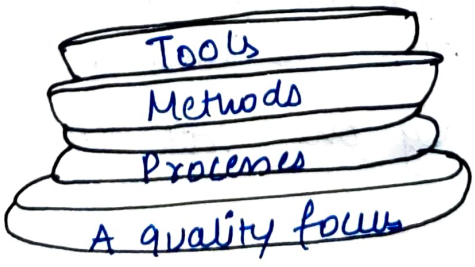
Introduction

→ **Software**:- (1) Instructions (computer programs) that when executed provide desired features, functions and performance
(2) data structures that manipulate information adequately.
(3) descriptive info. (documentation) that describes the operation and use of the programs.

* **Engineering**:- is the
Requirement analysis
design
construction
verification and
management of technical or social entities.

→ **Software Engineering**:
→ The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is application of Engineering to the software.

> Software Engineering is a Layered Technology



} Various layers

→ Total quality management fosters a continuous process improvement that ultimately leads to the development

of increasingly more effective approaches to the Software Engineering.

→ Quality is inverse of number of errors and is the bedrock that supports Software Engineering.

(2) :- The foundation for Software Engineering is the process layer

→ Process defines a framework that must be established for effective delivery of Software engineering technology

→ The Software process forms the basis of management control and establishes the context in which technical methods are applied, work products are produced, milestones are established, quality is ensured and change is properly managed.

* Software engineering methods provide the technical howtos for building Software.

→ Includes modeling activities and other descriptive techniques.

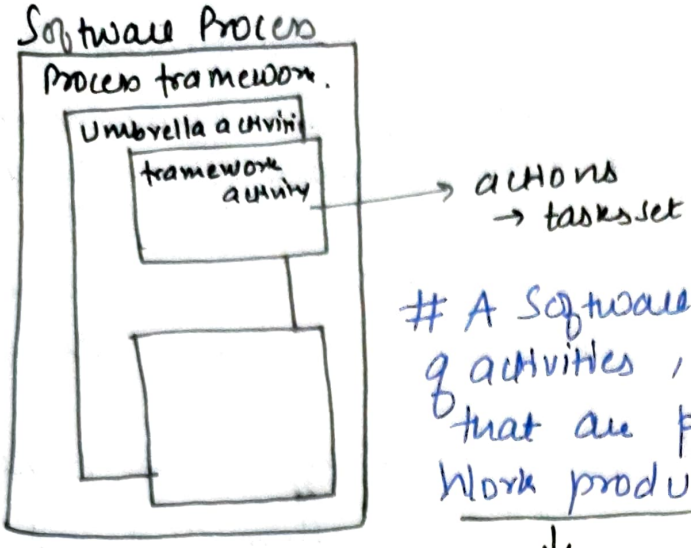
→ Methods: broad array of tasks that include communication, requirement analysis, design modeling program construction debugging, testing.

* Software Engineering tools to provide automated or semiautomated for the process and the methods. When tools are integrated so that information created by one tool can be used by another.

CASE :- Computer aided Software Engineering

→ System for the support of Software development

* Software process



A Software process is a collection of activities, actions and tasks that are performed when a Work product is created.

↓

intemediate output of each stage (2)

Activity :- Strives to achieve a broad objective that is applied regardless of the application domain, size of project, complexity of effort.

Actions :- encompasses a set of tasks that produces a major work product (architectual model).

Task :- focuses on a small, but well defined objective that produces a tangible outcome.

. Process framework

A process framework establishes the foundation for a complete Software Engineering process by identifying a small number of framework activities that are applicable to all software projects regardless of their size or complexity.

* Generic Activities of a process framework :-

① Communication : To gather requirements that help in defining software features and funcns with clients.

(2) Planning: prepare a map for software project
→ defines software engineering work by describing technical tasks to be conducted, risk management, resources that'll be required, work products to be produced and work schedule.

(3) Modeling:
→ creating models to better understand software requirements and design that will achieve those requirements.

(4) Construction: What you design must be built
→ This activity combines code generation and testing required to uncover errors in the code.

(5) Deployment: Delivery of product and evaluation of delivered product.

Umbrella Activities

→ Activities which are applied throughout a software project and help a software team manage and control progress, quality ~~and~~, change and risk.

→ Umbrella Activities Include:-

(1) Software project tracking and control allows software team to assess progress and take actions to maintain schedule.

(2) Risk Management Assess the Risks that may affect the outcome of the project or quality of project.

(3) Software quality assurance defines and conducts activities to ensure software quality.

5) Technical Reviews: aims SE work products in an effort to uncover and remove errors.

6) Measurement: defines and collects process, project and product measures and assist them to deliver project as per stakeholders need.

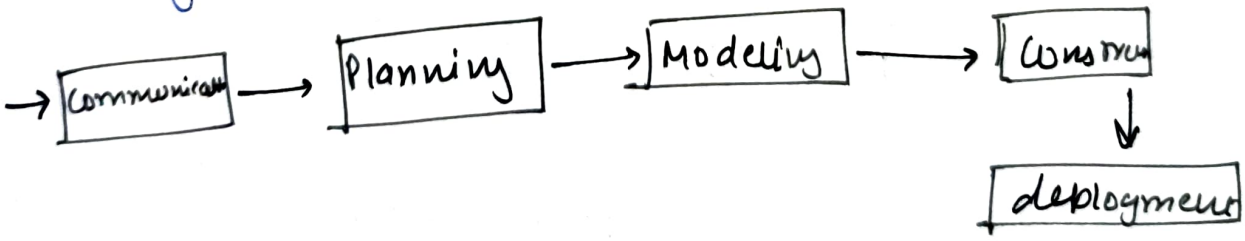
7) Software Configuration Management: manages the effects of change throughout the software process.

8) Reusability Management: defines criteria for work product reuse and achieve reusability components.

