Unit - I

Introduction

The term software engineering is composed of 2 words, software & engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose.
Software is collection of executable programming code, associated libraries & documentations.
Software, when made for a specific req. is called Software product.

Engineering, is all about developing products, using well defined, scientific principles & methods.

"Software engineering can be defined as engineering branch associated with the development of software product using well defined scientific principles, methods & procedures."

IEEE defines the application of a systematic, disciplined quantifiable, approach to the development, operation & maintanance software.

A small program can be written without help of software engineering principles but larger program would be difficult. Software engineering uses some principles, methodologies & procedures by which if the larger program becomes quite effective.
Software engineering follows two principles Abstraction & Decomposition. The main purpose of Abstraction is to consider only those aspects needs for problem & remove irrelavant details whereas decomposition breaks problem into subproblem through which we can easily deal with problem. These ways make software product effective.

Needs of Software Engineering

The needs of software engineering arise b/z of heigher rate of change in user req. & environment on which the software is working.

① Large Software :- for making large software, we need some scintific methods, principles of Software Engineering.

② Scalability :- If the software engineering techniques in software, then we can scale the software.

③ Cost :- We need heigh budget for making qualitable software.

④ Dynamic Nature :- The always growing & adapting nature of software hugely depends upon the environment in which the user work. If nature is always changing, new enhancement need to be done in existing one. This is software engineering plays a good role.

⑤ Quality Management :- Better process of software developed provides better & quality software product.

Characteristic of Good Software :-

A Software product can be judge by what it offers & how well it can be used.
The software following grounds—

- Operational
- Transitional
- Maintanance

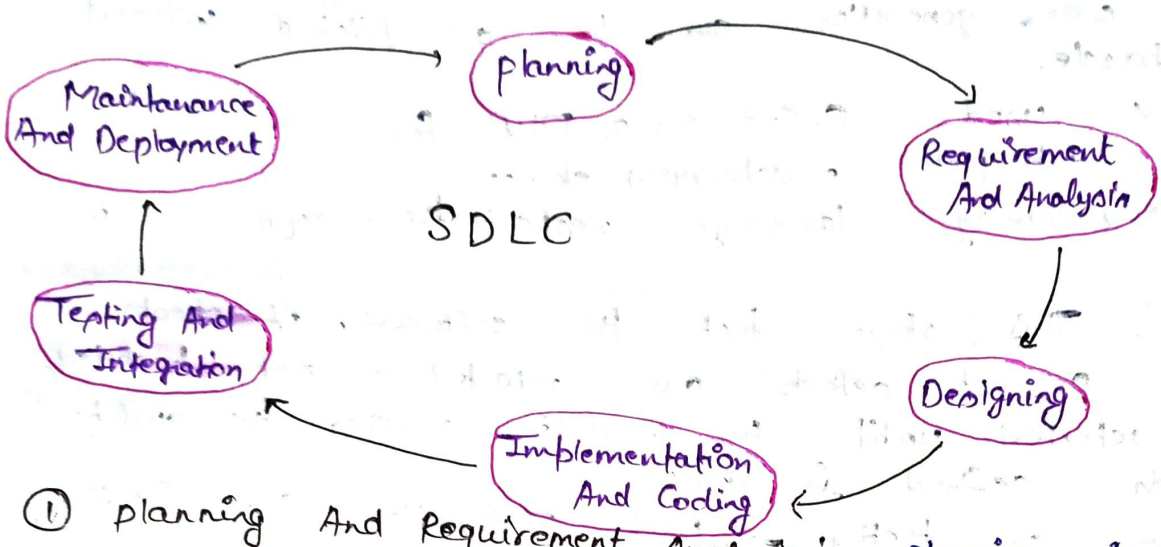
Operational :- This tell us how well software work in operation.
Budget, Usability, Efficient, Correctness, Functionality, Dependability, Security, Safety.

Transitional :- The aspect is important when the software is moved from one platform to another.
Portability, Interoperability (predicting data using previous data), Reusability, Adaptability

Maintanance :- The aspects briefs about how well a software has the capabilities to maintain itself in the ever changing environment.
Modularity, Maintainability, Flexibility, Scalability (Decomposition)

Software Development Life Cycle (SDLC)

SDLC is a process followed for a software project, within a software organization. It consist of detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving quality of software & overall the development process.



① Planning And Requirement Analysis :- planning is the quality assurance req. & identification of the risk associated with project is also done in the planning stage. The outcome of technical feasability study is to define the various technical approach that can be followed to implement the project sucessfully with minimum risk.

② Defining Requirements :- Once planning or req. analyses done the next step to clearly define & documents the product req. & get them approved from the customer or market analysts. This is done through an SRS (software Req. specification) which consist of all the product req. to be designed & developed the project life cycle.

③ Designing the Product Architecture :- SRS is the reference for product architecture to come out with best architects for the product to be developed. Based on the req. specified in SRS, usually more than one design approach for the product architecture is proposed &

documented in DDS - Design Document Specification. The DDS is reviewed by all the important parameters as risk assement, robustness, modularity, under budget, time contraints, for best design approach is selected for the product.

Implementation And Coding:- Now, actual development starts & product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.
Developers used C, C++, Java, PHP for coding with compiler, interpreter, debuggers etc...
The programming language respect the software is built.

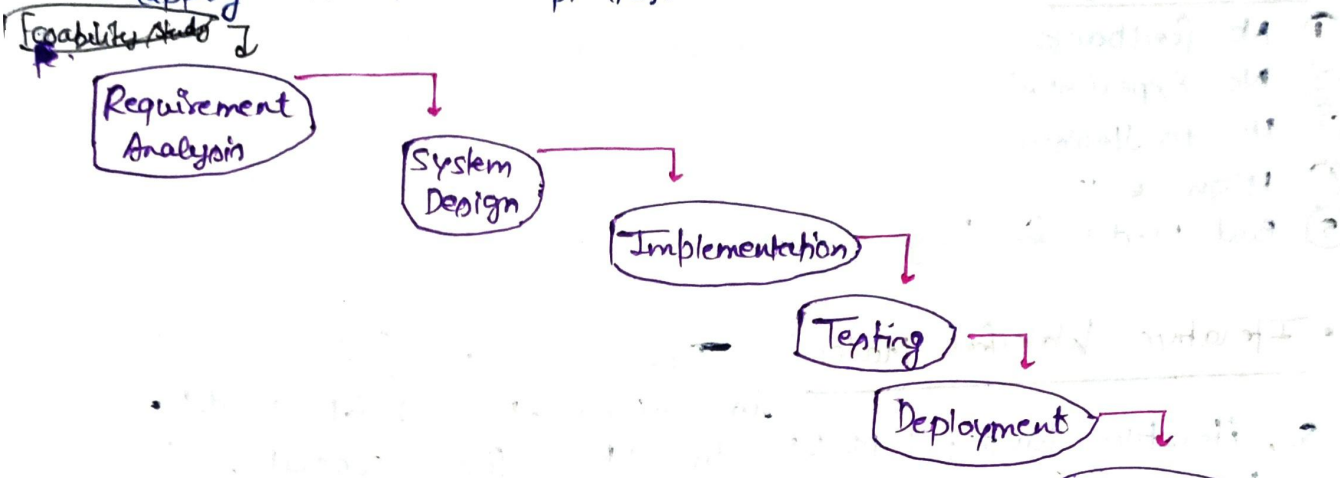
Testing:- This stage test the software, if checks product defects are reported or not, backeded, fixed, retested, until the product reaches the quality standards defined in SRS.
We can do testing - Unit, Integration, System Testing.

Maintanance And Deployment:- Once the product is tested & ready to be deployed it is released formally in appropiate market. The product may first be release in a limited segment & tested in real buisness environment (UAT - User Acceptance Testing).
Then based on feedback, the product may release as it same according to clients req. After it release.

SDLC Models:- There are various SDLC Models define and design for software development.

1. Waterfall Model
2. Iterative Waterfall Model
3. prototyping Model
4. Increamental Model
5. Evolutionary Model
6. Spiral Model

- **Classical Waterfall Model**: It is first process model to be introduced. It is referred to as a linear sequential life cycle model. In this, each phase must be completed before the next phase can be begin & there is no overlapping in the phases.



- **Requirement Gathering and analysis**:- All possible req. of the system to be developed are captured in this phase & documented in a req. specification document.

- **System Design**:- The req. specification from the first phase are studied, in this phase the system design is prepared. This system design helps in specifying hardware & system req. & helps in defining the overall system architecture.

- **Implementation And Coding**:- With inputs from the system design, the system is first developed in small program called units, After designing coding part can be done using C, C++, Java etc... language.

- **Integration & Testing**:- All implementation done, then testing part start on software in unit integration & system in testing.

- **Deployment And Maintanance**:- Once functional and non-functional testing done the product is deployed in the customer or released into the market. There are some issue which come up in the client environment when they used. So, Maintanance is done to deliver these changes in the customer environment.

Advantage:

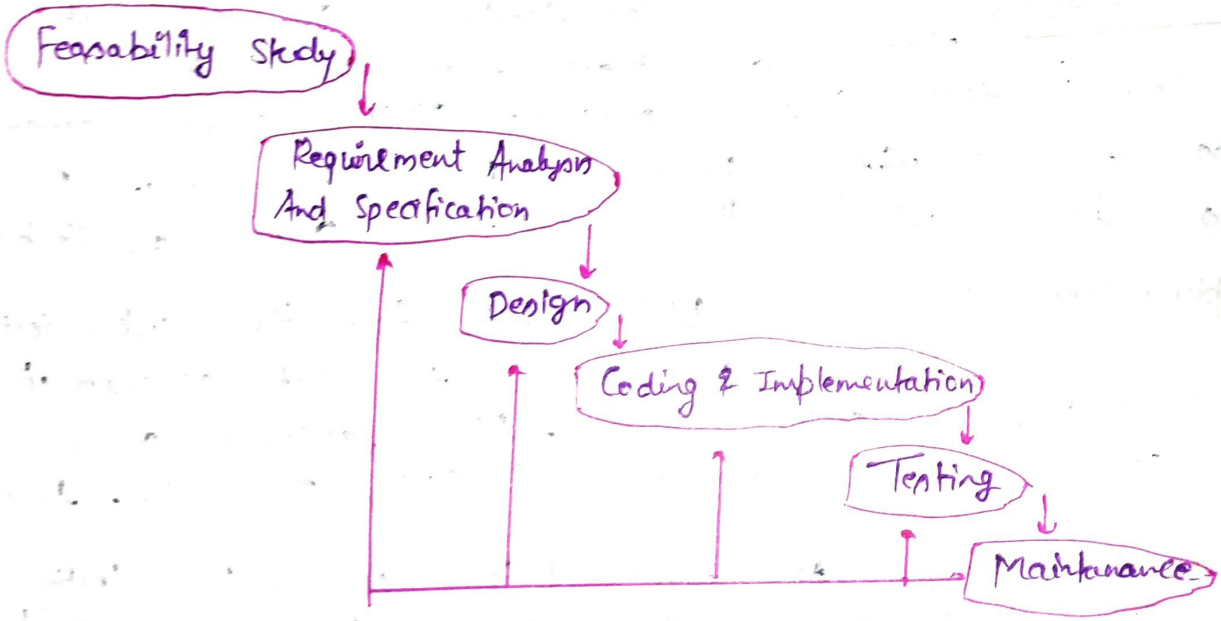
1. Base Model
2. Simple & Easy
3. Small projects

Disadvantages:

1. No feedback
2. No Experiment
3. No parallelism
4. High Risk
5. Bad Model for Large projects

- **Iterative Waterfall Model**:- There are no feedback system in classical waterfall model. So, iterative Waterfall Model thought for operating this problem.

The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.



When errors are detected some later phase, these feedback path allows correcting errors committing by programmers during some phase. There is no feedback path to be feasability study. Bcz once a project has bee taken, doesn't give up the project easily.

