

* Introduction to Software Engineering *

* what is software engineering ?

• Software :- It is a collection of integrated programs means it carefully organized instructions.

• Engineering :- It is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, processes etc.

• Software Engineering :-

It is an engineering branch related to the evolution of software product using well defined scientific principles, techniques and procedures.

• The Result of software engineering is an effective and reliable software product.

* Need of Software Engineering :-

1. Handling big projects

2. To manage the cost

3. To decrease time.

4. Reliable software

5. Effectiveness

we use softwares in our daily life for each and every purpose like college purpose, browsing purpose (gaming, shopping, social applications).

* characteristics of good software :-		
(ex: whatsapp) <u>Operational</u>	(ex: Turbo) <u>Transitional</u>	(version) <u>Maintenance</u>
+ Budget	- Interoperability	- flexibility
- Efficiency	- Reusability	- maintainability
- Usability	- Portability	- Modularity
- Dependability	- Adaptability	- scalability
- Correctness		
- Functionality		
- Safety		
- Security		

* The evolving Role of Software -

- A Software performs 2 task dual role.
- It is both a product and a vehicle for delivering a product.

* product : software is a product.

- produces, manages, and display information

* product (process)

* vehicle (process) :- It is information

transformer. (producing, managing, acquiring, modifying, displaying, vehicle)



Information Transformer



(single bit into/multimedia).

* Changing nature of software :-

The software has different natures; they are :

1. System software :-

- It is SW designed to provide platform for other software
- It is an interaction b/w hardware & application software.

Ex OS like MacOS, Linux, Android and Microsoft windows.

2. Application SW :-

- It is a Computer program designed to carry out a specific task as per user's business need.

Ex - Social media apps, Gaming apps, word processing apps, multimedia apps, Banking, Shopping, Booking apps etc.

3. Engineering & Scientific Software :-

- This is used to facilitate the engineering function and task take real time.
- It has high accuracy, complex formula evaluation & data analysis.

Ex Weather prediction apps, stock market apps, stress analysis, Body measurements.

4. Embedded S/W

It resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself.

Ex switches, Routers, Digital camera, washing machine functionalities, traffic control etc..

5. web application S/W

- It is a client-server computer program which the client runs on the web browser.

- web apps can be little more than a set of linked hypertext files that present information using text and limited graphics.

Ex online forms, shopping carts, Gmail, yahoo, photo editing, live conversation etc.

6. Artificial Intelligence S/W

- It makes use of a nonnumerical algorithm to solve a complex problem.

- Application within this area includes robotics, expert system, pattern recognition, artificial neural nets etc.. (AI gaming).

Ex Google cloud, Azure studio, tensorflow, Salesforce etc...

Software process (Role continuation)

* A slw process is the set of activities and associated outcome that produce a slw product.

• slw engineers mostly carry these activities.

1. slw specification - The functionality of slw and constraints on its operation must be defined.

2. slw Development :- The slw to meet the requirement must be produced.

3. slw validation :- The slw must be validated to ensure that it does what the customer wants.

4. software evolution - The slw must involve to meet changing client needs.

* Software Myths -

• myth :- It means ~~wrong~~ belief & misinformation.

* software myth :- software myths are beliefs about software and the process used to build it.

* myths have no. of attributes that causes serious problems on software.

* These myths can sometimes leads to false expectations.

There are 3 types -

1. Management myth :-
2. ~~Customer~~ Customer myth.
3. Developer / practitioner's myth.

1. Management myth - In management myth the managers are responsible for the developing the software.

Some of the management myths are given below : (i) myth : The management thinks we already have book with full of standards & precedures.

By reading those books we can develop software.
Reality - If all standards & procedures are present on books that help to develop a software but it is not possible for the maintain correct quality.

(ii) myth Adding new people into the project leads to finish the project in scheduled time.

Reality : Adding new people into the project cannot make the project can complete on intime and it reduces the time period to explain overall project and the money will also included.

2. Customer myths :-

customer myths are generally due to false expectations by customers and these myths end up leaving customers with dissatisfaction with the software developer.

Myth :- customer can give some requirements to start the project we can add more requirement later.

Reality :- It is not possible when we write overall program for design a program and few more changes to the written program. when new requirements is added then we can re-write overall program by the ~~developer~~.

(ii) Myth :- Any changes can occur it can be easily solved in software.

Reality :- Yes, it is true when any change is occurred, we can easily changes in software but the time waste is occurred.

3. Developer / practitioner's myth :-

Developers often work under management pressure to complete software within a time with fewer resources.

(i) myth :- once the software writes the code then the developer's work ends.

Reliability :- If the programmer write code & delivering to customer while it is working and when it is checked for the customer requirements any of the requirement is missed then we change entire program.

(ii) myth :- when project is ready there is no time to delay to deliver the project.

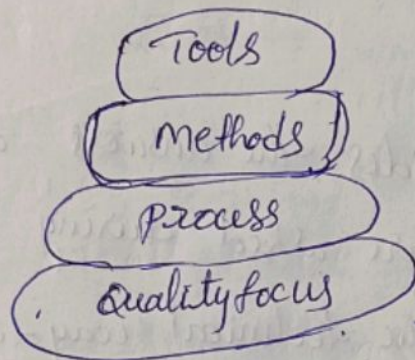
Reality :- If project is completed but any one option is not working in that s/w then we can send it again to the tester & find the bug for the code then the time delay is occurred to make changes in the software.

* A Generic view of process -

* Software Engineering - A layered technology

→ Software Engineering is a layered technology to develop a software, we need to go from one layer to another layer.

→ All There are 4 layers and these all layers are connected and each layer demands the fulfillment of the previous layer.



Flow chart of

A layered technology of software development.

1. Quality focus - (Foundation layer) (Bedrock)

The Characteristics of good quality software are:

- Correctness of the functions required to be performed by the software.
- Maintainability of the software
- Integrity i.e., providing security so that the unauthorized users cannot access information of data.
- Usability: i.e., the efforts required to use & operate the software.

2. process :-

(It is the base layer or foundation layer of the Software Engineering) ①

- The software process is the key to keep all levels together.

- It defines a framework that includes different activities and tasks.

- In short, It covers all activities, actions and tasks required to be carried out for Software development.

3. Methods :-

The methods provides the answers of all

"how-to" that are asked during the process

- It provides the technical way to implement the software.

- It includes collection of tasks starting from Communication, requirement analysis, analysis and design modelling, program construction, testing and support.

4. Tools :-

The Software Engineering tool is an automated support for the software development.

- The tools are integrated, i.e., the information created one tool can be used by the other tool.

SK

Ex The Microsoft publisher can be used ~~by~~ as a web designer tool.

* A process framework +

=> Software engineering has processes, methods and tools.

=> A Software process framework provides the organized steps (process) that guide how the methods and tools are used to ensure high-quality software.

* Software process:-

This is set of steps and tasks we follow to create a SW product, ensuring everything is done correctly.

* Framework :- It is standard way to build and release software, like a blueprint that guides developer.

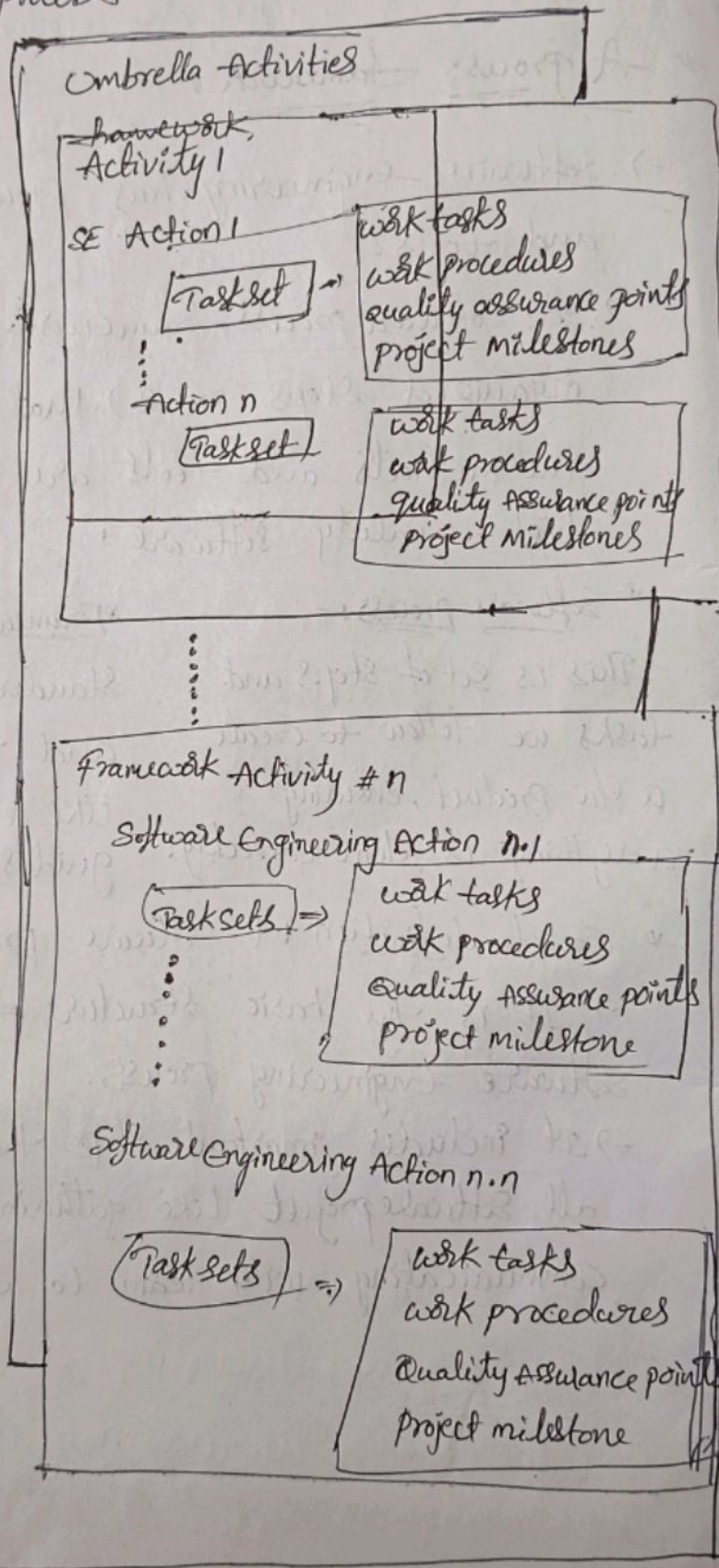
* Exact definition of software process framework:-

=> It is the basic structure for the entire software engineering process.

=> It includes important steps that apply to all software projects like gathering requirements, communicating with team to delivering the SW.

• Software process

process framework



It is also ←
blue print of
the generic process
framework.

Generic * process frame-work activities + (5 activities).

1. Communication :- The team talks with customers and stakeholders to understand what they need. This helps the team gather all the requirements.
2. planning :- The team makes a plan. They decide the what tasks to do, consider risks, gather resources like tools and people, and set a schedule.
3. modelling :- The creates a model of the software. This could be diagrams, flowcharts, or prototypes to show how the software will work.
4. Construction :- This is the coding phase. The team writes and tests to make sure it works correctly.
5. Deployment + The team delivers the software to the customer and takes feedback.

* Umbrella Activities +

1. Risk management (RM)
2. Software Quality Assurance (SQA)
3. Software Configuration Management (SCM)
4. Measurement
5. Formal Technical Reviews (FTR).

1. Risk management

- Managing Risks during software development.
- Helps the team handle problems that might arise and ensures the project stays on track.

Examples

- ⇒ Imagine you're building a new app.
- ⇒ What if the key developer gets sick, or a new version of the operating system is released?
- ⇒ Risk management is like planning for these "what if" scenarios.
- ⇒ You might create a backup plan where another team member is needed.

2. Software Quality Assurance (SQA)

- Planned activities that make sure the software meets quality standards.
- Ensures the final product is high quality & works as expected.

Example

- ⇒ This is like having a checklist to make sure every part of your app works correctly before launching it. (login page)
- ⇒ Before releasing the app, you test it on different devices, it doesn't crash and meets quality standards.

3. Software Configuration Management (SCM) :-

- Managing changes in the SW environment.
- Helps the team adapt to changes without disrupting the project.

Ex $\Rightarrow$ imagine you've added a new feature to your app, but it causes the older features.

$\Rightarrow$ SCM helps you manage these changes smoothly.

$\Rightarrow$ You keep the track of all versions of your app, so that you can ~~back~~ back to an earlier version if needed.

4. Measurement :-

- Tracking details like the cost of resources and work done.
- Keeps the project on budget and helps manage resources effectively.

5. Formal Technical Review (FTR) :-

- Regular meetings to check on the project's progress.
- Keeps everyone informed and ensures the project is moving forward smoothly.

Ex This is like having regular check-ins with your team to see how app development is going.

$\Rightarrow$ Every week, you and your team review the app's progress and decide what needs to be done next.

* Legacy Software :-

- It is a very old software.
- It was developed ~~by~~ decades of years ago.
- It is an inextendible design.
- It cannot be updated so it has poor quality.
- It has no documentation.
- It cannot meet (or) no need of new technologies.

* Solutions of Legacy Software :-

- ① Software should be adapted to meet new computing environment and technology.
- ② Software should be enhanced to meet new business requirements.
- ③ Extendible design → to make interoperable with other systems.
- ④ Re-architected, so that it is suitable to new environment technology.

* CMMI :-

The Capability Maturity Model Integration :-

- ⇒ The Software Engineering Institute (SEI) has developed to improve the capabilities of ~~the~~ the US Software Industry.
- ⇒ In the mid-1980s, the SEI initiated a study of ways of assessing the capabilities of software contractors.
- ⇒ To achieve these capabilities, the SEI contends that an organization should develop a process model that conforms to CMMI guidelines.
- ⇒ The CMMI model is very complex, so we have simplified it for discussion here:

1. Process areas : The CMMI identifies 24 process areas that relevant to software process capability and improvement. These are organised into four groups in the continuous CMMI model.

(i) process management

(ii) project management

(iii) Engineering

(iv) Support.

2. Goals

The CMMI has specific goals that are associated with each process area and that define the desirable state for that area.

3. practices - practices in the CMMI are descriptions of ways to achieve a goal.

⇒ A CMMI assessment involves examining the processes in an organisation and rating on a six-point scale assigns a level to a process as follows :-

(1) Not performed (incomplete)

one or more of the specific goals associated with the process area is not satisfied.

(2) performed + (Initial) The goals associated with the process area are satisfied.

(3) Managed (managed) - At this level, the goals associated with the process area are met and organisational policies.

(4) Defined - This level focuses on organisational standardisation and deployment of processes.

(5) Quantitatively managed - At this level, there is an organisational responsibility to use statistical and other quantitative methods to control sub-processes.

6. Optimizing : At this highest level, the organisation must use the process and product measurements to drive process improvement.

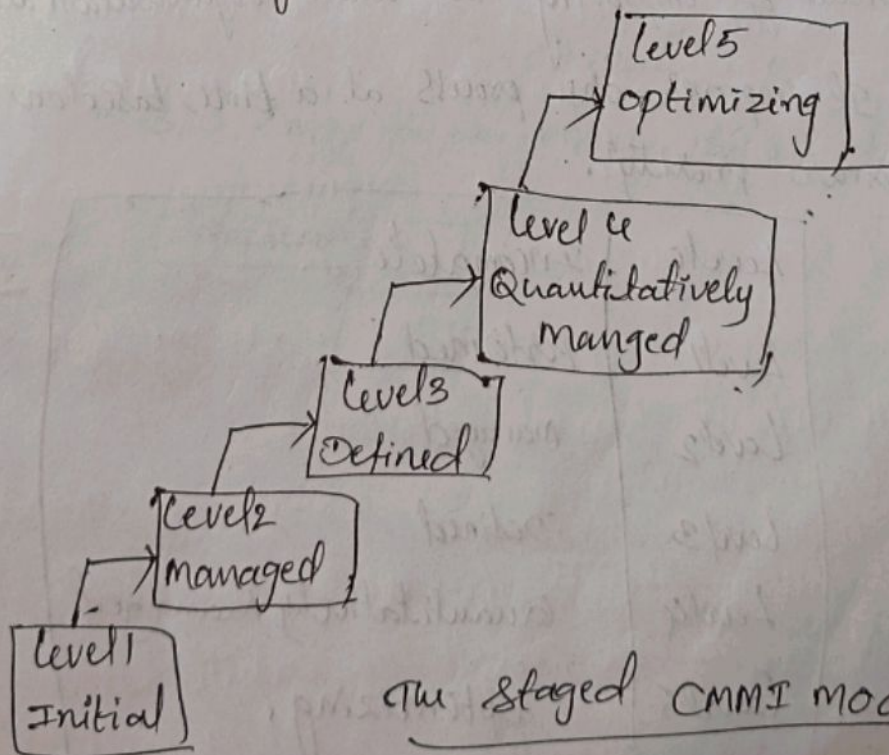
✓ CMMI Represents a process meta model in two different ways: as a

1. Staged CMMI model

2. The continuous CMMI model.

1. Staged CMMI model:- It is comparable with the software CMM in that it provides a means to assess an organisation's process capability at one of five levels.

⇒ It describes the goals that should be achieved at each level, moving from the lower to the higher levels in the model.



The Staged CMMI model

* Each maturity model level has an associated set of process areas and generic goals.

* The advantage of the staged CMMI, apart from its compatibility with the software CMM, is that it defines a clear improvement pathway for organisations.

* The disadvantage is that it may be more appropriate to introduce goals and practices at higher levels before lower-level practices.

2. The continuous CMMI model:-

→ This model is one way of applying CMMI (Capability maturity model Integration) where each process area is improved independently instead of improving the whole organisation at once.

"It improves one process at a time, based on business priority".

Level 0	Incomplete
Level 1	Performed
Level 2	Managed
Level 3	Defined
Level 4	Quantitatively managed
Level 5	Optimizing.

Advantages

→ Flexible

→ focus on important processes first.

⇒ Suitable for specific improvement goals. //

2. Waterfall model

→ Process models - It defines how software is developed, step by step, from requirements to maintenance. There are some models we will discuss

Waterfall Model (1970)

1. waterfall 2. Incremental

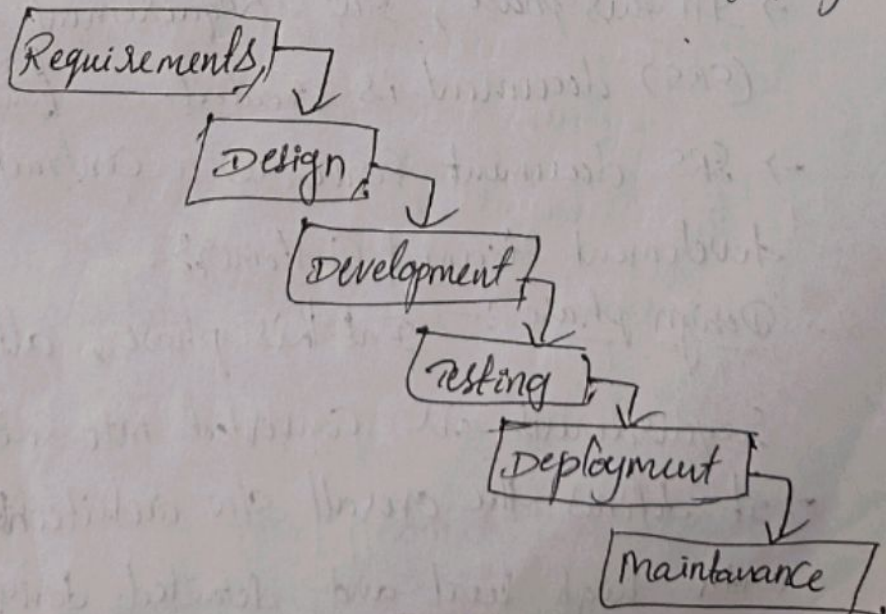
3. Iterative 4. Spiral

5. V-model 6. Agile

7. RAD (Rapid Application Development)

→ Winston Royce introduced the waterfall model in 1970 and also called as Linear Sequential Development model.

⇒ waterfall approach was first software development life cycle (SDLC) model, used widely in SW Engg. to ensure success of project.



* Each phase must be completed before the next phase can begin. output of one phase will be input of next phase.

* When to use waterfall model?

1. When the requirements are constant and not changed regularly.

2. Application is not complicated and big.

3. A project is short & simple

4. Where the tools and technology used is consistent and is not changing.

5. When resources are well prepared and available to work.

* Waterfall model phases:-

1. Requirement Analysis:-

→ The aim of this phase, is to understand the customer requirements to document them properly.

→ In this phase, SRS Requirement Specification, (SRS) document is created. ~~Need~~ ~~Want~~ SRS

→ SRS document serve as a contract b/w the development team & Customers.

2. Design phase:- In this phase, all gathered requirements are converted into suitable design.
→ It defines the overall S/W architecture together with high level and detailed design.

→ High level design include Algorithm, Flowcharts, Decision tree, Database design etc.

→ low level design include Rough paper design, User interface components etc.

→ Also finalized programming languages Database or other H/W & S/W requirements.

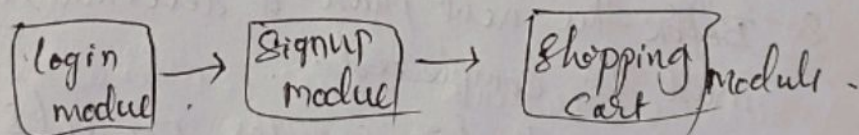
→ All this work is documented as S/W Design Document (SDD).