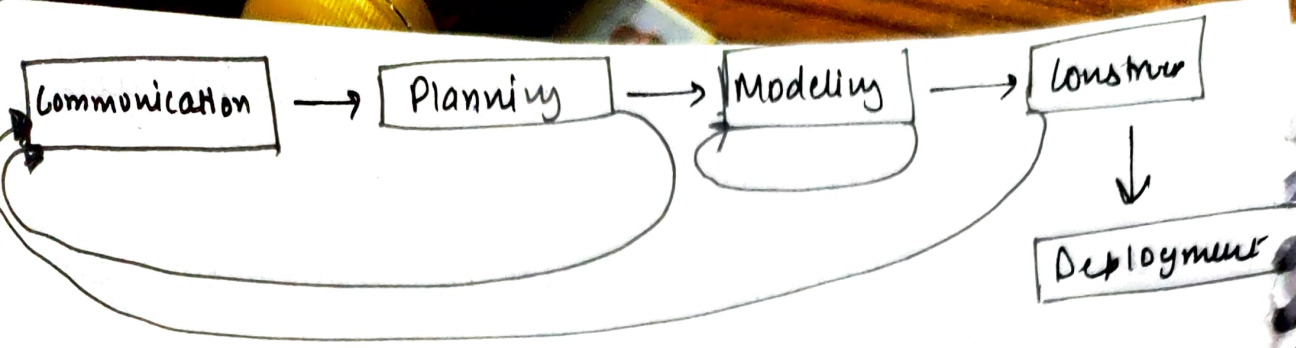
9) Work product preparation and production:
→ includes activities needed to create work products, such as models, docs, logs, forms.

: Process flow describes how the framework activities and the actions and tasks that occur within each framework activity are organised w.r.t sequence and time.

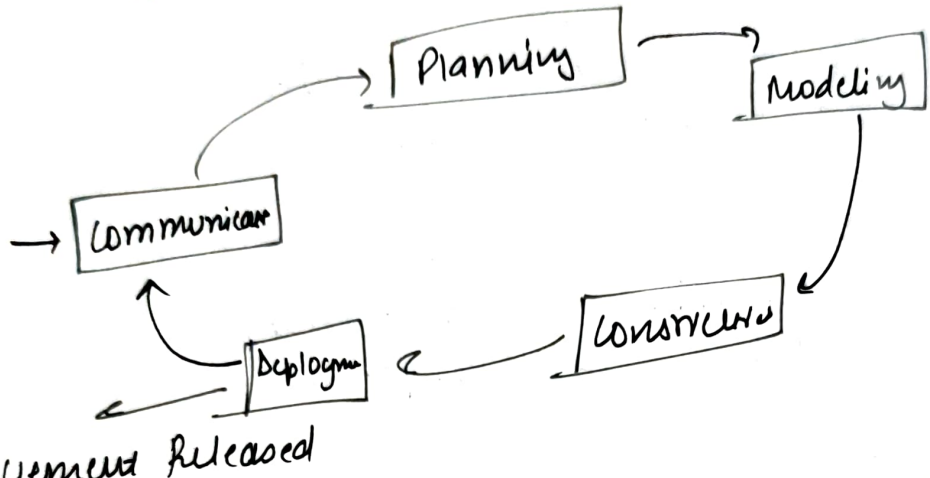
1) Linear process flow
→ executes the framework activities in sequence beginning at communication and ending at deployment



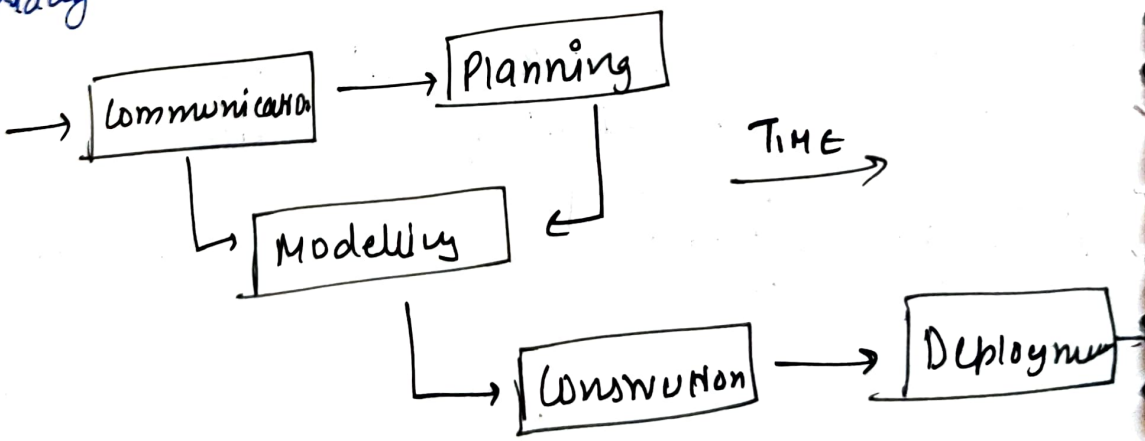
2) Iterative process flow
→ repeats one or more activities before beginning the next



Evolutionary:- executes the activities in circular manner
→ Increments the process set



Parallel process flow:- executes one or more activities in paralled with other activities.
→ It execute a activity before one activity has already completed execution.



framework activity changes the the nature of project changes

Prescriptive Process Model

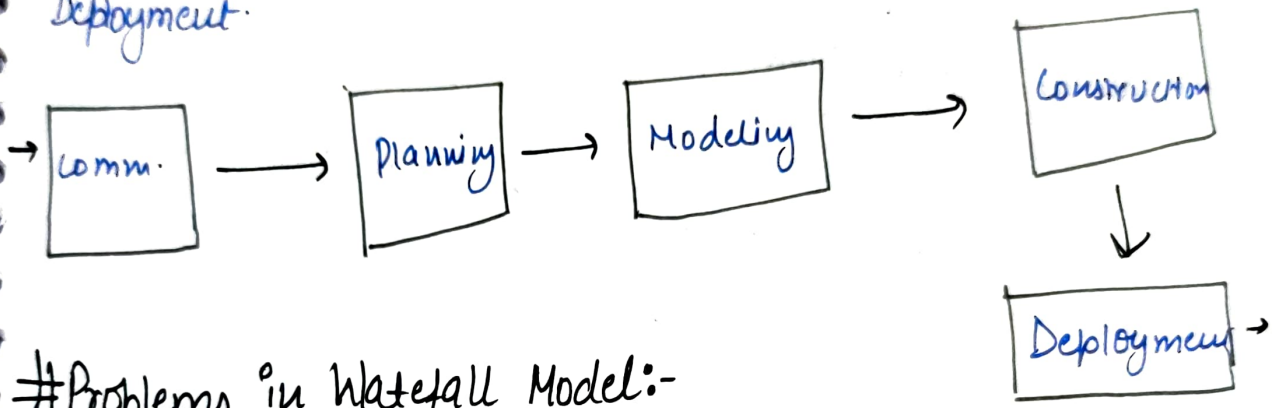
→ The prescriptive process model prescribes a set of process elements - framework activities, Software Engineering actions, tasks, work products, quality assurance and change control mechanism for each project.