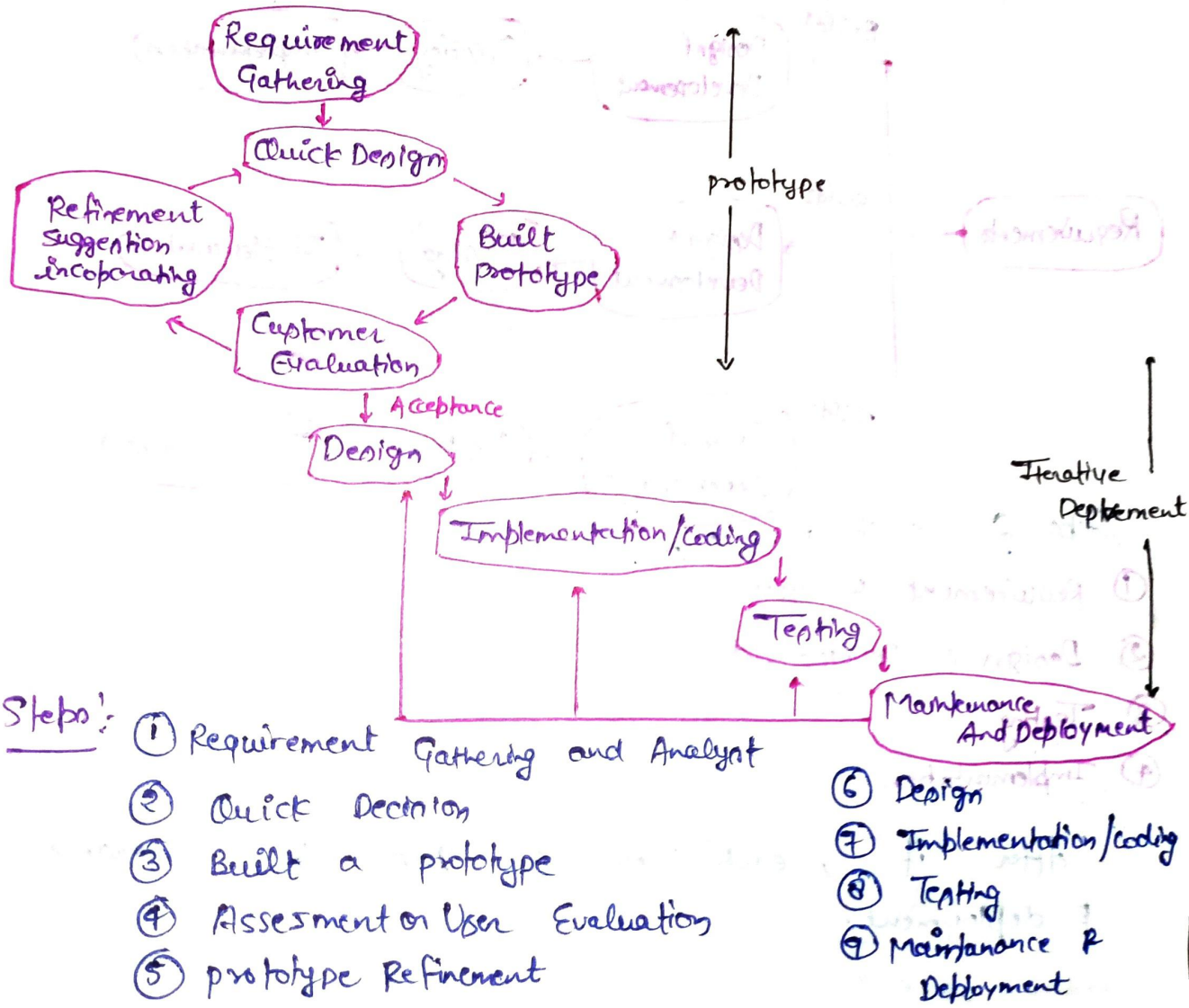
Advantage:-

1. feedback Path
2. Simple
3. ~~Design~~ cost Effective

④ In less time, make small projects in less time.

Disadvantages:
① Difficult to incoporate change request.
② Incremental Delivery not support
③ Overlaping of phase not support
④ Risk handling not supported
⑤ limited Customer Interactions

• Prototype Model: Before carrying development of actual software, a working of the system should be built. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limiting functional capibilities as compared to actual software.
It made bcz client not clear with its own idea. So in this case we made one prototype of the software. After satisfying the client, we go for further phases.
It is an throwaway model, good for technical & req. risk, increase in cost of development.

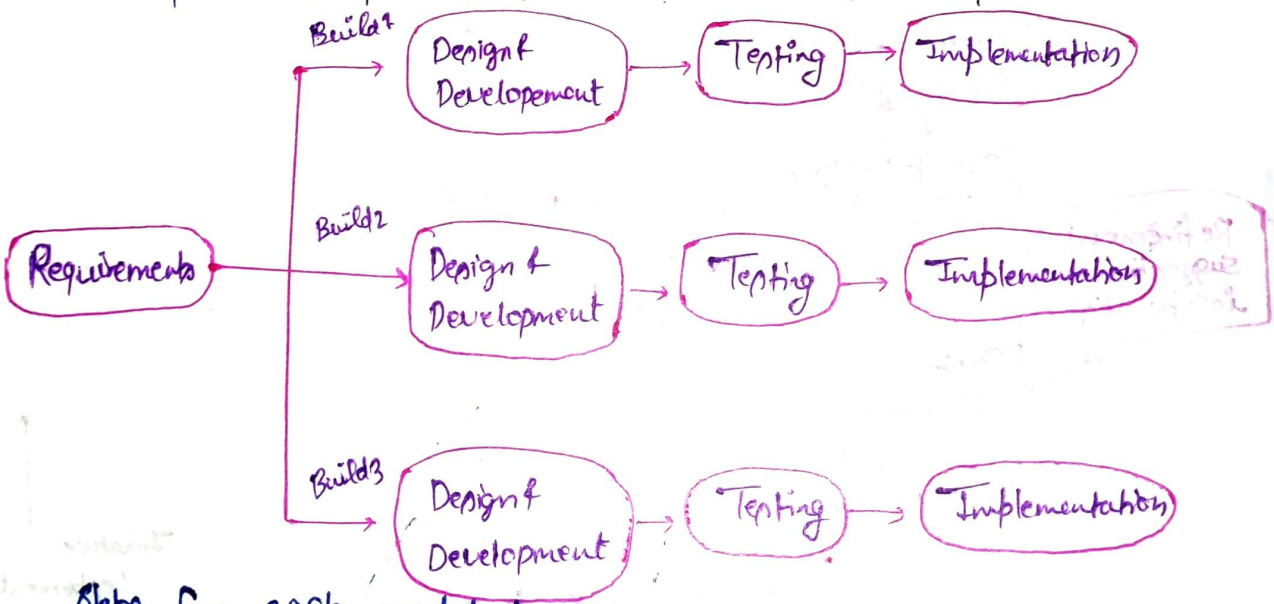


Steps:
① Requirement Gathering and Analyst
② Quick Decision
③ Built a prototype
④ Assesment or User Evaluation
⑤ prototype Refinement
⑥ Design
⑦ Implementation/coding
⑧ Testing
⑨ maintanance & Deployment

Advantage!- ① Reduce risk of incorrect user req.
② Good satisfication provided for clients.
③ Reduce maintanance cost
④ Errors can be detecting earlier.

Dis-advantage!- ① prototyping tools are expensive
② Time consuming process
③ Special tools & techniques are req. to built a prototype.
④ Req. cost customer money, need comitted customer, difficult if customer withdraw

• Incremental Model!- A process of software development where req. divide into multiple modules of software development cycle.
In which each module go with req., design, implementation & testing phase.
Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.



Steps for each module!:
① Requirement Analysis
② Design & Development
③ Testing
④ Implementation

After it, each modules goes for maintanance & deployment.

Advantage!- ① Customer Interaction Maximum
② More flexible
③ Easier to test & debug
④ Large Projects
⑤ Manage risk

Disadvantage!- ① Needs for good planning, otherwise whole project goes for wrong direction
② Total Cost is heigh
③ Well defined modules interfaces are need.

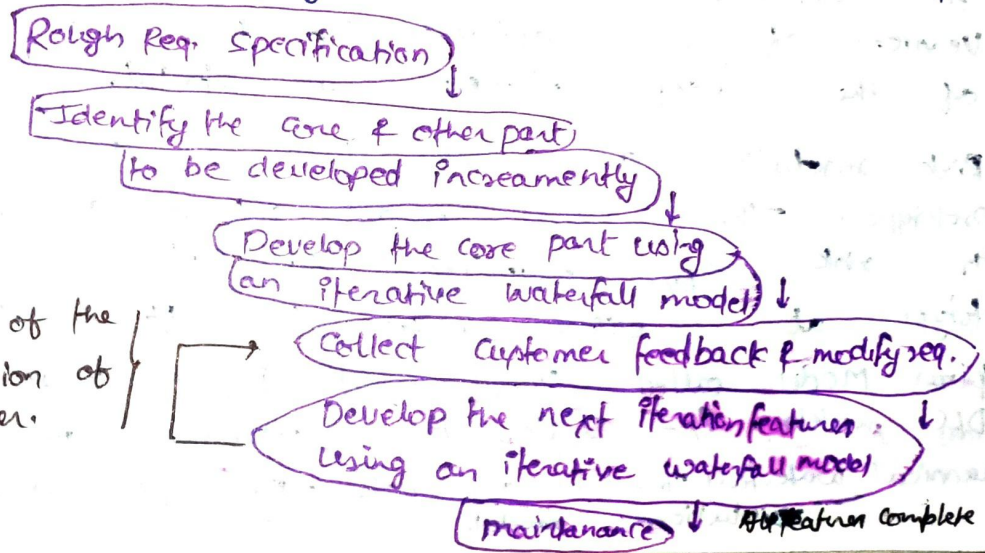
Evolutionary Model!- It is an combination of iterative and increamental model.

Increamental model first implement a basic few features & deliver again to the steps until the desired system is fully realized. Iterative model advantage gave feedback process in every phase.

Also known as "Design a little, build a little, test a little.

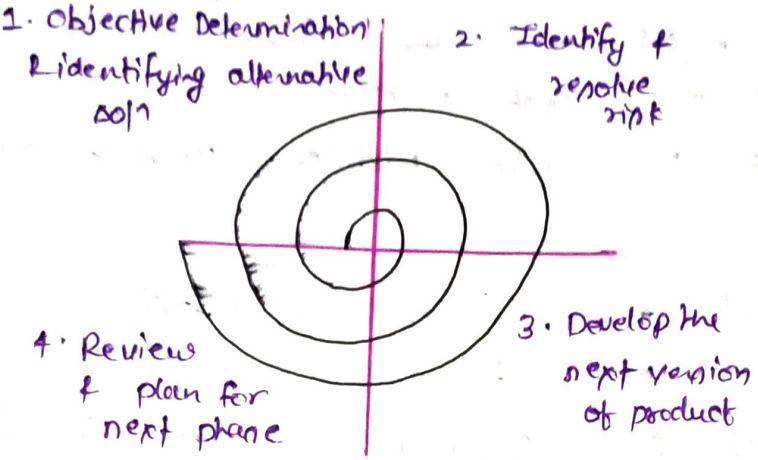
Advantage!- ① Customer req. are clearly specified.
② Risk analysis better
③ It support chaining environment, which is easy
④ Intial operating time is less
⑤ Suitable for large - mission critical specified.

Dis-advantage!- ① Not suitable for small projects
② Cost
③ Highly skilled resource are req.



Spiral Model :- It supports risk handling. In diagrammatic representation, it looks like a spiral with many loops. Each loop in the spiral called phase of the software development process.

Radius of spiral is equal to cost, and Angular dimension represents the progress made so far in the current phase.



1. Objective Determination & identifying alternative soln :- Req. are gathering from the customer & the objective are identified. Then, alternative soln possible for the phase are proposed in this quadrant.

2. Identify & resolve risk :- All the possible soln are evaluated to select the best possible soln, which resolved risk. At the end prototype is built for the best possible soln.

3. Develop next version of the product :- Identified the features are developed & verified through testing. At the end of 3rd quadrant the next version of the software is available.

4. Review & plan for the next Phase :- In the fourth quadrant, the customer evaluated the so far developed version of the software. In the end, planning for of the next phase is started.

Risk resolution are easier done by developing a prototype. The spiral model supports copying up with risk by providing the scope to build a prototype at every phase of software development.

Spiral Model called Metal model bcz it subsumes of all SDLC models. Ex- Single loop represent iterative next classical waterfall model approach, Prototype model, then evolutionary model.

Advantage!-
1. Risk Handling
2. Good for large projects
3. Flexible in Requirements
4. Customer satisfaction

Dis-advantage!-
1. Complex
2. Expensive
3. Too much dependable risk analysis
4. Difficult in time management

Software Requirements!- It is the description of features & functionalities of the target system.
It is the description what system should do.

Requirement Engineering (RE) refers to the model process of defining, documenting & maintaining req. in the engineering design process.

It is 4 steps process —
1. Feasability study (system is feasable)
2. Requirment Gathering / Elicitation (survey) // Req. elicitation.
3. Software Req. / Specification (well defined SRR) // Req. documentation
4. Software Req. Validation (Accuracy check) // Requirement conformation

Tools to support for Req. Engineering —
1. Observation reports (user observation)
2. Questionaries (interviews, surveys & polls)
3. Use Cases
4. Use stories
5. Req. workshop
6. Mind-mapping
7. Prototyping
8. Role playing

Software Requirement Specification (SRS)!-

SRS stands for Software Requirement Specification, is a document where all the software req. are written.
- It provided a detail overview of the software product, or it is a way to represent req. in a consistent format.

- It describes what the software will do & how it will be expected to perform.
- SRS is the complete description of the software system to be developed include all functional & non-functional Req.

Functional Req. which are related to functional/working aspect of software fall into this category

Non-functional Req. (security, storage, configuration (server), performance, cost, interportability, flexibility, disaster recovery, accessibility).

- SRS is given a complete picture of the entire project.
- This document is important b/z this is the base of the project which helps engineers to develop the software systematically.
- SRS document should be written in natural language, in an unambiguous manner that may also include charts, tables, DFD decision tables, and so -on.
- A specification can be written document, graphical model, A formal mathematical model, A prototype etc.
- SRS is written by business analyst, software architects software programmers, or any other analyst who is familiar with the product.

