining instantiation of the system
 once context design, Arch by

(further refinement is done iteratively).



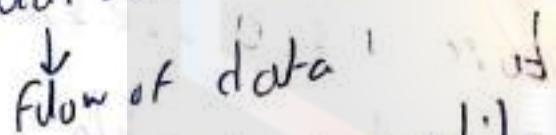
→ Alternative architectural designs:

→ Design results in a no. of architectural alternatives that are assessed to determine which is the most appropriate for the problem to be solved.

→ It can be a hybrid of multiple architecture if it best fits for the requirements (cost & approval from customer).

ATAM (Architecture Trade off analysis method)

- Developed by software engineering Institute (SEI)
- an iterative process
- Perform required activities in sequence.

- i) Collect Scenario's
→ usecases are developed to represent system from the user point-of-view.
- ii) Elicit requirements, constraints & environment description
→ information required for requirement engineering.
- iii) Describe architectural style/pattern chosen to address scenario & requirement
sub programs → (model view, process view, data-flow view) are represented.

flow of data
- iv) Evaluate quality attributes independently
e.g.: reliability, performance, security, maintainability, flexibility are checked & evaluated.

v) Identify sensitivity point for architecture (may attribute, bug)

vi) Critique candidate architectures
(from step 3) using (step 5)

These are the 6 steps based

on 5 & 6 , architecture are evaluated selected & moved to next iteration

II) Architectural complexity:

The overall complexity of a proposed architecture is assessed by considering the dependencies b/w component with the architecture.

i) Sharing dependencies: represent

dependence relationships among ^{consumers} producers who use the same source of

who produce for the same consumer.

iii) flow dependencies: represent dependence relationship b/n producers & consumer of resources

iii) constrained dependencies: represent constraints on the relative flow of control among a set of activities.

III) Architectural description language (ADL):

- provides syntax & semantics for describing s/w architecture
- provides designers:
 - with ability to decompose components
 - compose individual components (combine)
 - into large components
 - define interface b/n components (connection)

→ Mapping Data Flow in Software architecture

will be written soon / PDF will be posted.

→ Component Level design

→ Component level design occurs after the first iteration of the design

→ A component level design can be represented using some intermediate representation (e.g. graphical, tabular or text based) that can be translated into source code.

→ The design of data structures, interfaces, & algorithm should conform to well established guidelines to help us avoid the introduction of errors.

→ Component:

- A component is a modular building block for computer software.
- "A component is a modular, deployable & replaceable part of a system that encapsulates implementation & exposes a set of interfaces".
- based on different point of view
the meaning of component changes.
 - few views are
 - Object oriented view
 - Conventional view
 - Process related view

→ Designing class based component:

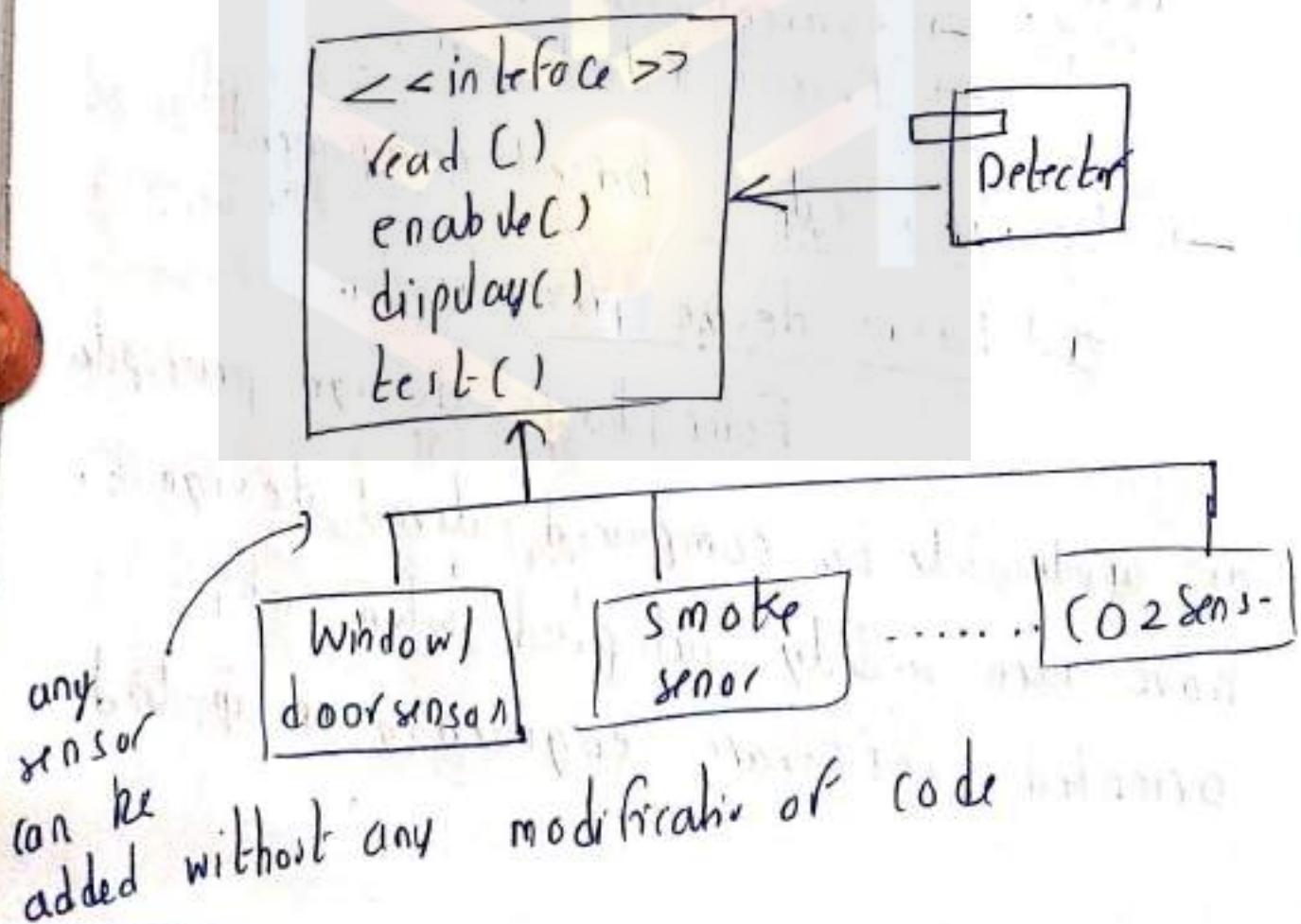
I) Basic design principles

Four basic design principles
are applicable to compound-level design &
have been widely adopted when obj-
oriented software engineering is applied.

i) The open closed principle (OCP):

→ A module or component should be open for extension but closed for modification.

→ The designer should specify component in a way that allows it to be extended without the need to make internal code or design modifications to the existing parts of the component.



ii) The Liskov Substitution Principle (LSP)

→ Subclasses should be substitutable for their base class.

For their base class:

iii) Dependency Inversion Principle:

→ Depend. on abstractions (i.e. interfaces)

don't depend on concreations.

→ The more a component depends on other concrete components (rather than on the interface) the more difficult it will be to extend.

iv) Interface Segregation Principle:-

→ many client-specific interface are better than one general purpose

interface → for several classes, specialized interfaces should be created to serve major categories of clients.

II) Component Level Design Guidelines

i) Components:

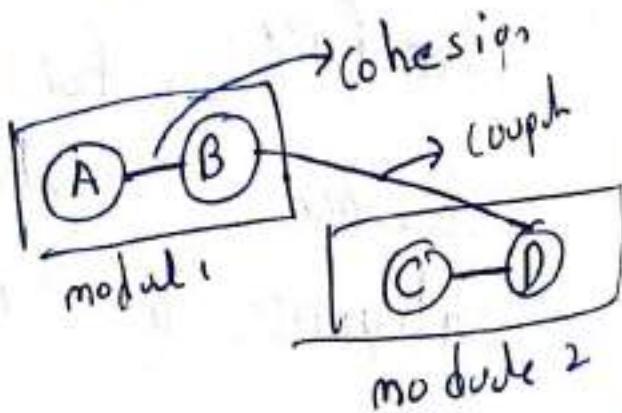
Naming convention should be established for components that are specified as part of the architectural model & then refined & elaborated as part of the component-level model.

ii) Interface:

Interface provides important information about communication & collaboration.

iii) Dependencies & Inheritance:-

It is a good idea to model dependencies from left to right & inheritance from bottom to top.



III) Cohesion : (max cohesion) is imp.

→ Cohesion is a measure of the degree to which the elements/components of the module are functionally related.

→ A good software design will have high cohesion.

function

sequence

communicational

procedural

Temporal

logical

coincidental

logical

the input for other elements data follow b/w the pair

high

Function cohesion:

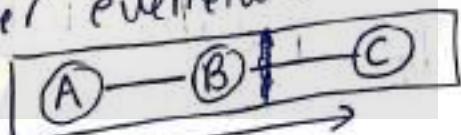
2 or more functions performing similar task should be placed in a single module (if they are related)

(A) (B)

Sequential cohesion:

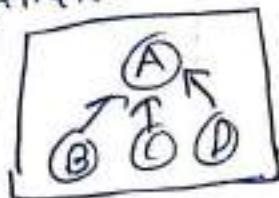
an element's outputs become some data that become

the input for other elements data follow

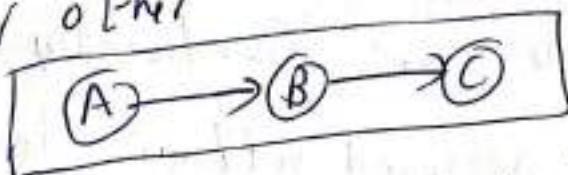


Communicational cohesion:-

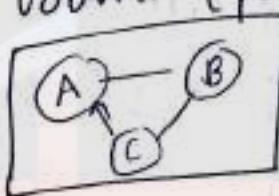
Two or more elements operate on the same input data



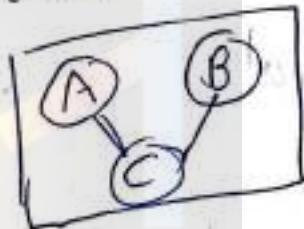
procedural cohesion: elements of procedural cohesion ensure the order of execution one after other



Temporal cohesion: elements are related by time involved (parallelly or at a time)



(Coincidental) cohesion: elements are not related to any module hence it can be kept anywhere



logical cohesion: The elements are logically related but not functionally

No. replace
elements

by task
in all above situation

IV) Coupling: (min coupling)

Coupling is the measure of the degree of independence b/w the modules. A good software will have low coupling.

types

Data coupling

stamp coupling

control coupling

external coupling

common coupling

context coupling

Best

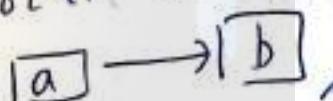
Data coupling: IF the dependency b/w the modules is based on the fact that they communicate by passing only data



Stamp coupling:

In stamp

coupling the complete data structure is passed from one module to another module



control coupling: IF the modules communicate by passing control info, then they are said to be control coupled, (Flag like).

External Coupling: In external coupling the module depends on other modules' external to the software (like DB/server)

Common Coupling: The modules have ^{global} shared data structures for ^{global} data 

Content Coupling: one module here
modifies data of another module 

→ Conducting Component Level Design:

Component level design is the definition & design of components & modules after the architectural design phase.

(or)

The designer must transform information from the analysis & architectural model, into a design representation that provides sufficient details to guide the construction (coding & testing) activity.

→ steps

Step 1: (Identify design class in problem domain)
All designer must transform informal classes that correspond to the problem domain are identified.

Step 2: (Identify Infrastructure design class).

All design class that correspond to infrastructure domain are identified

Step 3: (elaborate design classes)
All design classes that are not acquired as reusable components are elaborated

→ specify message details when
class or components collaborate

→ identify appropriate interface

for each component

→ elaborate attributes & define
data structures required to implement them

→ describe processing flow within
each operational in details.