→ It also prescribes a process flow - that is the manner in which the process elements are interelated to one another.

① Waterfall / Sequential Model:-

→ also called classic life cycle.

→ Suggests a systematic, sequential approach to Software development that begins at communication and ends at Deployment.



#Problems in Waterfall Model:-

→ Real projects rarely follow the linear approach.

→ Although linear model can accomodate iteration but indirectly.

→ Often difficult for the customer state all requirements explicitly.

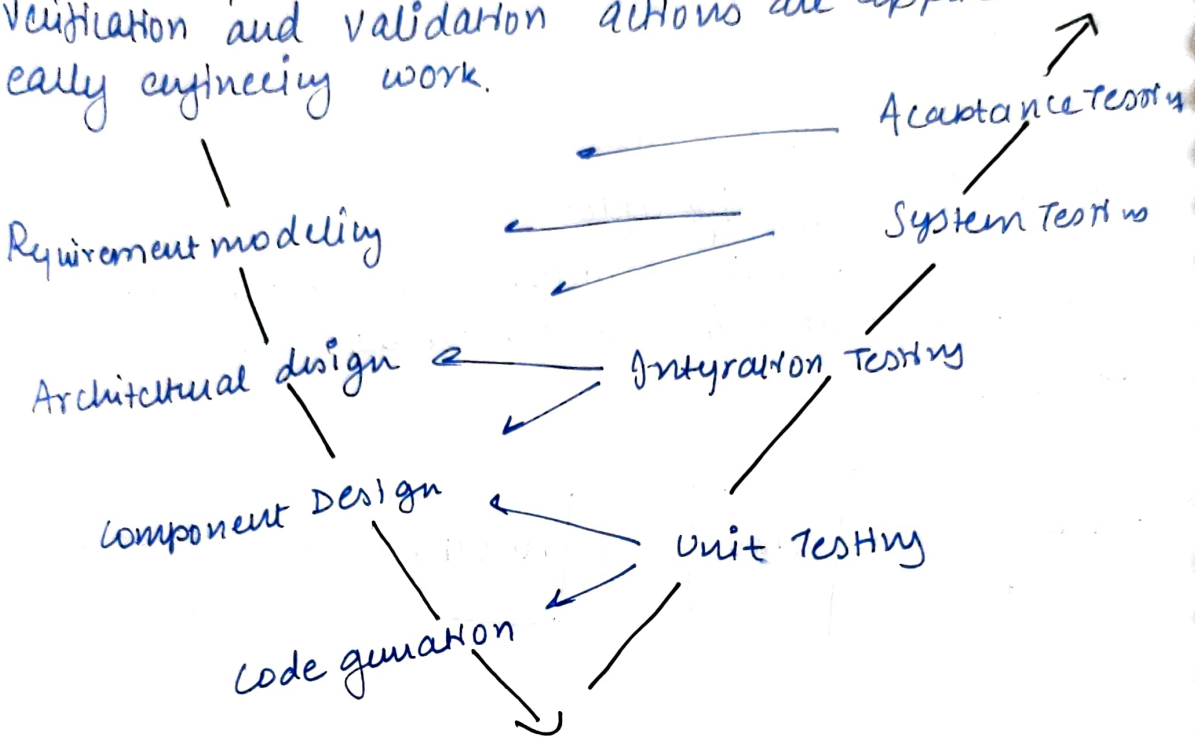
→ Customer must have patience. A working version of the program will be available at the time of Deployment only.

→ Linear nature leads to blocking state in which some project members must wait for other members to complete

V-Model

→ A variation of waterfall model, which depicts the relationship of quality assurance actions to be taken the actions associated with communication and early construction activities

→ Vmodel provides a way of visualizing how verification and validation actions are applied to early engineering work.



Incremental Model

→ Instead of delivering the entire system as a single delivery, the development and delivery is broken down into increments in which increments delivers a part of req. functionality.

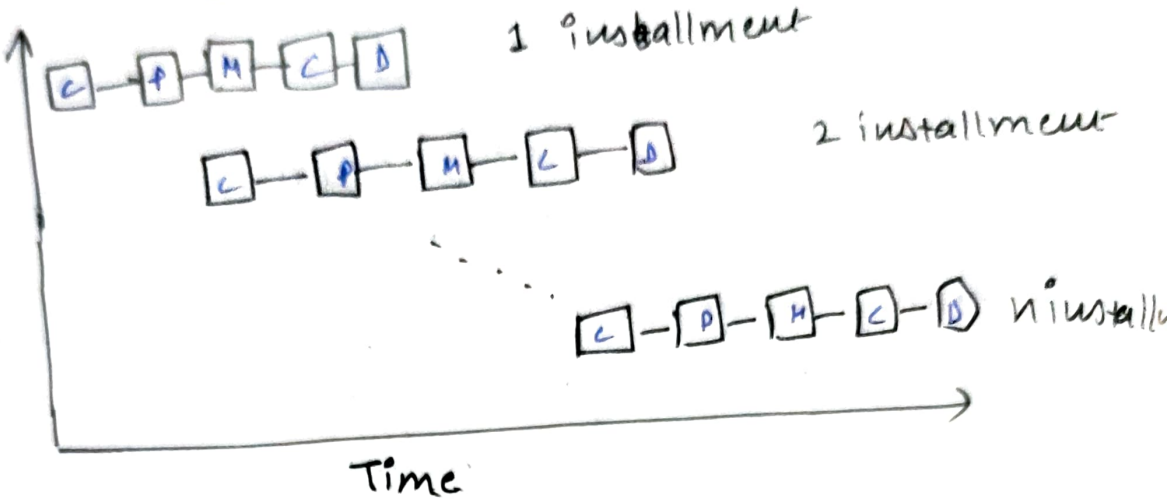
→ User requirements are prioritised and higher priority is included in early increments.