Uses of SRS document

1. Project manager base their plan, & estimate the schedule, effort or resources on it.
2. The development team needs it to develop the product.
3. The testing group needs to generate the test plans.
4. The maintenance and product support staff need it to understand what the software product supposed to do.
5. publication group write documentation
6. Customer rely on it to know what the product they can expect.
7. The SRS document can be used to resolve any disagreements b/w the developers & the customer which may arise in the future.

Problems without SRS document!-

1. The product will not develop in systematic way.
2. Without it, the system would not be implemented a/c to customer needs.
3. Software developers would not know whether what they are developing.
4. Without SRS, it will be difficult for maintanance engineer to understand the functionality of the system.
5. It will be very difficult to write the user manuals without understanding the SRS document.

Characteristics of good SRS document!-

1. **Completeness!-** SRS document contains functional, non-functional req., goals, purpose objective etc..
2. **Correctness!-** All requirement in SRS document should be correct
3. **Unambigous!-** The requirements stated have only meaning. Req. written in natural language with no ambiguity.
4. **Modifiable!-** It should be Modifiable bcz the user req. can be changed in future.
5. **feasable!-** All req. includes in the SRS document must be feasable to implement.
6. **Testable!-** SRS doament written in that way that can be tested in easy way.
7. **Consistency!-** should be consistent, no conflicts blw set of req.

Standard fomat of SRS Document

- Introduction
  - Purpose
  - Scope
  - Defination, ACRONYMS AND ABBREVIATIONS
  - REFERENCES
- Functional Requirements
- Overall Description
  - Product perpective
  - Product features
  - User Classes and Characteristics
  - Operating Environment
  - Design and implementation Constraints
  - Assumption and Dependancies

• External Interface Req.	• Non-functional Req.
• User Interface • System Interface • Hardware Interface • Software Interface • Communication Interface • Memory Constraints	• Performance • Security • Safety • Usability • Maintainability • Software Quality Attributes

Software Design

:- It is a process to transform user req. into some suitable form, which helps the programmer in software coding & implementation.

- For assesing user req., an SRS (Software Req. Specification) is created whereas for coding & implementation, there is a need of more specific & detailed req. in software terms.
- Software design is the first step in SDLC, which moves the concentration from problem domain to solution domain.

Software design yields three levels of results-

- **Architectural Design**:- It is the heighest abstract version of system, it identifies the software as a system with many components intracting with each other.
At this level, the designers get the idea of proposed solution domain.

- **High-level Design**:- It breaks the 'single entity-multiple component' concept of architectural design into less abstracted view of sub-systems & modules and their interaction with each other.
It focuses how the system along with all of its components can be implemented in forms of modules.
It recognize modular structure of each sub-system & their relation & intraction among each other.

- **Detailed Design**:- It defines logical structure of each module & their interfaces to communicate with other modules.
The detailed design deals with implementation part of what is seen as a system & its sub-system in the previous two designs.

Modularization:- It is a technique to divide a software system into multiple discrete & independent modules, which are expected to be capable of carrying out tasks independently.

Designers tend to design modules such that they can be executed and/or compiled seprately & independently.

Modular design unintentionally follows the rule of 'divide & conquer' problem-solving strategy this is beacaine there are many other benefits with modular design of a software.

Advantages:-

- Smaller components easy to maintain.
- Program can be divided on functional aspects.
- Concurrent execution can be made-possible.
- Desired from security aspect.
- Desired level of abstraction can be brought in the program.
- Concurrent with cohesion can be re-used again.

Concurrency:- Back in time, all softwares were meant to be executed sequentially. It's means coded instructions will be executed one after another implying only one portion of program, being activated at a given time.

"Concurrency Control provides capability to the software to execute more than one part of code be made parallel execution".

Example, the spell check feature in word processor is a module of software, which runs alongside the word processor itself.

Coupling And Cohesion:- When a software is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks.

They are though, considered as single entity but may refer to each other to work together.

There are measures by which of quality of a design of modules & their interaction among them can be measured. These measures are called coupling & cohesion.

Cohesion:- It measures that define the degree of intra-dependability within elements of a module. The greater cohesion, the better is program design.

Several types of cohesion —

(a) Co-incidental Cohesion :- It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization.
Because it is unplanned, it may serve confusion to the programmers & is generally not-accepted.

(b) Logical Cohesion :- When logically categorized elements are put together into a module, it is called logical Cohesion.

(c) Temporal Cohesion :- When a elements of module are organized such that they are processed at similar point in time, it is called temporal Cohesion.

(d) Procedural Cohesion :- When elements of module are group together, which are executed sequentially to perform a task, it is called procedural Cohesion.

(e) Sequential Cohesion :- When elements of module are grouped beacause the output of one element serves as input to another & so on ...,
it is called sequential Cohesion.

(f) Functional Cohesion :- It is considered as heighest degree of cohesion, & it is heighly expected.
Elements of module in functional cohesion are grouped beacause they all contributed to a single well-defination function. It can also be re-used.

Coupling :- It measures a that defines the level of interdependability among modules of a program. It tells at what level the modules interface & interact with each other.
The lower coupling, the better the program.
There are 5 coupling -

(a) Content Coupling :- When a module can directly acess or modify or refer to the content of another module,
it is called content level coupling.

(b) Common Coupling: When a multiple modules have read & write access to some global, it is called common or global coupling.

(c) Stamp Coupling: When multiple modules share common data structure & work on different part of it, it is called stamp coupling.

(d) Control Coupling: Two modules are called control-coupled, if one of them decided the function of the other module or changes its flow of execution.

(e) Data Coupling: Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then receiving module should use all its components.

Ideally no, coupling is considered to be the best.

Function Oriented Design: The system is comprised of many smaller sub-system known as functions. These functions are capable of performing task in the system.
It follows divide & conquer technique, & abstraction technique.

Structure Design: It mostly based upon 'divide & conquer' technique where several small problems & each small problem is individually solved until the whole problem is solved. Small pieces are solution modules. They arranged in hierarchy by which they communicate to each other & achieve precise soln.

Cohesion - group of all functionality related elements.

Coupling - Communication b/w different modules.

A good structured design has high cohesion & low coupling arrangements.

Object - Oriented Design: It works around real world entities which have some properties.

It has -
1. Object
2. Classes
3. Encapsulation
4. polymorphism
5. Inheritence

Software Design Approach: There are 2 generic approaches for software engineering.

- Top-down Design: Top-down design takes the whole software system as one entity & then decomposes it to achieve more than one sub-system or component based on some characteristic. And this process goes further next.
- Bottom to up Design: It proceeds with composing higher level of components by using lower level components.

DFD: DFD stands for Data flow Diagram. It represents the flow of data.

- It takes input & processed & provides output.
- DFD does not have control flow & no loops & no decision it simply have graphical structure design. • Easy to understand • We made many DFD until we reach our destination.
- It doesn't mention about data flows through the system. It is a good communication tool blw user & system designer.

Types of DFD – Data flow Diagram either logical or physical.

① Logical DFD: It shows flow of data in system.
For example in a Banking software system, how data is moved blw different entities.

② Physical DFD: It shows how the data flows is actually implemented in the system. It is more specific & close to the implementation.

DFD Components – It contains 4 components.

① Entities: Entities are source & destination of information data.
It represented by Rectangles.

② Processes: Activities & action or operation taken on data are represented by circle or round edge rectangle.

circle ◯ or ▢

③ Process, Data Storage:- It is a place where we store our data.

It can represented as a rectangle, with only one side missing. ⊏ or ═ open rectangle

④ Data flow:- Movement of data is shown by pointer arrow. Data arrow provides direction from source to destination.

→ Data flow (Arrow)

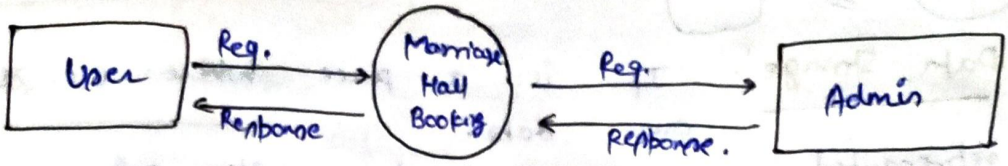
Logical DFD	Physical DFD
• It shows how data flow in a system. • It describes what data is moved from one entity to another. • It focus on what happens in data flow. • It shows how system works not how system be implemented. • It is implementation independent.	• It shows how data flow is actually implement in the system. • It describes how data is moved from one entity to another. • It shows how the current system operates & how a system will be implemented. • It is implementation dependent.

☐ There are mainly 3 levels in DFD—

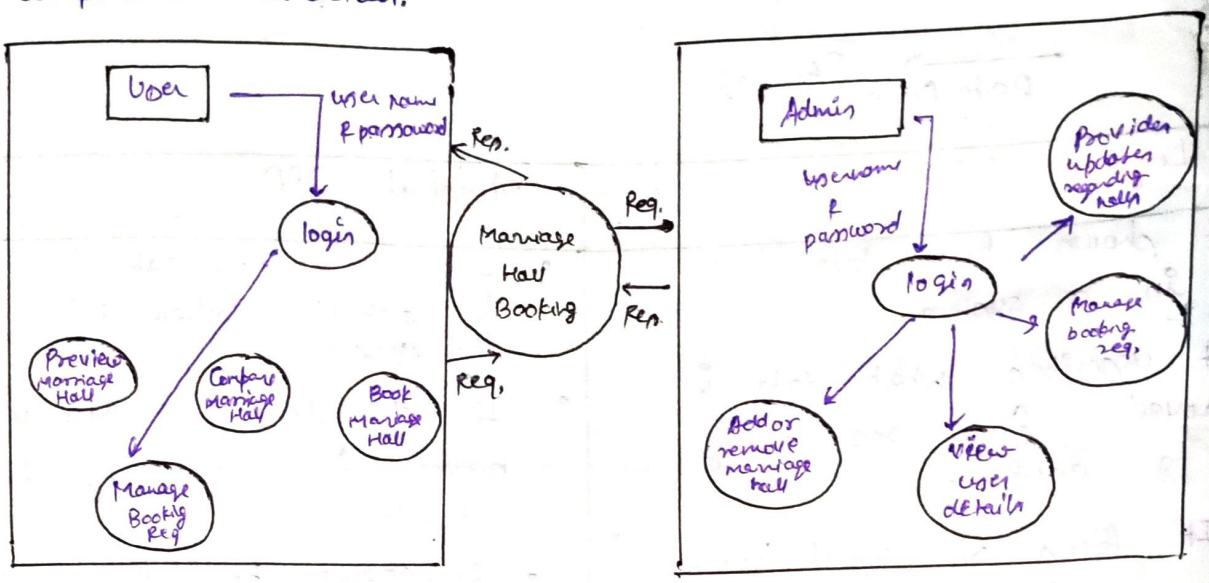
- 0- level DFD
- 1- level DFD
- 2- level DFD

0-level DFD:- It shows the system as a single process with its relationship to external entities.

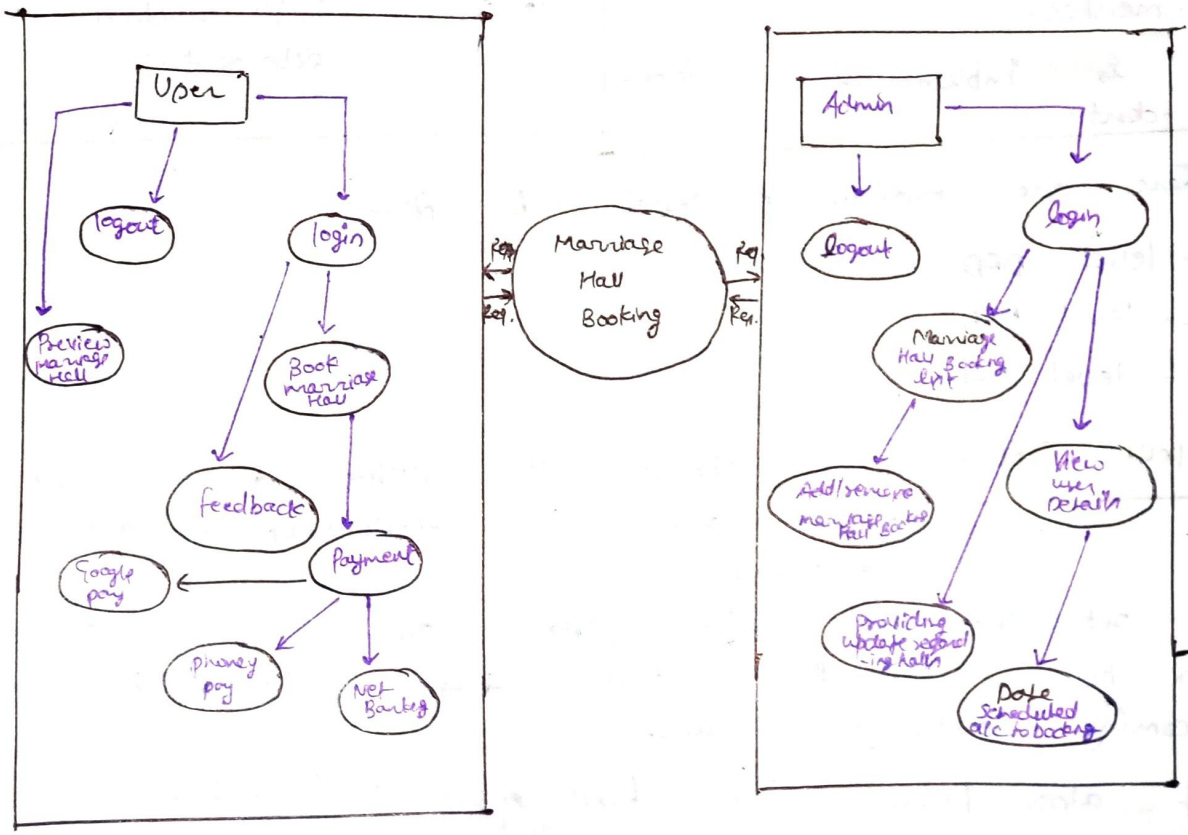
- It represents the entire system as a single process with the input & output data indicated by incoming / outgoing arrows.
- It also known as fundamental model & context design.
- It shows an abstract view (small view).