Step 3a) :- (Collaboration Details)
message can be elaborated by
expanding this syntax in the following manner

[guard condition] sequence ((return value) = message
name
expression)

(argument
list).

o
work order
number - (WON)

: Production Job

build Job(WON)

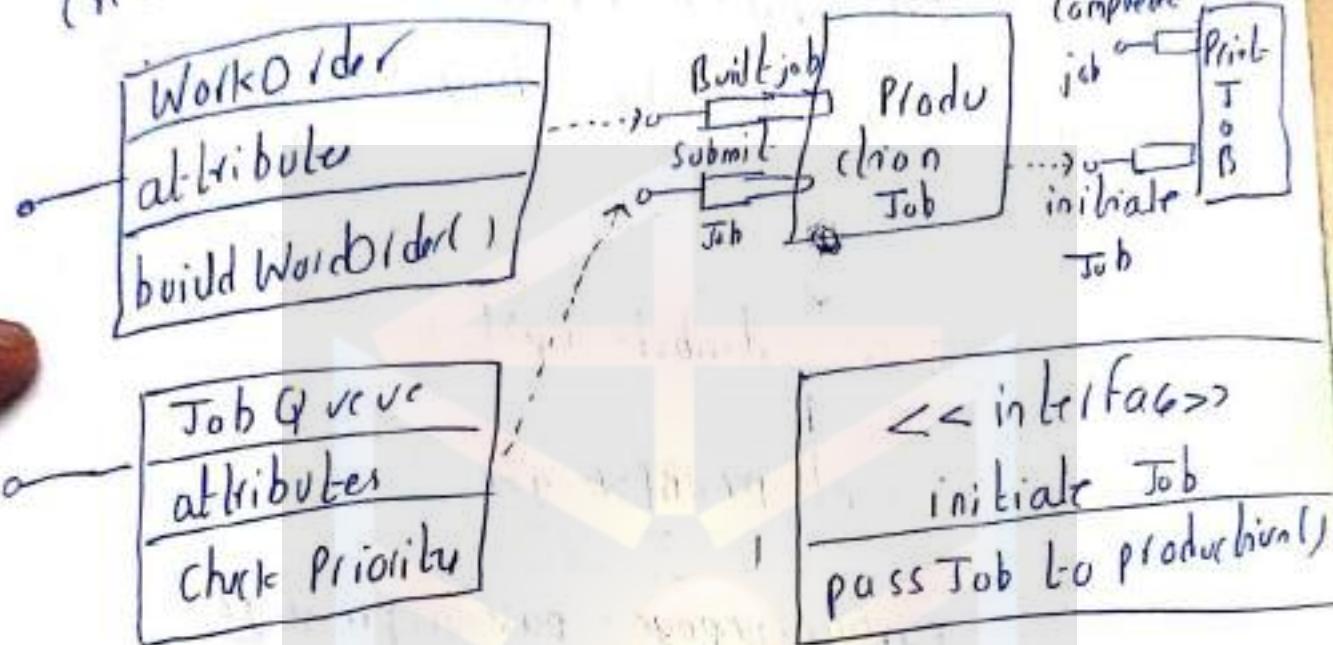
: workOrder

Submit Job()

: JobGiven

step 3b) (Appropriate Interface)

printjob interface "initiate Job", which does not exhibit sufficient cohesion because it performs three different sub functions (refactoring can be performed)



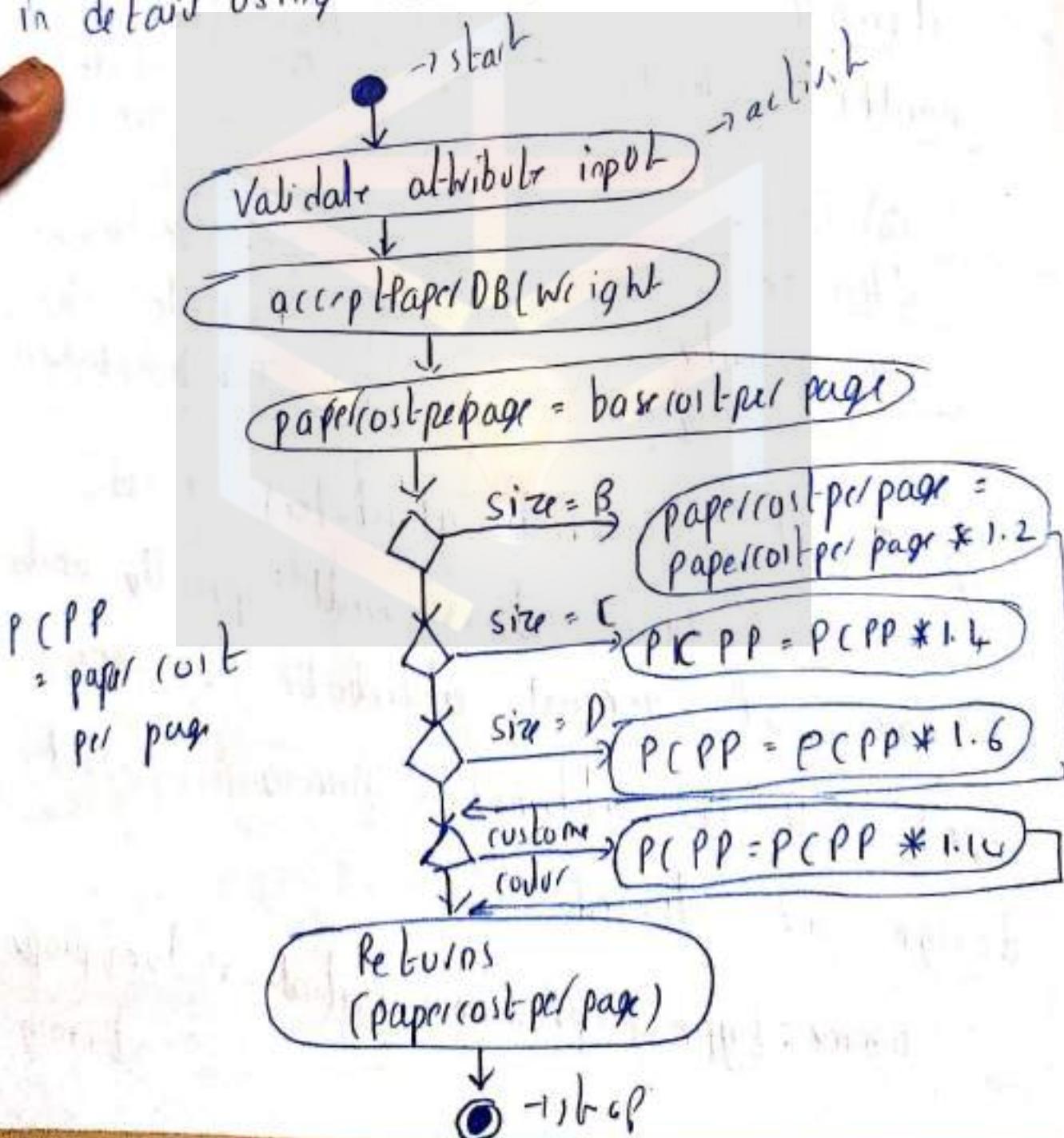
step 4 b c) (Establish attributes).

Analysis: classes will typically only list names of general attributes (ex paper) but names of the attributes during component-based. All the attributes during component design are listed

name:type-expression = initial-value{property string}

`paperType-wrigh : string = "A" { contains
1-4, A, B, C, D }`

step 3d) (describe processing flow)
The process flow is described
in detail using activity diagram



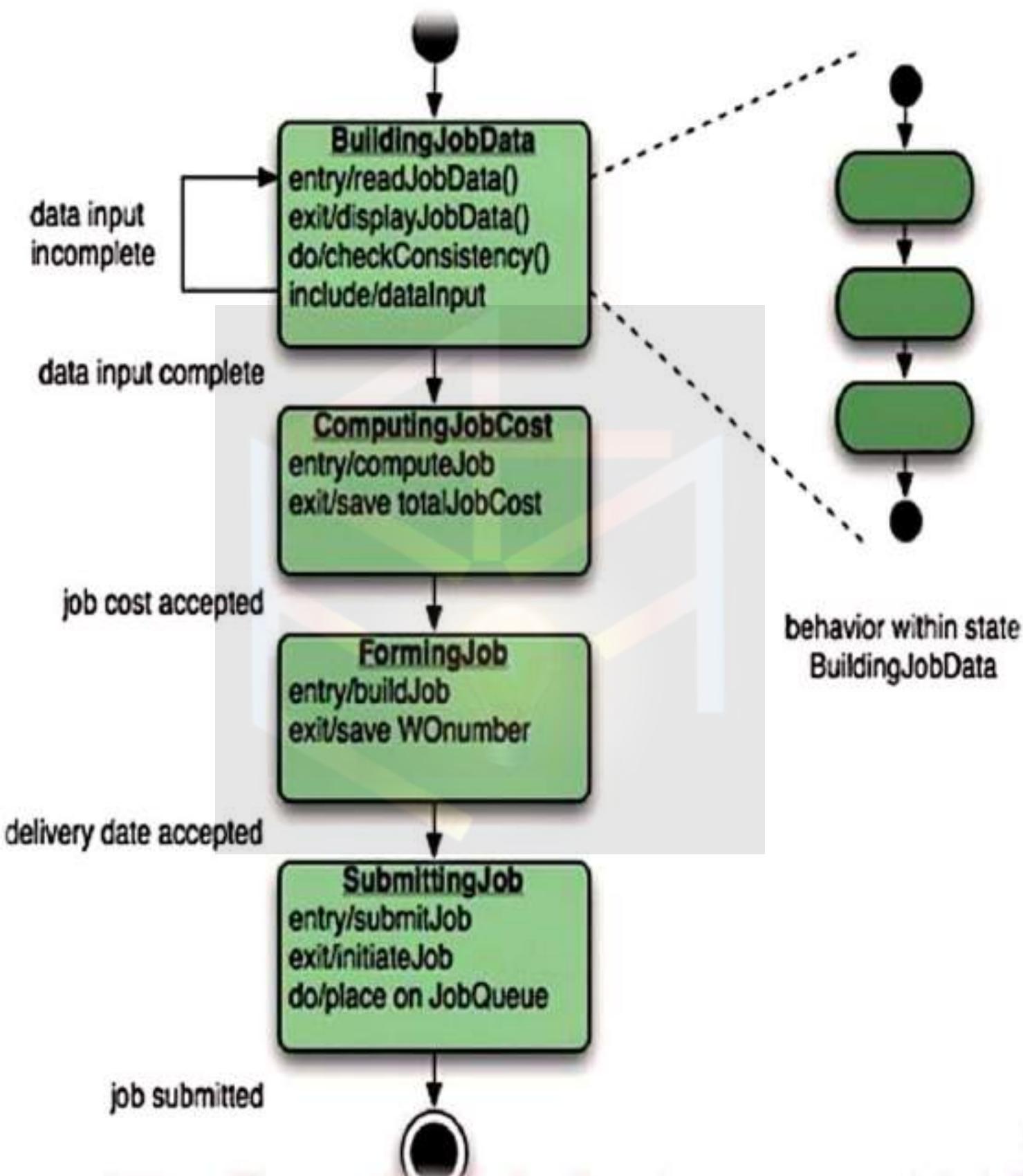
Step 4: (Persistent Data)

The persistent data sources are described (database & files) and the classes required to manage them are identified.

Step 5: (elaborate behaviour)

It is sometimes necessary to model the behavior of a design class. Transition from state to state having the form
event-name (parameter-list) [guard-condition]
/action expression.
(transition) of state during work flow

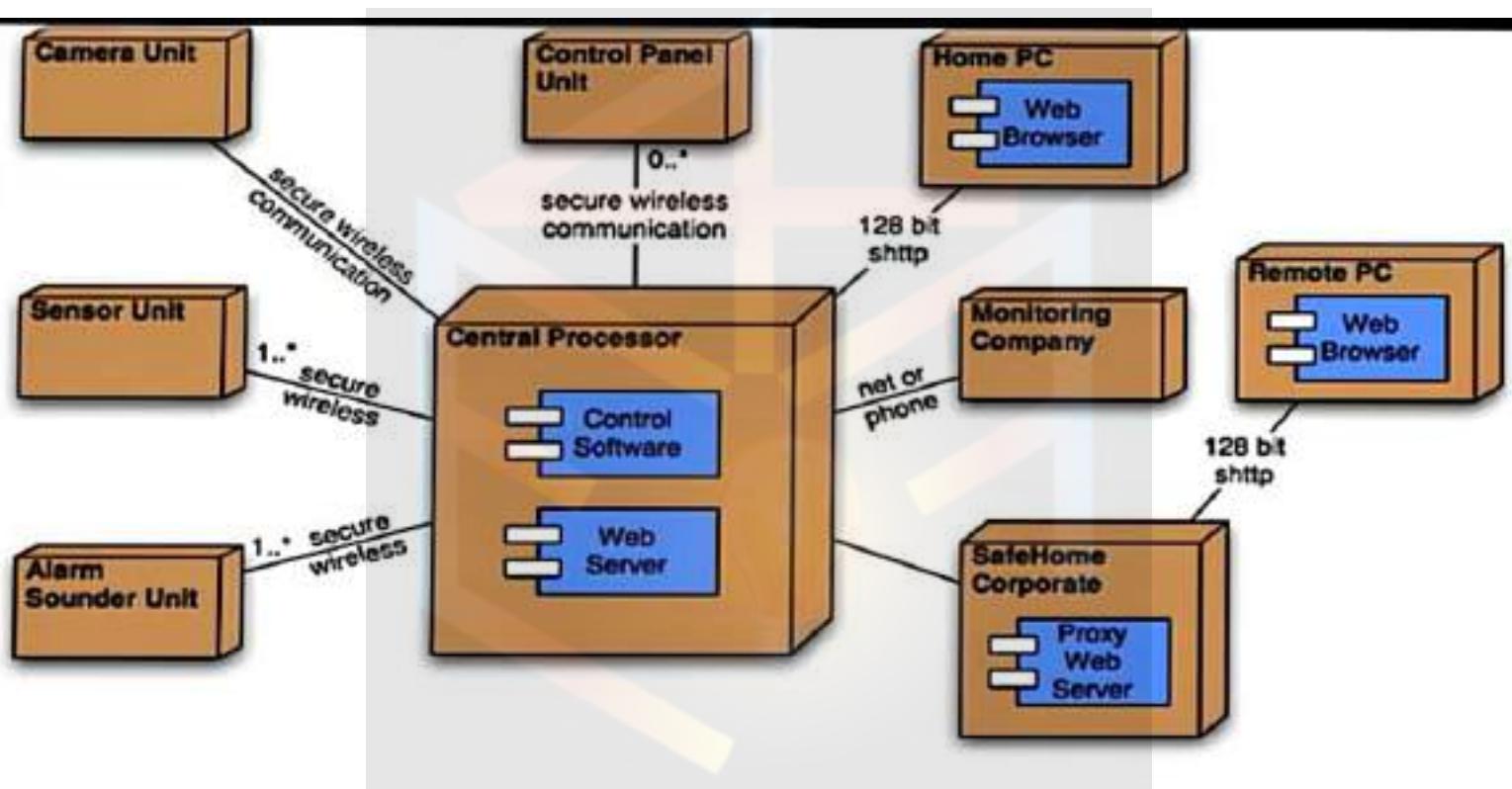
Dig..