3. Development phase -

⇒ In this phase, S/W design is translated into source code using any suitable programming language.

→ The developer first developed in small programs called units after that integrated it.

⇒ unit testing is done in this phase to check whether each module is working properly or not.



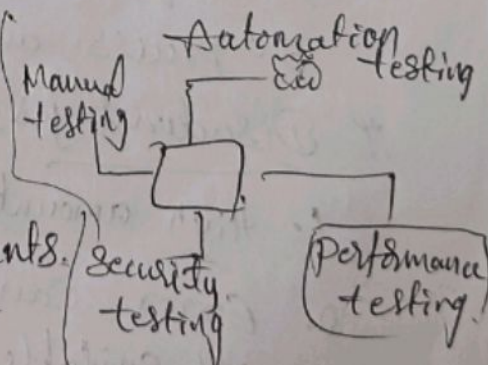
4. Testing phase -

→ In this phase, tester perform all the testing activities to make sure that the system meets the client requirements.

→ After combining all the unit modules, Integration testing is done for any faults and failures.

→ In case of any anomaly, report it.

→ It generates test cases and test reports.



5. Deployment & maintenance phase

⇒ The product is deployed in the customer environment or released into the market.

→ In maintenance phase, there are some issues which come up in the client environment.

→ To fix those issues, patches are released.

→ Also to enhance the product some better versions are released.

→ Maintenance is done to deliver these changes in the customer environment.

Monitoring the system



upgrading the system



fixing system errors.

* Advantages

1. Simple and easy to understand and use.
2. Before the next phase of development, each phase must be completed.
3. Works well for smaller projects.
4. Requirements are very well understood & fixed.
5. process and results are well documented.

* Disadvantages

1. High amounts of risks and uncertainty.
2. Error can be fixed only during the phase.
3. Not suitable for projects where req^s are changing.
4. It becomes very difficult to move back to the phase.
5. Not a good model for complex and object-oriented projects.

Spiral Model :-

- * Initially proposed by Boehm in 1986.
- * Spiral model is "a risk-driven software development process model."
- * It is a combination of waterfall model, Iterative model & prototyping model.
- * Software is developed in a series of incremental releases as per each spiral.

Ex Microsoft, OS versions, Gaming Industry etc.

* It also called as meta model.

→ Spiral model divided into the four parts.

- ① planning
- ② Risk analysis
- ③ Engineering & Execution
- ④ Customer Evaluation



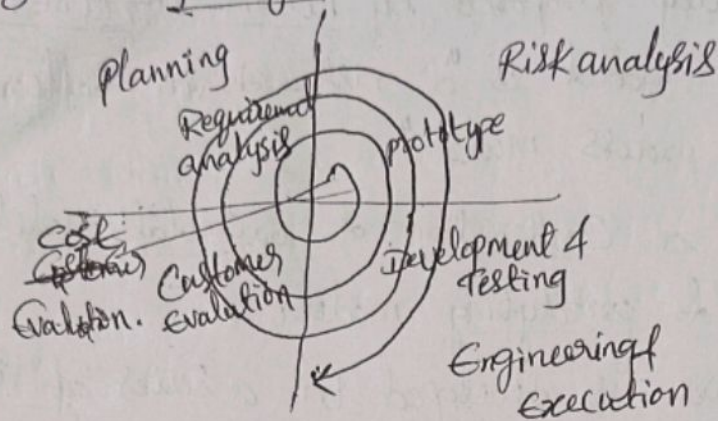
① Planning (Requirement Gathering & Analysis) :-

- * Communication b/w customers & project head.
- * Collect all the requirements from customers.
- * Analyses estimated cost, schedule & required resources.

② Risk Analysis

- * Identification of all the potential risks.
- * Risk mitigation strategy is planned for solving risks.

* Design a prototype of model.



③ Engineering & Execution +

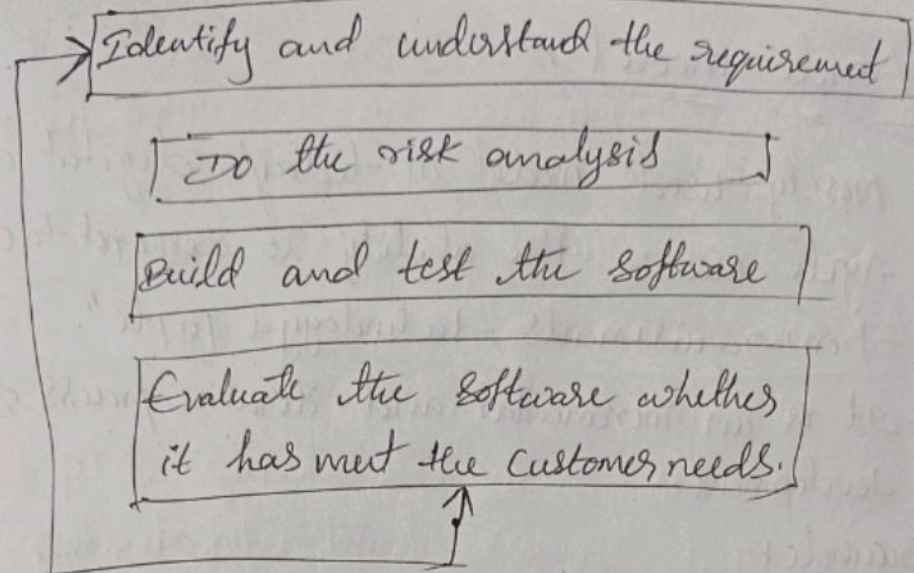
- * Actual Development start.
- * Designer design the product as per final prototype.
- * Developer perform actual coding or implementation.
- * Tester perform all testing methods.
- * Deploy or Release product to the customer environment.

④ Evaluation +

- * Take a feedback from customers.
- * If customer want any changes, goes to next planning OR next spiral iteration.

* When to use spiral model +

1. when the project is large and high budget.
2. when requirements are unclear & complex.
3. where the software needs continuous risk evaluation.
4. when creation of a prototype is applicable.
5. when changes may require at any time.
6. when long term project commitment is not feasible due to changes in economic priorities.



* Advantages of Spiral model.

1. High amount of risk analysis.
2. Risky parts can be developed earlier which helps in better risk management.
3. useful for large and mission-critical projects.
4. Allows extensive use of prototypes.
5. There is always a space for customer feedback.
6. Changing requirements can be accommodated.
7. Development is fast.

* Disadvantages

1. Risk analysis needed highly particular expertise.
2. Can be a costly model to use.
3. Doesn't work well for smaller projects.
4. Spiral process is complex sometimes because spiral may go infinitely.
5. large no. of spiral stages requires excessive documentation.

* Agile model +

- * Mostly used model in today's digital era.
- * Agile means the ability to respond to change from requirements, technology & people.
- * It is an incremental and iterative process of software development.

→ Example +

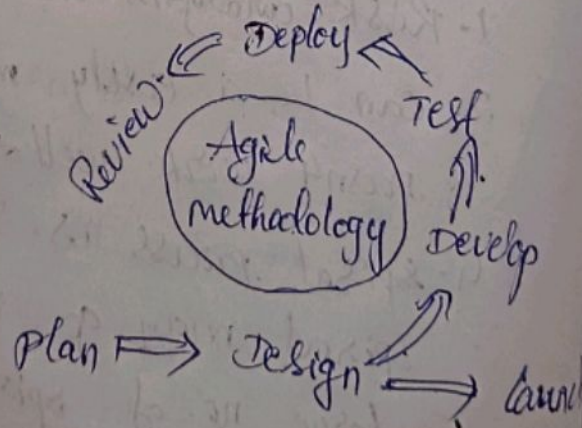
- * Divides requirements into multiple iterations & provide specific functionality for the release.
- * Delivers multiple software requirements.
- * Each iterations are lasts from two to 3 weeks.
- * Direct collaboration with Customers.
- * Rapid project development.

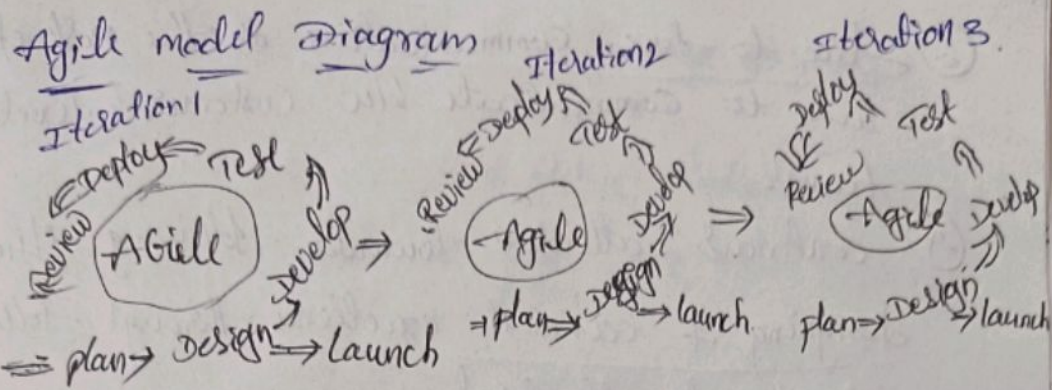
⇒ Traditional VS Agile model working with example

* For example +

- * Instagram social media application requirements are:

- ① Follow - unfollow option
- ② Edit - unfollow option
- ② Edit profile.
- ③ Search
- ④ messaging
- ⑤ Post photos
- ⑥ upload story
- ⑦ To make reels
- ⑧ Go live.





x when to use the Agile model?

- ① when project size is large.
- ② when frequent changes are required.
- ③ when a highly qualified and experienced team is available.
- ④ when a customer is ready to have a meeting with a software team all the time.
- ⑤ projects with flexible timelines and budget.

⇒ Agile principles (12).

- ① Highest priority is to satisfy the customers to early & continue delivery of sw.
- ② Being a flexible about changing requirements at any point of development.
- ③ working on frequent and short deliveries like couple of weeks or months with preference.
- ④ Transparency b/w business people & developers and requires them to work together.
- ⑤ By providing a better productive environment and providing them with all the support motivation it leads to better productivity.

⑥ Face to face communication as the most effective way to communicate b/w customer & development team.

⑦ Continuous attention towards following Effective designing & technical excellence through following optimal code standard.

⑧ It promotes Sustainable development by developers, users and sponsors should work together.

⑨ Continuous attention to technical excellence and good design enhances agility.

⑩ Simplicity is the art of maximum result & less hard work by removing unnecessary tasks and prioritizing activities.

⑪ The best architectures, requirements and design emerge from Self organizing & experience team.

⑫ For developing effective Software, regular analysis and work on improving the overall delivery of the development process.

* Advantages of Agile model :

① Supports customer involvement and customer satisfaction.

② Strong communication of the dev team with the customer.

③ Little planning required.

④ Efficient design and fulfils the business requirements.

⑤ ~~Efficient~~ design and anytime changes are acceptable.

⑥ provides a very realistic approach to SW development.

⑦ updated versions of functioning SW are released every week.

⑧ It reduces total development time.

* Disadvantages of Agile model

① Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

② Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

UNIT - II

Software Requirements

Functional and non-functional requirements

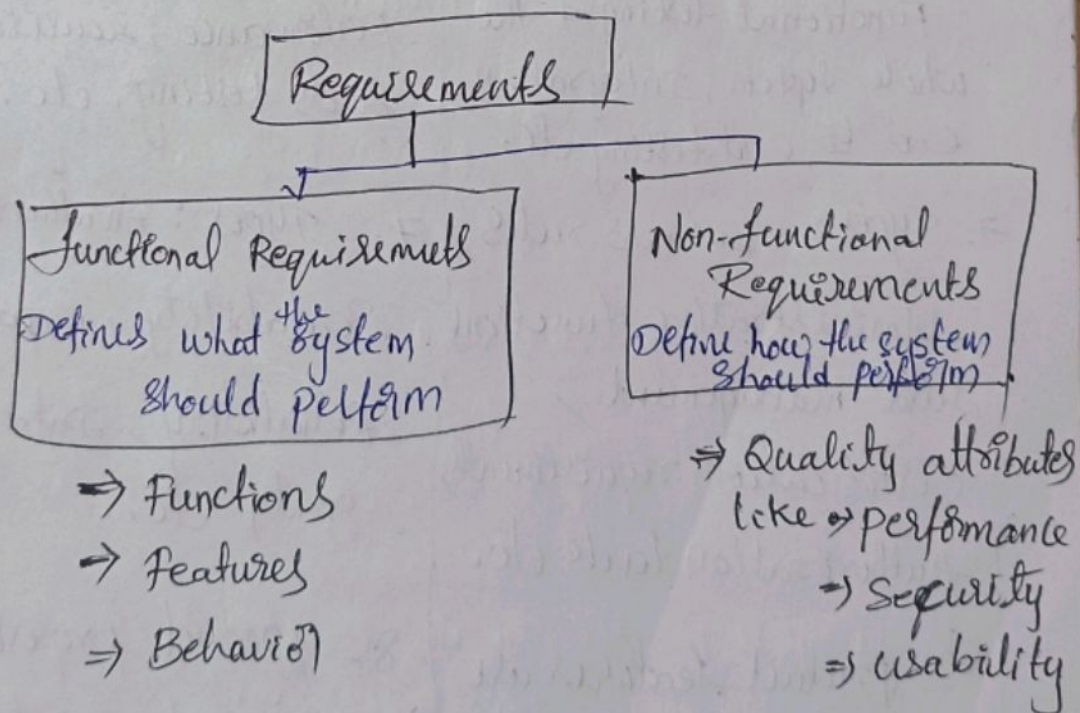
* What is SW Requirements

- Definition - According to IEEE standard, SW requirement is a Condition needed by a user to solve a problem or achieve an objective.
- Requirement should be clear, correct & well-defined.

* Types of Requirement

There are 2 types of requirements:

1. Functional requirements.
2. Non-functional requirements



* Difference b/w Functional and Non-functional Reqts

Functional Requirements	Non-function Requirements
1. Describe what the system should do.	1. Describe the how the system should perform. (partly testing)
2. Covers all the functions that the s/w must perform	2. Cover all aspects of good user experience.
3. Ensure all core functionality is well-performed.	3. Ensure user's needs are satisfied.
4. Easy to specify.	4. Difficult to specify
5. They are tested first.	5. They are tested after functional testing.
6. <u>what is tested</u> : API testing, Functional testing of the whole system, Integration, End to End testing etc.	6. <u>what is tested</u> : Usability, performance, security, stress testing, etc..
7. <u>Types</u> : Business rules, Administrative functions, Data management, Certification requirements, Authorization levels etc.	7. <u>Types</u> : Availability, Scalability, capacity, Reliability, Data integrity etc..
8. product features are end-result in FRs.	8. product properties are end-result in NFR'S

Examples for FRs & NFRs

1. ATM System (Real-time) :-

FRs :-

- The system shall allow users to insert ATM card.
- The system shall authenticate the PIN.
- The system shall allow users to withdraw cash.
- The system shall display account balance.
- The system shall print transaction receipt.

NFRs :-

- The system shall respond within 5 seconds after PIN entry.
- The system shall be available 24x7.
- The system shall ensure secure transactions using encryption.
- The system shall support multiple languages.
- The system shall handle 1000 transactions per minute.

2. Online shopping Application (Amazon-like)

3. Hospital Management System.

4. Student / college Management system.

* User Requirements +

* Definition + High level statements of services and constraints, often written by clients or users in simple, non-technical language.

* Purpose :- To explain the purpose of the application and the user's intended functionality.

* Components + often documented as user stories, use scenarios, or simple functional lists, such as "The user must be able to login".

* Focus :- problem domain, business goals and user interface needs.

① Ex Food delivery app

① "I want to see near by restaurants".

② "I want to order food and pay one".

③ "I want to get notifications about my order".

② Ex Railway Reservation system.

③ Ex College management.

④ Ex Online shopping App.

* System Requirements +

* Definition : Detailed, precise descriptions of the system's functions, services, and operational constraints.

* Purpose :- To guide developers in designing and building the system.

* Components : Documented in a SW requirements specifications (SRS) document, covering functional requirements.

(e.g. "The system shall store password requirements. hashes using SHA-256") and NFRs (ex. system speed, memory).

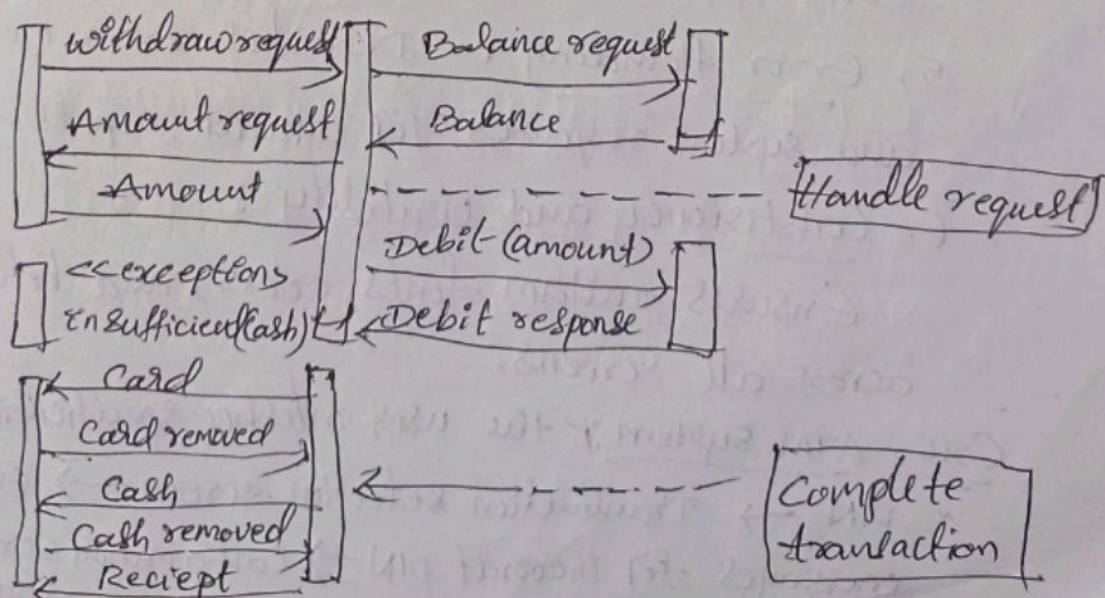
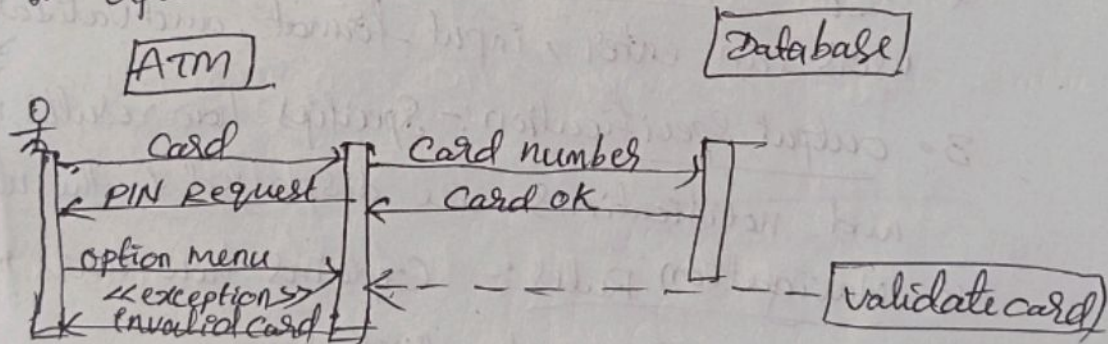
* Focus :- Solution domain, S/W logic, architecture, and technical interfaces.

Example :- ATM machine

User requirements :- I want to withdraw money from my account.

System requirements :-

- * The system shall verify the user's pin.
- * The system shall check account balance before withdrawal.
- * The system shall dispense cash within 5 seconds
- * Update the account balance after transaction.



* User Interface Specification - (UIS) (API)

* UIS is a document that describes how users interact with a software system. It defines the appearance, behavior and interaction of the user interface.

* Purpose of UIS :-

- To make the system easy and convenient to use
- To clearly define user-system interaction.
- To help developers, designers, and testers understand UI requirements.

* Contents of User Interface Specification :-

1. Screen Design :- Describes the layout of screens, including menus, buttons, icons and text fields.
2. Input Specification :- Defines the type of data users can enter, input format, and validation rules.
3. Output Specification :- Specifies how results, messages and notifications are displayed to the user.
4. Navigation Rules :- Explains how users move from one screen to another.
5. Error Handling :- Describes error messages and system responses for invalid inputs.
6. Consistency and Usability :- Ensures uniform fonts, colors, and behavior across all screens.

Ex ATM system if the user interface specification includes

* PIN → Transaction Selection screen → Error messages for incorrect PIN → Confirmation screen after transaction.

* Requirement Engineering process (REP)

* The process to gather the SW requirements from customers, analyze and document them is known as Requirement Engineering.