1-level DFD:- In 1 level DFD, we decompose the context level DFD (0-level DFD) into multiple components & define each component in detail.



2-level DFD:- In level 2 DFD, we decompose the 1 level DFD into more detail each necessary detail about system functioning properly.



Advantages:
- The Graphical view is easy to understand.
- Users easily understood the functionalities, the visualize contents.
- Anyone can understand either technical or non-technical person.

Dis-advantages:
- At the time DFD can confuse the programmers regarding the system (in 2 level DFD).
- It takes long time to made design.

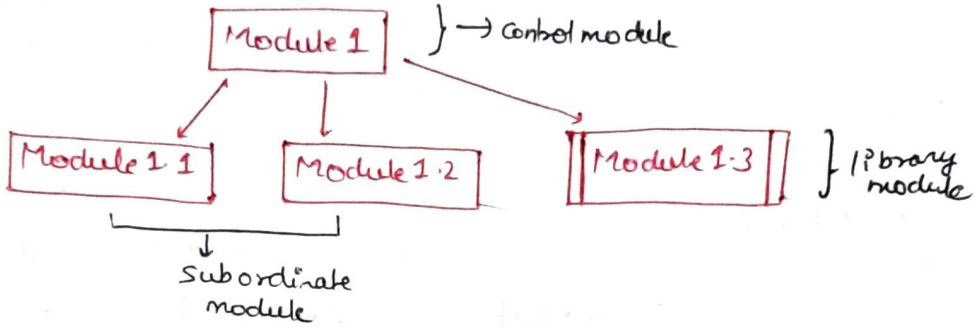
Structure Charts: It is an software engineering & organizational theory, is a chart shown the breakdown of a system to its lowest manageable levels. They are used in structure form to arrange modules into a tree. Each module is represented by a box, which contains module's name.

- Structure chart is a chart derived from Data Flow Diagram. It represent the system in more detail than DFD. It breaks down the entire system into lowest functional module, describe functions & sub functions of each module of the system to a greater detail than DFD.
- It is heiraarchical structure of modules. At each layer is specific task is performed.

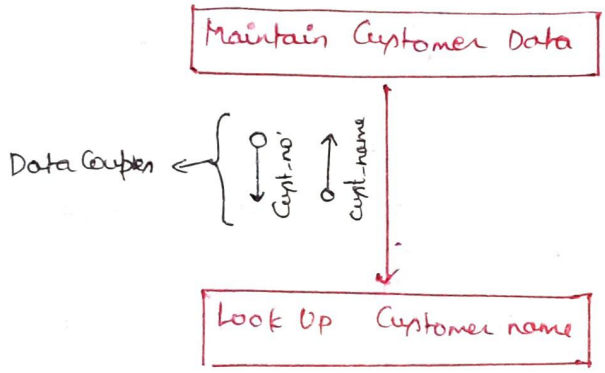
Uses –
- Describe functions & sub-functions of each part of system (in more detail than DFD).
- Show relationship blw common & unique modules of structure.
- Heirarichal, Modular Data Structure
  - (a) Each layer in a program performs specific activities.
  - (b) Each module perform a specific function.

Symbols –
- Modules (Seq. logic)
- Data Couples
- Conditions (decision logic)
- Loops (iterative logic)
- Control Coupling

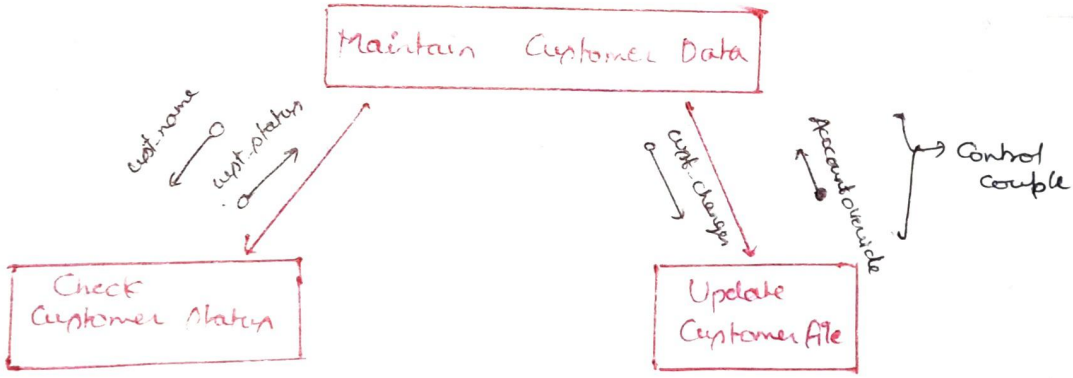
① **Module :-**
- Rectangle represent a module (program or subroutine).
- Control module (mainline) branch to sub-modules.
- Library module are reusable & can be invoked from more than one control module elsewhere in the system.



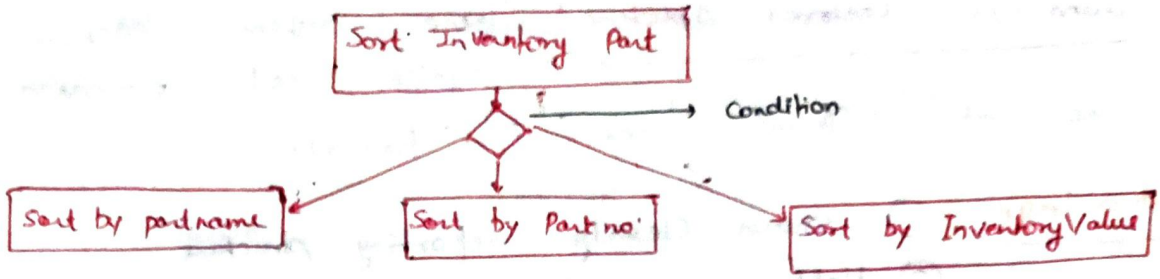
② **Data Couple :-**
- Arrow with an empty circle.
- Shows the data one module passes to another.



③ **Control Couple :-**
- Arrow with filled circle.
- Shows a message (flag) which one module sends to another. Module usage a flag to signal a specific condition or action to another module.

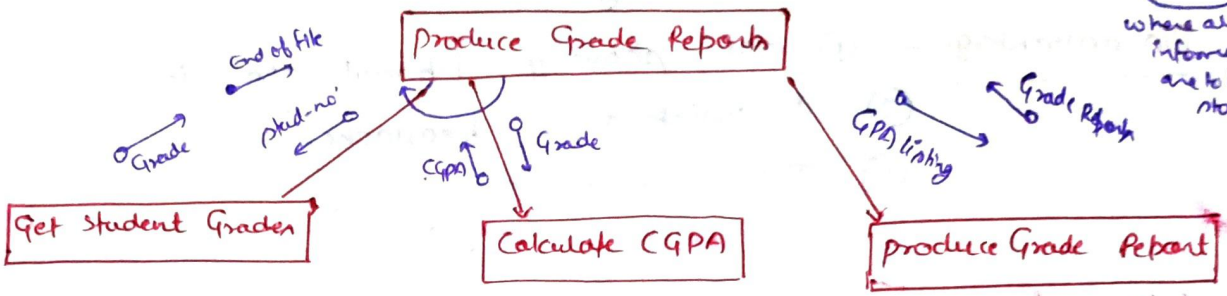


④ **Condition :-**
- A line with a diamond
- Idicate that a control module determines which subordinate module will be invoked.

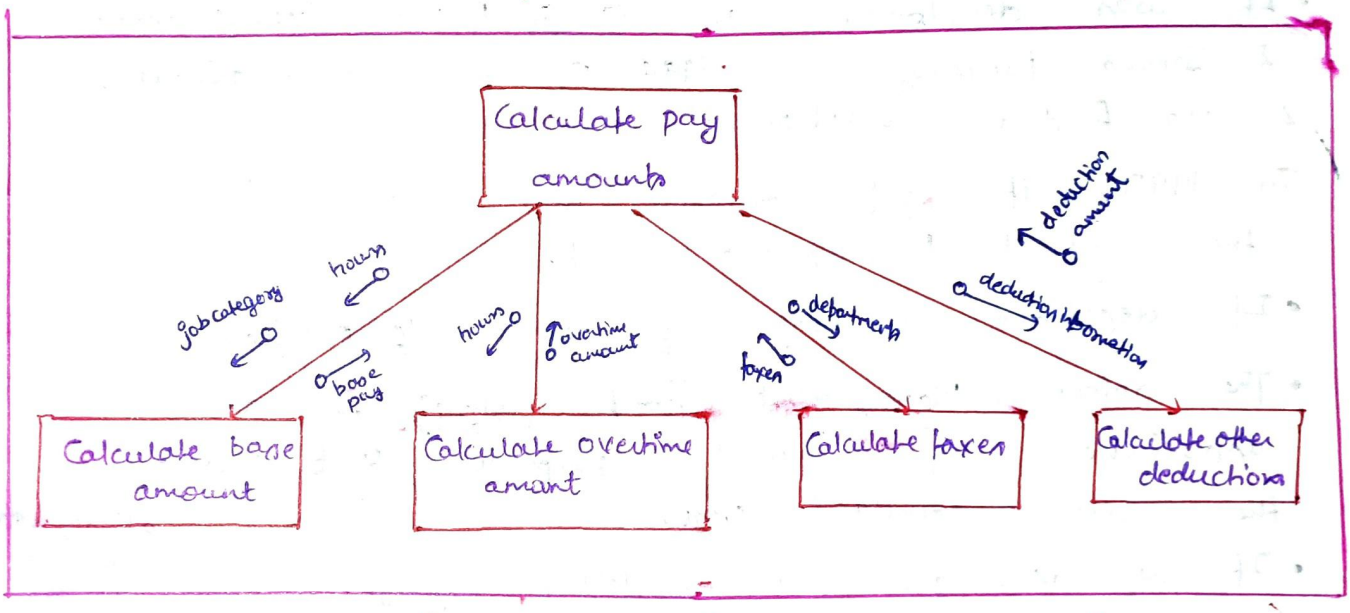


⑤ loop :- • A curved arrow representing loop
• One or more modules are repeated

⑥ physical storage - P.S where all information are to be stored.



Steps - ① Review DFD & object models
② Identify modules & relationship
③ Add couples, loops & conditions
④ Analyze structure chart, the DFDs, & the data dictionary.



Structure Chart for Calculate Pay Amounts

— Routine —

Types of Structure Charts -

① Transform Centered Structured - These type of structure designed for the system that receives input which is transformed by a seq. of operations being carried out by one module.

② **Transaction Centered Structure**: These system describes a system that processes a no of different types of transaction.

Advantage – ① Shows clearly reporting structure
② Helps new employees
③ Helps to manage workloads
④ Makes planning easier

Dis-advantage – ① Not showing informal channels
② A maintenance headache
③ Time Consuming.

UML :

- It stands for Unified Modelling language. (Scalable, secure & robust).
- It is an industory - standard graphical language for specifying, visualizing, constructing & documentation artifacts of s/w system.
- It was developed by Grady Booch, Ivar Jacobson, & James Rumbaugh in 1994 - 95 at relational software, & its further development was carried out in 1996. In 1997, it got adopted as a standard by the Object Management Group.
- It describes working of both s/f & H/w.
- The object Management Group (OMG) is an association of several companies that controls the open standard UML.
- It is an generalized modelling language.
- It is distinct from other programming language like C++, python etc. ...
- It is interrelated to object-oriented analysis & design. [pictorial language used to make blue prints]
- It is an pictorial language, used to generate powerful modelling artifacts.
- It is an important aspect for object-oriented software developement Use graphic notation to create visual models.

- Using UML diagrams we can generate code

- UML combines techniques data modelling, business modelling, object modelling, component modelling

"A diagram is a partial graphical representation of system model".

Following object-oriented concepts that are needed to begin with UML –

① Object!- An object is real world entity. There are many objects present with in a single system. It is an fundamental building block of UML.

② Class!- A class is a software blue print for objects, which it defines variables & methods inside it.

③ Abstraction!- It is an process of hiding irrelavant characteristics of an object to the user.

④ Inheritence!- It is an process by acquiring the properties of existing class then derived a new class.

⑤ polymorphism!- It is a mechanism of representing objects having multiple forms used for different purposes.

⑥ Encapsulation!- It binds data & the object together as a single unit,

Conceptual Model of UML!- It is the first step before drawing a UML diagram. It helps to understand the entities in the real world & how they interact to each other.