Step 6:- (elaborate Deployment Diagrams:-)
Deployment diagram or elaborated
to represent the location of key package
of component

Brows
are
deployment
units
below
are
component
units

Step 7:- (Redesign / reconsider)
The first component-level model
specification will not be a complete, analysis
or consistent at the nth iteration you
apply to the model.



→ Object constraint language: (OCL)

^{we know UML} → It is official part of UML

^{as only diagram} → The OCL complements UML

by allowing a software engineer

to use a formal grammar &

syntax to construct unambiguous statements

about various design model elements

(ex classes, objects, events, messages, interfaces)

→ adding more functionality to
our printing job (print upper cost bound &
drop dead as delivery date).

context PrintJob :: validate(upperCostBound:
integer, costDeliveryReq:integer)

pre: upper cost bound > 0

and, costDeliveryReq > 0

and self.job Authorization = 'no'

post: if self.totalJobCost <= upper cost
bound and self.deliveryDate <=

cost Delivery Req.

then

self.jobAuthorization = 'yes'

end if.

it should pass pre condition
then post condition

Supported by:

Rational Rose, ArgUML, Eclipse,
poseidon/octopus etc..

→ Designing conventional components:-

→ Conventional design constructs

emphasize the maintainability of a

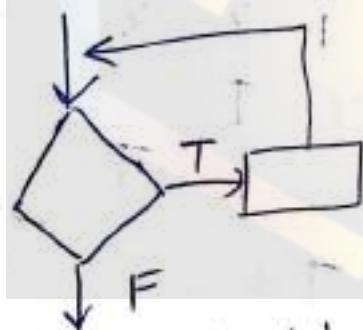
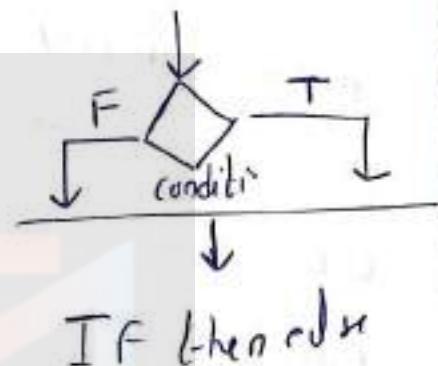
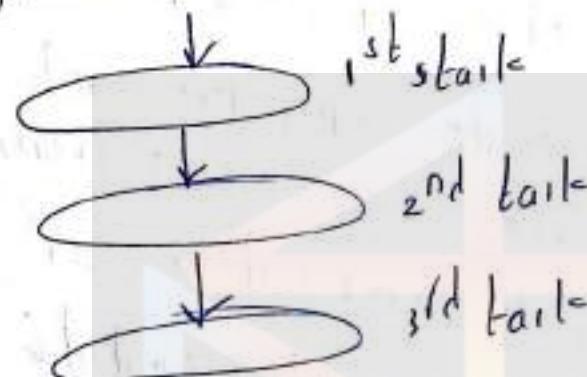
function/procedural program

- sequential, conditional & repetition

→ each construct has a predictable
logical structure where control enters at
the top & exits at the bottom enabling
a proper & early flow

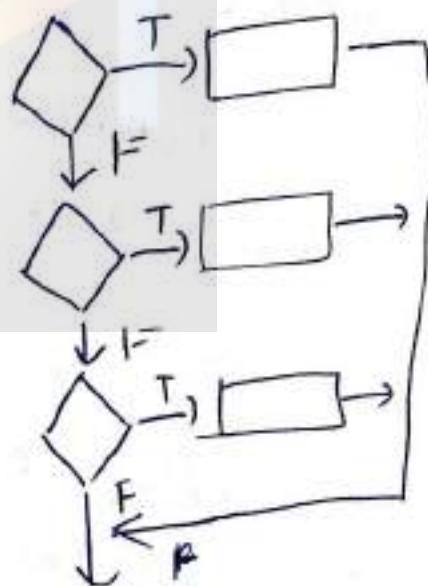
i) Graphical Design notation

The activity diagram allows a designer to represent sequence, condition & repetition all elements of structured programming called flow chart



of repetition
(loop)

With 1 or 2 lines data
according to marks



selection
(switch)

iii) Tabular Design Notation:

- list all actions that can be associated with a specific task (procedure)
- list all conditions (of decision made) during execution of the procedure
- associate specific set of conditions with specific action, eliminating impossible combinations of conditions (maintaining possible permutation of conditions)

		Rules			
		1	2	3	4
Condition	Condition A	T	T		F
	" B		F	T	
" C		T			T

Action	X	✓	✓	✓
Action Y				
" Z		✓	✓	
"				✓

→ User interface design:

→ User interface (UI) is the front end application view to which user interacts in order to use the software.

→ UI design focuses on

- The design of interfaces b/w software components
- The design of interfaces b/w the software & other nonhuman producers & consumers of information.
- The design of ~~the~~ interface b/w human & the computer

→ Graphical user interface (GUI's) have helped to eliminate many of the most irritating interface problems (using command line).

→ UI should be

- attractive
- simple to use
- responsive in short time
- clean to understand

→ Golden rules/principles (Golden rules)

There are three main golden rules given by "The Mandel"

- i) Place the user in control
- ii) Reduce the user's memory load
- iii) Make the interface consistent.

→ User interface design begins with the identification of user, task & environment requirements. Once user tasks have been identified, user scenarios created & analyzed to define a set of interface objects & actions

I) Place the user in control :-

→ Allow user interaction to be interruptible & undoable

- The user shall be able to easily interrupt a sequence of actions to do something else (without losing the work that has been done so far).

- The user be able to undo any action.
- hide technical internal from the casual user
 - The user should be stress free
 - The user should view only the output of the program, he should not be made aware of the implementation details of the program (or) abt system.
- design for direct interaction with objects that appear on the screen
 - The user, just by viewing once should be able to identify the purpose of a given object
ex: audio icon to play audio.
- Provide for flexible interaction:-
 - The user shall be able to perform the same action via keyboard, movement or touch etc.,
(Based on user preference).

→ Define interaction model in a way that does not force the user into unnecessary/undesired actions

The user shall be able to enter & exit mode with little or no effort.

→ Streamline interaction as skills develop & allow the interaction to be customized

The user shall be able to use a macro mechanism to perform a sequence of repeated interactions & to customize the interface.

II) Reduce demand on short term memory load
Reduce the User's Memory load → M.M

→ Reduce demand on short term memory:
- When users are involved in complex tasks, the demand on short term memory can be significant.

- The interface should be designed to reduce the requirement to remember past actions & results. (can be established by visual clues). [ex: calculator having past calculation history]

→ Establish meaningful defaults:

- The initial set of defaults should make sense for the average user, but a user should be able to specify individual preference, too as per requirement (e.g. amazon product search) (size company name etc...)

→ Define shortcuts that are intuitive:-

- When mnemonics are used to accomplish a system function (~~ctrl~~ + P for print) these also should be easy to remember control - print (ctrl + P) (first letter),

→ The visual layout of the interface should be based on a real world metaphor

- The screen layout of the user interface shall contain well understood visual

clues that the user can relate to
(real world action)

→ Disclose information in a progressive

Fashion

when interacting with a task, an object or some behavior, the interface shall be organized hierarchically by moving the user progressively in a step-wise fashion from an abstract concept to a concrete action.

(sign up, login pages, forms).

III) Make the interface consistent:

→ Allows the user to put current task into a meaningful context

An application will have many screens, pages, tabs, etc.. (hence giving a proper hints, icons, color coding) that enable the user to know the context of the work at hand (and time) ^{e.g.} website.

[continuation]

→ Maintain consistency across a family of applications

A set of apps performing complimentary functionality shall all implement same design rules so that consistency is maintained for all interactions.

e.g. Adobe Photoshop, Adobe Illustrator

→ If past interactive models have created user expectations, do not make changes unless there is a proper reason

Once a particular interaction sequence has become an actual standard, the application continues this expectation, irrespective of new functionalities. e.g. websites



→ User interface analysis & Design:-

The overall process of analyzing & designing a UI begin with the creation of different model of system model system function.

4 modules

i) User profile model:- (established by

human engineer or software engineer).

→ establish the profile of the end-user of the system. (Based on age, gender, education background & various other considerations)

→ This is an important criteria since, it is the end user who is going to deal with the product for ever developed by a given software development team.

→ Categorize user as

- Novices (new & inexperienced)
→ No syntactic knowledge of the system, little semantic knowledge of the application, only general computer usage.

- Knowledgeable, intermittent users
→ reasonable semantic knowledge of the system, a bit low syntactic knowledge.

- Knowledgeable, frequent users
→ good semantic & syntactic knowledge.

ii) Design model:- (created by a software engineer)

It is derived from analysis model of requirements & also based on users of the system.

iii) Users Mental model:- (Developed by user)

when interacting with the application).

→ often called the user's system perception
→ consists of the image of the system that users carry in their hands.

i) Implementation model :- The interface look & feel coupled with supporting information that describe interface semantics & syntax.

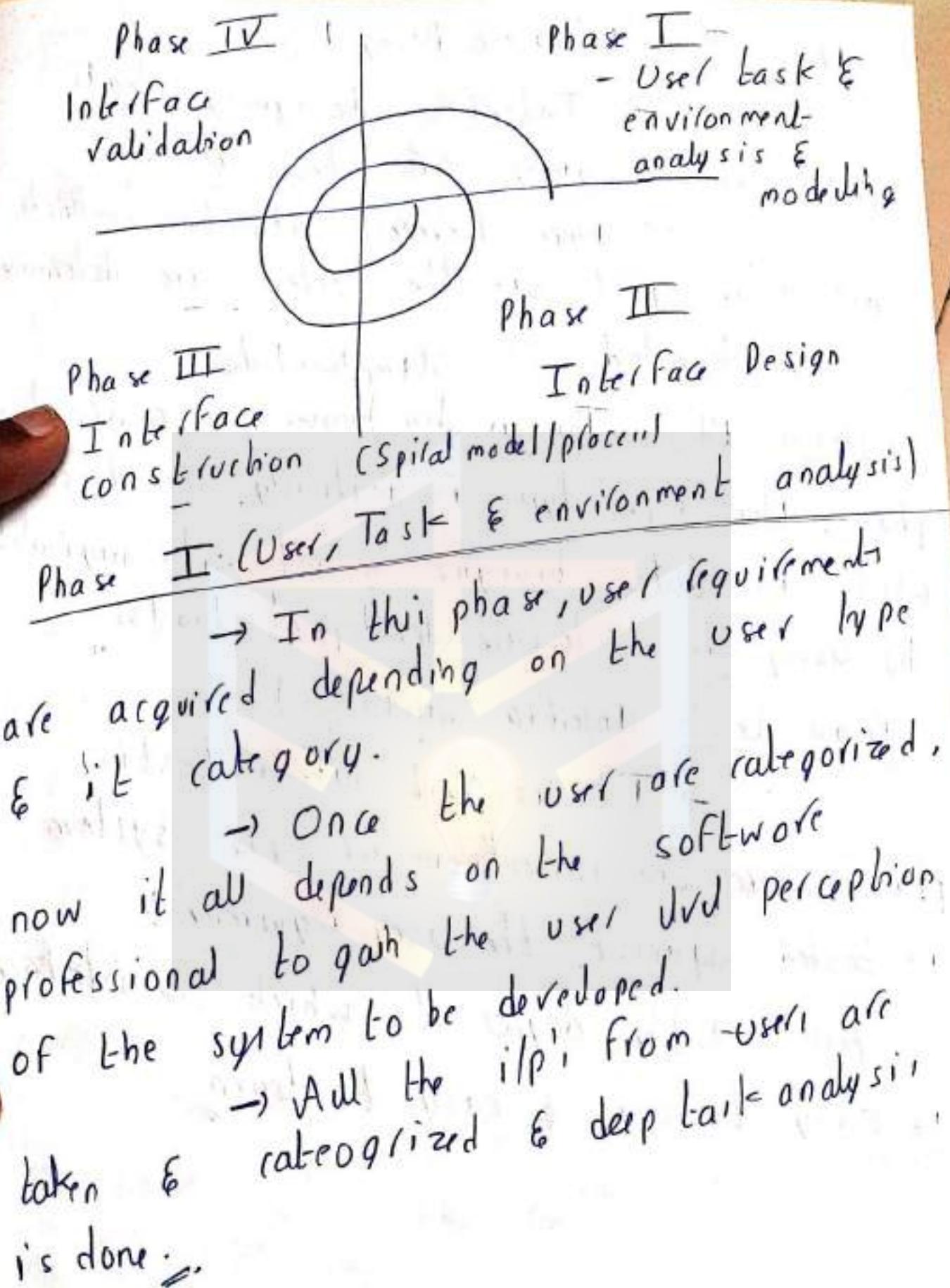
→ Design / Process :-

→ user interface analysis & design process is an iterative process & it can be represented as a spiral model.