* Following four steps involve in Requirements Engineering process:

Feasibility study

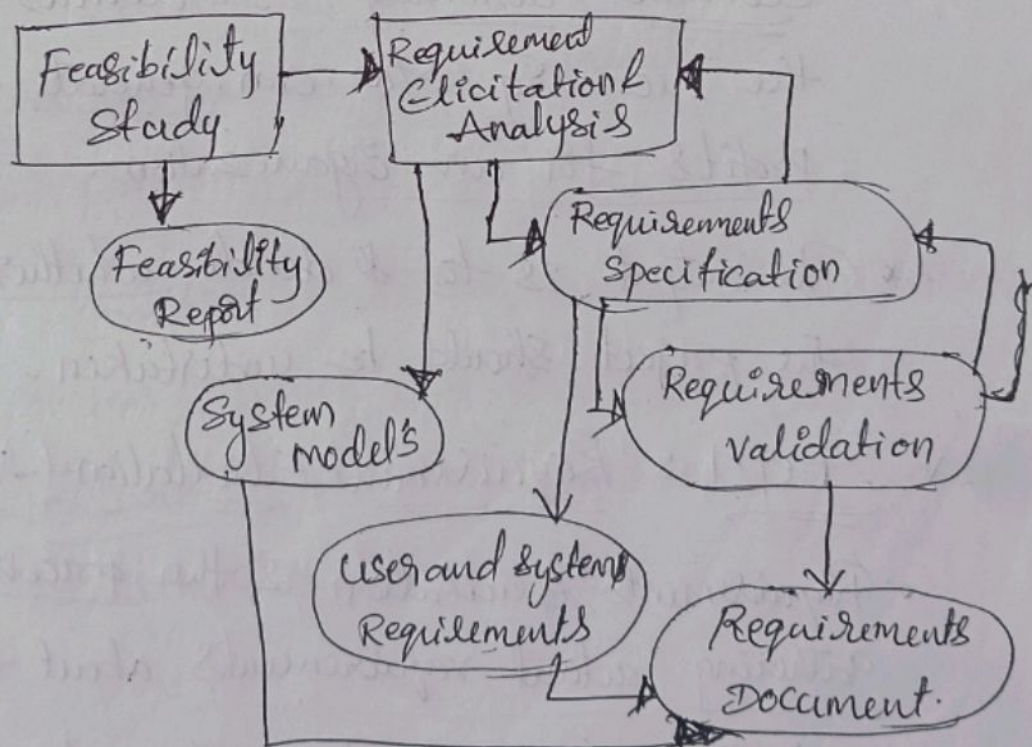
1. Requirement Elicitation and Analysis. (REA)

2. SW Requirement Specification. (SRS)

3. SW Requirement Validation. (SRV)

4. SW Requirement management. (SRM)

Requirement Engineering process Diagram:-



REP Diagram

* Feasibility Study

- To create and analyze the reasons for developing the software that is acceptable to users.
- To check aim and goal of the customer & organization behind implementation of software.

* Types of Feasibility Study

1. Technical Feasibility : Evaluates the current technologies, which are needed to achieve customer requirements within the time and budget.
2. Operational Feasibility :- Required SW solve the business problems and customer requirements.
3. Economic Feasibility :- It decides whether the necessary SW can generate financial profits for an organization.

* The output is to decide whether or not the project should be undertaken.

* Step 1: Requirement Elicitation & Analysis

- Requirements Elicitation is the process of gathering actual requirements about the needs and expectations of stakeholders for a SW system.

- To understand Customers, business manuals, the existing s/w of same type, standards and other stakeholders of the project.

• Techniques of the Requirement Elicitation -

1. Interviews: one-on-one conversations with stakeholders to gather information about their needs and expectations.

2. Surveys :- These are questionnaires that are distributed to stakeholders to gather information.

3. Focus Groups - Small groups of stakeholders who are brought together to discuss their needs.

4. Observations :-

observing the stakeholders in their work environment to gather information.

5. Prototyping :-

Creating a working model for the software system, which can be used to gather feedback from stakeholders and to validate requirements.

Step 2 : Sw Requirement Specification (SRS)

- Requirements received from client are written in natural language.
- After collecting, system Analyst to document the requirements in technical language.
- SRS include Data Flow Diagram, Data Dictionary, Entity Relationship Diagram.

➤ SRS Defines +

- How the intended Sw will interact with hardware.
- External Interface / GUI
- Speed of operation, Response time of System.
- portability of Software across various platforms.
- Maintability
- Speed of recovery after crashing.
- Security, Quality, limitations etc.

Step 3:- Software Requirement Validation

- After SRS Developed, the requirements discussed in this document are validated & tested.
- Requirement Validation done through Requirement reviews taken by customers, After developing prototyping check with customers & Test case generations.

* Requirements can be checked against the following conditions:-

- If they can practically implement.
- If they are correct and as per the functionality and specialty of software.
- If there are any ambiguities.
- If they can describe.
- Any changing or additional requirement.

Step 4: Software Requirement Management &

- Requirement management is the process of managing changing requirements during the requirements engineering process.

► Activities involved in Requirement Management

1. Tracking & Controlling Changes:

Handling changing requirements from customer

2. Version Control :- Keep tracking of different version of system.

3. Traceability : Trace to software is design, develop or test as per requirements or not.

4. Communication : If changing requirements has any issues contact with clients within time.

5. Monitoring & Reporting : Monitoring development process & report the status.

* Important questions

1. Explain in detail Requirement Engineering functions / process ? (7 marks)

2. What do you mean by Feasibility study in Requirement elicitation ? (6 marks)

3. State and Explain different methods of Requirement elicitation.

UNIT - III

Design Engineering

* Design engineering encompasses ^(including) the set of principles, concepts and practices that lead to the development of high-quality system or product.

(or)

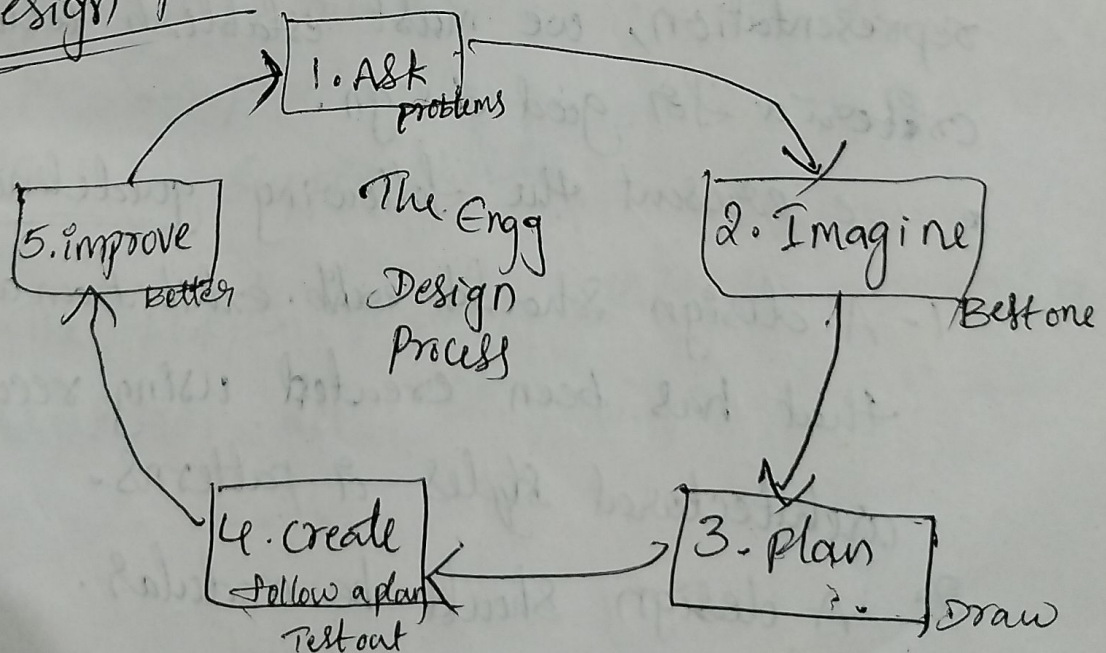
* Design Engineering is the process of applying scientific and Engg. principles to design, develop, and improve products, machines or systems.

* Design is what virtually every engineer wants to do.

* Design allows a software engineer to model the system or product that is to be built.

* Design depicts the flow in a new different way.

* Design process



* Design process and Design Quality

So design is an iterative process through which requirements are translated into a "blueprint" for constructing the software.

* 3 characteristics that serve as a guide for the evaluation of a good design.

1. The design must implement all of the explicit requirements in the analysis model.

2. The design must be readable.

3. The design should provide a complete picture of the software.

* Each of these characteristics is actually a goal for the design process.

* Quality of design :-

In order to evaluate the quality of a design representation, we must establish technical criteria for good design.

* we present the following guidelines :-

1. A design should ~~exib~~ exhibit an architecture that has been created using recognizable architectural styles or patterns.

2. A design should be modular.

3. A design should contain distinct representations of data, architecture, interfaces, and components.

4. A design should lead to data structures that are appropriate for the classes to be implemented.
5. A design should lead to interfaces that reduce the complexity of connections b/w components and with the external environment.
6. A design should be derived using a repeatable method.
7. A design should be represented using a notation.

⇒ Quality attributes :-

* ~~Qual~~ Hewlett-Packard developed a set of SW quality attributes that has been given the acronym FURPS:

F - Functionality
U - Usability
R - Reliability
P - Performance
S - Supportability.

* The FURPS quality attributes represent a target for all SW design.

Design Concepts +

* A set of fundamental software design concepts has evolved over the history of software engineering.

* M.A. Jackson once said. "The beginning of wisdom for a software engineer is to recognize the difficulty of getting a program to work, and getting it right."

1. Abstraction + It is one of the fundamental ways to that we as humans cope with complexity.

* As we move through different levels of Abstraction, we work to create procedural and data abstractions.

(i) A procedural abstraction refers to a sequence of instructions that have a specific and limited functions. ex: open for a door, walk to the door.

(ii) Data abstraction + is a named collection of data that describes a data object.

ex: Door (door type, swing direction, opening mechanism, weight, dimensions).

2. Architecture:-

3. patterns

4. modularity

5. Information Hiding

6. Functional independence

7. Refinement

8. Refactoring

9. Design classes

2. Software Architecture +

It alludes to "the overall structure of the S/W and the ways in which that structure provides conceptual integrity for a system."

* A S/W architecture is the development work product that gives the highest return on investment with respect to quality, schedule, and cost.

* The architectural designs can be represented using one or more of a no. of different models.

1. Structural models : represent architecture as an organized collection of program components.

2. Framework models: increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of Apps.

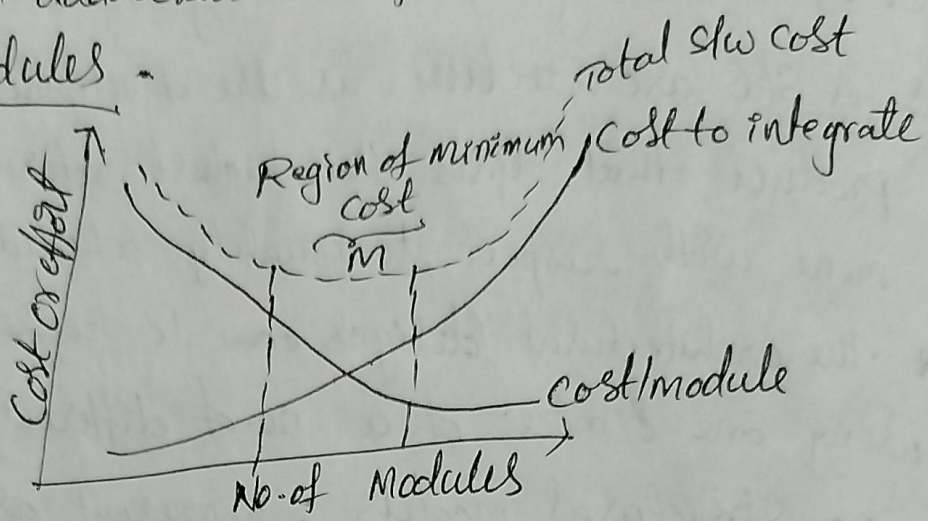
3. Dynamic models :- address the behavioral aspects of the program architecture.

4. Process models :- focus on the design of the business or technical of external events

5. Functional models + Can be used to represent the functional hierarchy of a system.

3. Patterns + A design pattern describes a design structure that solves a particular design problem within specific context and amid "Forces" that may have an impact on the manner in which the pattern is applied and used.

4. modularity: Software Architecture and design patterns embody modularity. (embody - represent) that is SW is divided into separately named and addressable components, sometimes called modules.



Modularity and software cost

* This leads to a "divide and conquer" strategy - It's easier to solve a complex problem when you break it into manageable pieces.

5. Information Hiding $\downarrow$

The principle of Information Hiding suggests that modules be "characterized by ~~design~~ design decisions that each module hides from all others

* The use of information hiding as a design criterion for modular systems provides greatest benefits when modifications are required during testing,
 because during software maintenance.

* Most data and procedure are hidden from other parts of the software.

6. Functional Independence It is achieved by developing modules with "single minded" function.
* we want to design SW so that each module addresses a specific subfunction of requirements.
* Independence is assessed using two qualitative criteria: cohesion and coupling

• Cohesion (^{unity} ^{Bonding} ^{togetherness})

1. Cohesion is an indication to which elements within a module are related.

2. It focuses inside a module.

3. High cohesion is a desirable good design.

4. performs single and well defined task.

5. It affects changes only in that module.

• Coupling (^{Dependency} ^{connection})

1. Coupling is an indication of dependency between modules.

2. Between modules.

3. Low coupling is a desirable good design.

4. performs in minimal interconnection.

5. change may affect other module.

* Cohesion It is indication of the relative functional strength of a module.

* Coupling is an indication of the relative interdependence among modules.

7. Refinement - Refinement is actually a process of elaboration. we begin with a statement of function that is defined at a high level of abstraction.
- * Stepwise refinement is a top-down design strategy originally proposed by Niklaus Wirth.
 - * A program is developed by refining levels of procedural detail.
 - * Refinement and ~~abstract~~ abstraction are complementary concepts.

8. Refactoring - Refactoring is a reorganization technique that simplifies the design or code of a component without changing its function or behavior.
- * Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure.

9. Design classes - Design classes represent the blue print of the software system. The analysis model defines a complete set of analysis classes.

- * Each of these classes describes some element of the problem domain, focusing on —

- * what objects exist
- * what data they store.
- * what operations (methods) they perform
- * how they interact with each other.

* 5 different types of design classes, each representing a different layer of the design architecture:

1. User interface classes.

2. Business Domain classes.

3. process classes.

4. persistent classes.

5. system classes.

1. User interface classes define all abstractions that are necessary for human-computer interaction.

* In many cases, HCI occurs within the context of a metaphor (eg. a checkbook, an order form, a fax machine).

2. Business classes are often refinements of the analysis classes defined earlier.

* The classes identify the attributes and services (methods) that are required to implement some element of the business domain.

3. process classes implement lower-level abstractions required to fully manage the business domain classes.

4. Persistent classes represent data stores (eg. Databases) that will persist beyond the execution of the software.

5. System classes implement software management and control functions that enable the system to operate and communicate within its computing environment and with the outside world.

* As the design model evolves, the software team must develop a complete set of attributes and operations for each design class.

* Design classes define 4 characteristics:

1. Complete & sufficient
2. Primitiveness
3. High cohesion
4. low coupling

* The Design model - A Design model is a blueprint of how the system will be built.

* It transforms the requirements into a structured solution that developers can implement.

* It comes after the analysis phase and before coding.

* Design model can be viewed in 2 different dimensions: The process Dimension and The Abstraction Dimension.

* The process Dimension indicates the evolution of the design model as design tasks are executed as part of the software process.

* The abstraction Dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.

* Design model Describes -

- The structure of the system.
- The components / modules.
- The relationships b/w them.
- The data flow and interaction.

* Elements of the Design model -

1. Data Design elements
2. Architecture Design elements.
3. Interface Design elements.
4. Component-level Design elements.
5. Deployment-level Design elements.

1. Design Data elements - Define how data is structured, stored, and organized in the system.
elements : *

- * Design of database
- * Data structures
- * Data organization.

2. Architectural Design elements - Describe the overall structure of the system and how its major components are arranged.

- * Overall system structure.
- * Subsystem & layers -
- * How Components are organized.

3. Interface Design elements :- Specify how users, components and external systems interact with the system.

1. User Interface (UI) 2. External Interfaces.

3. Internal Interfaces b/w Components -

4. Component level elements - Detail the internal logic and processing of individual modules.

- * Algorithms.

5. Deployment-level Design elements :-

Explain how the software will be installed and run on hardware and n/w environments.

- * Hardware configuration
- * System installation.
- * n/w setup.

* Creating an Architectural Design

→ Architectural design represents the structure of data and program components that are required to build a computer-based system.

→ A software engineer can design both data and architecture.

→ A database or data warehouse designer creates the data architecture for a system.

→ The "system architect" selects an appropriate architectural style for the requirements derived from system engineering and software requirements analysis.

→ Architectural design begins with data design and then proceeds to the derivation of one or more representations of the architectural structure of the system.

→ Alternative architectural styles and patterns are analyzed to derive the structure that is best suited to customer requirements and quality attributes.

→ Once an alternative has been selected, the architecture is elaborated using an architectural design method.

* What is work product?

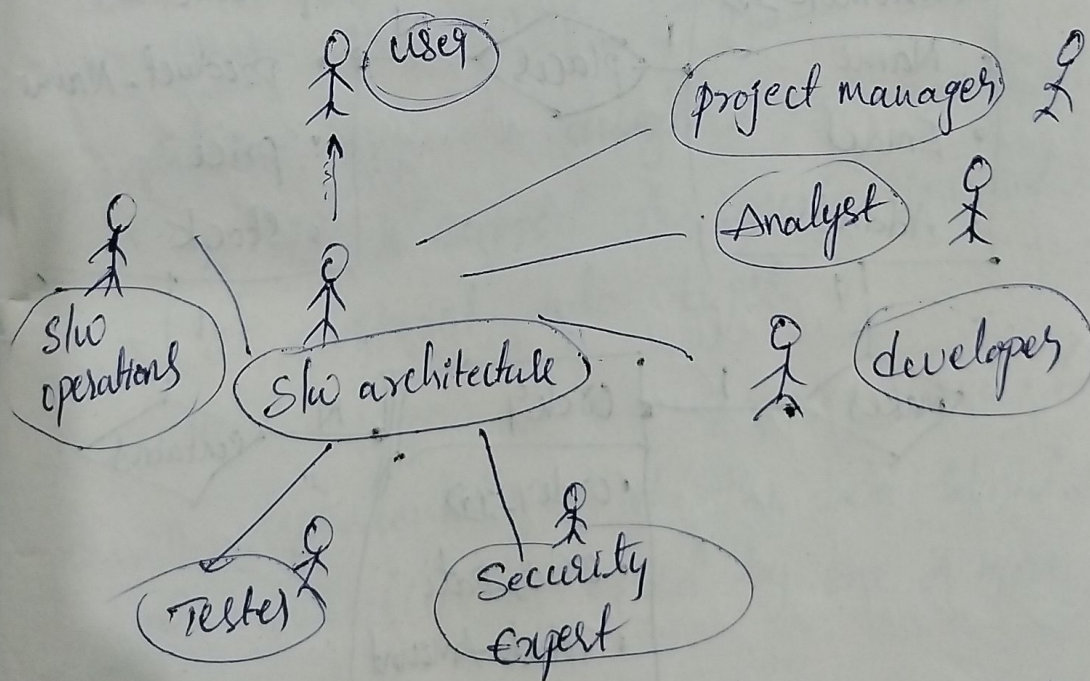
→ An architecture model encompassing data architecture and program structure is created during architectural design.

- * In addition, component properties and relationship (interaction) are described.
- * At each stage, sw. design work products are reviewed for clarity, correctness, completeness, and consistency with requirements and with one another.

* Software architecture +

→ It is the structure of the sw system like blue prints in building structure.

- It consists of sw components, details about datastructures and algorithms,
- relations among components.
 - data flow, control flow, dependencies.