Three major elements –

- UML building block
- Rules to connect the building block
- Common mechanism of UML

UML diagram hierarchy

graphical (things + relationships)

Diagram

(Static)

Structural Diagram

- Class Diagram
- Component Diagram
- Object Diagram
- Package Diagram
- Composite structural Diagram
- Profile Diagram
- Deployment Diagram

(Dynamic)

Behaviour Diagram

- Activity Diagram
- UseCase Diagram
- State Machine Diagram
- Interaction Diagram
  - Sequence Diagram
  - Interaction overview Diagram
  - Communication Diagram
  - Timing Diagram

Symbols:-

1. Class
2. interface
3. Use case
4. State
5. Component
6. Node
7. Activity
8. decision
9. Note
10. Package
11. pattern
12. Synchronization
13. Association
14. Refinement
15. Transition
16. Dependency

Building blocks of UML:-

1. Things
2. Relationship
3. Diagrams

Needs:-

1. Analysis
2. Specification
3. Code generation
4. Design
5. Visual & understand the problem
6. Testing

The UML diagram capture 5 views —

1. User View (external user interface).
2. Structural View (define classes, objects, operation relationship among them)
3. Behavioral View (collaboration among objects).
4. Implementation View (important components & their dependencies)
5. Environment View (how different components are implemented on different piece of hardware).

Advantages :-

1. Readable & flexible
2. proper communication for software architecture
3. planning tool before making a program
4. Easy Debug any issue

Dis-advantages :-

1. Time Consuming
2. It just a language
3. Designing (that is all)
4. Complexity

UML Use Case Diagram:- It is used to represent the dynamic behaviour of system.

It encapsulates the system functionality by incorporating use cases, actors & their relationships.

Purpose:

1. It gather the system needs
2. It depicts the external view of system
3. It recognize the internal as well as external factors that influences the system.
4. It represents the interaction b/w the actor.

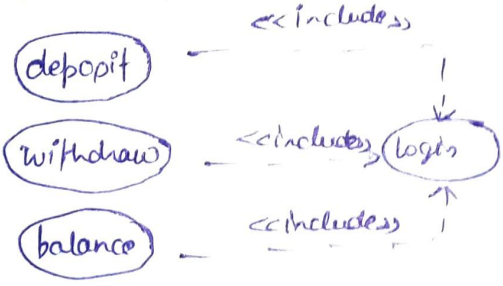
{ "Use case is a scenario by which user can use system & pictorially we represent those scenerios by use case diagram." }

Parts / Components:-

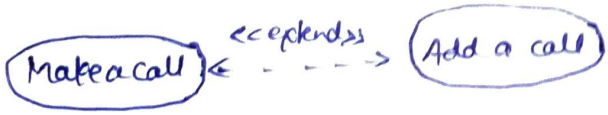
1.  (Actor) – role play (system) (real person) → who interact with system outside the boundary (i) primary actor (initiate) (ii) secondary actor (involved in use case)
2.  (use case) – capability (functionality or service provided by use case)
3. ——— (connector) – interaction
4. ⟶ (generalization)
5. - - - -> (stereotype) – relationship (relation b/w Actor & use case)

Relationship:- It categorized into 2 ways –

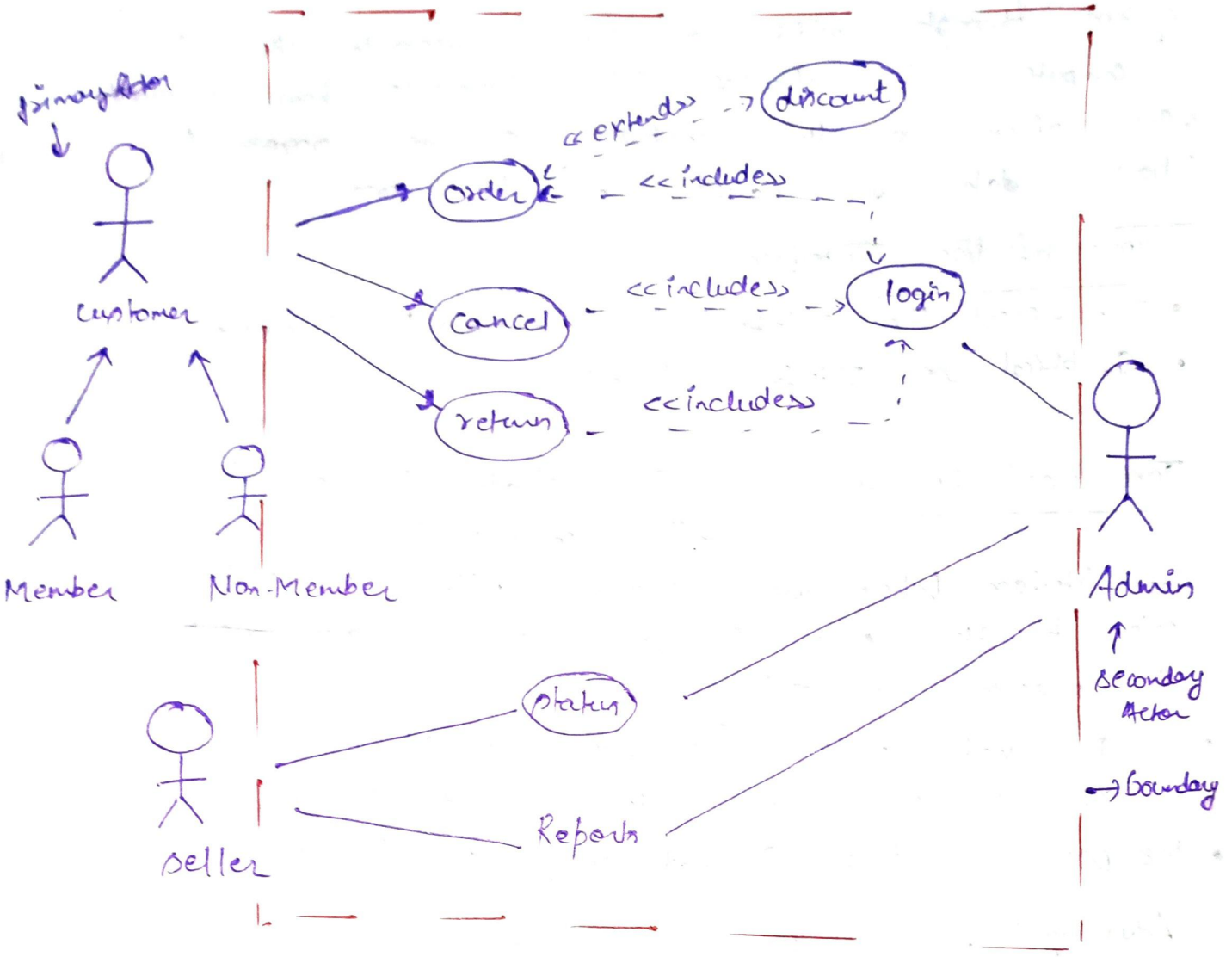
1. <<include>> – Implicit function (important to use)



2. <<extend>> – Explicit function (optional)



Shopping App Diagram Using UseCase!



Steps!-

1. Identify Actor
2. Identify Usecase
3. Identify include & extend usecases
4. Identify generalization
5. Identify relations
6. Draw Usecase

User Interface Design:- The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program & how data is displayed on the screen.

Types of User Interface —

- Text-Based User Interface or Command Line Interface
- Graphical User Interface (GUI)

Text-Based User Interface:- This methods used for interacting with computer. It is a user interface where user interacts with the computer solely through keyboard & seq. a command through to perform a task.

- CUI used earlier (precursor of GUI).
- Using commands user interacts with computer.
- MS-DOS & Windows Command Prompt, Linux(Unix)

Advantage:-
- Many & easier customized options
- Typically capable of more important tasks.

Dis-advantage:-
- Less Memory comparison to GUI
- Less expensive user b/z a lower resolution screen may be used.
- Navigation is difficult.
- Single user, & at a time.

GUI [Graphical User Interface]:- GUI relies much more heavily on the mouse.

Example - Windows operating systems

Characteristics → Multiple Windows, Icons based System, Graphics, Multi-User, multi-task, less time to take to perform any tasks.

Advantage:-
- Less expert knowledge
- Easy to navigate
- Multi-task perform at a time.

Dis-advantage:- • Not resource efficient (high power, memory consumption)
• User heavily depend on icon based system. So, they do not have knowledge, how the task is execute.

CUI/CLI	GUI
• Character User Interface or Command line Interface	• Graphical User Interface
• User use Commands like text for interaction	• User use graphics like icons for intraction.
• Navigation not easy	• Navigation is easy
• Must knowledge of commands for intractions.	• No need that much prior knowledge. Just clicked on icons.
• Low memory req.	• High memory req.
• Little flexible	• High flexible
• Single-User	• Multi User

UI Design Principles:- UI Design principles are — [Intractions blw user & software]

• Structure – Design should organize the user interface purposefully, consistent models & recogonize to users, putting related things together & seprate unrelated things.
The structure principle is concerned with overall user interface architecture.

• Simplicity:- Design should simple, easy, communicating clearly in directly user's language & providing good shortcuts that are meaningfully related to longer procedures.

• Visibility:- The design should keep users informed of actions. The design should make all req. options & materials for a given functions visible without distracting the user with redundant data.

Feedback!- The design should keep user informed of actions or interpretation, changes of state or condition & bugs or exception that are relevant & of interest to the user through clear, concise, language familiar to users.

Tolerance!- The design should flexible & tolerate, decreasing the cost of errors & misuse by allowing undoing & redoing while also preventing bugs, wherever possible by tolerating varied inputs & sequences & by interpretating all reasonable actions.

Of → [Simplicity, Know your user, Efficiency, Stay Consistent, User Visual Hierarchy, feedback].

Coding!- It is an process of transforming the design of a system into a computer language format.

- The coding phase of software development is concerned with software translate design specification into the source code.
- Coding is done by coder or programmers who are independent people than the designer.
- The cost of testing & maintenance can be significantly reduced the effort & cost of the coding.

Goals!- ① To translate the design of system into a computer language format.

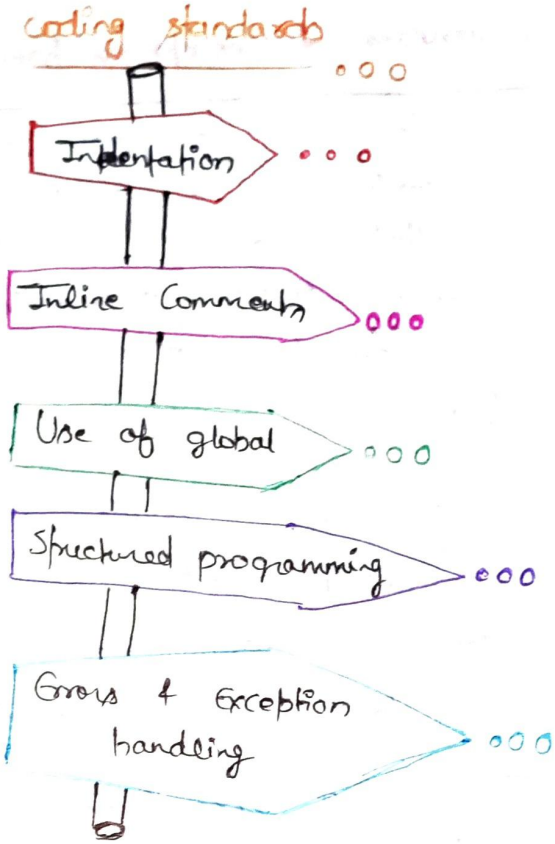
② To reduce the cost of later phases.

③ Making the program more readable

Characteristics!-

① Readability
② Error checking
③ Cost
④ Portability
⑤ Modularity
⑥ Generality
⑦ Quick Translation
⑧ Brevity
⑨ Efficiency
⑩ Widely available

Coding Standards: Generall coding standards refers how the developer write code. There are some essential coding standards regardless of the programming language being used.



- **Indentation**:- Proper & consistent indentation is essential in producing easy to read & maintainable programs.
  - It emphasize the body of a control structure such as loop or a select statement.
  - Emphasize the body of a conditional statement.
  - Emphasize a new scope block.
- **Inline Comments**:- Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
- **Use of global**:- These rules file what types of data can be declared & what can't.
- **Structured Programming**:- GoTo statement makes the program unstructured, it reduce the understandability of program. It dividing program into heirarchy modules, which may have, contain other elements.

- **Naming Conventions for global, local & constants identifiers!**

A global variable always begin with a capital letter, local variable names made of small letter (camelCase) & constant names always capital letters.

- **Error returning conventions & exceptions handling system!**

Different function in a program report the way of error conditions are handled should be standard within an organization.
Ex- different tasks while encountering an error condition should either return 0 or 1 consistently.

Coding Guildeliness:- If provide the programmers with set of the bests methods which can be used to make programs more comfortable to read & maintain.

- **Line length!**:- The source code lines at or below 80 characters.
- **Spacing!**:- Appropriate use of space with in a line of code.