It consists 4 frame work activities

- i) User task & environment analysis
- ii) Interface design (objects & action) (requirements gathering)
- iii) Interface construction (user interface development tools are used)
- iv) Interface validation.

↓
(correctness)
of system &
validation !



Phase II : (Interface Design) :-

- Interface designing deals with the designing of the system.
- Here different objects, which are active part of the system are determined & implemented.

(Implementation)

Phase III :- This is also known as construction phase. Here a prototype is initially constructed, which is later brought up into its originality by using the interface development tools.

Phase IV : (Interface Validation)

→ It is just like a testing phase where the correctness of the system is tested against the user requirement.
tested → the degree to which the interface is easy to use & easy to learn.

→ Interface design steps:-

→ end of interface analysis marks the begining of interface design issues.

→ Interface design is a ^{by step} ~~step~~ process in which each step may get repeated multiple times & the new step generated contains upgraded qualities than previous steps.

Steps:-

→ Initially identify the object & its associated operations. This is usually done by traversing through the use case written. later these are elaborated.

→ Now, categorize these objects under the following types i.e., source, application & target objects.

→ Transfer the source objects onto the target objects, hence, in this way a hard copy of the application is created.

→ When all possible objects are identified & their relative operations are defined, then start creating the screen layout

→ To make the layout attractive, several text, graphic images, icons etc.. are placed at suitable locations. (Layout be interactive!)

→ Various Design issues associated with the User Interface Design

While designing anything we do face multiple issues.

If Response Time is system response time is the primary complaint for most interactive application.

→ It is nothing but the time taken by the system to respond to a user activity.

→ In general, system response time is measured from the point at which

the user performs some control action
(e.g. hit return button or clicks a mouse).
until software responds with desired info
or action
→ min response time is suggested.

iii) help facilities/options:-
→ Almost every software includes
help topics which may guide the users for
the proper usage of the software.
→ Almost every user of an
interactive computer based system requires
help now & then.
Nowadays → In most cases, however, modern
software provides online help facilities
that enable a user to get a question
answered or resolve a problem without
leaving the interface.

iii) Error handling :-

→ every error message or warning produced by an interactive system will have specific characteristics

- The message should describe the problem in descriptive that the user can understand.

- The message should provide constructive advice for recovering from the error.

- The message/error should never blame on the user.

iv) Menu & command labeling :-

→ The typed command was once the most common mode of interaction b/w user & system software & was commonly used for applications of every type (now labels are commonly used even though in back end commands are executed).

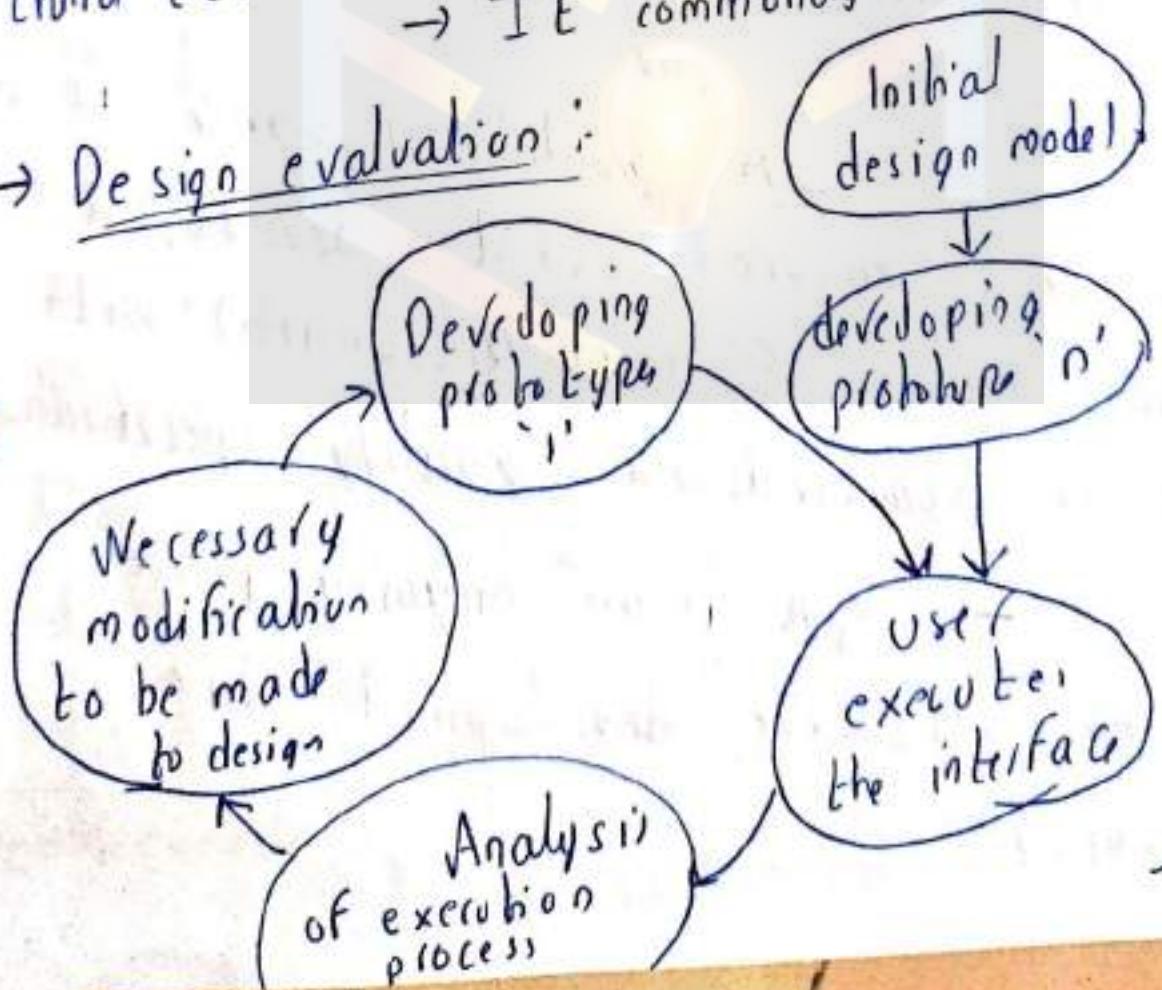
v) Application accessibility:-

As computing application become more common now a day, software engineer must ensure that interface design encompasses mechanism that enable easy access for those with special needs.

vi) Internationalization:-

→ As when using global applications if a user open's the application the language should automatically be in china etc... → It commonly found now.

→ Design evaluation:-



Unit - IV

Software Quality assurance, Testing
strategies, Testing Strategies, Testing
Tactics, Product metrics

→ Software Quality assurance (SQA)

SQA is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedure as well as standards suitable for the project & implemented correctly.

(or)

SQA quality assurance is a process that ensures that developed software meets & complies (consists) with defined or standardized quality specification.

→ SQA is an ongoing process within the software development life cycle (SDLC).

SQA activities:-

- prepare sqa plan for the project.
- participate in the development of the project's software process description.
- review software engineering activities to verify compliance within the defined s/w process.

SQA activities:-

i) SQA management plan :-

Make a plan how you will carry out the sqa throughout the project.

ii) Set The check Points:-

SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

iii) Multi testing strategy:- Donot depend

on single testing approach. when you have a lot testing approaches available use them.

iv) Measure change Impact :-

The changes for making the correction of an error sometimes re-introduce more errors. Keep the measure of impact of change on project.

v) Manage Good Relations :-

In working environment managing the good relation with other teams involved in the project development is mandatory.

→ Elements of Software quality assurance:

i) Standards: The IEE, ISO & other standard organizations have produced a broad array of software engineering standards & related documents.

Hence the Job of SQA is to ensure that stand. that are adopted & followed properly / perfectly.

iii) Reviews & audits: Technical reviews are a quality control activity performed by software engineers for the project.

→ Audits are also performed to check how well they are following standards.

iii) Testing: Software Testing is a quality control function, that have one primary goal - to find errors.

→ The goal of SQA is to plan testcases properly & efficiently to ensure the quality.

iv) Error / defect collection & analysis: SQA collects & analyzes errors & defects data to better understand how errors are introduced & how to eliminate them.

v) Change management: Change is one of the most disruptive aspect of any software project.

→ Hence it's mandatory to check quality even after changes^(plan) made.

vii) Education: Every software organization should plan/improve its software engineering practices.

viii) Security management/ safety:-

with increase in cyber crime all the data collected in the software should be saved safely.

- as softwares in every electronic device which is use today it must be safe to use (mainly autonomous vehicles, air crafts etc..)

viii) Risk management:-

eliminating of risks is also important.

→ SQA Tasks :-

- SQA plan
- Process description
- Review work activities
- Audit work products
- Documents deviations
- Reports to senior management.

→ Goals :-

- Requirement Quality
- Design Quality
- Code Quality
- Quality control effectiveness
- Statistical software quality assurance
- Statistical SQA - Follow

4 steps

- i) information about software errors & defects is collected & categorized.
- ii) An attempt is made to trace each error & defect to its underlying cause.

iii) Using the Pareto principle (80%).
of the defects can be traced to 20% of all possible cause, isolate/solve the 20% defect first

iv) slowly start solving 20% & you will end up solving all.

common defects identification:

- software defects can be identified before & after the publish/use.
- incomplete testing
- ambiguous or inconsistent
- error in design logic
- error in data representation
- inaccurate or incomplete documentation
- error in programming

etc

→ Six Sigma for software engineering
(6 rules/principles)

- 1) Define customer requirements & deliverables, project goals etc..
- 2) measure the existing process & its output to determine current quality.
- 3) analyze defects.
- 4) improve the process by eliminating the root causes of defects.
- 5) control the process to ensure that further work does not reintroduce the cause of defects.
- 6) verify to avoid defects.

→ Reliability of software :-
Reliability of a software specifies probability of failure, free operation for a given time duration

→ ISO 9000 Quality standards:-

→ ISO (International standards

Organization) is a group or consortium
of 63 countries established to put an

→ ISO declared its 9000 series
of standards in 1987.

→ ISO 9000 standards determines
the guidelines for maintaining a quality
system.

→ ISO 9000 series of standards
is based on the assumption that if a proper
stage is followed for production, then
good quality products are bound to
follow automatically.

image of ISO certification

How to get ISO 9000 Certification?

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

ISO 9000 Certification



- 1. Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.
- 2. Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
- 3. Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
- 4. Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
- 5. Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
- 6. Continued Inspection:** The registrar continued to monitor the organization time by time.

→ Testing: Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end users.

A process of executing a program with an intent of finding an error.

→ The main objective of testing is to prove that the software product meets a set of pre-established acceptance criteria under a prescribed set of environmental circumstances.

[Testing can be done using manually / software tools]

→ A strategic Approach to software

Testing

(General characteristics)

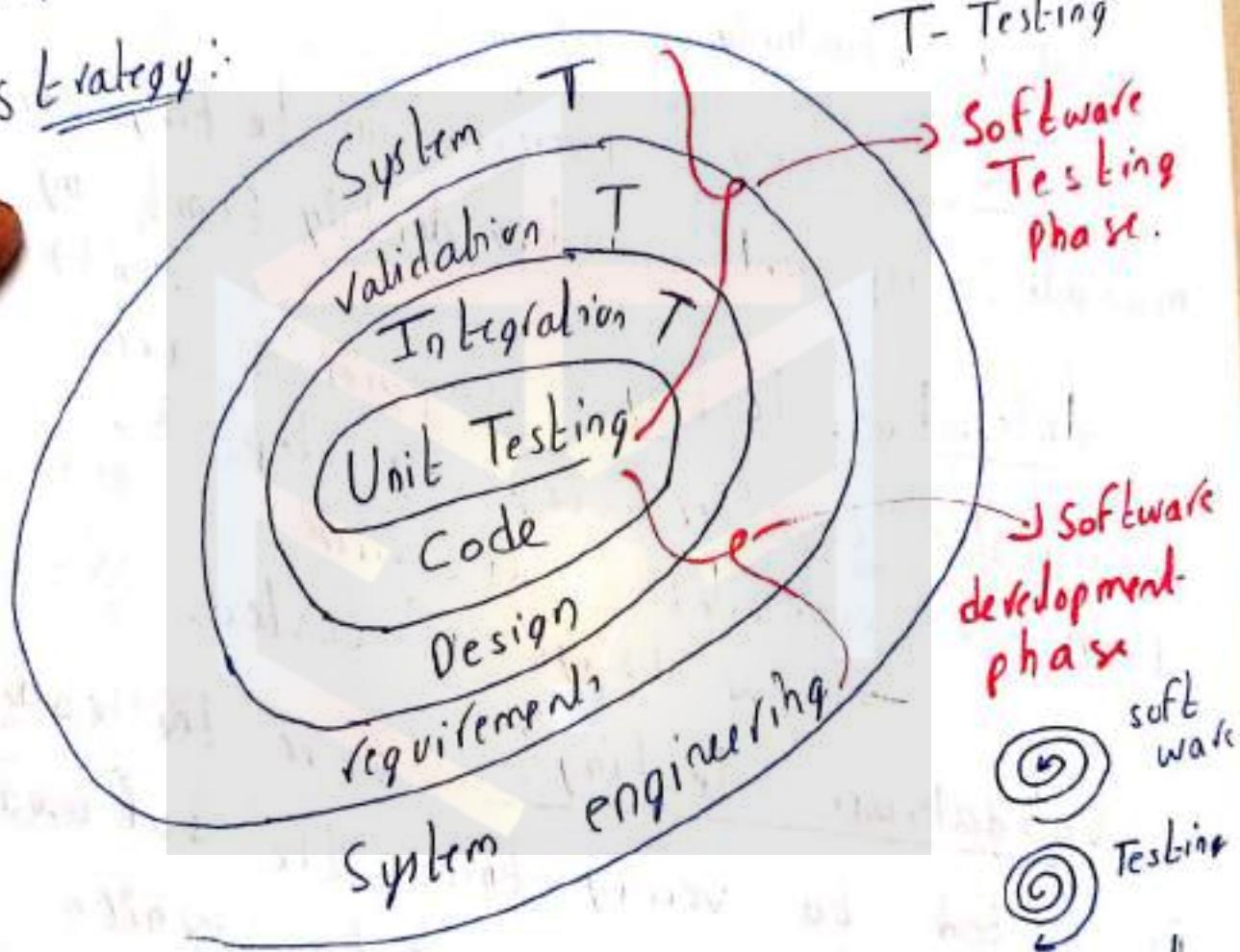
→ To perform effective testing, a software team should conduct technical reviews

- Testing begins at component-level & continues till end.
- Different testing techniques are appropriate at different points in time.
- Testing is done by developer of s/w & for large projects by an independent test group.
- Testing & debugging are different activities.
- Verification & validation are two important aspects of software testing process.
 - Verification: Does the product meet its specifications?
 - The set of activities that ensure that software correctly implements a specific function or algorithm.