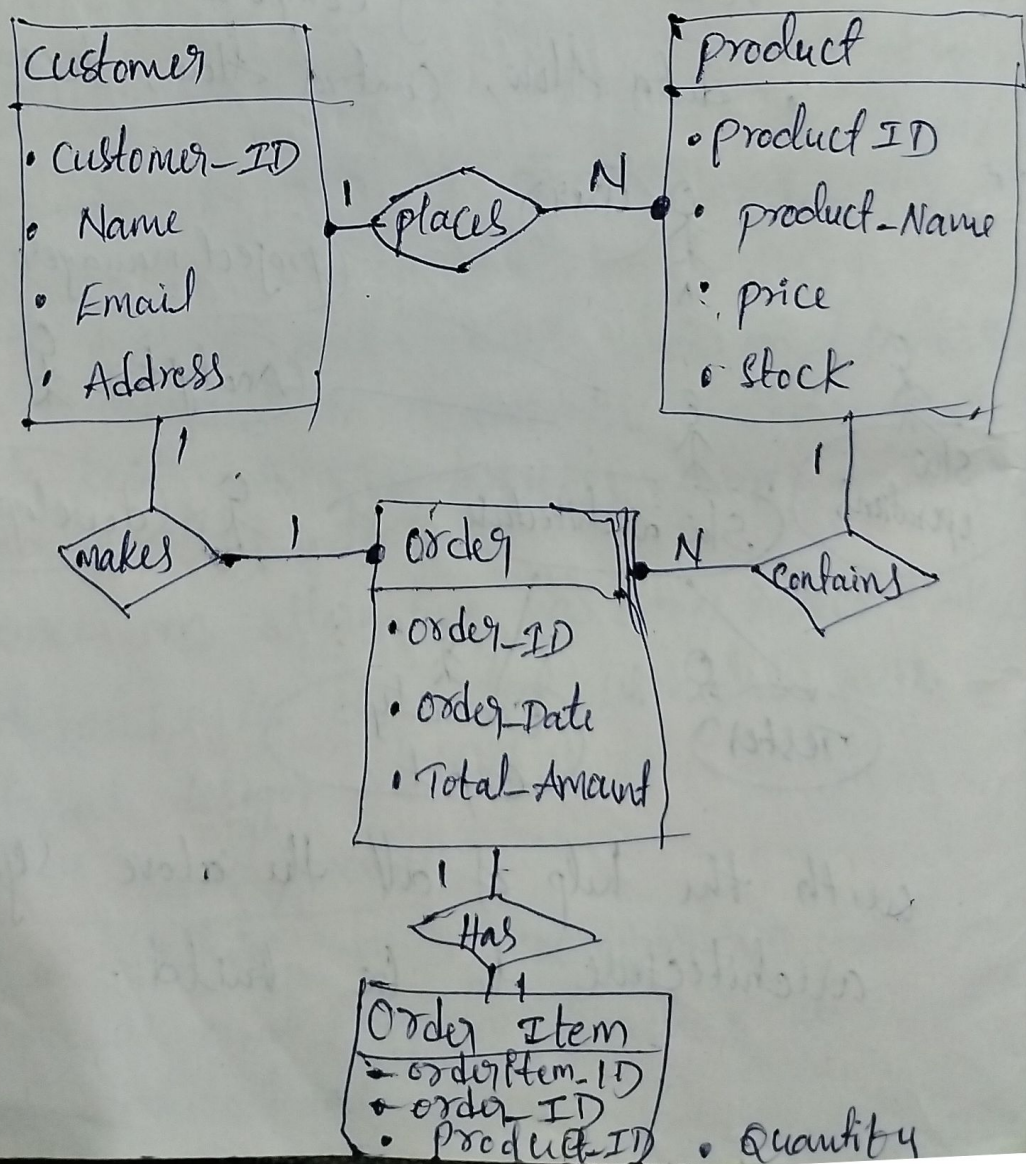


with the help of all the above system sw architecture to be build.

* Elements of S/W Architecture

- ⇒ Components → (abstract unit of S/W ^{instructions & internal state} which provides transformation of data)
- ⇒ Connectors - communicate, co-operate among components
- ⇒ Configuration topologies → structure of architectural relationships
- ⇒ System models → different architectural styles used.

* Data Design + It is the process of transforming data requirements identified during requirements analysis into a detailed structure specification. ex 18. E-commerce System.



* Data design at architectural level :- 5

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 company & community has developed data mining techniques also called knowledge discovery in database (KDD) to extract ~~import~~ info from database.

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

* Data Design at component level :-

(principle) A software design & programming languages should support the specification and realization of abstract data types.)

→ Data design at the component level, focuses on representation of the data structures that are directly accessed by one or more S/W components.

* Data specification, following set of principles:-

1. The systematic analysis principles.
2. All data structures and the operations.

3. A mechanism for defining the content of each object.

4. Low-level data design decisions &

5. The representation of data structure.

6. A library of useful data structures and the operations.

7. A software design and programming language.

* Architectural styles and patterns:-

* These are fundamental concepts in SE that define how a system is structured and how its components interact.

* Architectural style defines a family of systems in terms of a pattern of structural organization. It gives a high level of structure of the system.

* Here, each style describes a system category that encompasses:

1) a set of components : (eg: a database, modules)

2) a set of connections : that enable communication, co-operation, co-ordination among components

3) Constraints : defines how components can be integrated to form the system. and

4) Semantic modules : enables a design to understand the overall properties of a system

by analysing the known properties of its constituent parts.

There are some Architectural styles:

① Data-centered Architectural styles

② Data-flow

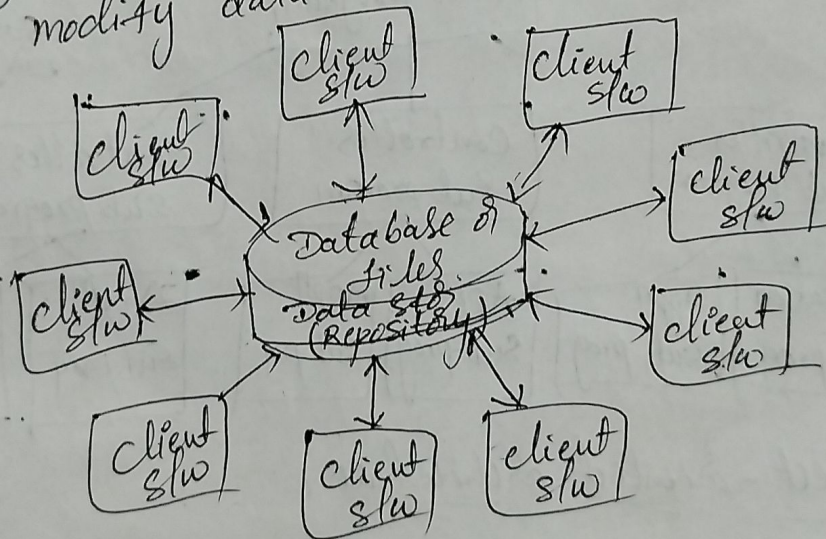
③ call & return

④ object-oriented

⑤ layered architecture styles.

① Data-centered Architecture style :-

A data store (files 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.

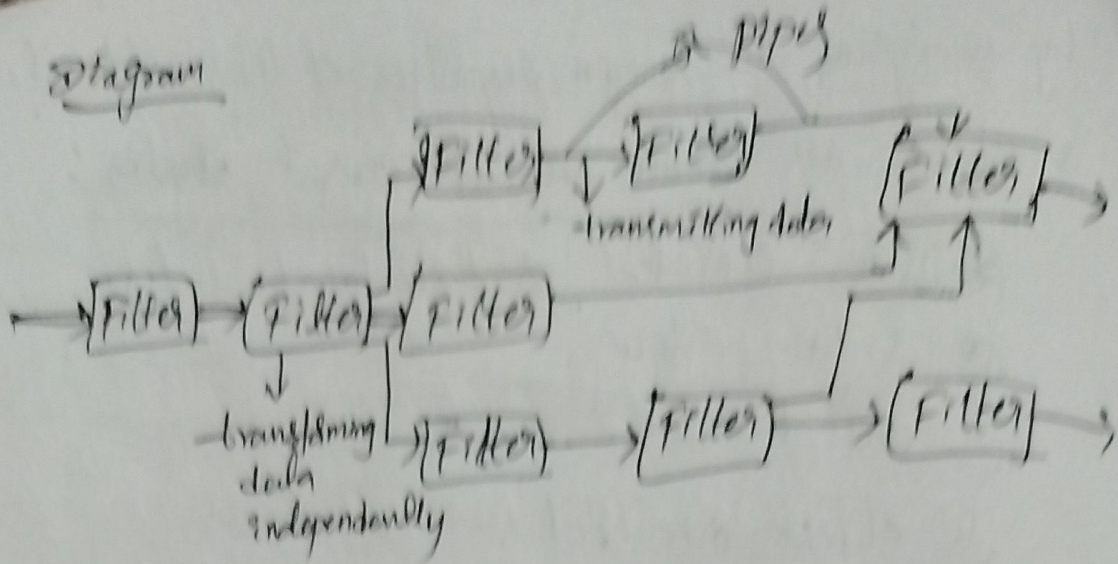


② Data-flow Architecture &

This architecture is applied when i/p data are to be transformed through a series of computational or manipulative components into o/p data.

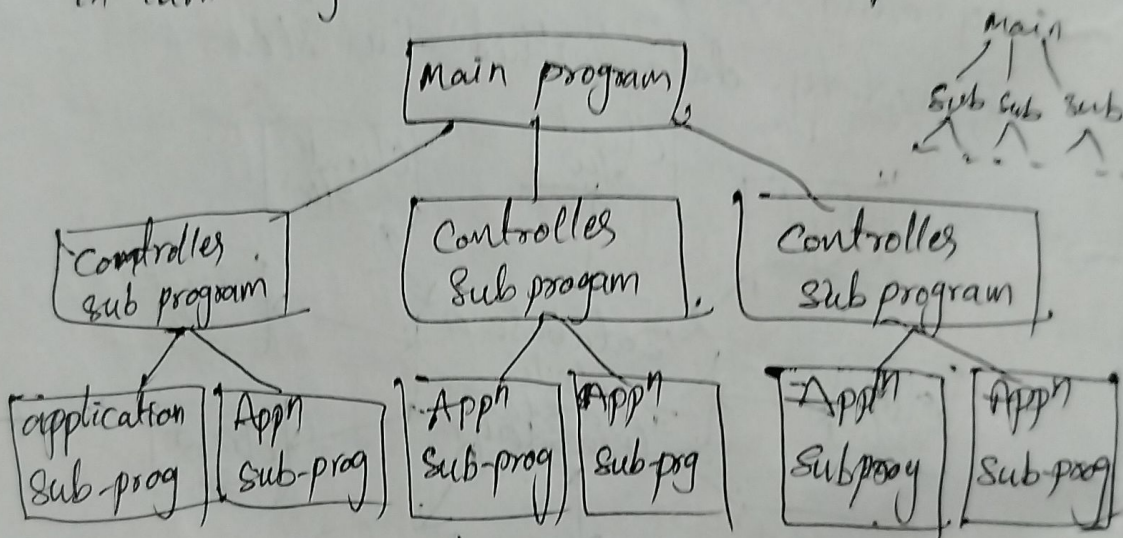
Eg: pipe and filter pattern :- filters transforming data independently of other components, pipes transmitting data.

Diagram



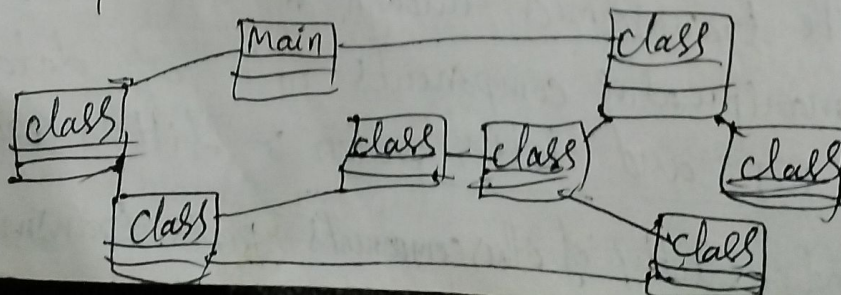
③ Call & Return architecture :-

A ^{Main} program / sub program Architecture :- Decompose function into a control hierarchy where a "main" program invokes a no. of program components, which in turn may invoke still other components.



④ Object-oriented architecture :-

The components of a system encapsulates data and operations that must be applied to manipulate the data.

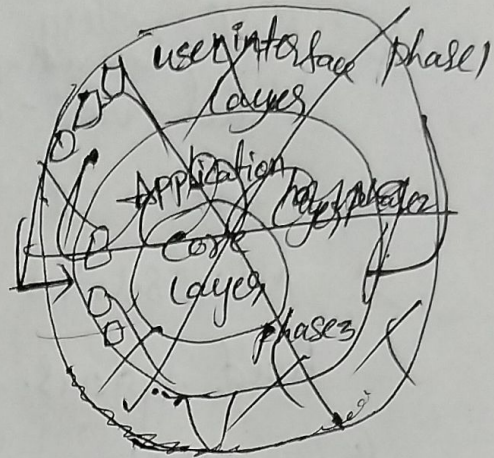


15X

- * The communication & co-ordination b/w components is accomplished by a message passing.

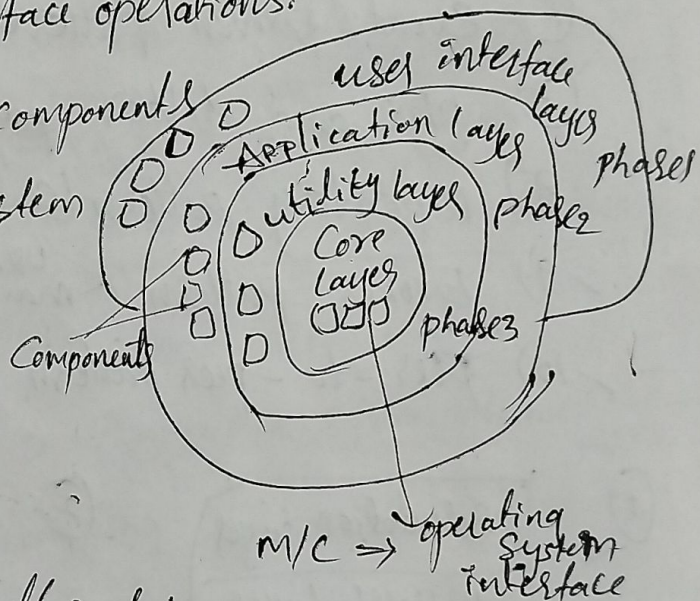
⑤ Layered Architectural

- * No. of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.



- * The outer layer components service user interface operations.

- * The inner layer components perform operating system interfacing.



* Architectural patterns

It provides reusable solutions to commonly occurring problems in software architecture.

- * They define how components interact, how data flows and responsibilities are organized.

Here are most important architectural

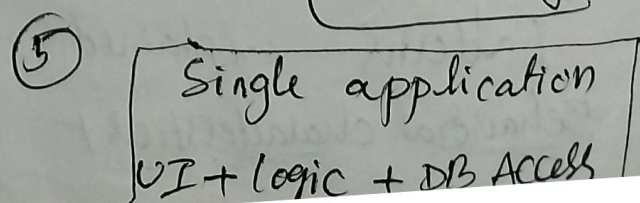
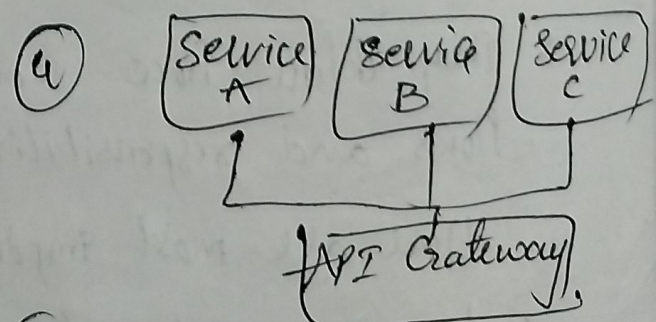
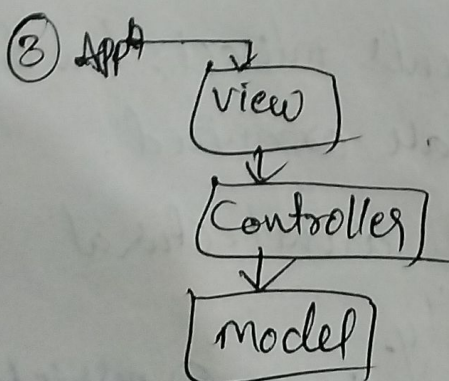
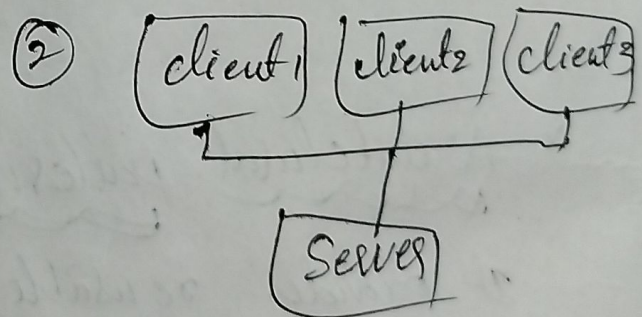
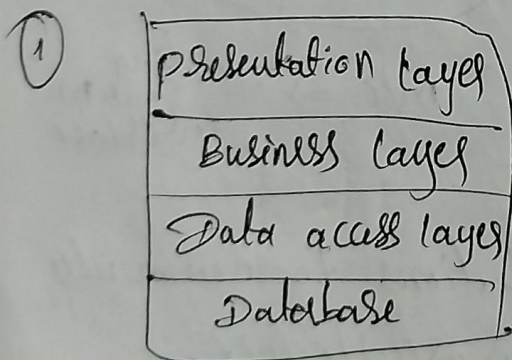
Patterns explained clearly.

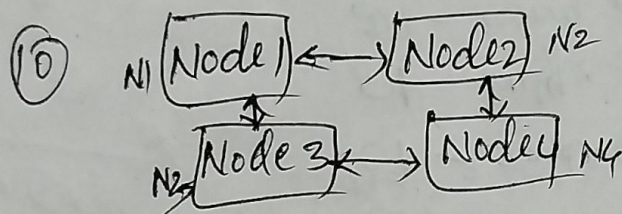
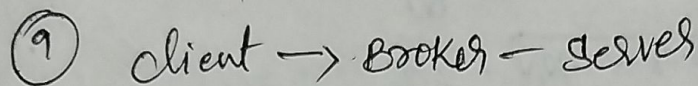
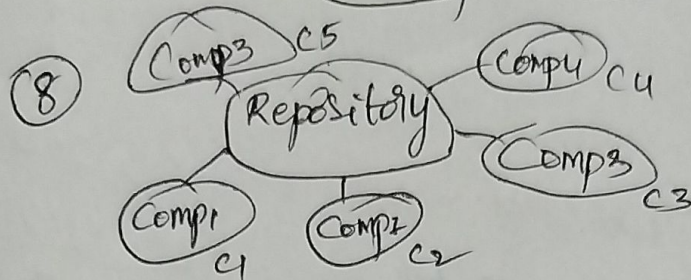
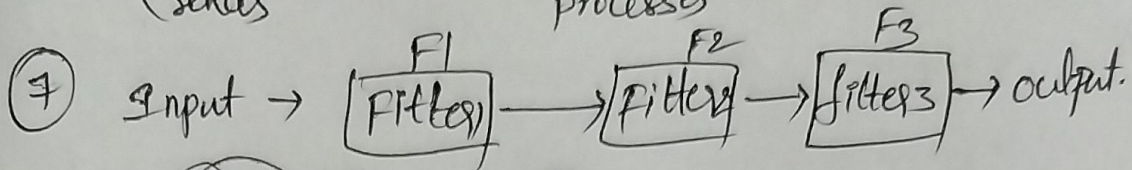
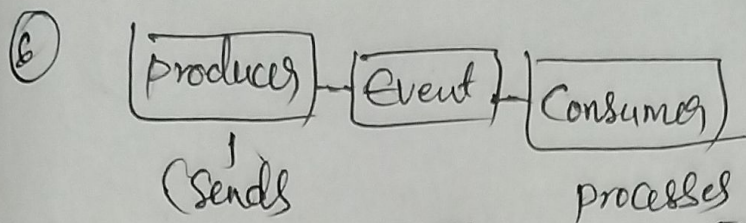
Behavioral characteristics +

① concurrency ② persistence

③ Distribution.

- ① Layered pattern (n-tier architecture)
- ② Client-server pattern (ex: UI → Business Logic → DB)
(browser (client) - web server)
- ③ MVC pattern ^{separates app into} improves maintainability & scalability
(Model - View - Controller)
Data UI Logic
- ④ Microservices pattern: used in modern cloud-based systems.
- ⑤ Monolithic pattern (Entire app is built as a single unit)
- ⑥ Event-Driven pattern (Notification Systems)
- ⑦ pipe and filter pattern (Compiler Design)
- ⑧ Repository pattern (Central place to store and manage data)
- ⑨ Broker pattern ^{used in distributed systems} manages communication ^{between components}
- ⑩ peer-to-peer pattern (file sharing system)





Arch. style - overall structure of a system
 pattern - Reusable solution to design problems.

* Advantages styles and patterns -

- styles
1. Clear structure - organizes the system into well-defined parts.
 2. Easy understanding
 3. Better communication (teams can discuss design using common styles (eg: layered))
 4. Scalability and maintainability.

- patterns
1. Better problem solving (provides standard solⁿs)
 2. Improved quality. (patterns are tested and reliable)
 3. Reusability. (solutions can be reused to multiple projects)
 4. Reduced development time.
 5. ~~Scal~~ Flexibility and consistency.

* Architectural Design :-

It represents the structure of data and program components.

* that are required to build a computer-based system.

* Architectural design, representing the system in context :-

- system engineer must model the context.

- Context diagram - models in which the system interacts with external entities.