Example!- ~~cout~~ cout << "The total cost" << cost << endl;

- **Code should be well-documented!**:- There must be at least one comment line on the average for every 3-source line.
- **The length of function!**:- A length of function should be 10 source lines.
- **Donot Use goto statements!**:- Because it makes program unstructured difficult to understand.
- **Inline Comments!**:- Inline comments promote readability
- **Error message!**:- Error handling is an essential aspects of computer programming. It involves making error messages meaningful.

Software Testing :- It is a widely used technology beacause it is compulsory to test each & every software before deployment.

" It is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability Usability) & evaluating the execution of s/w components to find s/w bugs & errors or defects ".

Attributes of Software :-

1. Reliability
2. Scalability
3. Portability
4. Reusability
5. Usability

Advantages of Testing

1. Highly efficient
2. Produce s/w with longer effective operation life.
3. Produce system with more closely meet user needs & req.
4. Some hidden problem can revealed by testing.
5. It reduce maintenance & deployment cost if we test our s/w in testing phase.
6. Client Trust build if we made effective product by testing the s/w.

Goals → Defect Detection, Defect Prevention, User Satisfication

What has to be tested ?

1. GUI 2. Security 3. User friendness 4. code 5. Modularity 6. efficiency 7. design 8. Accessibility

Fundamentals of Software Testing (good test & testability)

(good test → maximum ability to find errors; testability → ease of computer testing)

① operability – s/w high quality & less bugs
The better it work the more efficient it can be tested.

② Observability – o/p are generated for each i/p
"what you see what you test".

③ Controllability – The better we controlled the s/w the more testing will be automated.

④ decomposibility – Divide large module into multi sub module & test individually then combined.

⑤ simplicity – The less we test the more quickly we test.
- functional simplicity (function in usage should be simple)
- Structural simplicity (architecture used should be simple)
- Code Simplicity (Coding standard used should be simple).

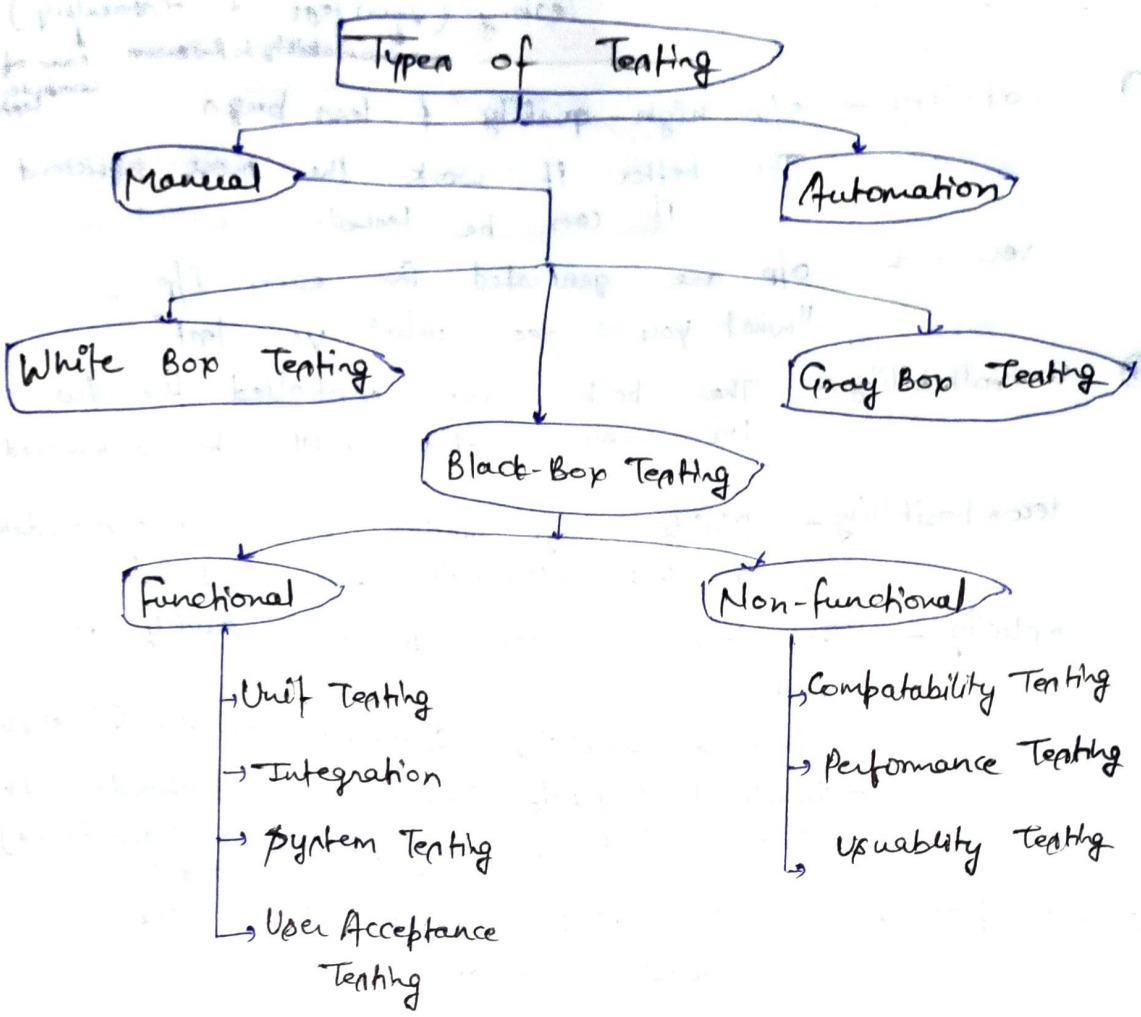
⑥ stability – fewer the changes, the fewer the disruption to test.

Roles of Testing :

① Identify error - bugs.
② Reduce the cost of s/w.
③ Test reports
④ Severity of error
⑤ Test - cases
⑥ Categorizes the errors

Impacts of Bugs :

① personal / Individual
② Society
③ Country
④ Organizational



Manual Testing :- The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual Testing.

" We do not need any specific knowledge of any testing tool, rather have a proper understanding of the product so we can easily prepare the test document ".

Types -

① White Box Testing :-

The white box testing is done by Developer where they check every line of code before giving it to the Test Engineer.

Since the code is visible for the developer during testing, that's why it is also known as White Box Testing.

It is also known as glass box testing, structural testing, clear box testing, open box testing & transparent box testing. It tests internal coding & infrastructure of a S/w focus on checking of predefined inputs against expected & desired o/ps.

- It is based upon internal structure, & skill of test programming are req. to design test cases.
- Developer do white-box testing. After it, the app send to testing team then they do black box testing.
- Inside it developer test path, loop, condition, tested based on memory perspective, test performance.

(2) Black Box Testing: It is done by Test Engineer, where they can check the functionality of an application or the S/w a/c to client needs.

- In this the code is not visible while testing, that's why it is known as black-box testing.
- In this method, tester selects a function & given input value to examine a functionality & check whether a function is giving expected output or not. If function produces correct O/P then it is passed in testing, otherwise fails. The test team report the result to the development team & the tests the next function. After completing of all functions if there are severe problems, then it go back to the development team for correction.

Techniques in Black Box Testing –

1. Decision Table Technique
2. Boundary Value Technique
3. State Transition Technique
4. All-pair Testing Technique
5. Cause-Effect Testing
6. Equivalence partition Testing Technique
7. Error Guessing Technique
8. Use Case Technique
9. Syntax testing
10. Use story Testing

Automation Testing!- It is a process which is done by with the help of automated tools.

- Converting any manual test cases into the test scripts with the help automation tools, or any programming language. Through if we can increase the speed of our test execution because here, we do n't req. any human efforts. We need to write a test scripts & execute those scripts.

Coverage Based Testing!- The aim of coverage based testing methods is to 'cover' the program with test cases that satisfy some fixed condition / criteria.
It is an method of White Box Testing.

It's types—

1. Statement coverage
2. Branch Coverage
3. Condition Coverage
4. Path Coverage
5. Multiple Condition Coverage

Statement Coverage!- Every statement of program should be checked at least once

Branch Coverage!- Every possible alternative in a branch or decision of the program should be checked at least once.

Condition Coverage!- Each condition in a branch is made to evaluate to both true & false.

Multiple Condition Coverage!- All possible combinations of conditions outcomes within each branch should be checked at least once.

path Coverage!- Each & every execution path of the program should be checked at least once.

Ex.. Control flow coverage, Data flow Coverage, Modified Coverage etc.

Test Case Design Techniques:-

A test case is a document which has a set of conditions or actions that are performed on the s/w application to verify the expected functionality of the feature.

⇒ Boundary Value Analysis (BVA)

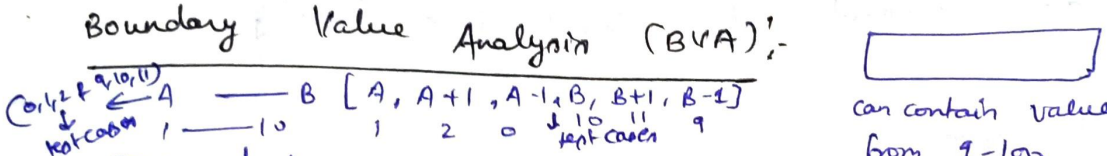
⇒ Equivalance Class Partitioning

⇒ Decision Table based Testing

⇒ State Transition

⇒ Error Guessing

Boundary Value Analysis (BVA):-



⇒ One test case for exact boundary of input domain each means 1 to 100.

⇒ One test case for just below of inputs domains each domain 0 to 99.

⇒ One test case for just above boundary values of input domain each means 2 to 101.

Equivalence Partitioning:-

It is also known as eq. class partitioning testing

In this, we divide in classes for testing problem. And check one value from each class if they verified then that the class test are pass. And we go through with next class. This saves our times.

Ex- 1 - 500 → classes

1 - 100 → if 99 in passed then this class is pass.
101 - 200
201 - 300
301 - 400
401 - 500
501 - 6000 → 502 → this value not pass then test class is fail.

Decision Table based Testing:-

The technique to the correct combination of inputs & determines the result of various combinations of inputs. To design the test cases by decision table, we need to consider conditions as inputs & action as output

Ex:- When we use email account, on any app we have to login with email & its password then their some test condition are checked —

	(1)	(2)	(3)	(4) → Test cases
1 Email (condition)	T	T	F	F
2 Password (condition)	T	F	T	F
3 Result (Action)	Accountpage	Incorrect password	Incorrect email	Incorrect email

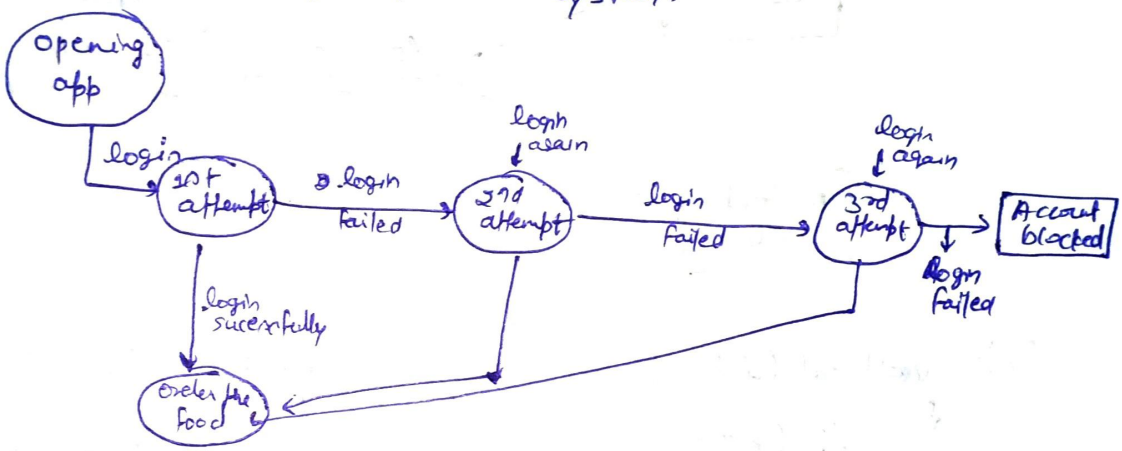
No of Test cases = 2^n ∴ n - no of condition

email & password → 2 condition

Test cases = $2^2 = 4$

State Transition Technique :- Stages you have all different stages of machine you need to test it all different phases of your system.

Ex-



Error Guessing :- Technique where there is no specific method for identifying the error. It is based upon the experience of test analyst, where the test uses the experience to guess the problematic areas of S/w.

In this we guess any i/p & check there is no kind of rules to perform error guessing technique.