→ Validation: Does the product perform as desired.

- The set of activities to ensure working as per the goal of customer requirements.

Strategy:



- Software is developed by moving inward into the spiral
- Testing is conducted by moving outward in the spiral.

System engineering: It describes the role of the software going to be developed.

requirements, design, code are already defined in unit 1 detailedly.

Unit Testing: focuses on testing each module/component independently (unit by unit).

Integration Testing: Components are integrated together step by step to form a complete software.
→ s/w design is tested.

Validation Testing: here the code is tested to verify that the software satisfies all the requirements which were laid during the requirement phase of software development.

System testing: The entire system is tested as a whole.

→ Testing strategies for conventional software

Unit testing:

→ Unit testing usually refers to the process of testing the small unit or module of the entire system.

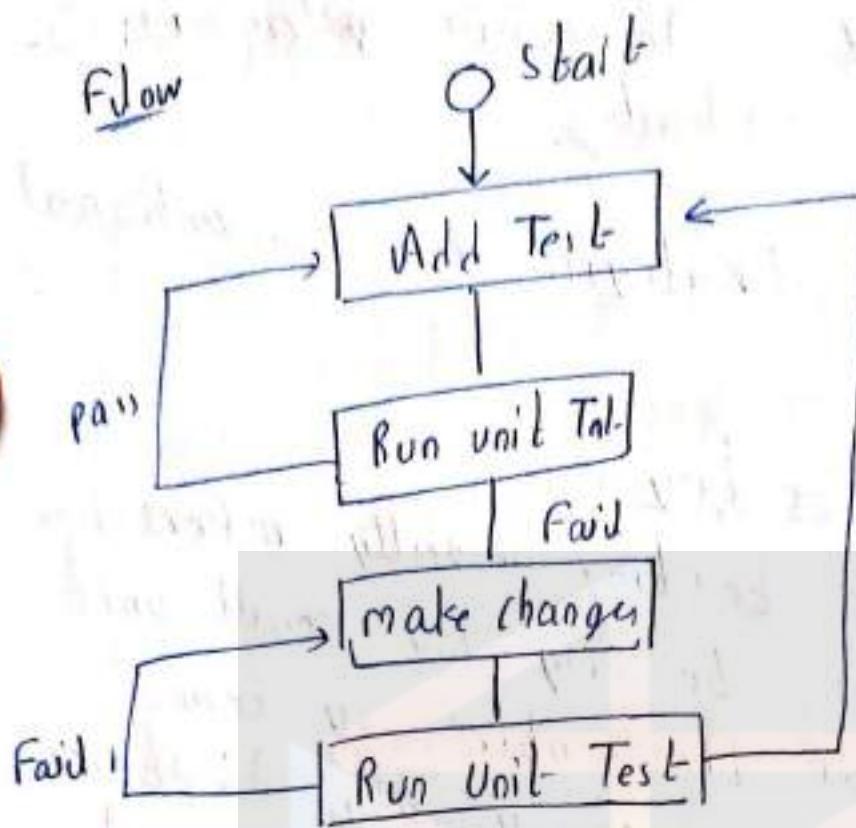
→ This process usually exposes the bugs, existing in the control path located internal to a given unit. (in that unit)

→ Unit testing can be carried out simultaneously on multiple units of the system.

code
def divider(a, b)
 return a/b
end

→ a small unit code

can be tested -
if it's returning
correct value & if again
tweak change & success



Benefits of Unit Testing :-

- makes coding practice more agile/computer friendly
- improves the quality of the code
- Helps find software bugs early
- facilitates changes & simplifies integration
- makes debugging easier & quicker
- helps create better software design
- Reduces Testing cost.

→ Targets of Unit test cases:-

- Module interface testing :-

→ ensure that information flow properly into and out of the module.

- local data structure :-

→ ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution.

- Boundary condition testing :-

→ For loops work perfectly at boundary conditions.

- Independent path testing :-

→ all statements in a module should be executed (by varying inputs).

- Error handling path testing :-

→ ensure that the algorithm respond correctly to specific error conditions

—

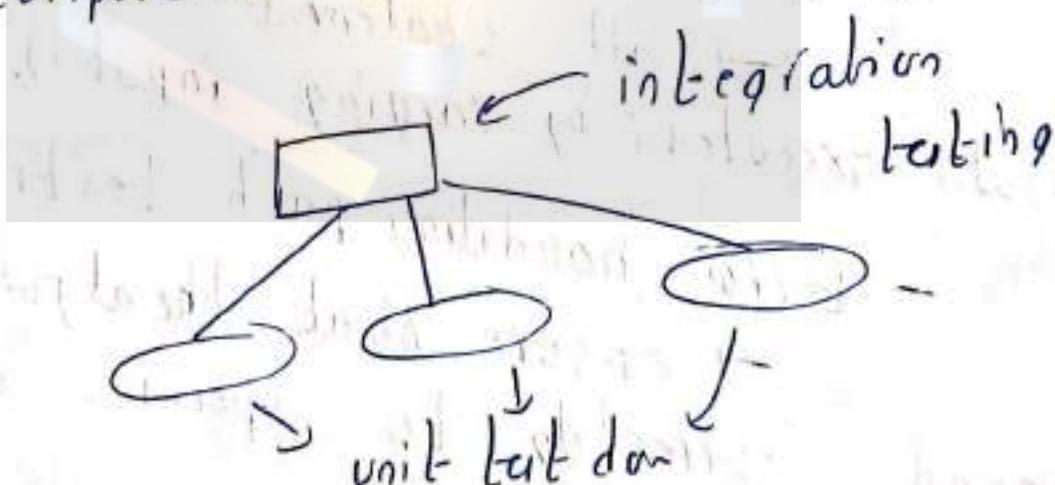
→ Integration Testing:

→ The purpose of integration testing is to expose faults in the interaction b/w integrated unit.

→ The integration testing is the process of testing the interface b/w units or modules.

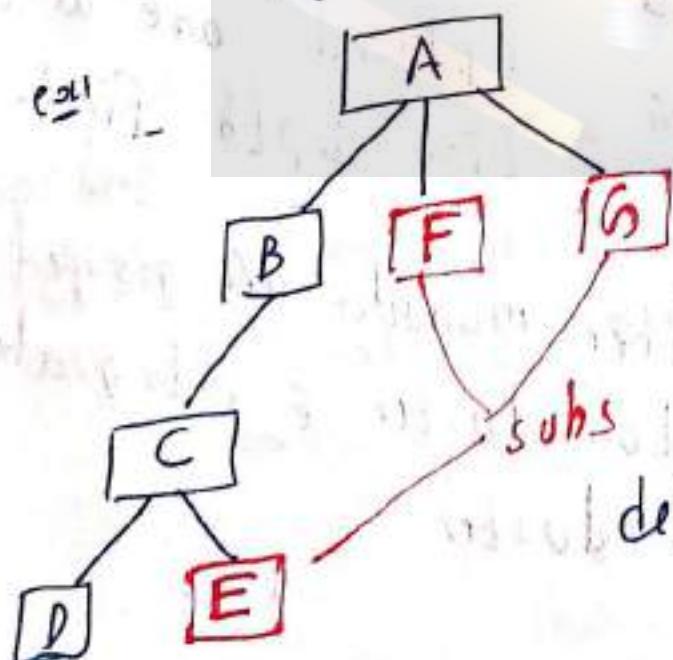
I) Big Bang approach:

→ Combining all the modules once and verifying the functionality after compilation of individual module testing.

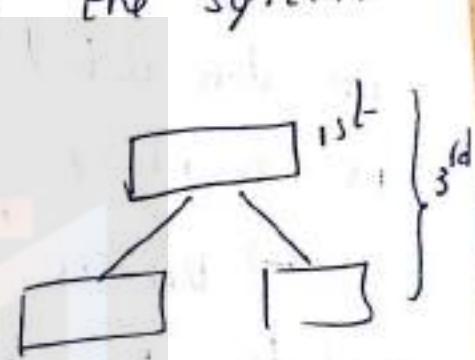


II) Top down approach:

→ In top down approach, testing takes place from top to bottom
→ high level modules are tested first and then low level modules & finally integrating the lower level module to high level to ensure the system is working as intended.
→ stubs are used as a temporary module is not ready for integration testing (just dummy)

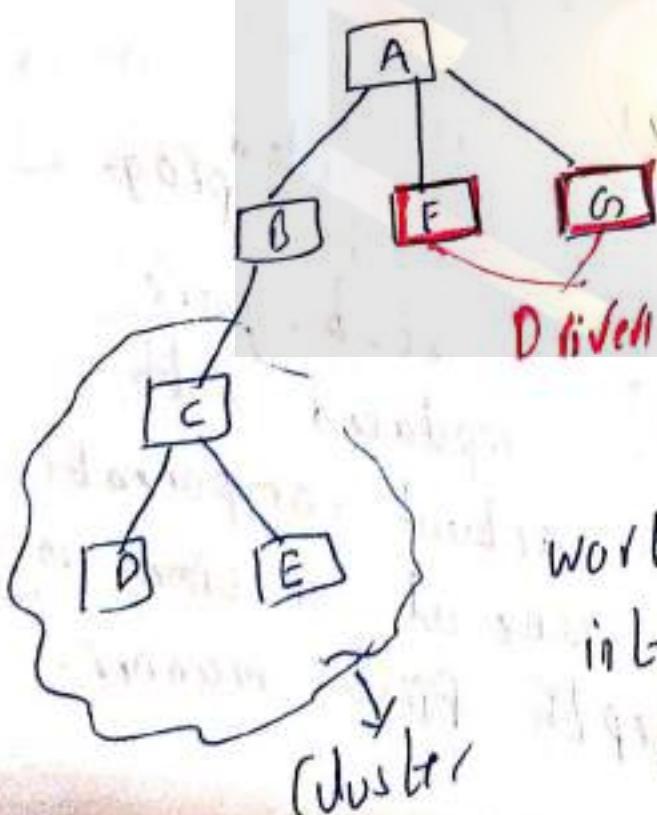


call
called prog.
↑
stubs are replaced with actual components one at a time in a depth first manner.