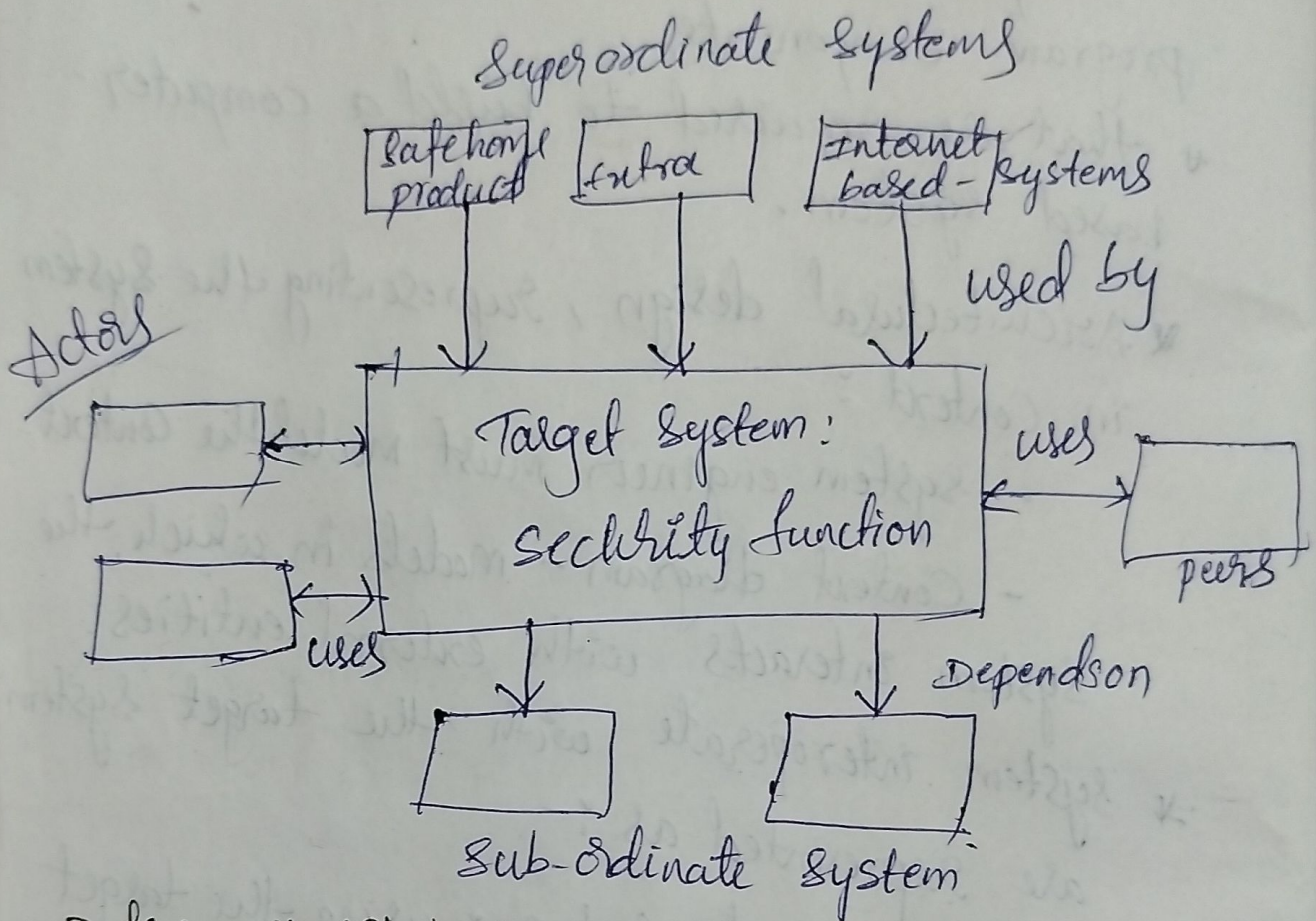
* system interoperate with the target system are represented as :-

- 1) Superordinate system :- using the target system as part of some high level processing scheme.

- 2) sub ordinate system : used by the target system to provide data & processing needed to complete the target system.

- 3) peer level system :- producing or consuming information needed by peers and the target system.

* Actors :- those entities that interact with the target system by producing or consuming information that is necessary for requisite.



* Defining (Architecture) Archetypes :- An archetype is a class or pattern that represents a case abstraction that is critical to design of an architecture for the target system. In general, a relatively small set of archetypes is required to design even relatively complex systems.

There are 4 types of Archetypes

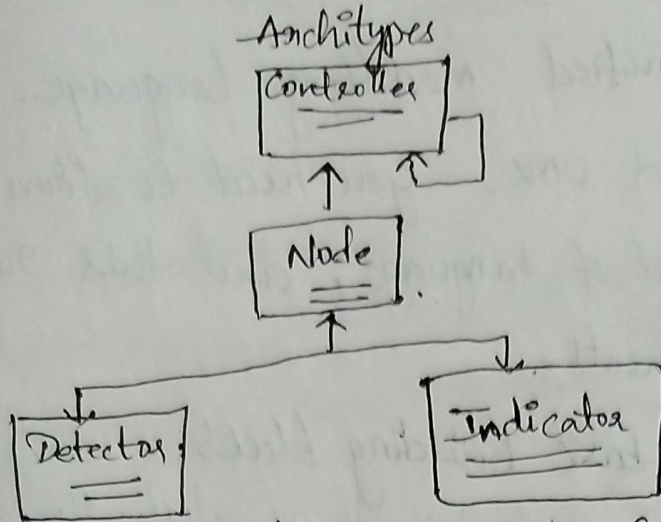
1) Node.

2) Detector.

3) Indicator.

4) Controller.

UML (Unified Modelling Language) relationship for safe home Security function archetypes:-

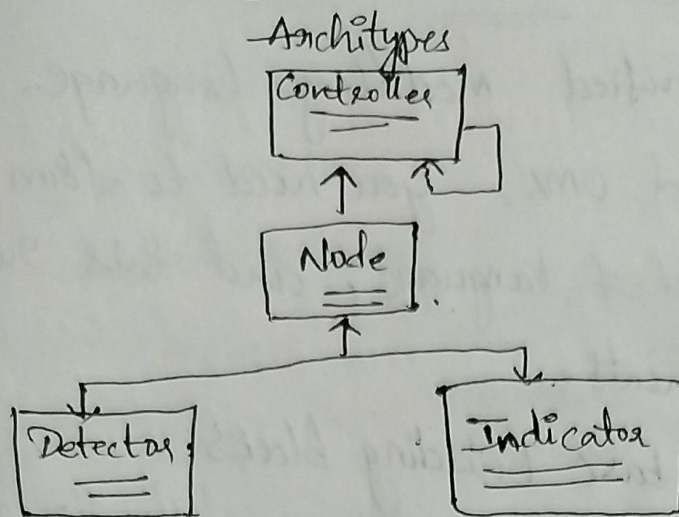


Refining the Archetype into Components
 As a shw architecture is refined into Components, the structure of System begins emerge.
 → the Architectural designer begins with the classes that were described as part of the analysis model. These analysis classes (domains) represent entities within the application domain that must be addressed within the shw architecture. Hence the application domain is one source for derivation & refinement of Component.

- to.
Describing instantiations for the System's

1. Architectural design that has been modelled to this point is still related the high level.
2. The Context of the System has been represented, archetypes that indicates the important abstractions, overall structure for the System, is occurrence & the major shw Components have been identified & further refinements is still necessary.
3. An actual instantiation of the architecture is developed. By this the architecture is applied to a Specific problem with intent of demonstrating that the structure & Components are appropriate.

UML (Unified Modelling Language) relationship for safe home Security function architectures:-



Refining the Architecture into Components

As a SW architecture is refined into components, the structure of system begins to emerge.

→ the Architectural designer begins with the classes that were described as part of the analysis model. These analysis classes (domains) represent entities within the application domain that must be addressed within the SW architecture. Hence the application domain is one source for derivation & refinement of components.

- to -

Describing instantiations for the system

1. Architectural design that has been modelled to this point is still related to the high level.
2. The context of the system has been represented by archetypes that indicate the important abstractions, overall structure for the system, its occurrence & the major SW components have been identified & further refinement is still necessary.
3. An actual instantiation of the architecture is developed. By this the architecture is applied to a specific problem with intent of demonstrating that the structure & components are appropriate.

* Conceptual Model of UML

UML - Unified Modelling Language.

→ To understand UML, you need to form a Conceptual model of language, and this requires 3 major elements.

① UML's basic building blocks:

② The rules.

③ Mechanism.

UML is an industry-standard graphical language for specifying, visualizing, constructing & documenting the artifacts of SW system.

* ① Building blocks of UML from: - UML combines techniques from: - data modelling - Business u - object modelling - component u

① Things. (abstraction & components)

(ii) Relationships. (connection b/w components)

(iii) Diagrams. (interactions of things & relationships)

xi) Things in the UML

There are four kinds of things in the UML:

a) Structural things.

b) Behavioral things.

c) Grouping things.

d) Annotational things.

① Structural things - static part of a model, representing elements that are either physical or Conceptual.

- classes

- interfaces

- use cases

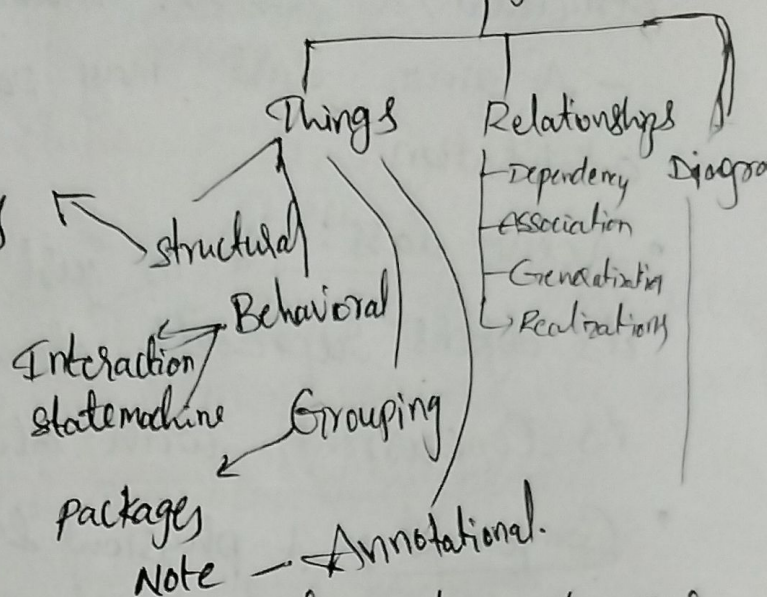
- collaborations

- Activity classes

- Components

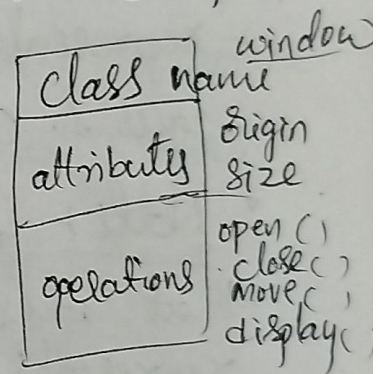
- Nodes

UML
building blocks

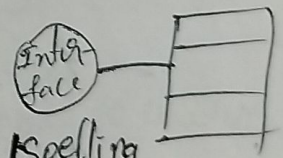


- Class: a class is a ~~set~~ of description of a set of objects that share the same attributes, operations, relationships and semantics.

- class implements one or more interfaces



- Interfaces A collection of operation that specify a service of a class or component.
 - Describes externally visible behavior of the element.
 - attached to class or component that spelling realizes the interfaces.



- usecase: a description of set of sequence of action that a system performs that yields to observable result of value to a particular actor.
 - It is realized by collaboration.

(place order)

• collaboration :- defines an interaction, which has structural, as well as behavioral dimensions.

- A given class may participate in several collaboration

• Active class: it is just like a class except that its objects represent elements whose behavior is concurrent with other elements. $\Downarrow$

• Component: A physical & replaceable part of system that conforms to and provides the realization of a set of interfaces.

- represents physical packaging of classes, interfaces & collaboration

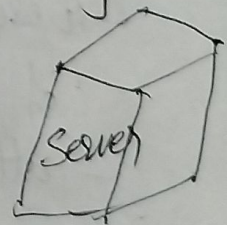
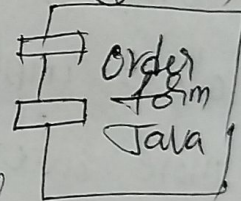
• Node: A physical element that exists at run-time and represents ~~physical packaging of classes~~, a computational resource, having at least some memory and often processing capability.

- A set of components may reside on a node & may also migrate from node to node.

Chain of Responsibility

EventManager

Suspend()
Flush()

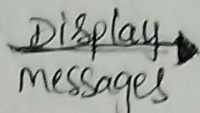


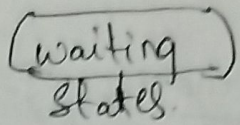
(b)

Behavioral things These are dynamic parts of UML models.

$\Rightarrow$ These are the verbs of a model, representing behavior and space.

⇒ Two primary kinds of behavioral things:

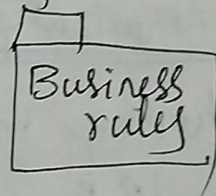
① interaction 

② state machine 

③ Grouping things :- these are the organizational parts of UML models. These are the boxes into which a model can be decomposed.

⇒ There is one primary kind of grouping.

• packages :- (mechanism for organizing elements into groups) - a package is purely conceptual meaning that it exists only at development time.



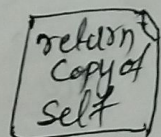
④ Annotational things :-

Annotational things are the explanatory parts of UML models.

⇒ These are the comments, you may apply to describe and remark about any element in a model.

⇒ There is one primary kind of AT.

• Note :- (a symbol for sending constraints and comments attached to an element or a collection of elements.



* Relationships in the UML :-

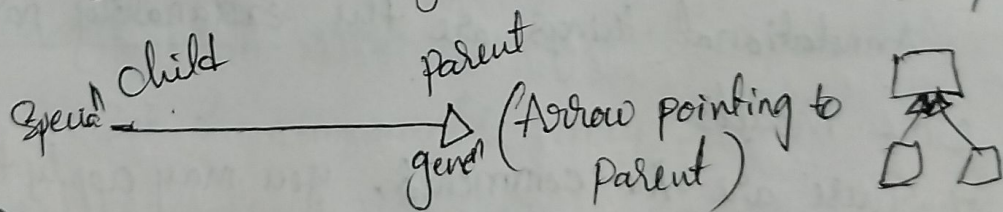
- There are four kinds of relationships in the UML :-

① Dependency :- Change of thing may affect the other. (----->)

② Association :- A structural relationship that describes a set of links, a link being a connection among objects.

Employer ¹-----^{*} Employee (It contains a multiplicity & role names)

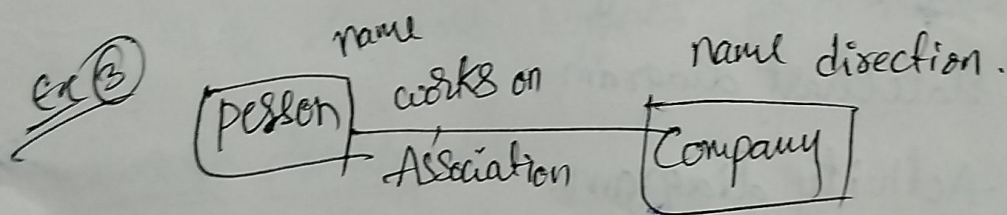
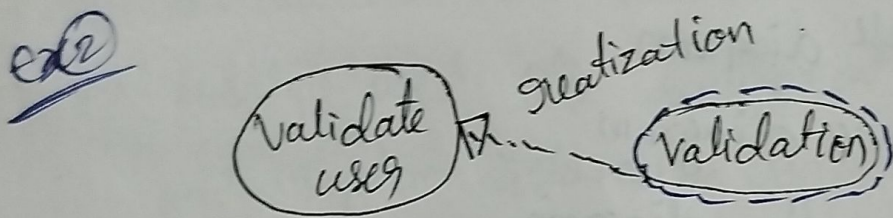
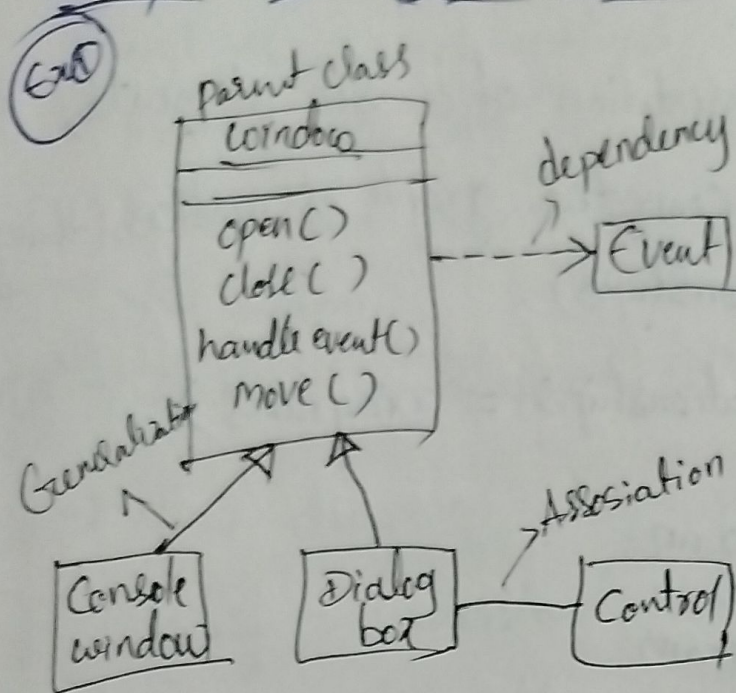
③ Generalization :- a specialization/generalization relationship in which objects of the specialized elements (the child) are substitutable for objects of the generalized element (the parent)



④ Realization :- A semantic relationship b/w classifiers, where in one classifier specifies a contract that another classifier guarantee to carry out.

Symbol :- ----->

Examples for how relationships communicating



- * A relationship means connection b/w different parts of the system, like classes, objects & components.
- * It help in organizing, reusing and managing SW systems efficiently.
- * It shows how one element is linked to interacts with another in a program.

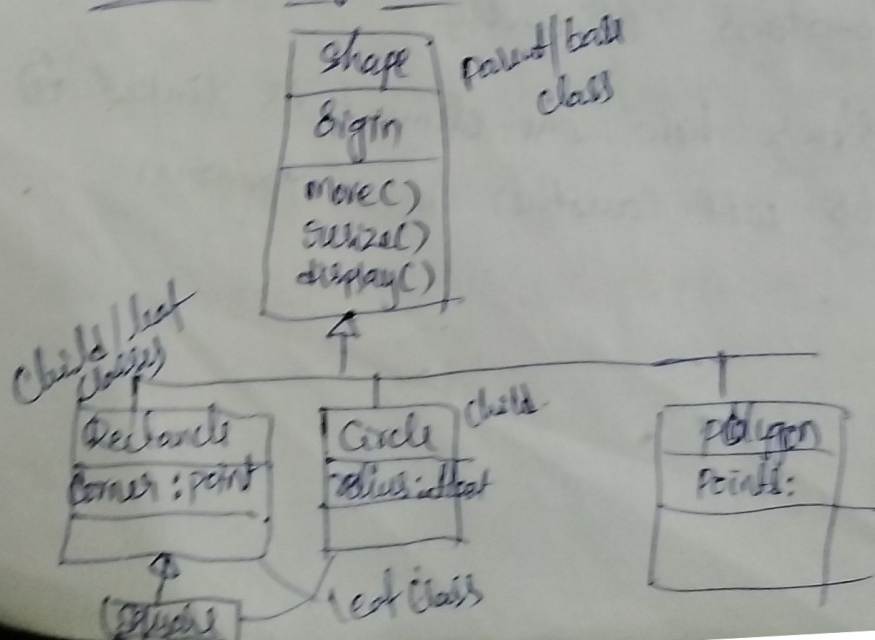
③ Diagrams in the OMT :-

→ Graphical presentation of a set elements, rendered as a connected graph of vertices (thing) and arcs (relationships).

(Things + relationships = diagrams).

- ① class diagram
- ② object diagram
- ③ usecase diagram
- ④ sequence diagram
- ⑤ Collaboration diagram.
- ⑥ statechart diagram
- ⑦ Activity diagram.
- ⑧ Component diagram.
- ⑨ Deployment diagram.

① Class diagram :-



- diagrams.
- class ~~diag~~ shows a set of classes, interfaces & collaborations and their relationships.
 - most common ~~diag~~ found in modelling object oriented systems and address the static design view of the system.

⑧ → class is a blueprint for objects in SW system.

→ These diagrams helps in system design visualization.

→ Easy to understand relationships and improves communication b/w developers.

→ mostly used SW development, database design and system architecture planning.

→ These diagrams supports object-oriented design.

⑨ UML diagrams

* Basic components of a class diagrams:-

* A class is represented as a rectangle with 3 sections :

• Top → class name

• middle → Attributes

• Bottom → methods/operations.

Attributes + These are variables or properties of a class.