→ Once the development has started the requirements are frozen.

→ Early increments acts a prototype to help explain elicit requirements for later increments.

→ Lower Risk of overall project failure

→ Highest priority system services tend to receive the most testing.



Evolutionary Process Model

→ Produces more complete version of software with Each iteration.

→ Buisness and product requirements often change as development proceeds making linear path unrealistic

→ It is being designed to accomodate a product that grows or changes.

→ Prototyping Model and Spiral Modeling.

① Prototyping [no prototype/specifications known prior]

→ It begins with communication

→ Prototype iteration is planned quickly and modeling occurs.

→ Quick design focuses on a representation of those

aspects which will be visible to end users.

→ quick design leads to construction of prototype

which is deployed and evaluated by stakeholders who provide feedback that is used to further refine requirements

→ Gather Requirements

→ Developer and Customer define overall objectives, identify areas needing more investigation.

→ Use existing program fragments, program generators to throw together working version

→ Prototype evaluated and requirements refined.

→ Iteration occurs at the prototype to satisfy the Customer and developer.

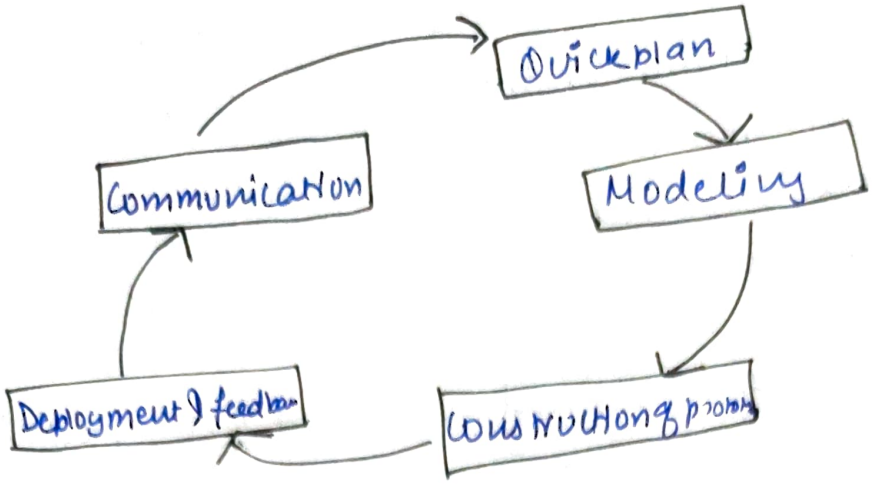
→ then throw away the prototype and rebuild system to high quality.

→ alternatively we can have ~~alternative~~ evolutionary prototyping - start with understood requirements

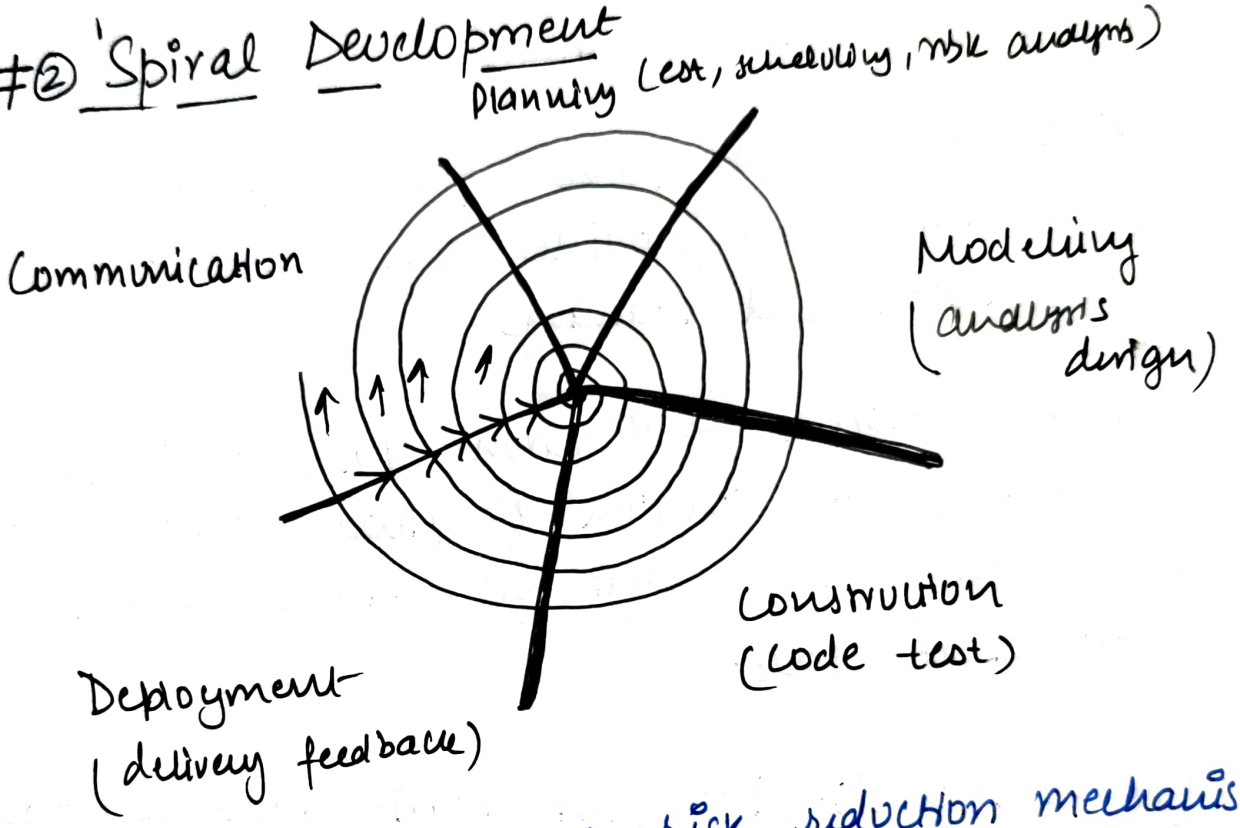
Problems:- Customer may want to stay on to the first version and want them fix than rebuild.