III) Bottom up approach:

- Here testing takes place from bottom to up.
- Lowest level modules are tested first & then high level modules are finally integrating the high level modules to low level to ensure the system is working as intended.
- Drivers ^{calling pgs.} are used as a temporary module for integration testing

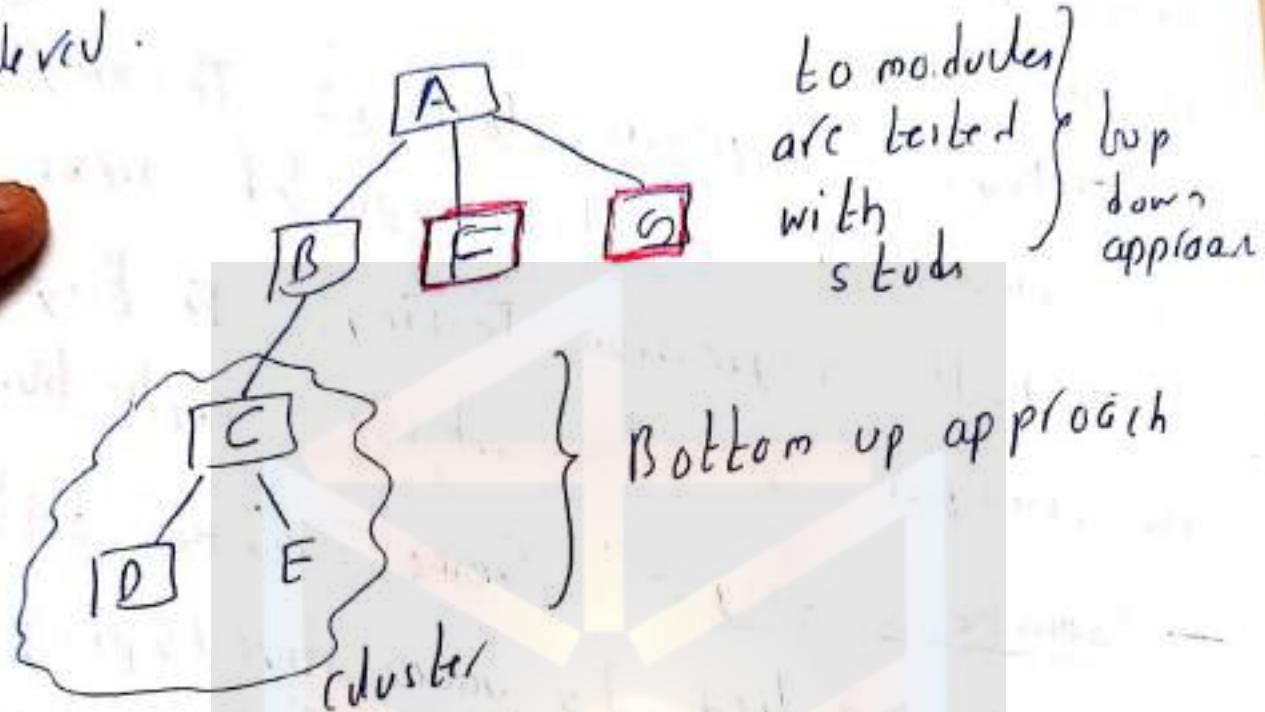


drivers are replaced one at a time depth first

worker modules are grouped into clusters & integrated

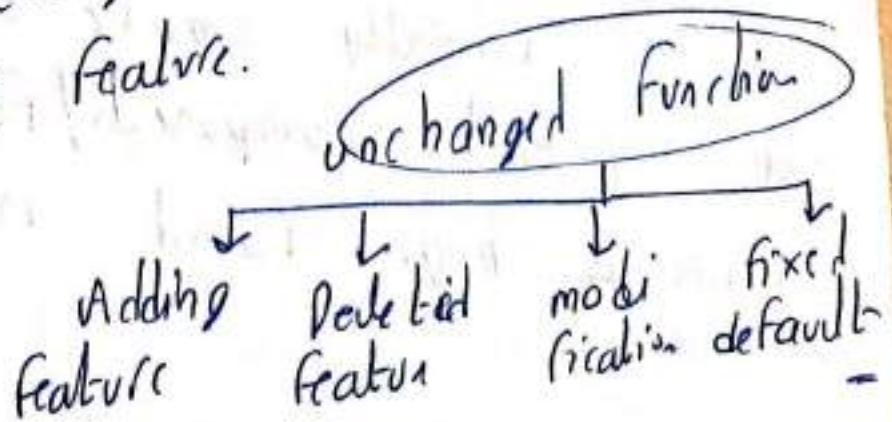
IV) Sand width testing:-

→ Top down strategy for upper level & bottom up strategy for subordination level.



Regression testing :-

→ It is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing feature.



3 types

→ Unit Regression Testing: Testing done by only the changes or modification done by the developer.

→ Regional Regression Testing: Testing the changed & also impacted areas

→ Full Regression Testing: Testing the changed & also whole application

→ Smoke testing :- Some Smoke testing is usually applied to under developing software entities. The testing is applied to those projects, which it needed to be completed in short period of time.

initially smoke testing is done on important components/functions to avoid issues or bugs found in rigorous testing.

2 types
Formal smoke testing: pre written specification
test cases for smoke testing.
informal smoke testing: smoke testing
is not mandatory, no pre written smoke
testing.

→ Validation Testing:

- validation testing follows after integration testing.
- When individual components have been exercised the software is completely assembled as a package, & interfacing errors have been corrected (if any).
- The scenario of validation testing is that to what extent, the customer is satisfied with the developed software. This is the process of validation.

→ configuration reviews :-

→ An important element for the customer, a series of validation process is configuration reviews.

→ The objective of the review is to ensure that all elements of the software configuration have been properly developed.

→ also called as audits

→ Acceptance Testing / Red box testing

→ When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.

→ conducted by the end user rather than software engineer.

→ "It is an end to end testing which is done by the customer for a particular period of time to make sure that software

is able to handle real time business scenario's or not"

→ Alpha testing:

→ During alpha test, several user are called to the developer's site & are exposed to the product

→ Under the guidance of the software engineer, the user uses the product while he is doing so, the software engineer notes all the problems/bugs/problems faced by user.

→ ITL is conducted in a controlled environment.

→ Beta testing:

→ During beta test, the product is delivered to the user. Here in the absence of software engineer, the product is tested & a report is prepared by the user.

→ The report contains details of difficulties faced by the user during the product operation.

Alpha Testing vs Beta Testing

→ Done by the customer at developer site → Done by the customer at user's (customer) site

→ conducted in a controlled environment → conducted in real time environment, not under developer's control.

→ not open to the market & public → open to the market & public.

→ carried out before the release of the product to customers. → carried out after the release of product to customers.

→ errors / failures are recorded → failures are reported.

→ white + Black → Black box
box testing testing.

→ System Testing:

→ System testing is a level of testing that validates the complete & fully integrated software product.

→ The purpose of a system test is to evaluate the end-to-end system usage of all modules till end (like a customer).

Type → recovery testing:

→ ensures that the system must recover from faults & resume processing with little or no down time.

→ any processing fault should not bring down the entire system.
→ software must recover from faults & resume normal processing within a specified time period.

→ Security Testing:

→ Verifies that the protection mechanism built into the system will actually protect product from attack.

→ Stress Testing:

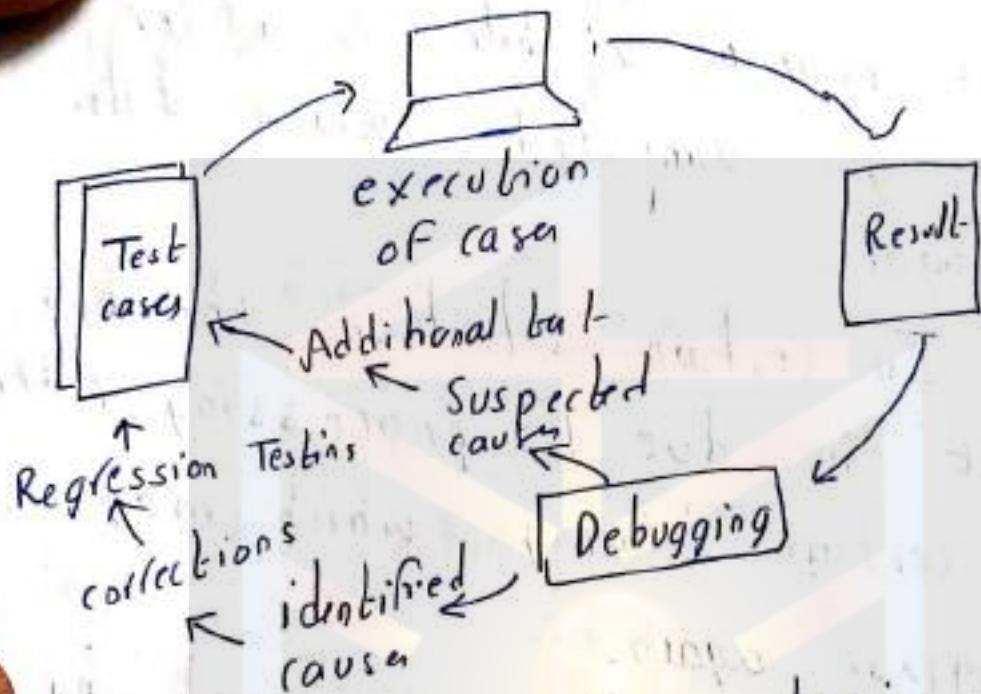
→ It executes the system in a manner that demands resources in abnormal quantity (many users at once).
like → excessive memory requests
→ database request
→ many user at a time.

→ Performance Testing:

→ aimed at testing the run-time performance of the software
→ It is done through out the testing phases
→ speed, time, etc..

→ Debugging / The art of debugging:

→ debugging occurs as a consequence of successful testing.



- debugging process begins with the execution of a test case.
- results are assessed & the difference b/w expected & actual performance is encountered
- "Debugging is the process of detecting & removing existing & potential errors (bugs) in a software code."

→ Complexity of Debugging

→ Sometimes the symptoms get vanished, where other errors are qualified.

→ In certain situations the symptoms may be visible & after sometime get vanished which later appears again.

→ In certain situations the bug may not be due to processing, rather due to certain timing which may later appear again.

→ The symptom may be caused by human error that is not easy to track.

→ Debugging Strategies:

→ Objective of debugging is to find & correct the cause of a software error

→ normal debugging methods & tools
are not a substitute for careful evaluation
based system & to make clear code.

3 strategies for debugging

i) Brute Force: (adding print statement)

→ This is the foremost common
technique of debugging however is that the
least economic method

→ During this approach, the program
is loaded with print statements to print
intermediate values hoping that a number
of the written values can facilitate to
perform spot the statements in error (cause)
→ commonly used & less efficient.

ii) Back tracking:

→ This is additionally a reasonable
common approach. during this approach, starting
from the statement at which error
symptom has been discovered. the source code
is derived/back tracked till error

→ Sadly best only for small programs.

iii) Cause elimination method :-

→ In this approach a list of causes that may have contributed to the error symptom is developed & tests are conducted to eliminate every error.

→ Fundamentals of software testing:-

↳ To find errors

good test: has maximum ability / high probability to find errors

Testability: ease of computer testing

i) Operability:-

software high quality - less bugs
The better it works - the more efficiently it can be tested.

ii) Observability:

outputs are generated for each if
"what you see is what you test"

iii) Controllability:

The better we can control the software the more testing can be automated.

iv) Decomposability:

dividing a large module into sub independent modules & tested individually the combined & retested.

v) Simplicity: The less there is to test the more quickly we can test it

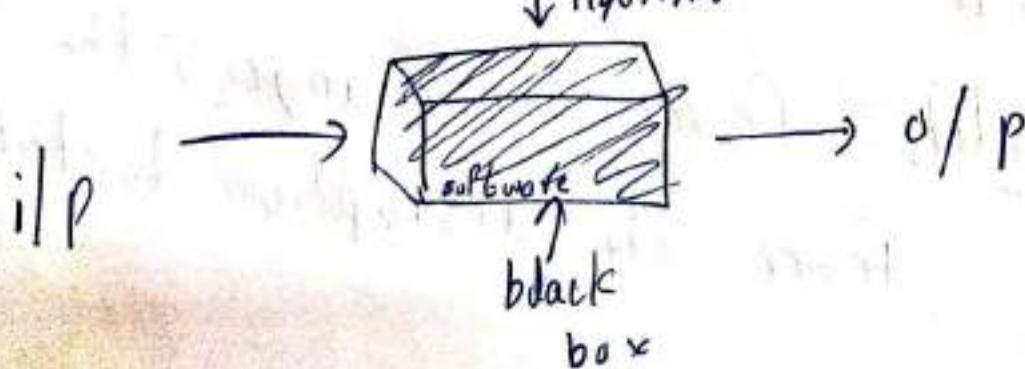
- Functional simplicity (functions should be simple)
- Structural " (architecture should be simple)
- code " (coding standards should be used).

vi) Stability: Fewer the changes, the fewer the disruption to test

vii) Understandability The more info we have the smaller we will test.

→ Black box Testing -

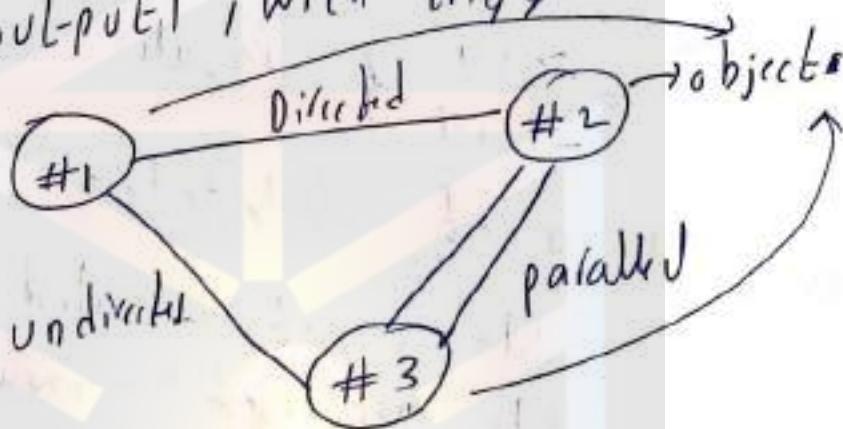
- The black box testing is done without the internal knowledge of the software
- This method is applied at every level of software testing i.e., unit testing, Integration testing, system testing, Acceptance testing.
- It doesn't check the internal code of the software
- It checks the behaviour, hence it is also called behavioural testing



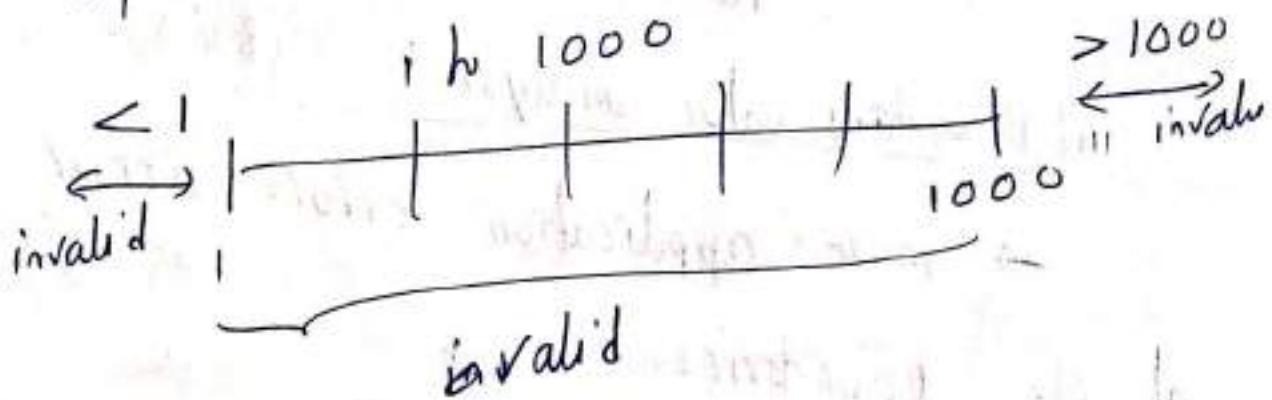
→ Tests are based on requirements & functionality.

methods
i) Graph based testing:

This technique of Black box testing involves a graph drawing that depicts the link b/w cause (input) & the effect (output), with trigger b/w objects.



ii) Equivalence Partitioning: It divides the input domain into various



e.g. value b/n 1 to 99
1 - 99 — valid } 1

< 1 } invalid 2
> 99 } 2 (Jasim

val = 19

19 — valid

> 19 } invalid

< 19 }

in a set { 1, 2, 3, 4, 5 }

val = 7

value in set — valid
value not in set — invalid.

in boolean

True — valid

False — invalid

iii) Boundary value analysis: (BVA)

→ more application errors occur
at the boundaries.