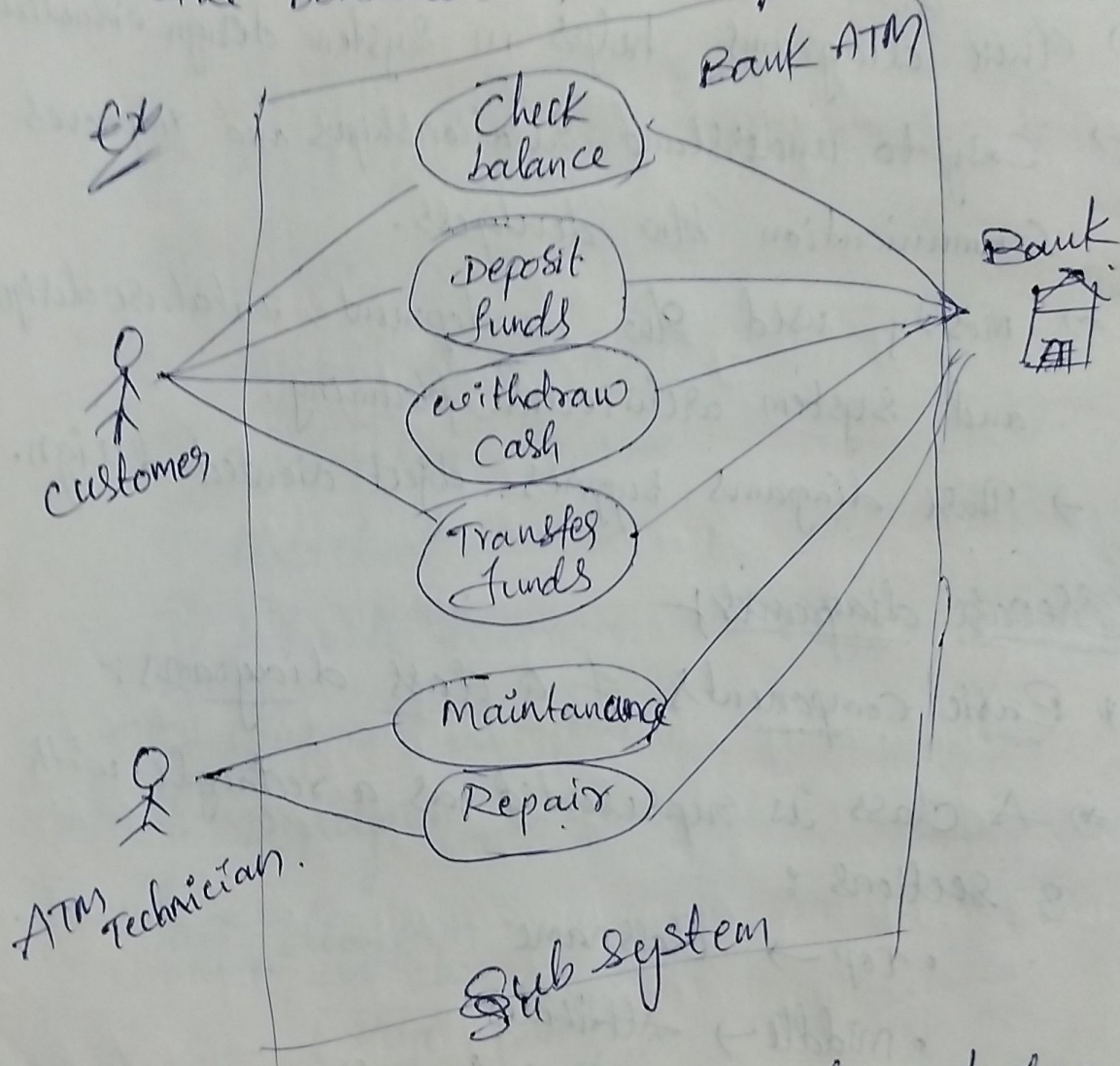
Example :- Name, age, id.

operations :- These are functions inside the class.

Ex: display(), calculate marks().

② Usecase diagrams

- * A usecase diagram shows a set of usecases and actors (special kind of class) and their relationships.
- * Address the static use case view of a system.
- * Specially important in organizing, understanding the behaviours of the system.



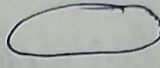
- * A usecase diagram is a type of diagram in UML used to represent the functional requirements of a system - that is, what system does from a user's perspective.
- * Main components - Actors, usecase, system boundary Relationships.

① Actor - Represents a user or external system.
ex Student, Admin, Customer.

Figure : stick figure 

② Use case - Represents a function or action performed by the system.

ex Login, Register, place order.

Symbol :- oval shape 

③ System boundary -

- Defines the scope of the system.
- All use cases are inside the boundary.

④ Relationships -

(a) Association - connects actor & usecase
- shows interaction.

(b) Generalization - Inheritance b/w actors & usecases

③^u Sequence diagram & collaboration diagram

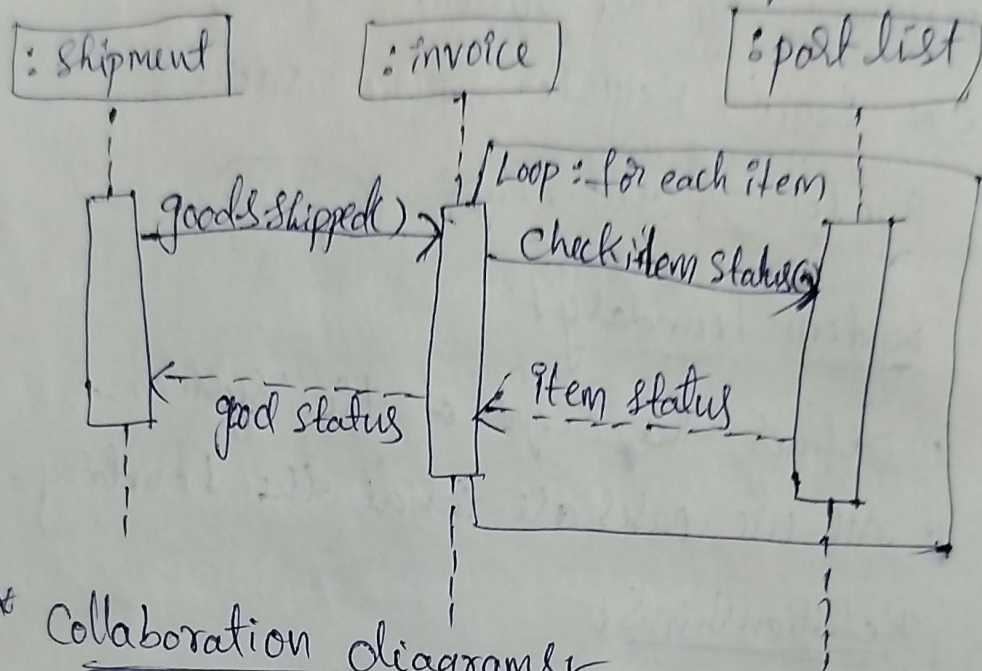
Interaction diagrams

⇒ Interaction shows an arc, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.

→ Interaction diagrams address the dynamic view of a system.

* Sequence diagram

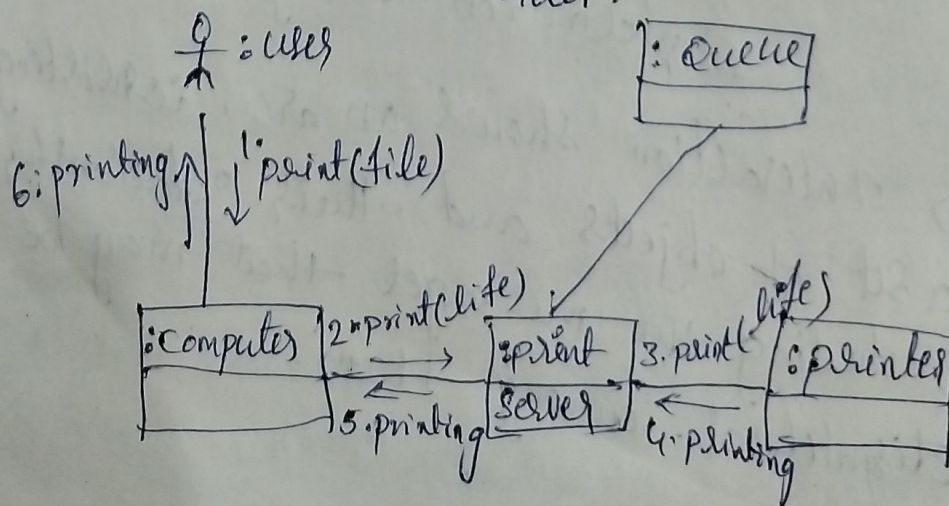
→ A sequence diagram is an interaction diagram that emphasizes the time-ordering of the messages.



* Collaboration diagram

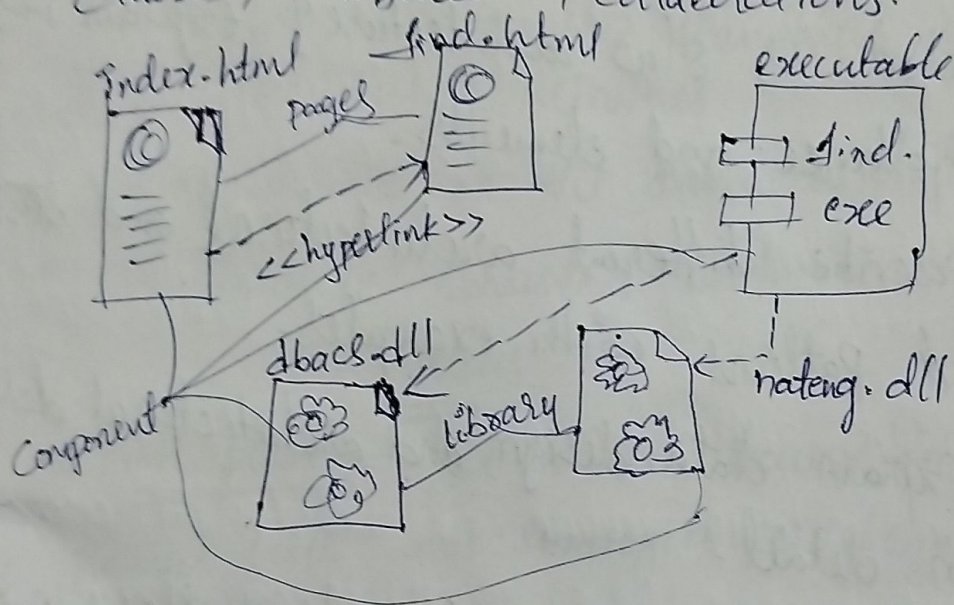
→ Collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects send and receive messages.

Note:- Sequence and Collaboration diagrams are isomorphic means, that you can take one and transfer it into the other.



⑤ Component diagram

- > A component ^{diagram} shows the organization and dependence among a set of components.
- > Component diagram address the static implementation view of a system.
- > They are related to class diagrams in that a component typically maps to one or more classes, interfaces & collaborations.



* Important questions /

1. Explain the SW design process in detail.
2. What is design quality? Explain its key attributes.
3. Describe the fundamental design concepts used in SW Engg?
4. Explain the design model and its components in SW Engg.
5. What is SW architecture? Explain its importance and elements.
6. Describe different architectural styles and patterns with examples.
7. Explain data design and architectural design in detail.
8. What is OMT? Explain its importance in SW design.
9. Explain class diagrams, ~~comp~~ and their components with an example.
10. Describe sequence diagrams, use case diagrams, ~~and~~ collaboration diagrams and Component diagrams with examples.

UNIT-4 Testing strategies

* testing strategy - It defines the s/w testing methodology used to manage the risk of a product or project.

* So, it acts as a guideline of how testing needs to be done by defining the different testing levels, test environments, standards, and other high-level testing activities.

* It is defined at the organizational level.

* It should be consistent with the test policy.

* test plan :- versus It is more focused on how the planned tests and strategies will be executed and implemented.

* So, it functions as a document that defines the scope & objectives of the tests, the entry & exit criteria, the input & output of each test, the testing schedule, and so on.

* It is defined at the project level.

* It should be consistent with both the test policy & the test strategy.

* A Strategic approach to Software Testing :-

* Software testing :- It is the process of checking SW to ensure it works as expected, meets user needs, and is free of bugs.

* It is an important step in delivering high-quality, reliable, and secure SW.

* Why SW testing is important?

- Identifies and fixes problems early.
- Ensures smooth functioning in real-world conditions.
- protects sensitive data and enhances user trust.

* Example Testing an online shopping app to ~~ensure~~

Testing a login feature to verify it accepts correct credentials and shows an error for wrong ones.

* Purpose of Testing + (Verification + validation) process

- Detect errors + find bugs or issues in the SW
- Verify requirements + Ensure the SW matches design and specifications.
- Validate functionality :- Confirm that it works as expected for users.

Example :- Testing an online shopping app to ensure users can search for items, add them to the cart, and complete the payment process.

* Understanding their roles +
Testing :- Identifies problems by checking how
 slw behaves.

Debugging :- Fixes the problems found during testing

Aspect	Testing	Debugging
Focus	Finds problems	Fixes the root cause of problems
Example	Testing reveals login failure	Debugging fixes the code causing the failure.

* Strategic Approach to Testing

* What is a testing Strategy?

A testing strategy is a systematic plan to evaluate slw at every level, from small components to the entire system.

* Key goals

- fix problems early to save time and costs.
- Test software in all possible scenarios.
- Confirm slw fulfills both functional and user requirements.

* Types of Testing Strategies (for conventional slw)

- ① unit testing
- ② Integration testing
- ③ system testing
- ④ validation testing

* A strategy for testing Conventional SW -

① Unit testing - Testing individual components & pieces of the SW.

- Example - Testing if a "submit" button correctly performs its action.

② Integration testing - Ensures different parts of the SW work together. 3 types $\leftarrow$ Big-bang appn.
Top-to-bottom
Bottom-to-top

- Ex - Checking if the login connects to the user dashboard.

③ System testing - Evaluates the entire SW system end-to-end.

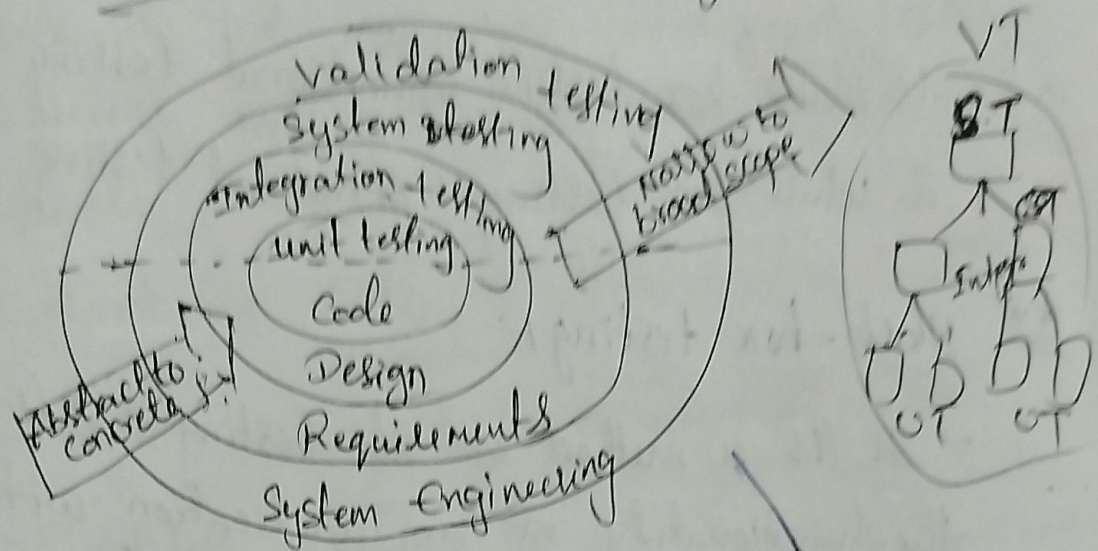
Ex - Testing an e-commerce app to confirm ~~each~~ search, add-to-cart, and payment work seamlessly.

④ Validation testing

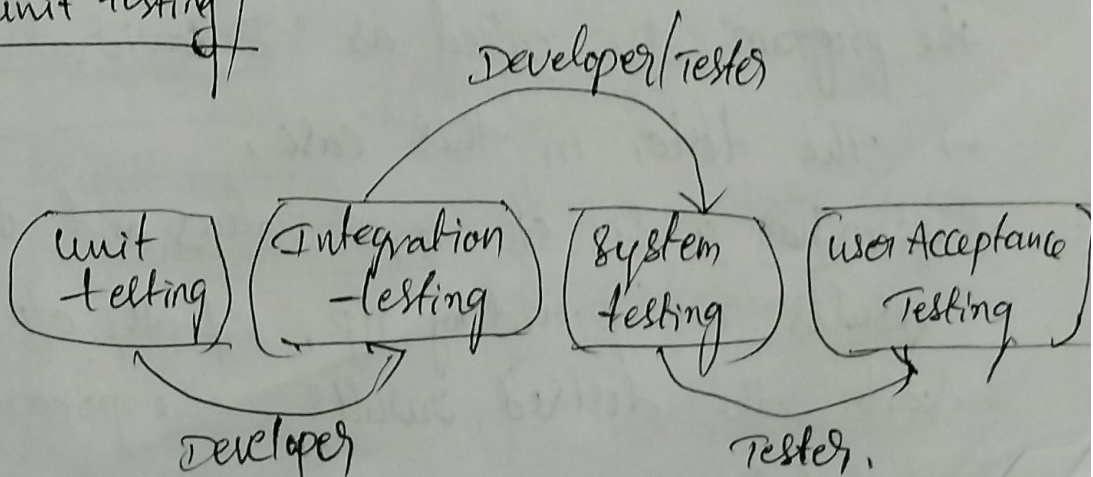
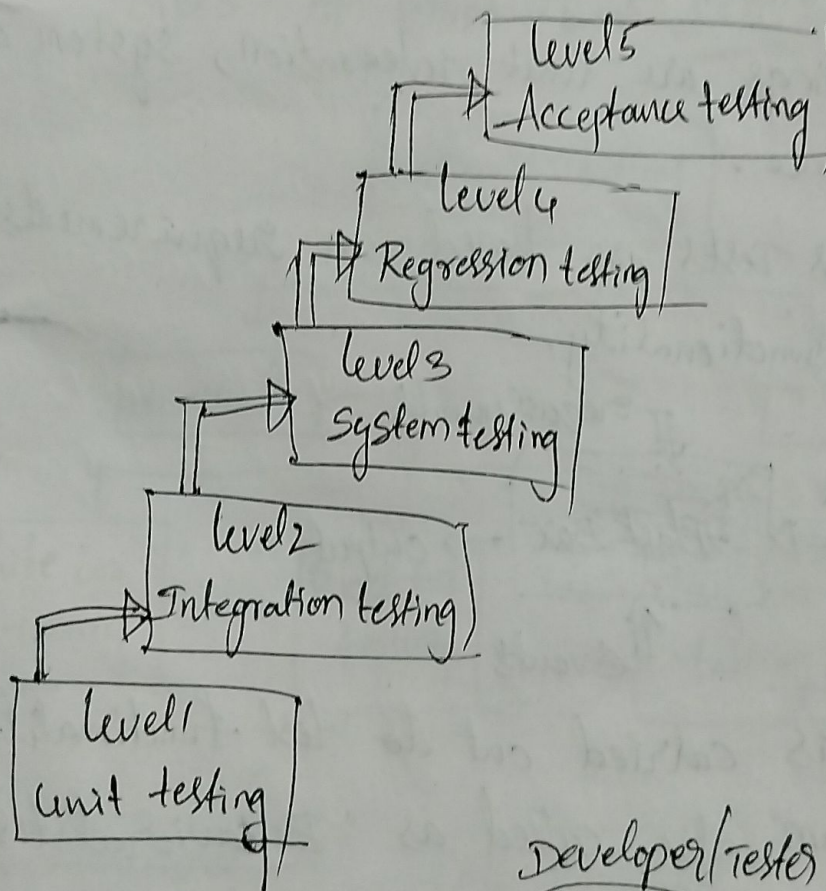
Ensures the SW meets user needs and expectations.

Ex - Allowing users to test a beta version of the app.

Conventional SW testing



* Levels of Testing



* Testing techniques are divided into two types:

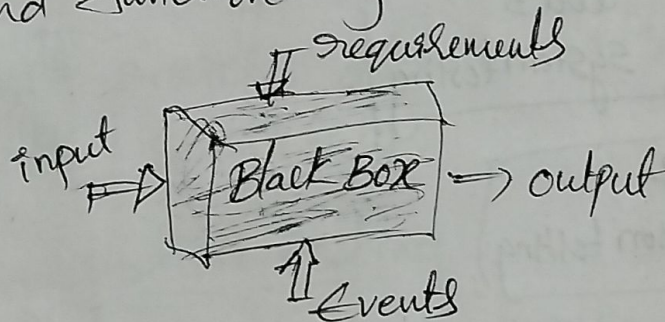
1. Black-box testing / functional testing.
2. white-box testing / structural testing.

1* Black-box testing

⇒ It is a method of software testing that examines the functionality of an application without looking into its internal structures or workings.

⇒ This method is applied at every level of software testing are unit, integration, system and acceptance.

⇒ ~~Tests~~ Tests are based on requirements and functionality.



⇒ It is carried out to test functionality of the program also called as "Behavioral testing".

⇒ The tester in this case,

given ~~has~~ a set of input values and desired results on providing i/p, if the o/p matches with the desired results, the program is tested "OK".