→ Developer may become comfortable with compromise and forget why are they inappropriate.

→



#② Spiral Development



→ It uses prototyping as a risk reduction mechanism but enables to apply the prototyping approach at any stage in evolution of the product.

→ Process is represented as a spiral rather than a sequence of activities with backtracking.

→ Spiral Model is a realistic approach to the development of large scale systems and software.

Customer Communication: tasks required to establish effective communication b/w developer and customer.

Planning: task required to define resources, timelines and other project related information.

Risk Analysis: Tasks required to asses both technical (code) and management risk. (resources)

Engineering: tasks required to build one or more representations of the application.

Construction and Release: Tasks required to construct, test, install and provide user support. (documentation and Training)

Deployment / Customer Evaluation: Tasks required to get feedback on evolution of software representations created during the Engineering stage and implemented at deployment stage

Spirals in Model:- The first circuit might result in development of a product specification.

→ Subsequent passes may develop a prototype and progressively more sophisticated versions of software.

→ Each pass through the planning region results in adjustment to project plan.

In addition the project Manager adjusts the planned number of iterations required to complete the software.

Agile Process:-

- Agility means quick to respond.
- Agility is dynamic, content specific, aggressively change embracing and growth oriented.
- Agile process is quick to respond and should respond to followings:

1) It is difficult to predict in advance which software requirement will change and which will persist. It is also difficult to predict the customer priority changes.

2) For many types it is difficult to predict how much design is necessary before construction is used to proove the design.

3) Analysis, Design, construction and testing are not as predictable.

CHARACTERISTICS :-

→ Agile process must be adaptable, Unpredictibility can be resolved by adaptibility.

→ Agile process should adapt incrementally

→ Iterative approach enables the customer to evaluate the software increments regularly, provide necessary feedback to the software team and influence the process adaptions that are made to accomodate changes.

Agility Principles

The following are the principles of agility:-

→

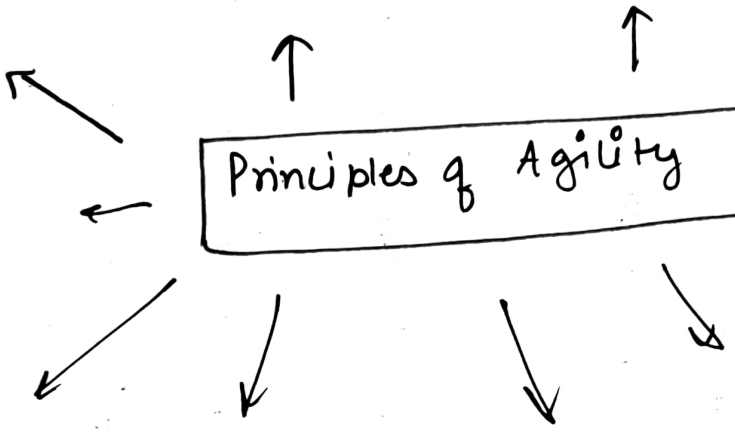
1) Highest priority is to satisfy the customer through early and continous delivery of valuable Software.

(2) Welcome changing requirements, even in late development. Agile process harness change for customer's advantage.

(3) Deliver Software frequently over short time periods

(4) Business People and developer must work daily throughout the project.

(5) Build projects around motivated individuals.



6) The most effective way of conveying the information is face to face conversation

(12) At regular intervals the team reflects on how to become more effective then tunes and adjusts its behaviour accordingly

7 Working Software is the primary measure of progress

(8) Agile process Maintain Sustainable development throughout

(9) Continous attention to technical excellence and good design enhances agility

(10) Simplicity — art of maximizing the work not done

(11) Best Architecture emerge from self-organizing team

SCRUM

→ It is an Agile Software Development Method

→ Frame work Activities include:
- Requirement analysis, design evolution and delivery
- Work Tasks are called Sprint. The work conducted within a sprint and number of sprints required for each framework activity depends upon product size and complexity.

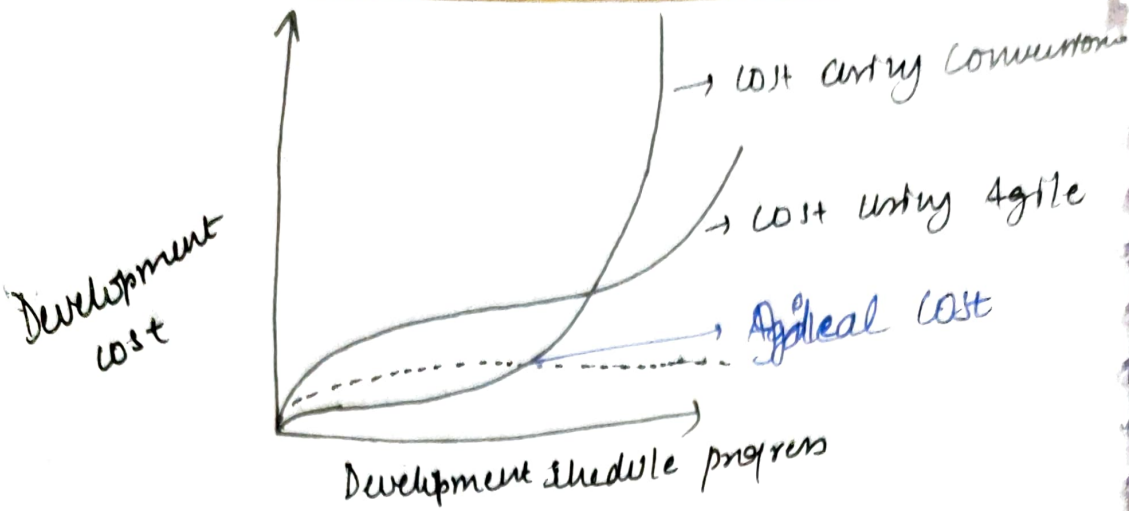
- SCRUM incorporates a set of process patterns that emphasize project priorities, compartmentalized work units, communication and frequent customer feedback.