→ It is used to identify errors at boundaries rather than finding those errors exist in the center of the input domain.

→ while writing programs with loop we always think twice loop should start from $i=0$ or $i=1$ & condition $>$, or \geq and $<$ or \leq we also think a lot
→ How to resolve this we always check values beside both boundaries.

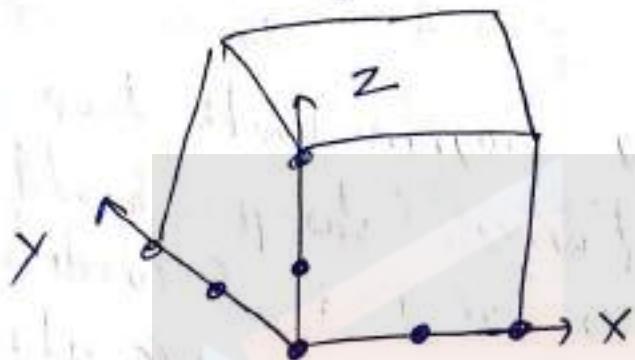
-1 0 1 1 ... 11 10 11 12

iv) Orthogonal array testing:

→ The problem in which the input domain is small but it is large enough to accommodate the exhaustive testing.

ex 3, in prob
 α, β, γ
 3 distinct values \rightarrow No.
 \rightarrow may be.

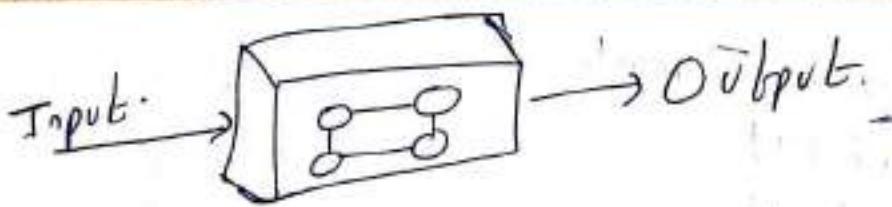
$$3^3 = 27 \text{ case.}$$



now based
 on 1 value
 we keep deciding
 & solving.

→ White box testing:-

- White box testing techniques analyze the internal structures of the software.
- also called as glass box testing or clear box testing or structural testing.
- It can be applied at all levels of SDLC.
- Tests are based on coverage of code statements, branches, paths, conditions etc..



I) Basic Path Testing:

5 step process
 i) Using the design or code, draw the corresponding flow graph.

- i) Draw the DD Path Graph
- ii) Determine the cyclomatic complexity of the flow graph
- iii) Determine a basis set of independent paths
- iv) Prepare test cases that will force execution of each path in the basis set.

Control Flow graph:

→ The control flow of a program can be analyzed using a graphical representation known as flow graph. (like flowchart)

small code
program Triangle

1 input (a)

2 input (b)

3 input (c)

4 if ($a < b < c$) and

5 ($b < a + c$) and

6 ($c < a + b$)

7 then isATriangle = T

8 else isATriangle = F

9 endif

10 if isATriangle

11 then if ($a = b$) and ($b = c$)

12 then output = "Equilateral"

13 endif

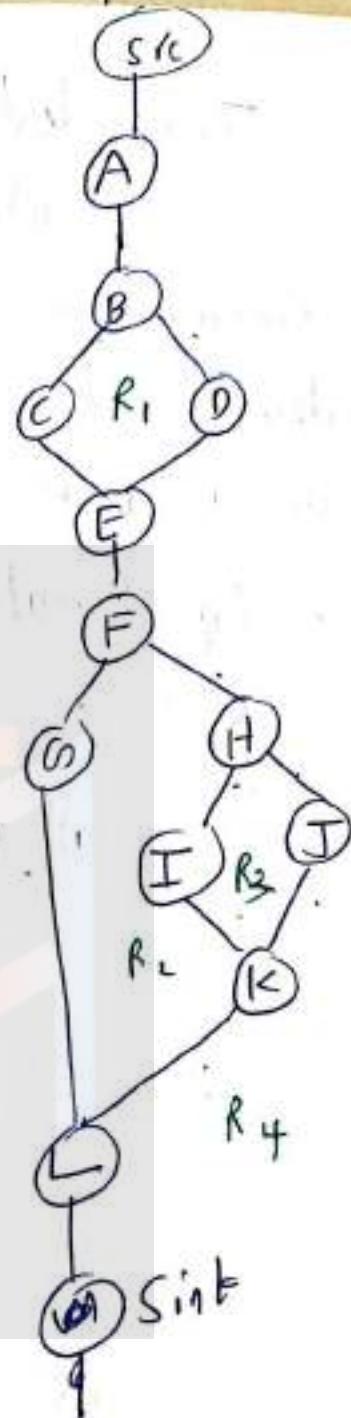
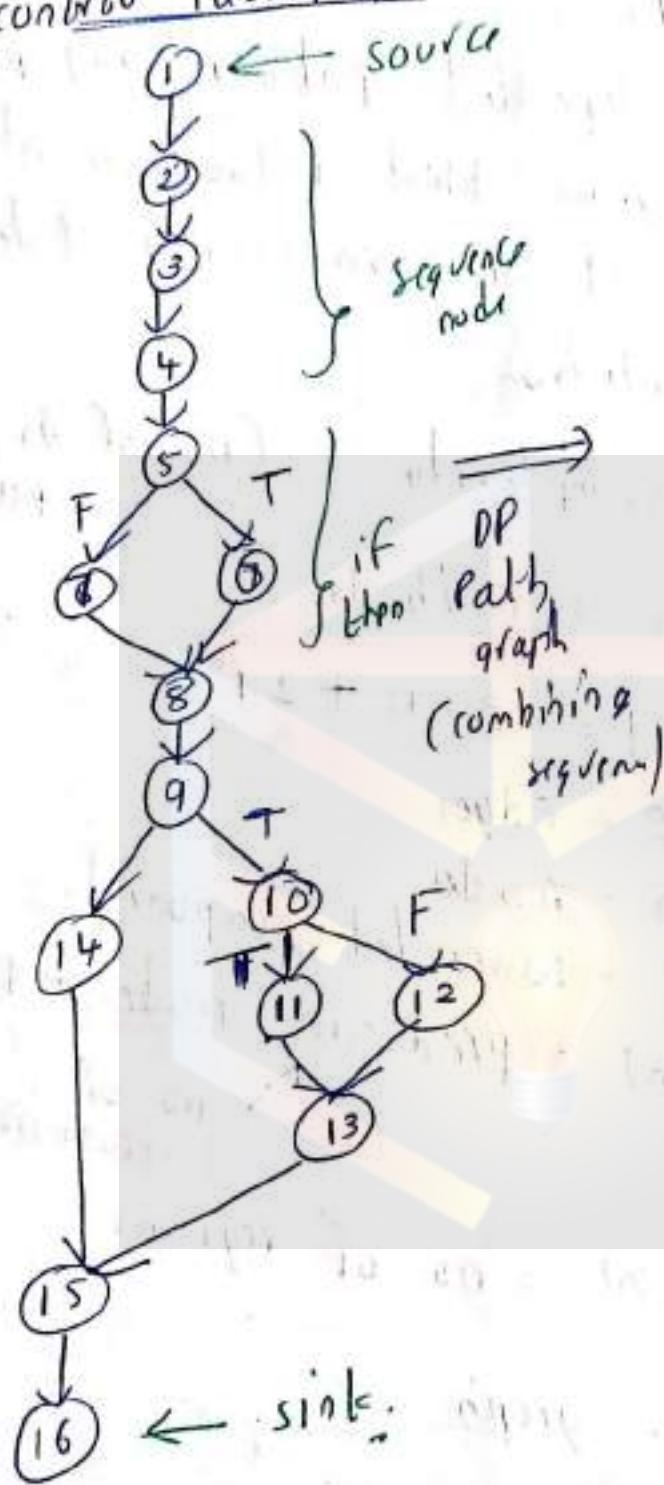
14 else output = "Not a triangle."

15 endif

16 end Triangle.

17 else output = "Not Equilateral"

control flow graph



$$\frac{16 - 14 + 2}{4}$$

Independent path :- An independent path is any path through the program that introduce at least one new set of processing statement or a new condition.

Cyclomatic Complexity :- (no. of unique paths)

$V(n) \rightarrow$ we have 3 methods

i) $V(n) = e - n + 2p$
e = edges

n = no. of nodes

p = connected components

ii) $V(n) = \text{predicate node} + 1$

 ↑
 no. of condition node

iii) $V(n) = \text{no. of regions}$

for same previous graph

$$e = 16, n = 14, p = 1$$

$$\therefore 16 - 14 + 2 = 4$$

$$\begin{aligned}
 &= \overset{(o)}{\text{predicatr node}} + 1 \\
 &= (\text{D}, \text{F}, \text{H}) \\
 &= 3 + 1 = 4 \\
 &= \overset{o}{\text{no. of f lgv}} \\
 &= (\text{R}_1, \text{R}_2, \text{R}_3, \text{R}_4) \\
 &= 4
 \end{aligned}$$

Independent paths

- 4 paths
- ① → A'src - A - B - D - E - F - H - IJK - L - sink
 - ② → src - A - B - D - E - F + IJK - L - sink
 - ③ → src - A - B - D - E - F - G - L - sink
 - ④ → src - A - B - C - E - F - G - L - sink

5 test cases

- | | |
|-----|--------------------------|
| 1 → | not equivalent |
| 2 → | equivalent |
| 3 → | not a triangle |
| 4 → | not a triangle. |
| 4 | test cases are mandatory |

II) Control structure testing:-

here testing control
structure is done here

i) Condition testing:-

Condition testing test
the logical conditions in the program
module

relational expression $\rightarrow E_1 \text{ relational op } E_2$

$<, \leq, =, ! =, >, \geq$

in simple condition \rightarrow unary operators
(\sim Not).

compound condition $\rightarrow (\underline{E_1 \& E_2}) \parallel (\underline{E_2 \& E_3})$

①

②

③ in ①.

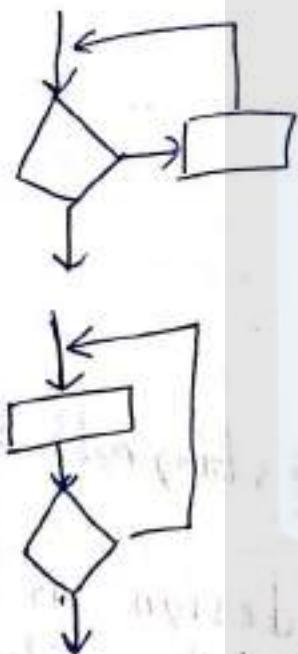
Boolean express. \rightarrow True / False.

1

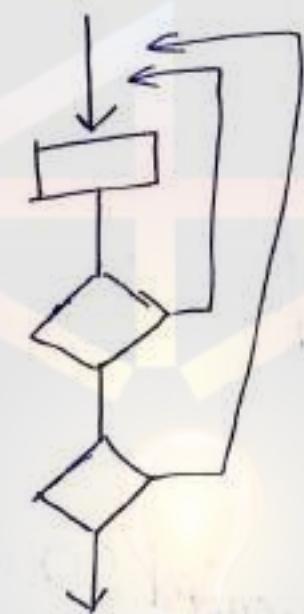
iii) Data Flow Testing:

Data flow testing tests
the paths of a program according to
the location of definition & use
of variables in the program.

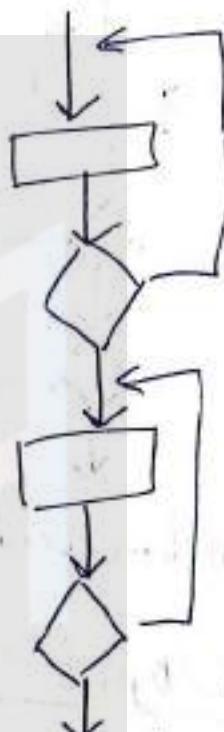
iii) Loop Testing:



Simple loop



Nested loops



concatenated loops.