It's main purpose to identify common errors —

- → Enter blank space into the text fields
- → Null pointer Exception
- → Divide by zero
- → Use maximum limit of files to be uploads
- → check button without entering values
- → Enter valid parameters

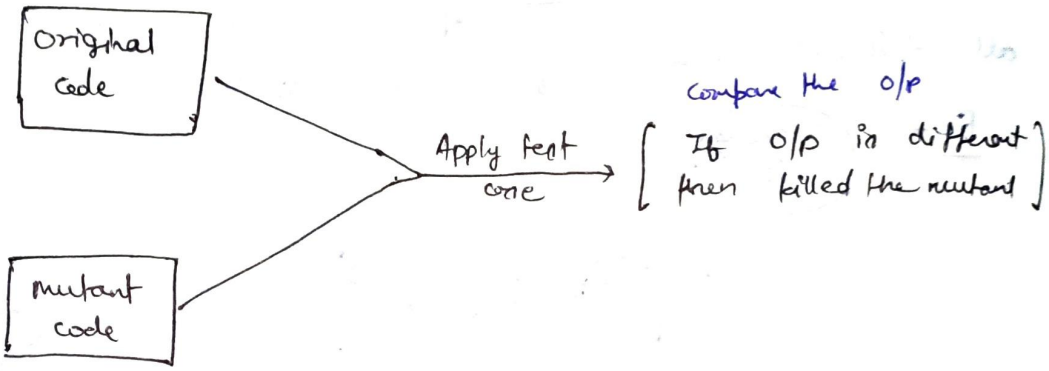
Ex- Mobile No' field —
(1) Text about to below digit by entering digits
(2) left blank
(3) Enter alphanumeric character

It's main purpose to find informal errors which can't find by any systematic approach or technique.

Mutation Testing:- It is a s/w testing where we mutate (change/alter) certain statements in the program & check if the test case are able to find errors.

- The change in the mutate program are kept extremely small, so it doesn't effect the overall objective of the program.
- It's goal to develop effective testcases. It's makes test cases robust.
- Also called, fault based testing

How to execute Mutation Testing →



Ex:-

```
void cal (int n)
{
    if (n%2==0)
      cout<<"Even";
    else
      cout<<"odd";
    return 0;
}
```

→ original code

```
void cal (int n)
{
    if (n/2==0)
      cout<<"Even";
    else
      cout<<"odd";
    return
}
```

→ mutant code

Test case at n = 10 original code = even
mutant code = odd
So, kill the mutant code.

Types of Mutation Testing:-

① Value Mutations:-

```
int sum
int c=10;
int b=20;
Sum=a+b;
cout<<sum;
```

```
int sum;
int a=5;   → value
int b=20;
Sum=a+b;
cout<<sum;
```

2. Decision Mutation :-

```
if (a<b)                    if (a>b)  → Decision
    c=10;                       c=10;
else                        else
    c=20;                       c=20;
```

3. Statement Mutation :-

```
if (a<b)                    if (a<b)
    c=10;                       a=a+b;  → Statement
else                        else
    c=20;                       c=20;
```

Mutation Score :-

$$\frac{\text{killed Mutant}}{\text{Total no of Mutant}} \times 100$$

If we can develop best test cases then mutation score will be 100%.

Advantages –

- It brings a good level of error detection in the program.
- It discovers ambiguities in the source code.

Dis-advantages –

- It is highly costly time-consuming.
- It is not for black box testing.

Static Analysis :- It involves going through the code in order to find out any possible defects in the code.

- It is analysis which can done before execution of program like the coding standards are used or not.
- This analysis most of the time at coding. While coding we check syntax errors, loop structure, colon termination.
- program run only after static analysis verification.
- It is non-run time environment.
- It is done after coding & before execution of program.
- Static analysis can be done by a machine to automatically walk through the source code & detect syntax errors. Ex- Eclipse. Or it can also analysis by person or programmer who knows coding standards.

Advantage - ① It can point out the exact spot in code where there is an error, so we can easily fix it.

② Problems can found earlier in the development life cycle, reducing time cost to fix.

③ Unique defect can be found that can't found at dynamic analysis.

Disadvantages:- ① It is time-consuming if conducted manually.

② Sometimes automated tools produce false result

③ It doesn't going to catch some runtime errors.

Dynamic Analysis:- ① It involves executing the code & analyzing the o/p.

② In this we checked that desired o/p is right or wrong

③ If any wrong or undesired o/p occur then we fix in program's code.

④ It is fast when the program is running.

⑤ A dynamic test will monitor system memory, function behaviour, response time & overall performance of system.

⑥ It is done after/at execution of program.

Advantages:- ① It identifies error while program running.

② It analysis for application in which you don't have acess to the actual code.

③ It find errors that are not found by static analysis.

Dis-advantages - ① It taken longer to fix error bcz if can't find exactly point where the error occurs.

② Automated tools in dynamic analysis not work properly sometimes.

Static Analysis	Dynamic Analysis
① It performs at non-runtime	① It performs at run-time
② works on source code	② works on executed code
③ Req. large amount of time & resource for debugging	③ Req. less time compared to static Analysis
④ It is a preventive Action.	④ It is a corrective action.
⑤ Can find exact location of errors.	⑤ can't find exact location of error
⑥ At static analysis, more defects occurs.	⑥ less defects occurs
⑦ It is performed before dynamic analysis.	⑦ It is performed after static analysis.
⑧ It is dependent on source code.	⑧ It is independent of source code.

Software Maintenance: ① s/w is very important activity in an organization b/c we want to do more better performance with our s/w.

② s/w must be upgraded & maintain regularly either depending upon the user req. or change in technology.

③ s/w maintenance is need to correct the error in the existing product, enhance the features & try to port or implement the s/w into new platforms.

④ In earlier, days s/w maintenance was the most neglected area by an organization but in the current trend we provide more focus on s/w maintenance in comparision to s/w development.

Types of s/w Maintanance:-

① **Corrective Maintenance:-** The problems reported by the users at a time of using the s/w produced. [20%]

② **Adaptive Maintenace:-** It describes it has the ability to run on different platforms, different OS & change GUI depending upon the new s/w or H/w. (Add some features). [25%]

③ **Perfective maintenance:-** The maintenance will describe that a s/w should be support new features, different functions and we can extend, the s/w according to the user req.
Better, faster, cleaner. [50%]

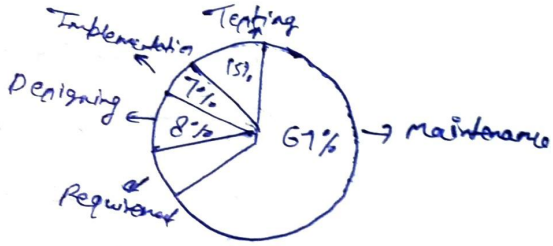
Needs of Software Maintenance:

1. for removing errors // error detections
2. For making our s/w user-friendly by taking feedback of users.
3. During maintenance replacing optimized codes
4. Unwanted side effects save by Software maintenance.
5. To satisfy user req. during maintenance.
6. Delete unwanted version/data
7. Enhancement of capabilities
8. optimization

4. Preventive Maintenance :- making the program easier to understand. // ability to prevent unwanted actions before these actions happening //
5% Helps us in future maintenance work.
e.g→ optimization code, documentation update.

5. Inspection :- These are usually made as a result of code inspection to reduce the likehood of a failure.

// It's primary goal to modify & update s/w applications after delivery to correct errors & improve performance //



Cost & efforts of S/w maintenance :-

The cost & effort of s/w maintenance can vary depending on the type of maintenance being performed & the complexity of the s/w system.
In general s/w maintenance can be a significant cost of organizations, as it typically involves a combination of labor, hardware & s/w costs.

- Labour costs :- software developers, engineers & technicians.
- Hardware & S/w costs :- servers, software licenses, development tools.
- Training costs :- maintenance tasks, such as S/w developers, engineers, & technicians.
- Tools costs
- Efforts →
  - Time & resources :- Time req. to identify & fix the problem, test the solution, & implement the solution.
  - Communication & Cordination :- efforts req. to communicate & coordinate with stakeholders such as customer & other teams.
  - Testing & validation :- Effort req. to test & validate the s/w to ensure that it is correctly & do not cause a problem.
  - Previous study.

Costs factors →
1. Complexity of S/w
2. Size of S/w
3. No of users
4. Change rate of S/w systems
5. Availability of personnel (skills)
6. Tools & technologies
7. location
8. Age of S/w systems

Software Reverse Engineering :- It is also called back engineering. It is the process of recovering & understand the basic methods, design & the req. specification of legacy S/w product.

(2) First we analyze the already created s/w source code after that recreating it so we can findout the original code with some improvement.

(3) Simple word, It in a process to re-think re-structure & re-built our legacy s/w.

Why?

- Sometimes situations arise that req. the use of reverse - engineering

Ex- If a part of an - super computer is broken which is not freely sold in market, then reverse engineering will be used to understand the design & [marketing] manufacturing process of that part. After that, the part will be manufactured.

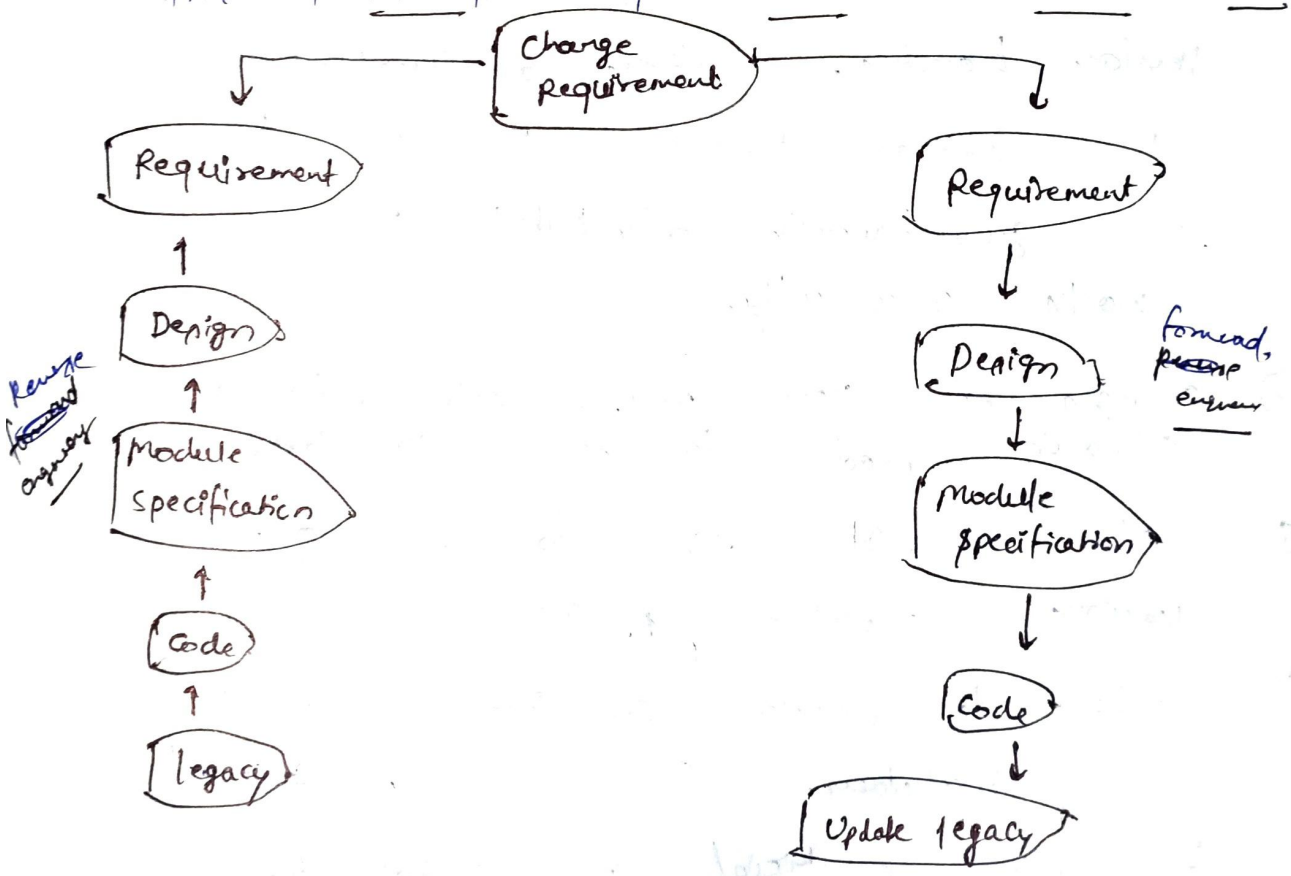


fig- s/w Reverse Engineering

Software Configuration Management Activities :-

S/w configuration Management (SCM) is defined as a process to systematically manage, organize & control changes in the documents, codes & other entities during S/w Development life cycle.

→ The primary goal is to increase productivity with minimal mistakes.

Needs :-

(1) There are multiple people working on S/w which is continually updating.

(2) It may be a case where multiple version, branches, authors are involved in a software project, & the team is geographically distributed & works concurrently.

(3) Changes in user req., policy, budget, schedule need to be accomodated.

(4) S/w should able to run on various machines & OS.

(5) Helps to develop co-ordination among stakeholders.

(6) It also beneficial to control the costs involved in making changes to a system.

SCM process defines a number of tasks —

Version Control!- Helps to make from existing product to new product

It is a tool that captures the changes to source code element : file, folder, image binary.

Change Control!- It is a systematic approach to managing all changes made to a product or system. The purpose to ensure that no unecessary changes are made, that all changes are documented, that services are not unecessarily disturbed & that resource are used efficiently.

Status Reporting!- Providing accurate status & current configuration data to developers, testers & end users, customers & stakeholders through admin ~~guest~~ guides, user guides, FAQ's, Release notes, etc..

Configuration Audit!- S/w Configuration audit verify that all S/w product satisfies the baseline needs. It that what is built is what is delivered.

identification!- Identification of object.