Scrum Process flows:

Backlog:- a prioritized list of project requirements or features that provide a business value for the customer. Items can be added to backlog at any time. Product manager assesses the backlog and update priorities as required.

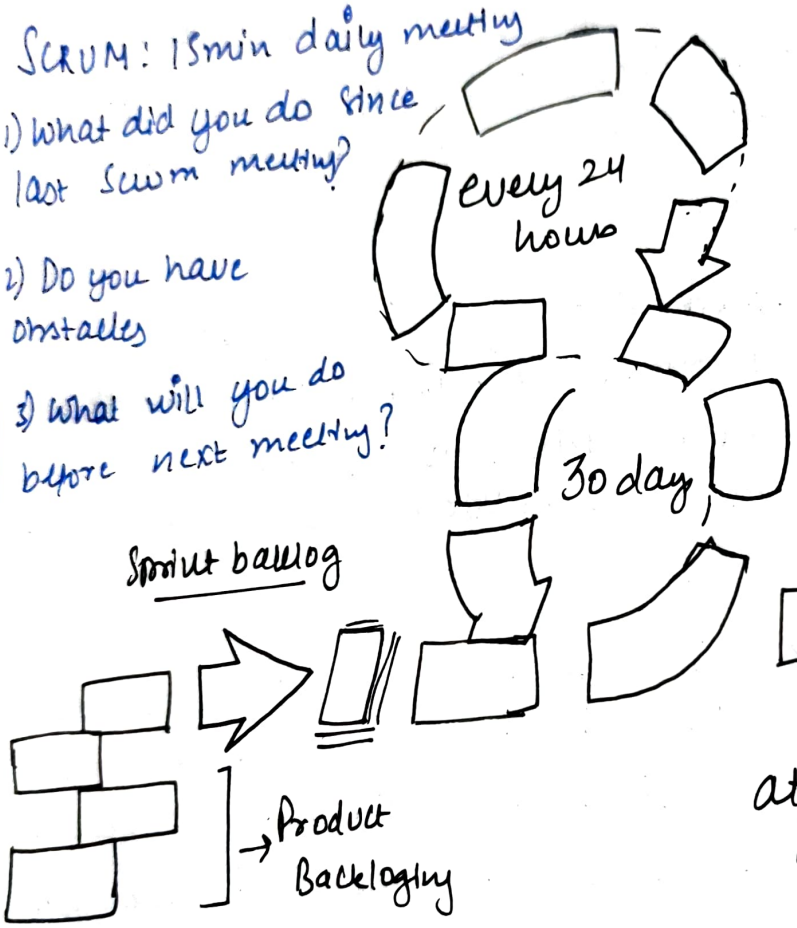
Sprints:-
- consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time box 10. (Changes (backlog work items) are not introduced during the sprint. Hence sprint allows to work in stable environment but in shorter spans.
- Scrum meetings are short
- Scrum master leads the meetings

Demos:- deliver software increment to the customer so that implemented functionality can be demonstrated and evaluated by customer.



SCRUM: 15min daily meeting

1) What did you do since last Scrum meeting?
2) Do you have obstacles
3) What will you do before next meeting?



New function daily demonstrated at the end of Sprint

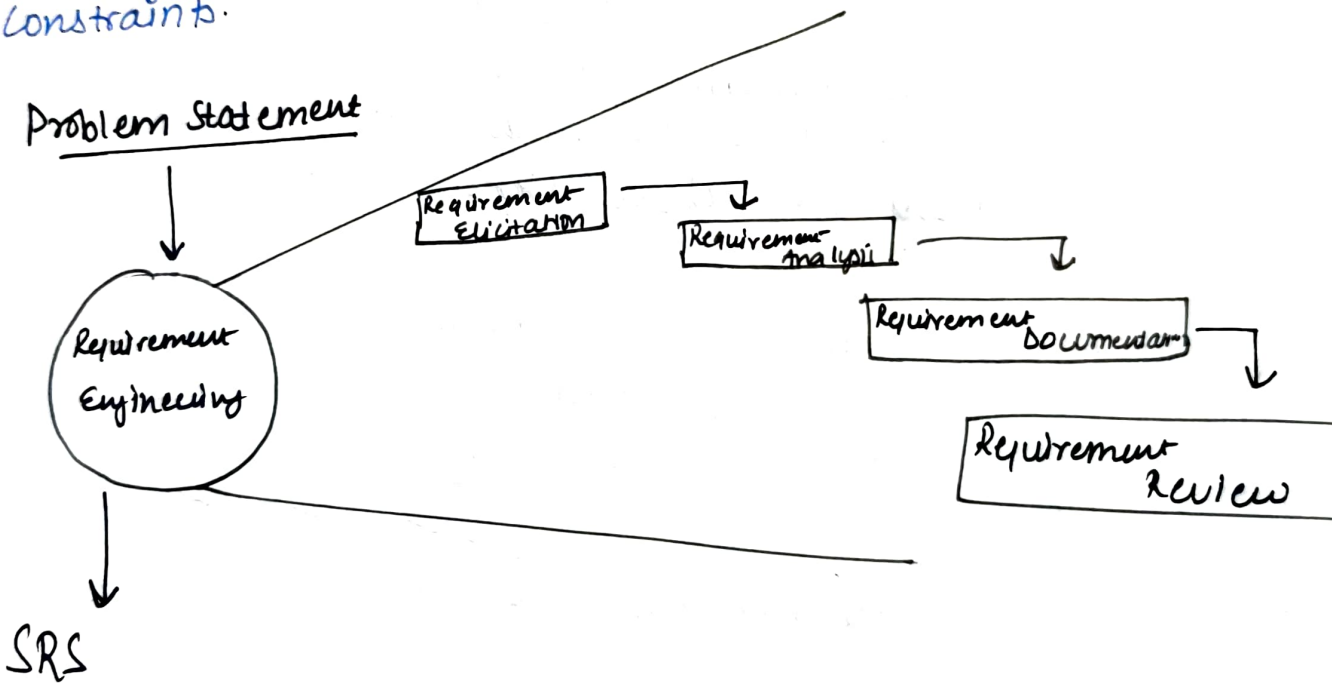
Software Requirements Analysis and Specifications.