Techniques

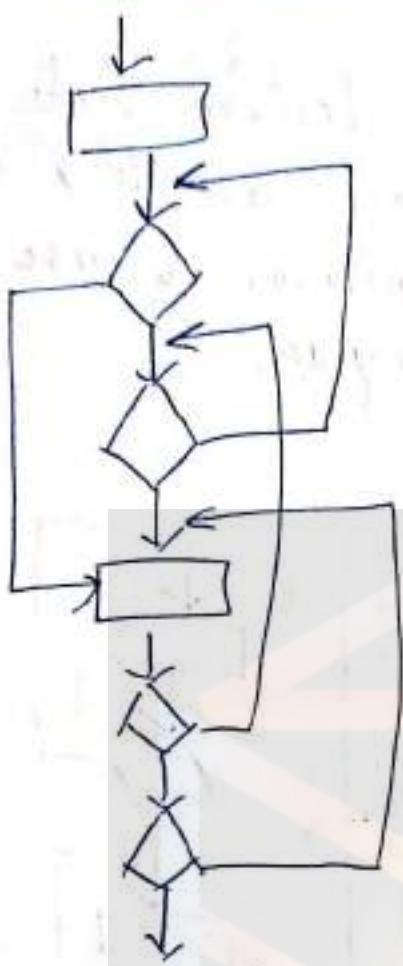
- skip the loop
- one pass
- 2 passes
- m passes

Techniques

- inside for specific bit
- output for specific trim

Techniques

- use same pair simple loops



unstructured.

Better to reduce
to structured &
unprevious testing.

→ Object-Oriented (O-O) Testing method

→ When we wrote code/design in conventional methods like ^{structure} we used software testing /

→ If program/code/design uses object oriented concepts (class, encapsulation, objects, Inheritance) etc.. Then we go for (O-O) testing.

Methods

i) Fault-based Testing:

→ designing test cases that have high probability of detecting high no. of errors.

→ depends on tester's completeness

→ Good result with less efforts

→ It misses

- Incorrect specification

- Interaction among subsystems.

ii) Scenario-Based Testing:

→ to avoid above 2 misses

→ It concentrates on what the user

does, (not what product does)

→ This means capturing the tasks

(via use case) that user has to perform.

iii) Class Testing Based on method

→ Simplified approach to test classes

→ each module in can be tested

using unit testing.

→ Therefore all the methods of a class can be involved at least once to test the class.

iv) Random Testing:

→ It is supposed by developing a random test sequence that tries the minimum variety of operations typically to the behavior of the categories.

v) Partition Testing:

→ This methodology categorizes the input & output of a category so as to check them severally.

→ Interclass Test case design

i) multiple class Testing:

ii) Tests Derived from Behavior model

note/
theory not found if you have unclear share

→ Product metrics :- measurements

→ Product metrics measure the characteristics of the software product that has been developed.

→ Help software engineer to better understand the attribute of model & assess the quality of the software.

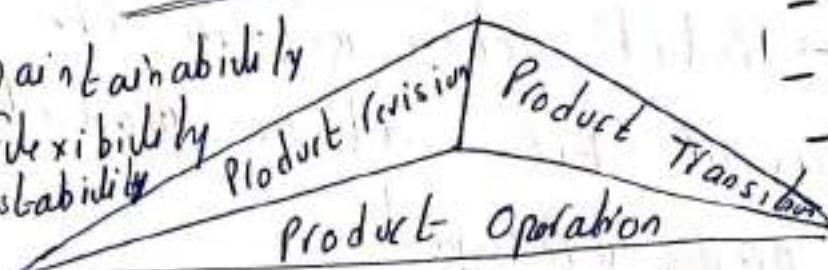
→ Software quality :-
→ Software quality is defined as the conformance to explicitly stated functional & performance requirements.

→ Software quality product is defined in term of its fitness of purpose.
→ i.e., the quality product does precisely what the user wants it to do.

→ Mc Call's Quality Factors:-

- maintainability
- flexibility
- Testability

- Interoperability
- Reusability
- Portability



- Efficiency
- Integrity
- Usability
- Reliability
- Correctness

1) Correctness:- The ability of the given software should satisfy its purpose of developing (& satisfying customer).

2) Reliability:- Software should be more reliable (if any failure occurs it should be rectified as fast as possible).

3) Usability:- The effort required to learn, operate, prepare input for & interpret output of a program.

4) Integrity:- The factor deals with the software security, that is to prevent access to unauthorized persons.

5) Efficiency:- The amount of computing resources & code required by a program to perform its function.

6) Portability:- The effort required to transfer the program from one hardware and/or software system environment to another.

7) Reusability:- The extent to which a program (or part of program) can be reused in other application.

8) Interoperability:- The effort required to couple one system to another.

9) Maintainability:- The effort required to locate & fix an error in a program.

10) Flexibility:- The effort required to modify an operational program (easy).

ii) Testability: The effort required to test a program to ensure that it performs its intended function.

→ ISO 9126 Software Quality Factors

i) Functionality: The degree to which the software satisfies stated needs that the software is in a failure state.
(it should more reliable)

iii) Usability: The degree to which the software is easy to use.

iv) Efficiency: The degree to which the software make optimal use of system resources

v) Maintainability: The ease with which repair & enhancement may be made to the software.

Software Portability: The ease with which the software can be transposed from one environment to another.

→ Measures, Metrics & Indicators

Measure: provides a quantitative indication of the extent, amount, dimension, capacity or size of some attribute or a product or process.

Metric: (IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Indicator: A metric or combination of metrics that provide insight into the software process, a software project, or the product itself.

(reverse order in syllabus)

→ Metric for maintenance:

With regard to metrics for maintenance, IEEE came out with a value referred as software maturity index, normally known as SMI.

$$SMI = \left[M_L - (F_c + F_a + F_d) \right] / M_L$$

M_L = No. of units/modules in the current release/program ✓

F_c = the no. of modules in the current release that have been changed

F_a = the no. of modules in the current release that have been added.

$SMI = 1.0^{(nat)}$
is stability F_d = the no. of modules from that are deleted/unavailable in current release.

→ Metric for testing:

The metric is derived by considering the implementation of process of testing.

→ halstead metric applied to Testing

$$PL = \frac{1}{O_1 \times O_2}$$

(Program Jvd)

$$e = \frac{V}{PL}$$

(halstead effort)

O_1 = no. of operators available in code

O_2 = sum of occurrences of operators

O_2 = no. of independent operands

available in the code.

PL = Program level

V = volume / no. of bits accommodating the program.

e = halstead effort

Metric for source code

This is done after the completion of writing of the required code, or after the completion of designing phase of the software project.

Length $W = n_1 \log_2 n_1 + n_2 \log_2 n_2$

Program volume $V = W \log_2 (n_1 + n_2)$

Volume ration $L = 2/n_1 * n_2 / V^2$

n_1 = no. of distinct operators in a program.

n_2 = no. of distinct operands in a program

n_1 = total no. of operator occurrences

n_2 = total no. of operand occurrences

→ Metrics for analysis model:-

→ Metrics for the analysis model are useful in estimating the project.

→ These metrics examine the analysis model with the intent of predicting the size of the resultant system.

→ Function-based metric / functional point

→ To measure the standard worth of the software, as a unit of software worth. Function Point was developed.

→ From user's point of view.

5 Functional units :-

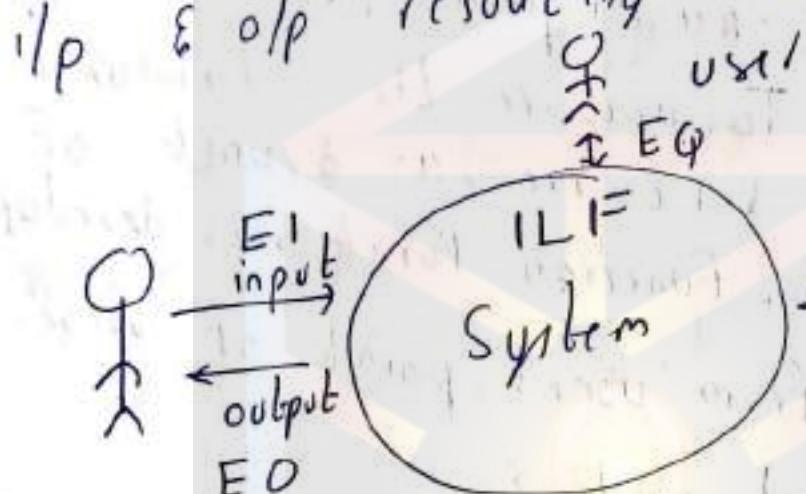
i) Internal Logic File (ILF) :- The control info or logically related data that is present within the system.

ii) External Interface File (EIF) :- The control data referred by the system but present in another system.

iii) External Inputs (EI): Data/control info that comes from outside our system

iv) External output (EO): Data that goes out of the system after generation.

v) External enquired (EQ): Combination of resulting data retrieval



$$FP = UFP \times CAF$$

Functional point Unadjusted Functional point Complexity adjustment factor.

$$FP = \text{Total} \times [0.65 + 0.01 \times [F_1 b^{14}]]$$

→ Metric for design model:

→ Architectural design metrics:

→ It focus on characteristics of the program architecture & efficiency of modules or components within the architecture.

→ Card & Galli define three software design complexity measures

- i) Structural complexity
- ii) Data "
- iii) System complexity

i) Structural complexity
structural complexity of module i is defined in manner as

$$S(i) = F_{out}^2$$

F_{out} $F_{out}(i)$ is the fan out module

↓
No. of modules of i invoked by module i

iii) Data Complexity

It provides an indication of the complexity in the internal interface for a module.

defined as

$$D[i] = \frac{V[i]}{F_{out}(i) + 1}$$

$V[i]$ = No. of i/p/o/p variables that are passed to & from module i

iii) System Complexity

It is defined as the sum of the structural & data complexities

$$C[i] = S[i] + D[i]$$

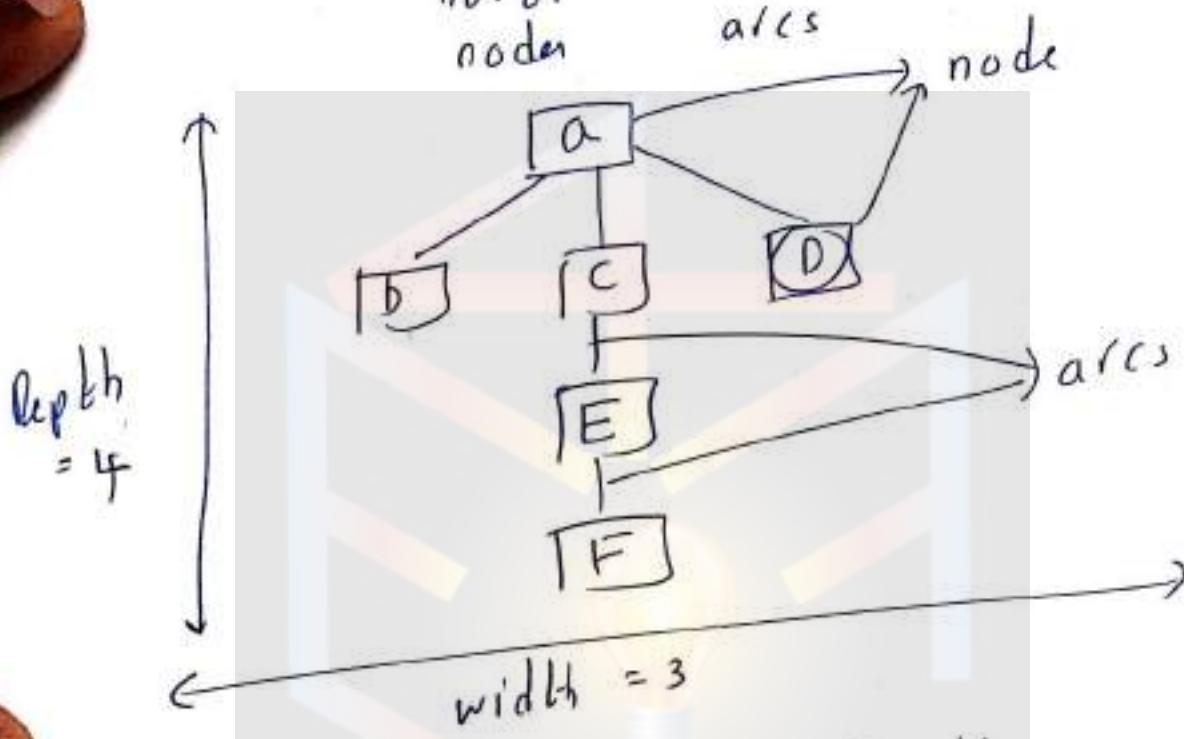
\uparrow
structural data
complexity complexity

Fenton suggest a number of simple morphology (shape) metrics that enable

different program architecture to be compared using a set of straight-forward dimensions.

$$\text{size} = n + a$$

↓ ↓
no. of no. of
nodes arcs



$$\text{size} = 6 + 5 = 11$$

$$\text{edge to node ratio} = \frac{5}{6} = 0.83$$

$$\therefore \text{DSGI} = \sum w_i D_i$$

$i=1 \text{ to } 6$

Design structure
grability index

↓
weight of
each intermediate
node.