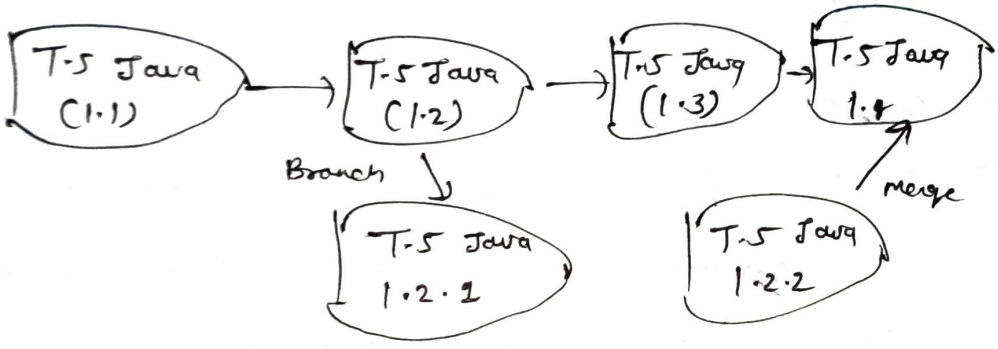
It is collection of base object & aggregate object. Every object has defined feature that it uniquely identified.

Software Version Control :- ⇨ Control new version

⇨ Also known as revision control.

⇨ Each revision has timestamp & associate person making change.

Ex –



– change made to source is backed along with :-

(a) Who made with change?

(b) Why they made it?

(c) References to problem fixed

(d) Enhancements introduced by the change.

– A version control system is a repository of files with monitored acess.

⇨ It is a category of S/w tools that help a S/w team manage to source code over time.

⇨ It keeps track of every modification to the code in a special kind of database.

⇨ If a developer made mistake, then developers turns back the clock compare earlier version of code to fix mistake.

It is divided into four sub activities -

→ identifying new version (After previous version the new version version 2 with some modification).

→ Numbering scheme (It have following format

Version X Y Z

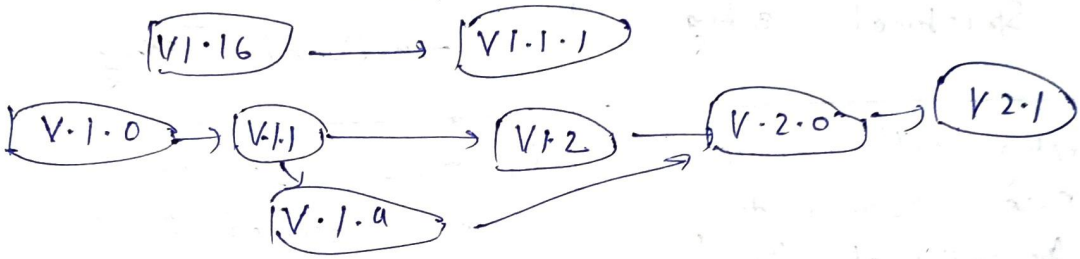
X → entire SCI

Y → Component of SCI

Z → section of component of SCI

→ Visibility (The version no will be visible either in frame or below title.).

→ Tracking (Best way keep track diff version is version evalution graph).



Verification & Validation :-

Verification comes after Verification ↗ At the time of de-development the software hits to s/w req. customer req.

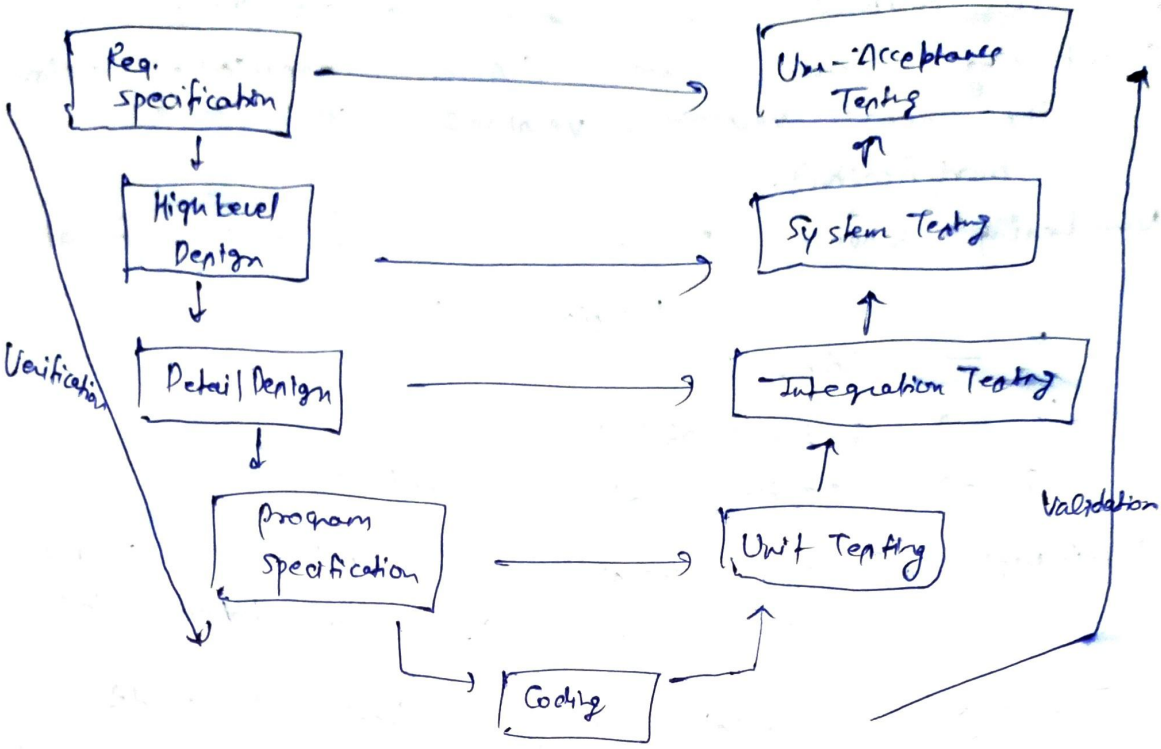
Validation

↓

when actual s/w made then actual testing on s/w that s/w works properly or not.

Quality assurance ← Verification (Before Validation) → Test testing team Validation (After Verification)

- find bugs early stage of development
- It can only find bugs that could not find at verification process
- It's checks that the development of s/w hits customer req. at the development.
- At the time development
- Are we building the s/w right?
- Static testing
- doesn't include the execution of code.
- Methods of verification are reviews, walk throughs, inspection & desk checking.
- It's checks the s/w specification met or not.
- It checks that the development of s/w hits customer req. after development.
- After development
- Are we build the right s/w?
- Dynamic testing
- includes the execution of code.
- Method for validation are Black-Box Testing, White Box Testing and non-functional Testing.
- It's check whether the s/w meets the req. & expectation of customer or not.

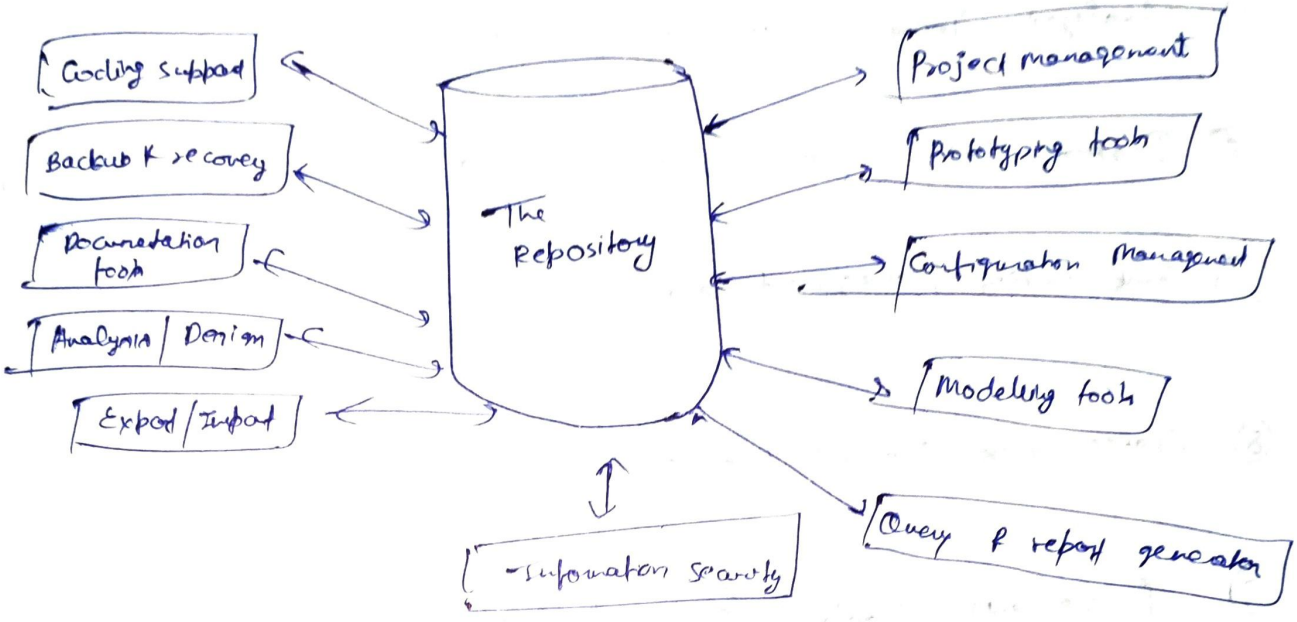


Structural Testing	functional Testing
(1) Testing will uncover error occured during the coding of the program.	(1) Testing will uncover error occured during the implementation & design specification.
(2) If concerned about both result & the process.	(2) If only concerned on result not on processing
(3) White Box Testing Types – (1) Mutation Testing (2) Data flow Testing (3) control flow Testing (4) slice-based Testing Control Testing Data flow Check code of statement over a control structure Slice complete code into small chunks then evaluate portion carefully	(2) Black Box Testing Types – (1) Unit (2) Integration (3) System (4) User Acceptance

Case Tools :

C → Computer S → Software
A → Aided E → Engineering

- It means development & maintenance of s/w projects with the help of various automated s/w tools.
- Case Tools are set of s/w application programs, which are used to automated SDLC activities.
- Used by project managers, analyst & engineers to develop s/w system.
- Tools → (1) Analytic tools (2) Design tools (3) Project management tools
  (4) Database management tools (5) Documentation Tools



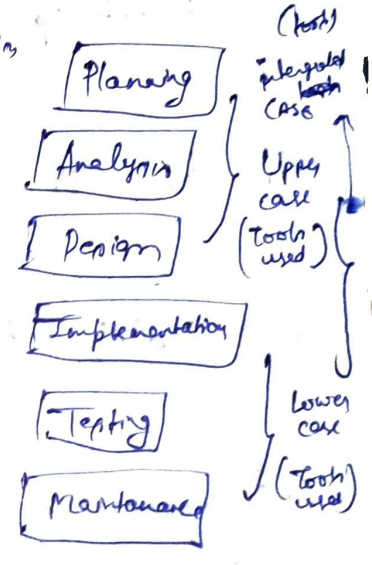
Case tools

Central Repository :- CASE Tools req. repository, which can have all information from which all tools access the data.

Upper Case Tools — (planning, analysis & design) tools are used

Lower Case Tools — Implementation, Testing, maintance

Integrated Case Tools : work in Upper case or lower case.
Or in all stages used tools.



Advantages :-

1. Cost of product reduces
2. quality improve
3. meet req. & design to customer req.
4. ~~high quality~~ security

Dis-advantages-

1. cost increase when we use more CASE tools
2. learning curve to use case tools increase time.
3. Tool mix problems

Case Tools Types

1. Diagram Tools :- (Structure / data & control flow) tools.
2. Process Modelling Tools :- (process modelling tools)
3. Project Management Tools :- (planning, cost, effort project) tools
4. Documentation Tools :- (documents made)
5. Analysis Tools :- (inconsistency checks)
6. Design Tools :- (block structure of S/w)
7. Configuration Management Tools :-
8. Change control Tools :- (Changes or modification) tools
9. Programming Tools :- (IDE to code)
10. Prototyping Tools :- prototype design tools
11. Web Development Tools :- designing web pages
12. Quality Assurance Tools :- Quality manage
13. Maintenance Tools :- modifications (automatic / manually)

Risk management :- A risk is uncertain event or condition that, if it occurs, has a +ve or -ve effect on a project objective.

" Risk Management plan is a document that a project manager control the risk & then uncertain event can be prevented. "

Ex :- Software → develop → the req. of that s/w in user environment (x). (Teacher).

~~Types~~ Strategies :-

(1) Reactive Management Strategies :- Risk has occured, we concern what would our strategy to minimize the damage or risk.

Ex :- When we drive a car, then we have an insurance

(2) Proactive Management Strategies :- those event even happening in the first place.

Ex :- while driving, seat belt, helmet etc.

Types —

(1) Business Risk :- When we build a project then it doesn't committed in business environment or out of budget.

(2) Technical Risk :- Concerned with quality, design, maintance risk.

(3) Project Risk :- Concerned with cost, resources, schedule customer related issues.

Example :- Teacher.