→ Hardest part of building a software is "what to build" to decide

→ Requirements describe what of a system and how

→ Requirement Engineering produces one large document that describes of What the system will do without telling how it will do.

Requirement Engineering:- is the disciplined application of proven principles, methods, tools and notation to describe a proposed system's intended behaviour and its associated constraints.



Requirement Engineering is one of the most crucial Activity in this creation process. Without well written requirement specification —

→ developer donot know what to build
→ customer donot know what to expect.
→ no way to validate the system.

(1) Requirements Elicitation:- Gathering of Requirements

→ Here requirements are identified with the help of customer and existing system processes if available

(2) Requirements Analysis :-
Analysis of Requirements start with requirement elicitation. The requirements are analysed in order to identify inconsistencies, defects, omissions etc.

(3) Requirements Documentation: This is the end product of requirement elicitation and analysis. The documentation is very crucial as it will be the foundation for design of software. Also called SRS document

(4) Requirements Review: It is carried out to improve the quality of SRS. It may be also called as requirement verification for maximum benefits review and verification should be treated as a continous activity that is incorporated along the above steps.

The Software requirement specification should be internally consistent, consistent with document correct and complete with respect to satisfying needs.

SRS is a contract between the developer and customer.

Types of Requirements:-

Known: Something a stakeholder believes to be implemented.

Unknown: Forgotten by stakeholder because they are not needed right now or needed by other Stakeholder

(iii) Unknown requirement : Stakeholder may not be able to think about new requirement due to limited domain knowledge.

Stakeholder : Who may have some direct or indirect influence on system requirement.

They are further classified into:-

Functional : functional requirement are mostly those requirements which describe what software has to do. often called as product features.

Non functional : quality requirements that stipulate how well the software does, what it has to do. It is important to users to include specification of desired performance, availability, reliability, flexibility, usability.

System Requirements:- are architectural, component naming compatibility, interfacability, upgradibility etc. Constraints can come from user or organization may be functional or non functional.

Requirement ~~Analysis~~ Elicitation :-

It is perhaps most difficult, most critical most error prone and most communication intensive aspect of software development. Elicitation can succeed only through effective customer-developer partnership.

USE CASE APPROACH :-

→ developers use this methadology in order to gather requirements

→ The focus on what user needs to do with the system is much more powerful than traditional elicitation approach of asking the user what they want the system to do.

→ This approach is a combination of text and pictures. In order to improve the understanding of requirements.

→ Use case give functional view of the system and discribe what of a system and not how.

→ Use case diagrams are graphical representations that may be decomposed into further levels of abstraion

Use cases are structured outline or templates for discription of user requirements.

Use case scenarios are unstructured discription of user requirement.

Components of Use case diagram

(1) ACTOR : represented by stick figure.

→ agent or external agent that lies outside the system model but interacts with it in some way.

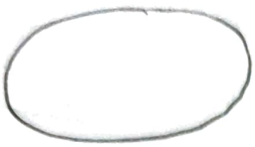
→ Actor may be a person, machine or external info system that is external to the system.

Cockburn :- It distinguishes between primary and secondary actors.

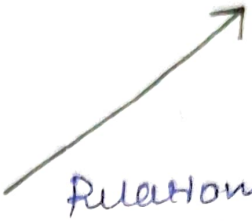
A primary goal is one having a goal requiring the anistance of the system.
A seconday actor is whom from which the system needs anistance.

Components of Use case diagram:-

Actor



Use case



Relationship between actors and use case and/or between the use cases.

Use case template

Introduction – Describe brief purpose of use case

Actors – List of actors that uses and participate in this use case

Pre conditions – condition that needs to be satisfied for use case to execute

Post conditions – After execution of use case different states of use case are defined here.

Flow of Events

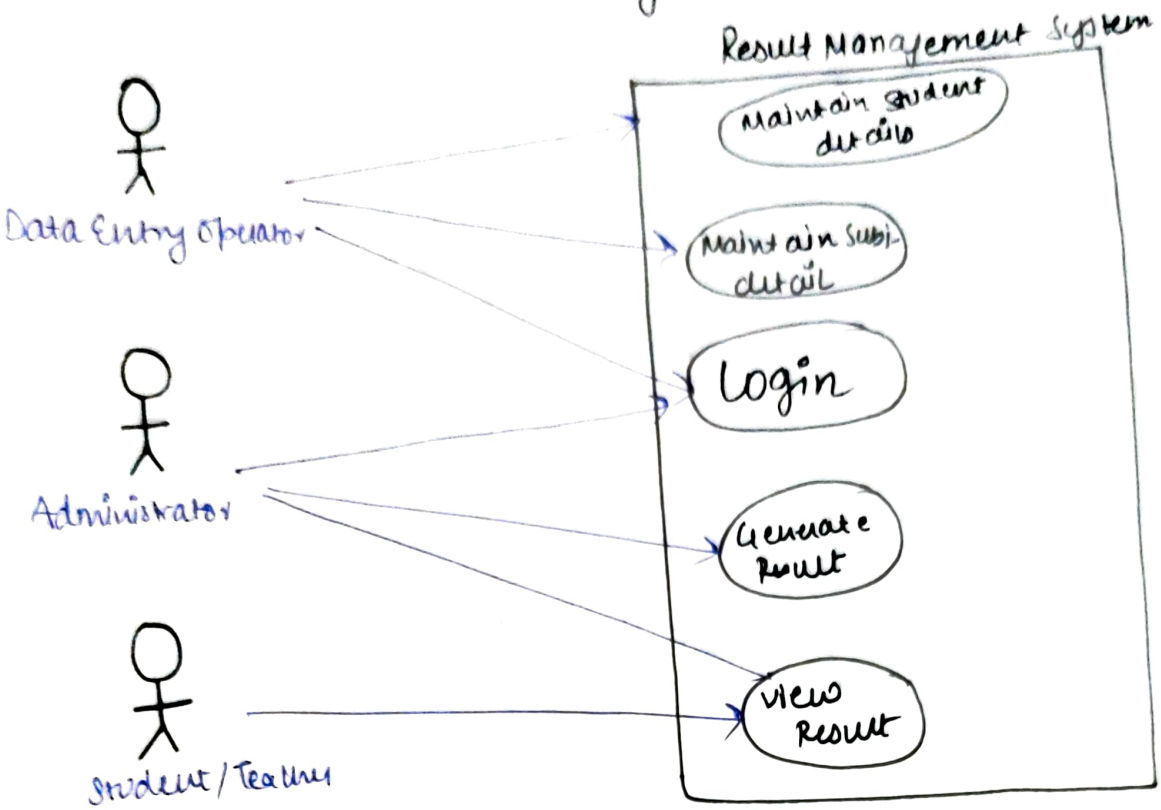
5.1 Basic flow – Use of primary events that occurs when use case is executed.