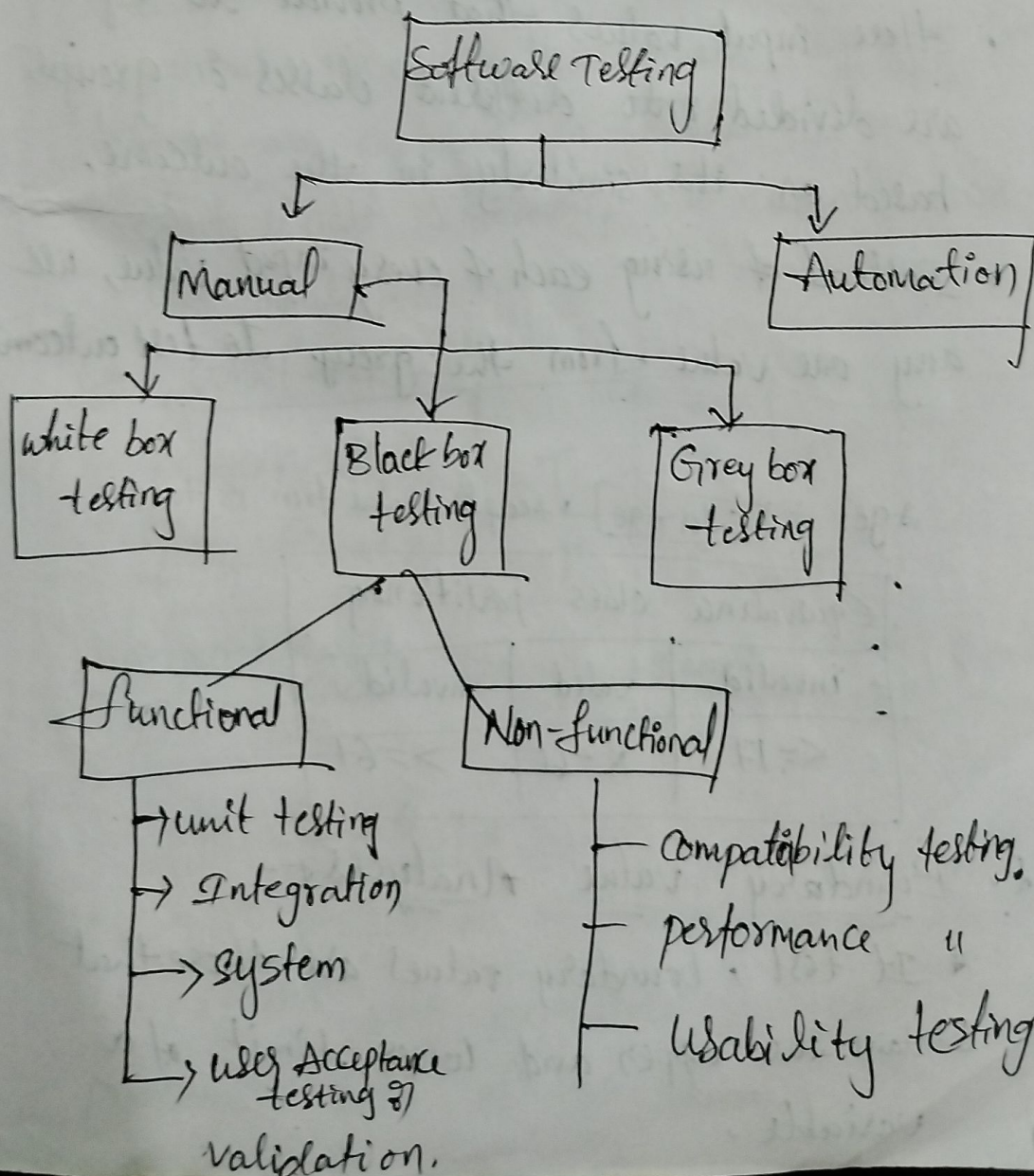
→ In this method, the design and structure of code are not known to tester and testing engineers and end users conduct this test on software.

→ Black box testing tools:-

QTP, selenium, loadrunner & Jmeter.

HP → Quick test
 now → professional
 (UFT - unified functional testing)
 ↓ open source automation testing tool
 ↓ web browser testing only
 ↓ support Java, python, C#, etc.
 ↓ Java meter
 ↓ performance load, stress testing of applications

→ Types of Black box testing:-



* Black box testing techniques

1. Equivalence partitioning
2. Boundary value Analysis.
3. Decision table testing
4. Error Guessing
5. State transition testing
6. All pairs testing techniques.

① Equivalence partitioning

- Here input values that provide to system are divided into different classes or groups based on its similarity in the outcome.
- Instead of using each & every input value, use any one value from the group to test outcome.

Age [Enter Age] → Accepts value from 18 to 60.

Equivalence class partitioning		
invalid	valid	invalid
≤ 17	$18-60$	≥ 61

② Boundary value Analysis

- * In test, boundary values are those that contain the upper and lower limit of a variable.

- It tests, while entering boundary value whether the S/W is producing correct output or not.

Age : Enter age * accepts value 18 to 56

Boundary value Analysis		
invalid (min-1)	valid (min, min, max, max)	Invalid (max+1)
17.	18, 19, 55, 56	57

3. Decision Table testing

- Various i/p combination & their respective system behavior are captured in tabular form.
- Check logical Relationship b/w two and more than two inputs.

Ex Gmail Account.

Email (Cond ¹)	T	T	F	F
password (Cond ²)	T	F	T	F
Expected Result (Action)	Account page	Incorrect password	incorrect email	Incorrect email

4. Error Guessing

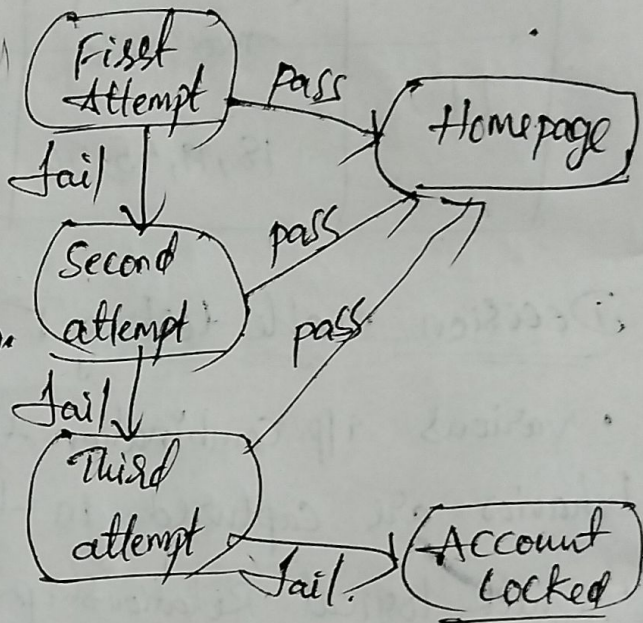
- It is based on the experience of the tester, where tester uses experience to guess the problematic areas of the S/W.

• example

Divide by zero, Handling all null values in text fields, Accepting the submit button without any value, file upload without attachment, file upload with less than or more than the limit size.

5. State transition testing

- It is used to capture behavior of the S/W Applⁿ when diff. exp values are given to the same function.
- Applies to those types of Appl^s that provide specific no. of attempts to access application.



6. All pairs testing technique

- It is used to test all the possible discrete combinations of values.
- This Combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box etc..

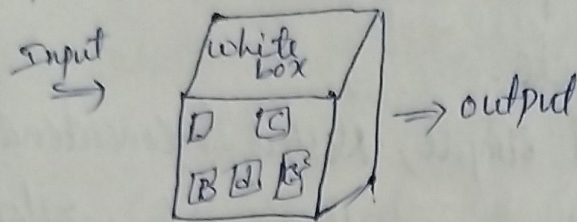
* Advantages of Black box testing

1. The tester does not need to have more functional knowledge or programming skills.
2. It is efficient for implementing the tests in the larger system.
3. Tests are executed from the user's or client's point of view.
4. It is used in finding the ambiguity and contradictions in the functional specifications.

* Disadvantages

1. without clear programming knowledge, test cases are difficult to implement.
2. It does not reveal the errors in the control ~~system~~ structure.
3. working with a large sample of inputs can be exhaustive and consumes a lot of time.

- * Each statement of your program.
- * Functions & modules on an individual basis.
- * Expected output.



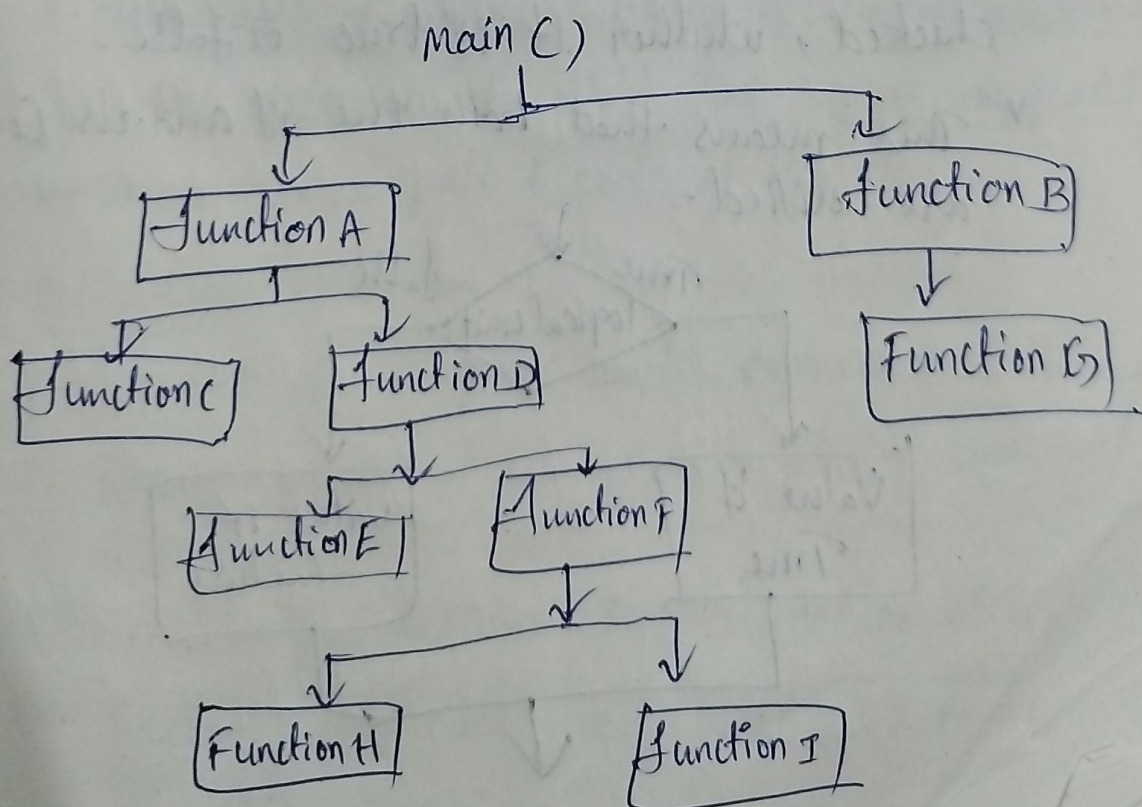
* white box testing techniques & types

- ① path coverage / testing
- ② Loop testing
- ③ Branch coverage / condition testing
- ④ statement coverage - testing.

① path coverage testing &

* Testing is dependent on the program's control or flow structure.

* All possible paths are defined with entry to exit points in the system.



② Loop testing :-

- * There are loops such as while, for and Do-while loops etc.
- * It checks all simple, nested & concatenated loops
- * It checks ending condition, if working correct and if the size of the conditions is enough.

while (Condition 1)

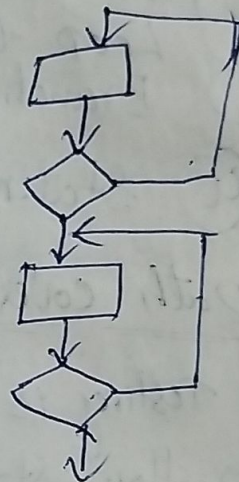
{
 Statement (S);

}

while (Condition 2)

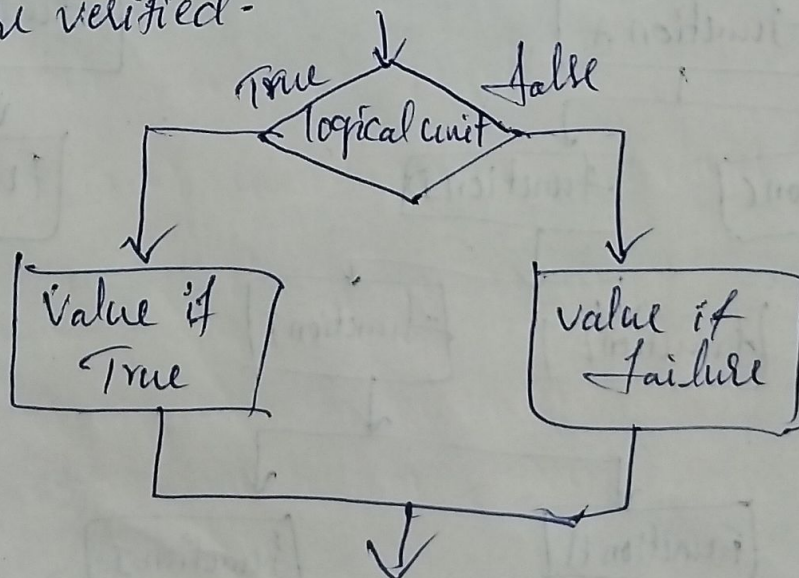
{
 Statement (S);

}



③ Branch coverage / Condition testing

- * The logical conditions for every value are checked, whether it is true or false.
- * This means that both the if and else conditions are verified.



4. Statement Coverage

- * Statement coverage is a testing in which tester traverse all statement in code.
- * It ensure that each statement of the code is executed at least once.
- * Hence each line of the code is verified.

```
1. printf (int a, int b) {  
2.   int result = a+b;  
3.   if (result > 0)  
4.     print ("positive", result)  
5.   else  
6.     printf ("Negative", result)  
7. }
```

* Advantages

1. It optimizes code so hidden errors can be identified.
2. Test cases of white box testing can be easily automated.
3. Early detection of errors that improve code quality.
4. It cover most path & code of the sw.

* Disadvantages

1. W.B.T can be quite complex & expensive & time-consuming.
2. Requires professional resources with detailed understanding of programming.
3. Missing functionalities cannot be detected.
4. Redesign of code needs test cases to be written again.

* * Validation testing *

- Validation defines as "that validation succeeds when the functions in a manner that can be reasonably ~~are~~ accepted by the customer.

1. Validation - Test Criteria :-

- After each validation test case has been conducted, one of two possible conditions exists:

(1) The function & performance characteristic confirms to specification and is accepted, or

(2) A deviation from specification is uncovered and a deficiency list is created.

2. Configuration Review :-

- An important element of the validation process is a Configuration Review.

- The intent of the review is to ensure that all elements of the software configuration have been properly developed or not.

- The configuration review, sometimes called an audit

③ Alpha and Beta testing

- The Alpha test is conducted at the developer's site by a representative group of end users -

- The beta test is conducted at one or more end-user sites. Unlike Alpha testing, the developer generally is not present.

~~• A variation of stress testing is a technique called sensitivity testing.~~

- A variation on beta testing called customer acceptance testing.

System Testing

- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.

Types of system testing

1. Recovery Testing is a test that forces the S/W to fail in a variety of ways and verifies that recovery is properly performed.
 2. Security testing it attempts to verify that protection mechanisms, built into a system will, in fact, protect it from improper penetration.
 3. Stress Testing It executes a system in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called "sensitivity testing."

4. Performance testing - it is designed to test the run-time performance of SW within the context of an integrated system.

5. Deployment testing, sometimes called Configuration testing.

* The art of Debugging

Debugging occurs as a consequence of successful testing.

① The Debugging process

* The debugging process will usually have one of "two" outcomes:

(1) The cause will be found and corrected (2)

(2) The cause will not be found.

* A few characteristics of bugs provide some clues

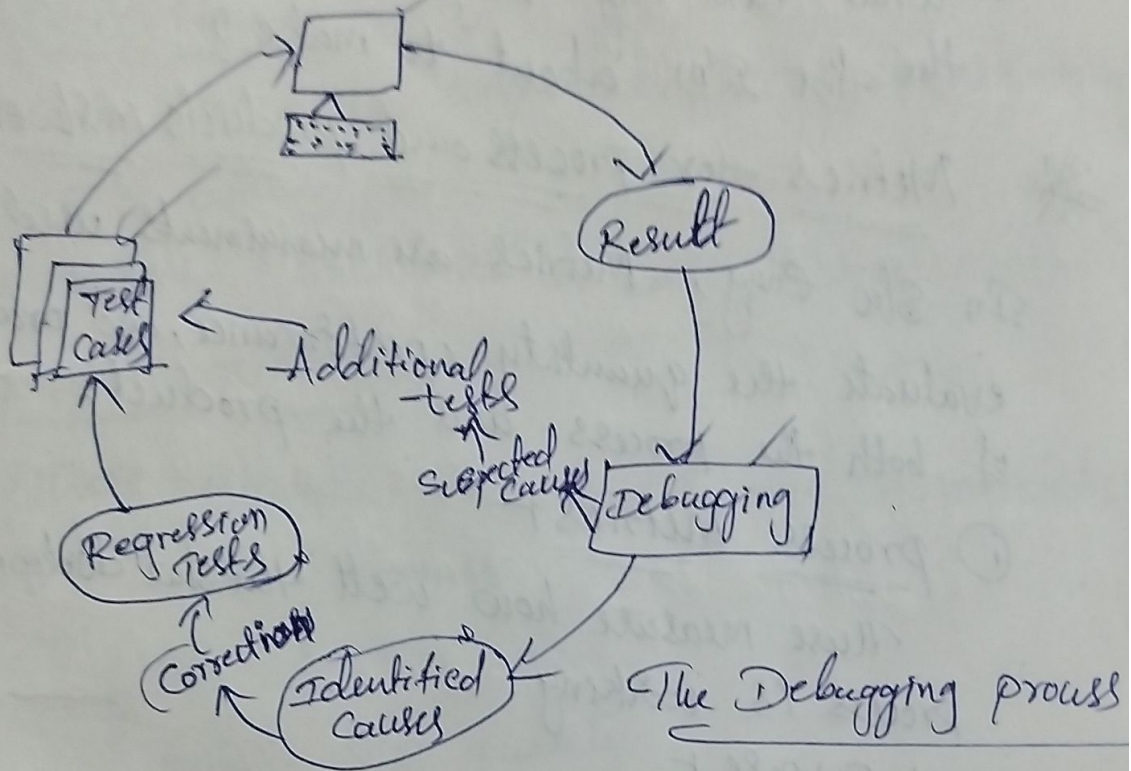
① The symptom and the cause may be geographically remote.

② The symptom disappears (temporarily), when another error is corrected.

③ The symptom may actually be caused by non-errors.

(4) The symptom may be caused by human error that is not easily traced.

(5) The symptom may be a result of timing problems rather than.



(2) psychological Consideration

* Unfortunately, there appears to be some evidence that debugging process is an innate human trait. Some people are good at it and others & not.

(3) Debugging Strategies (Debugging tactics)

• Broadly, describes 3 debugging strategies:

(1) Brute force.

(2) Backtracking, and

(3) Cause elimination.

④ Correcting the error

- Van Vleck suggest "3" simple questions.
- 1. Is the cause of the bug reproduced in another part of the program?
- 2. What "next bug" might be introduced by the fix I'm about to make?

* Metrics for process and products in SW Engg

In SW Engg, Metrics are measurements used to evaluate the quantity, performance, and progress of both the process and the product.

① process metrics

These measure how well the SW development process is working.

=> Purpose

- Improve development efficiency.
- Reduce defects.
- Monitor project progress.
- Enhance productivity.

=> Common process metrics

① Effort metrics:

- Measurable time spent on tasks.

Example person-hours → person-days.

② Productivity metrics

* output per unit effort.

Ex: Lines of Code (LOC) per developer.

③ Defect Removal Efficiency (DRE) :-

* measures how effectively defects are removed during development.

Formula

$$DRE = \frac{(\text{Defects removed before release} / \text{Total defects}) \times 100}{100}$$

④ Cycle Time

* Time taken to complete a process (e.g, coding-testing)

⑤ Review Efficiency

* Measures effectiveness of code / design reviews

* Product Metrics

These measure the quality and characteristics of the software product.

→ Purpose :

* Ensure software quality

* Evaluate performance & reliability.

* Improve Maintainability.

→ Common product metrics

① Size metrics : Measure software size.

Ex • Lines of code (LOC)

• Function points (FP)

② Complexity metrics .

* Measure how complex the code is ,

Ex cyclomatic complexity.

③ Quality metrics

- * Defect density (defects per LOC)
- * Reliability (failure rate).

④ Performance metrics

- Response time
- Throughput
- Memory usage.

⑤ Maintainability metrics

- * ease of fixing bugs or modifying code.

Finally +

- process metrics help improve how sw is built.
- product metrics help evaluate what is built.

* Software measurement +

It is the process of quantifying attributes of sw such as size, complexity, quality and performance.

- * It helps in :-
 - Estimation of cost and effort
 - Improving sw quality
 - Tracking project progress.

* Types of sw measurements

① Product metrics :- measure the sw product itself.

- Size (Lines of code - LOC)
- Complexity
- performance
- quality.

② process metrics :-

• measure the development process

- Development time
- Cost
- Defect removal efficiency

③ project metrics :-

• measure the project management aspects.

- Effort
- Schedule
- Resource utilization.

* Key software metrics :-

① Size metrics :- LOC, Function points (FP).

• Complexity metrics :- cyclomatic & control flow complexity

• Quality metrics :- Defect Density, Reliability, Maintainability.

• Productivity metrics :-
- output per developer,
- effort per module.

* Measurement process

① Define goals

② Identify metrics

③ Collect data

④ Analyze results

⑤ Improve process.

} Explain in detail.

* Advantages :-

- Better estimation
- Improved quality
- Risk reduction
- Performance tracking -

* metrics for S/W quality

S/W quality metrics are quantitative measures used to evaluate the quality of S/W products and processes.

- They help ensure the S/W is reliable, efficient, maintainance and user friendly.

* Types of S/W quality metrics

① product quality metrics +

measure the quality of the final S/W product

② process quality metrics :-

measure the development process quality.

③ project quality metrics +

measure project-level quality.

* Important S/W quality metrics

① Defect Density + measures no. of defects in S/W.

$$DD = \frac{\text{No. of defects}}{\text{Size of S/W (LOC or FP)}}$$

Lower value = Better quality.

② Defect Removal Efficiency (DRE) +

measures how effectively defects are removed.

$$DRE = \frac{\text{Defects Removed before Release}}{\text{Total defects}} \times 100$$

Higher value = Better quality.

③ Mean Time To Failure (MTTF) :-

- Average time b/w failures.
- Higher MTTF = more reliable S/w.

④ Mean Time To Repair (MTTR) :-

- Time taken to fix a failure
- Lower MTTR = Better maintainability.

⑤ Availability :- measures system Availability

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

⑥ Customer Satisfaction :-

- user feedback
- surveys
- Ratings.

⑦ Code coverage :-

- percentage of code tested.
- Higher coverage = Better testing quality.

⑧ Cyclomatic Complexity :-

measures code complexity.

- lower complexity = Easier maintenance.

⑨ Response time :-

time taken to respond to user requests.

- lower Response time = Better performance.

* Characteristics of good quality metrics

- Measurable.
- Easy to understand.
- Objective
- Actionable.

* Advantages

- Improves slw reliability
- Help detect defects early.
- Supports better decision-making.
- Enhances customer satisfaction.

* Important questions -

- ① Explain the strategic approach to slw testing
- ② Differentiate slw Black box and white-box testing with examples.
- ③ What is validation and system testing? Explain their objectives and differences
- ④ Explain the art of ~~the~~ debugging in slw Engg.
- ⑤ What is slw measurement? Explain various metrics for slw quality.

UNIT- 5

NOTES

- **Risk management:** Reactive vs. Proactive Risk strategies, software risks, Risk identification, Risk projection, Risk refinement, RMMM, RMMM Plan.
- **Quality Management:** Quality concepts, Software quality assurance, Software Reviews, Formal technical reviews, Statistical Software quality Assurance, Software reliability, The ISO 9000 quality standards.

RISK MANAGEMENT:

- **Risk:** is an undesired event or circumstance that occur while a project is underway
- It is necessary for the project manager to anticipate and identify different risks that a project may be susceptible to
- To reduce the impact of all kinds of risk that may affect a project by identifying, analyzing and managing those risks.

REACTIVE VS PROACTIVE RISK STRATEGIES:

Reactive Risk strategy:

- If you don't actively attack the risks, they will actively attack you.
- If some is facing great difficulty and still says that "don't worry, I will think of something".
- Means never worrying about problems until they occurred.
- Most software teams rely on reactive risk strategies.
- Reactive strategy monitors the projects future risk and resources are set aside.
- Software team does nothing about risks until something goes wrong.
- Then team does some action and tries to correct the problem rapidly.
- It is called fire fighting mode.
- When it fails, "crisis management" takes over and project will be in trouble.

Proactive Risk strategy:

- It is more intelligent strategy than reactive strategy for risk management.
- Proactive strategy begins long before technical work is initiated.
- In this, Risk is identified, their probability and impact is accessed and they are ranked by importance.

- Then the software team makes a plan for managing risk.
- Main objective is to avoid risk, but because all risks cannot be avoided, so team always works to make plan.

SOFTWARE RISKS:

Software risk always involves two characteristics:

1. **Uncertainty:** Risk may or may not happen, i.e. there are no 100% probable risks.
2. **Loss:** If risk is reality, unwanted loss or consequences will occur.

Different categories of software risks are considered:

1. Project Risks:

- Threaten the project plan and affect schedule and resultant cost.
- It means, if project risks become real, it is possible that project schedule will slip and that cost will increase.
- Project Risks identify budget problems, schedule problems, resource problems, and requirements problems and their impact on project.

2. Technical Risks:

- Threaten the quality and timeliness of the software to be produced.
- If a technical risk becomes reality, implementation may be difficult or impossible.
- Technical risks identify possible design problems, implementation problems, interface problems, verification problems, and maintenance problems.

3. Business Risks:

- Threaten the viability of the software to be built.
- **Top five business risks are :**
 1. Building a good product that no one wants
 2. Building a product that no longer fits into business strategy
 3. Building a product about which salesman does not know how to sell
 4. Losing the support of senior management
 5. Losing budgetary commitment.

4. Known Risks:

- These risks can be uncovered from careful evaluation of project plan.

5. Unpredictable Risks:

- Risks that occur and may be very difficult to identify.

6. Predictable Risks:

- Risks are identified by past project experience (e.g. staff turnover, poor communication with customer).

RISK IDENTIFICATION:

- It deals with identification of risk and attempt to specify threats to the project plan (estimates, schedule, resource loading etc.)
- After identifying known and predictable risks, project manager takes first step to avoid them when possible, and control them when necessary.
- **Generic risks** are threats to every software project.
- **Product-specific risks** are identified only by those who are specific to software with clear understanding of technology, people, and environment.

After product specific risk identification, answer of following question is given:

- "What special features of the product can threaten the project plan".

Steps for Risk Identification:

Step1: Identify all possible risks.

Step2: Create risk item check list.

- **Checklist is used for risk identification and focuses on some known and predictable risks in following subcategories :**
- Product size :
- Business impact :
- Customer characteristics :
- Process definition :
- Development environment :
- Technology to be built :
- Staff size and experience :

Step3: Assessing overall project risk:

Step4: Categorize into risk components: -

- **Performance risk:** degree of uncertainty that the product will meet its requirements and be fit for its intended use
- **Cost risk :** degree of uncertainty that project budget will be maintained
- **Support risk:** degree of uncertainty that resultant software will be easy to correct, adapt, and enhance

- **Schedule risk:** degree of uncertainty that project schedule will be maintained and that product will be delivered on time.

Step5: Divide the risk into one of 4 categories: The impact of each risk driver on the risk component is divided into one of four impact categories

- Negligible-0
- Marginal-1
- Critical-2
- Catastrophic-3

RISK PROJECTION:

- Also called risk estimation.

Risk projection gives rate to each risk in two ways:

- 1. The likelihood or probability that risk is real(L_i)
- 2. The Consequences (X_i) of that risk.
- It estimates the impact of risk on the project and the product.
- Estimation is done by using Risk Table

Steps in Risk Projection:

1. Estimate L_i for each risk
2. Estimate the consequence X_i
3. Estimate the impact
4. Draw the risk table
5. Ignore the risk where the management concern is low i.e., risk having impact high or low with low probability of occurrence
6. Consider all risks where management concern is high i.e., high impact with high or moderate probability of occurrence or low impact with high probability of occurrence

Developing a risk table:

- A risk table provides a project manager with simple technique for risk projection.
- Project team begins by listing all risks in first column.
- Each risk is categorized in the second column (e.g. PS means project size risk, BU means business risk).
- The probability of occurrence of each risk is entered in next column of table.
- In next column impact of each risk is assessed.

- The category for each four risk components- performance, cost, support, schedule – are averaged to determine overall impact value.
- A risk table is then sorted by probability and impact to rank risks.

Assessing the risk impact:

Three factors affect the consequences that are likely if a risk occurs:

- Its nature, scope, and timing.
- The nature of risk indicates problems that are likely if it occurs.
- The scope of a risk combines the severity (how serious is it?, how much project will be affected or how many customers are harmed).
- Timing of risk considers when and for how long the impact will be felt.

Figure: Risk Projection Table

Risk	Category	Probability~	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger no. of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End user resist system	BU	40%	3	

The impact of each risk is assessed by Impact values:

- Catastrophic-1
- Critical-2
- Marginal-3
- Negligible-4

Following steps are recommended to determine overall consequence of a risk:

- 1. Determine the average probability of occurrence value for each risk component
- 2. Determine the impact for each component.
- Complete the risk table and analyze the results.
- The risk exposure ,RE is determined using following relationship :
- $RE = P \times C$, where p is probability of occurrence of a risk, and C is cost to the project.

RISK REFINEMENT:

- Also called Risk assessment.
- As time passes and more about the project and risk is learned, then it is possible to refine the risk into more detailed risks.
- One way to do this is to represent the risk in *condition-transition-consequences (CTC) FORMAT*.
- So, risk is stated in following form:
- Given that <condition> then there is concern that (possibly) <consequences>.

The general condition can be refined in following manner:

- **Sub condition 1:** Certain reusable components were developed by third party with no knowledge of internal design standards.
- **Sub condition 2:** Design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
- **Sub condition 3:** Certain reusable components have been implemented in a language that is not supported on target environment.

Risk Mitigation Monitoring and Management (RMMM):

- Its goal is to assist project team in developing a strategy for dealing with risk.

An effective RMMM strategy must consider three issues:

- 1) Risk Avoidance
- 2) Risk Monitoring
- 3) Risk Management

Risk Mitigation:

- If software team adopts a proactive approach to risk, avoidance is best strategy.
- We do Proactive planning for risk avoidance.
- It is done by developing a plan for risk mitigation.

To mitigate the risk, possible steps for reducing turnover are taken:

- Meet the current staff to determine the cause (poor working conditions, low pay)
- Mitigate those causes that are under control before project starts.
- Develop techniques to make sure the continuity when people leave.

- Organize the teams so that information is distributed
- Define standards and mechanisms so that documents are developed timely
- Conduct peer reviews of all work and assign a backup staff member for critical

Risk Monitoring:

- Assessing whether predicted risk occur or not.
- Ensuring risk aversion steps are being properly applied.
- Collection of information for future risk analysis.
- Determine which risks caused which problems.
- Also project manager monitor the effectiveness of risk mitigation steps.

For example, in case of high staff turnover, following factors can be monitored:

- General attitude of team members on project pressures
- Inter personal relationships among team members and problems.
- The availability of jobs with company and outside.

Risk Management:

- Risk management and Contingency planning assumes that mitigation efforts have failed and risk has become a reality.
- Actions to be taken in the event that mitigation steps have failed and the risk has become a live problem
- Devise RMMP(Risk Mitigation Monitoring And Management Plan)
- RMMM steps provide additional project cost.
- For example, giving the time to backup persons needs cost.

RMMM Plan:

- RMMM Plan documents all work performed as a part of risk analysis.
- Each risk is documented individually by using a Risk Information Sheet (RIS).
- **Risk Information Sheet** is maintained by using a database system.
- Once RMMM Plan has been documented and project is started, risk mitigation, monitoring steps starts.
- Risk mitigation is risk avoidance activity.

Risk monitoring is project tracking activity with three objectives:

- To assess whether predicted risks occur or not
- To make sure that risk aversion steps defined for risk are properly applied
- To collect information that is used for future risk analysis.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/04	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: RE computed to be \$20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/04			
Current status: 5/12/04: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

QUALITY CONCEPTS:

- People will forget how fast you did a job, but they remember how well you did it.
- It is not enough to say that software quality is important.
- But we also have to define what software quality is and how to get it.
- **Variation Control is heart of quality control.**
- Controlling the variation is the key of high quality product.
- From one software project to another, we want to minimize the difference between predicted resources and actual resources, including staff, equipment, time, cost, number of defects in the upcoming release of software project. etc.

Following are the four Quality Concepts:

1. Quality:

- The American Heritage dictionary defines quality as "a characteristics or attribute of something".

- As an attribute, quality refers to measurable properties.
- These properties include complexity, number of function points, SLOC, etc.

Two kinds of quality when we measure based on characteristics:

- 1. **Quality of design:** Refers to characteristics that designers specify for an item.
- 2. **Quality of conformance:** Refers to the degree to which design specifications are followed during construction of product.
- Quality of design deals with requirements, specification, and design.
- Quality of conformance deals with implementation.
- If implementation is according to design and the result is as per requirement and goals, then quality of conformance is high.

Robert Glass says that:

- **User satisfaction** = Compliant product + good quality + delivery within budget and schedule.
- So, he says that quality is important, but if user is not satisfied, nothing else will really matter.

Demarco defines Quality as:

"A product quality is a function of how much it changes the world for better".

2. Quality Control:

- **Series of** inspections, reviews, feedback and tests are used throughout the process to make sure that product will get the required result.

3. Quality Assurance:

- It consists of auditing and reporting functions that evaluate effectiveness and completeness of quality control activities.
- Goal of QA is to provide necessary data to management to inform about quality.
- If that data is identifying some problems, then management will tell the problems and apply resources to solve quality issues.

4. Cost of Quality:

- It includes all costs occurred in achieving the quality or in performing quality related activities.
- Study of cost of quality provides a base for current cost of quality; identify new methods for reducing the cost.

Cost of quality is divided into:

- **Prevention Cost:** Includes quality planning, formal technical reviews, testing equipment, and training.
- **Appraisal Cost:** Includes activities to get product condition by going through it.
- In-process and inter-process inspection, equipment calibration and maintenance, testing
- **Failure Cost:** It disappears if no defects appeared before giving product to users.
- It can be internal failure like rework, repair, failure mode analysis cost
- External failure cost like Complaint resolution, product return and replacement, help line support, warranty work

SOFTWARE QUALITY ASSURANCE (SQA):

- How do we define quality is more important.
- "Every program or product does something right for someone".

For our purpose, software quality is defined as:

- "Conformance to explicitly specified functional and performance requirements and implicit characteristics that are expected from all software".

Definition of Software Quality serves to emphasize:

1. Conformance to software requirements is the foundation from which software quality is measured.
2. Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.
3. Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

Software Quality Assurance:

- It is a "planned and systematic pattern or actions that are required to make sure high quality in software".
- For software quality assurance – software engineers, project managers, customers, individual who are in SQA group.
- The people who perform SQA must look the software from customer's view.
- SQA group will ensure that software quality is maintained.
- Software quality assurance (SQA) is the concern of every software engineer to reduce cost and improve product time-to-market.
- A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are included in an SQA plan.
- SQA activities are performed on every software project.

- Use of metrics is an important part of developing a strategy to improve the quality of both software processes and work products.

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 with the defined software process.
- Audit designated software work products to verify compliance with those defined as part of the software process.
- Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
- Record any evidence of noncompliance and reports them to management.

SOFTWARE REVIEWS:

- Software reviews are the "filter" for the software process.
- Reviews are applied at various points and try to uncover errors and defects that can be removed.
- Software reviews "purify" SE activities like analysis, design, and coding.
- Informal meetings are conducted for reviews.
- Formal presentation to customers, management, and technical staff is also a review.
- Software engineers (and others) conduct formal technical reviews (FTRs) for software quality assurance.
- Purpose of FTR is to find errors before they are passed on to another software engineering activity or released to the customer.
- Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

Cost impact of Software Defects:

- Primary objective of formal technical reviews is to find errors during the process so they do not become defects after release of software.
- Industry studies suggest that design activities introduce 50-65% of all defects or errors during the software process.

- Formal Review techniques have been shown to be up to 75% effective in uncovering design flaws
- So it ultimately reduces the cost of subsequent activities in the software process.
- Assume that an error uncovered during design will cost 1 unit of cost.
- The same error uncovered just before testing will cost 6.5 units, during testing will cost 15 units, and after release it cost between 60 to 100 units.

Formal Technical Review (FTR):

- A FTR is a software quality control activity performed by software engineers and others.

The objectives of FTR are:

1. To uncover errors in function, logic or implementation for any representation of the software.
 2. To verify that the software under review meets its requirements.
 3. To ensure that the software has been represented according to predefined standards.
 4. To achieve software that is developed in a uniform manner.
 5. To make projects more manageable.
- Each FTR is conducted as meeting and will be successful only if it is properly planned, controlled, and attended.

Review meeting in FTR:

The Review meeting in a FTR should abide to the following constraints:

1. Review meeting members should be between three and five.
 2. Every person should prepare for the meeting and should not require more than two hours of work for each person.
 3. The duration of the review meeting should be less than two hours.
- The focus of FTR is on a work product that is requirement specification, a detailed component design, a source code listing for a component.
 - The individual who has developed the work product i.e., **the producer** informs the project leader that the work product is complete and that a review is required.
 - The project leader contacts a **review leader**, who evaluates the product for readiness, generates copy of product material and distributes them to two or three review members for advance preparation
 - Each reviewer is expected to spend between one and two hours reviewing the product, making notes.
 - Review leader also reviews product and establish an agenda for review meeting
 - The review meeting is attended by review leader, all reviewers and the producer.

- One of reviewer act as **recorder**, who notes important points discussed in meeting.
- The meeting (FTR) is started by introducing the agenda of meeting and then the producer introduces his product.
- Then the producer "walkthrough" the product, the reviewers raise issues which they have prepared in advance.
- If errors are found the recorder notes down each.

At the end of the review, all participants of FTR must decide whether to:

- 1. Accept the product without further modification
- 2. Or reject the product due to severe errors
- 3. Or accept the product provisionally.

Review reporting and Record keeping:

- During the FTR, a reviewer(recorder) records all issues that have been raised

A review summary report answers three questions:

- 1. What was reviewed?
- 2. Who reviewed it?
- 3. What were the findings and conclusion?
- Review summary report is a single page form with possible attachments

The review issues list serves two purposes:

- 1. To identify problem areas in the product
- 2. To serve as an action item checklist that guides producer as corrections are made

Review Guidelines:

- Guidelines for conducting FTR must be established in advance, distributed to all reviewers, and then followed.

Following are the guidelines for formal technical reviews:

- Review the product, not the producer
- Set an agenda and maintain it.
- Limit debate and rebuttal.
- Enunciate problem areas, but don't attempt to solve every problem noted.
- Take return notes.
- Limit the number of participants and insist upon advance preparation.
- Develop a checklist for each product i.e. likely to be reviewed.

- Allocate resources and schedule time for FT.
- Conduct meaningful training for all reviewer
- Review your early reviews.

Sample – Driven Reviews:

- Thelin and his colleagues suggest a sample driven review process.
- Samples of all software engineering work products are inspected to determine which work products is most error prone.
- To be effective, sample driven review process must attempt to quantify those work products that are primary targets for full FTRs.

To do this, following steps are mentioned:

1. Inspect a fraction A_i of each software work product, i .
2. Record number of faults F_i found within A_i .
3. Develop gross estimate of number of faults within work product i by multiplying F_i by $1/A_i$.
4. Sort work product in descending order according to the gross estimate of the number of faults in each.
5. Focus available review resources on those work products that have highest estimated number of faults.

The fraction of work product that is sampled must be :

- Representative of the whole work product.
- Must be large enough to be meaningful to the reviewer.

STATISTICAL QUALITY ASSURANCE:

- Information about software defects is collected and categorized for some time period.
- Each defect is traced back to its cause.
- Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the "vital few" defect causes.
- Move to correct the problems that caused the defects in the "vital few".
- Some defects are uncovered as software is being developed.

Hundreds of different defects are uncovered; all can be tracked to one of following causes:

- Incomplete or erroneous specifications (IES)
- Misinterpretation of customer communication (MCC)

- Intentional deviation from specifications.(IDS)
- Violation of programming standards.(VPS)
- Error in data representation(EDR)
- Inconsistent component interface(ICI)
- Error in design logic(EDL)
- Incomplete or erroneous testing.(IET)
- Inaccurate or incomplete documentation(IID)
- Error in programming language translation of design(PLT)
- Ambiguous or inconsistent human /computer interface(HCI)
- Miscellaneous(MIS)

Six Sigma for Software Engineering:

- The most widely used strategy for statistical quality assurance.
- **Six Sigma strategies** is disciplined method that uses data and statistical to measure and improve a company's operational performance by identifying and eliminating "defects" in manufacturing process.
- Six Sigma is derived from six standard deviations – defects per million occurrences which mean an extremely high quality standard.

Three core steps:

- 1. **Define** customer requirements, deliverables, and project goals via well-defined methods of customer communication.
- 2. **Measure** each existing process and its output to determine current quality performance (e.g., compute defect metrics).
- 3. **Analyze** defect metrics and determine vital few causes.

For an existing software process that needs improvement, six sigma suggest two additional steps:

- 1. **Improve** process by eliminating the root causes for defects
- 2. **Control** future work to ensure that future work does not reintroduce causes of defects
- These core and additional steps are called **DMAIC** (define, measure, analyze, improve, and control) method.

If new processes are being developed by an organization (rather than improving an existing process) the additional steps are:

- 1. **Design** each new process to avoid root causes of defects and to meet customer requirements.
- 2. **Verify** that the process model will avoid defects and meet customer requirements.

- This variation is called **DMADV** (define, measure, analyze, design, and verify) method.

SOFTWARE Reliability:

- **Software Reliability is defined as:** the "probability of failure free operation of a computer program in a specified environment for a specified time period".
- Can be measured directly and estimated using historical and developmental data.
- Software reliability problems can usually be traced back to errors in design or implementation.

Measures of Reliability and Availability:

Simple measure of reliability is MTBF.

- ☐ Mean time between failure ($MTBF = MTTF + MTTR$)
- ☐ MTTF = mean time to failure
- ☐ MTTR = mean time to repair
- **Software Availability :** It is defined as the probability that a program is operating according to requirements at a given point in time and is defined as :
- ☐ Availability = $[MTTF / (MTTF + MTTR)] \times 100\%$

Software Safety:

- It is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- Modeling and analysis process is conducted as a part of software safety.
- Initially hazards are identified and categorized by criticality and risk.

ISO 9000 QUALITY STANDARDS:

- **Quality assurance systems are defined as:**
- The organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
- Quality assurance systems are created to help organizations to make sure their products and services satisfy customer expectations.

- ISO 9000 describes the quality elements that must be present for a quality assurance system to be compliant with the standard.
- But it does not describe how an organization should implement these elements.
- ISO 9001:2000 is the quality standard that contains 20 requirements that must be present in an effective software quality assurance system
- **Requirements delineated by ISO 9001:2000 address following topics :**
- Management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, control of quality records, internal quality audits, training, servicing, and statistical techniques.