5.2 Alternative flow – Another posible flow in this use case.

Special Requirements – Buisness Rules for flows, Scenes and failure Scenarios to be described.

Related use case: A use case is related to some other use case or not

· Use Case Diagram ·



1. USE CASE FOR LOGIN:-

1.1 Introduction: This use case describes how a user logs into the Result Management System

1.2 Actors: (i) Data Entry Operator (DEO)
(ii) Administrator / Deputy Registrar.

1.3 Pre Conditions: None

1.4 Post Conditions: If use case is successful the actor is logged into the system. If not, the system state is unchanged.

1.5 Basic Flow: This use case starts when User wishes to login into the Result Management System.

(i) System requests that the actor enter his/her name and password.

(ii) The actor enters his/her name & password.

(iii) System validates name and password, if finds correct allow the actor to login into the system.

1.6 Alternate Flows: Invalid Name and password.
If in basic flow, the actor enters an invalid name and/or password the system displays an error message. The actor can choose to either return to beginning of the basic flow or cancel the login, at that point the usecase ends.

1.7 Special Requirements: None

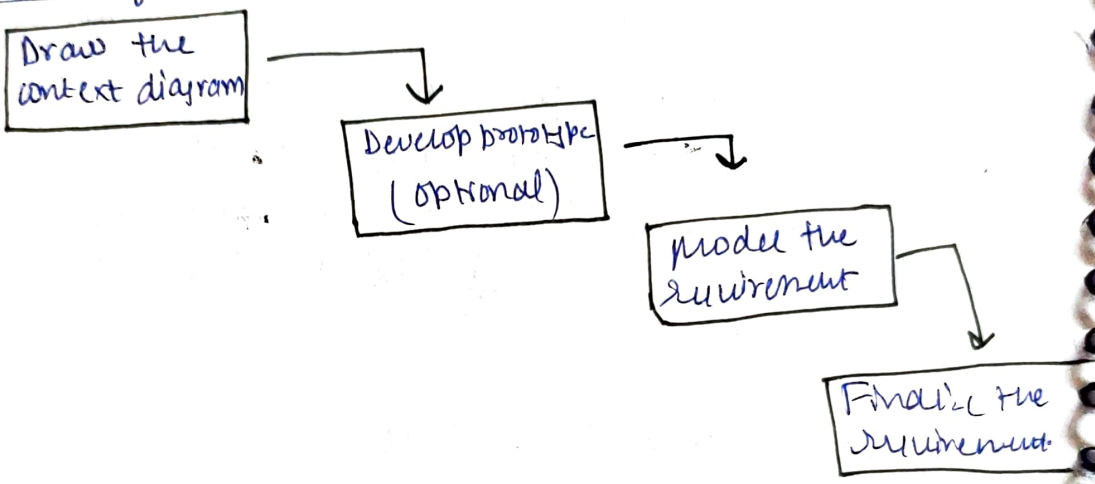
1.8 Use Case Relationship: None.

Requirement Analysis:-

→ Very essential and important activity after elicitation We analyze, refine, structure the gathered requirements in order to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system.

→ After completion of analysis it is expected that understand of the project may improve significantly.

Various Steps of Requirement Analysis:-

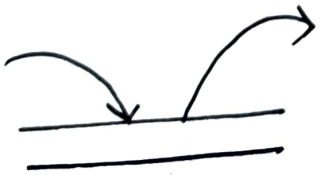


Data flow Diagrams

→ widely used for modeling the requirements
→ DFD shows the flow of data thro'g the system.
→ DFD is also known as a data flow graph or a bubble chart.
→ The system may be a company, an organisation, set of procedures, hardware system, software system.

★
→ All names in DFD must be unique. To make it easier to refer to items in DFD.
→ It is not a flow chart. It only represents the flow of data

Symbols in DFD:-

		function
	Data flow	Use to connect processes to each other, to sources to sinks, the overhead indicates the direction of data flow.
	Process	Performs some transformation of input data to yeild output data.
	Source or Sink of Data	A source of data input or a sink of system outputs.
	Data Store	A repository or storage of data overhead indicates the inputs and the outputs to the data store.

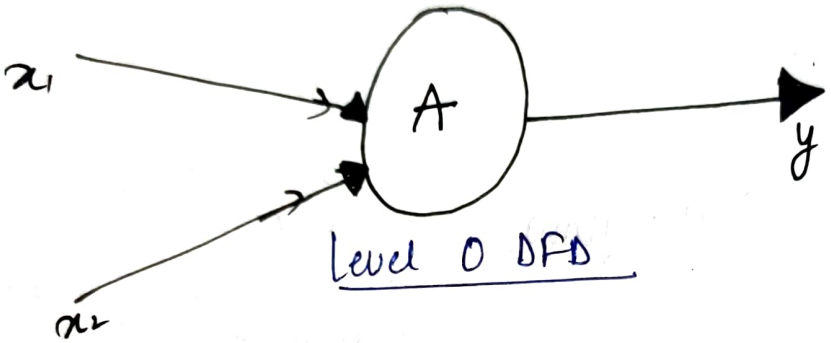
A circle or a bubble shows a process.

Levelling in DFD :-

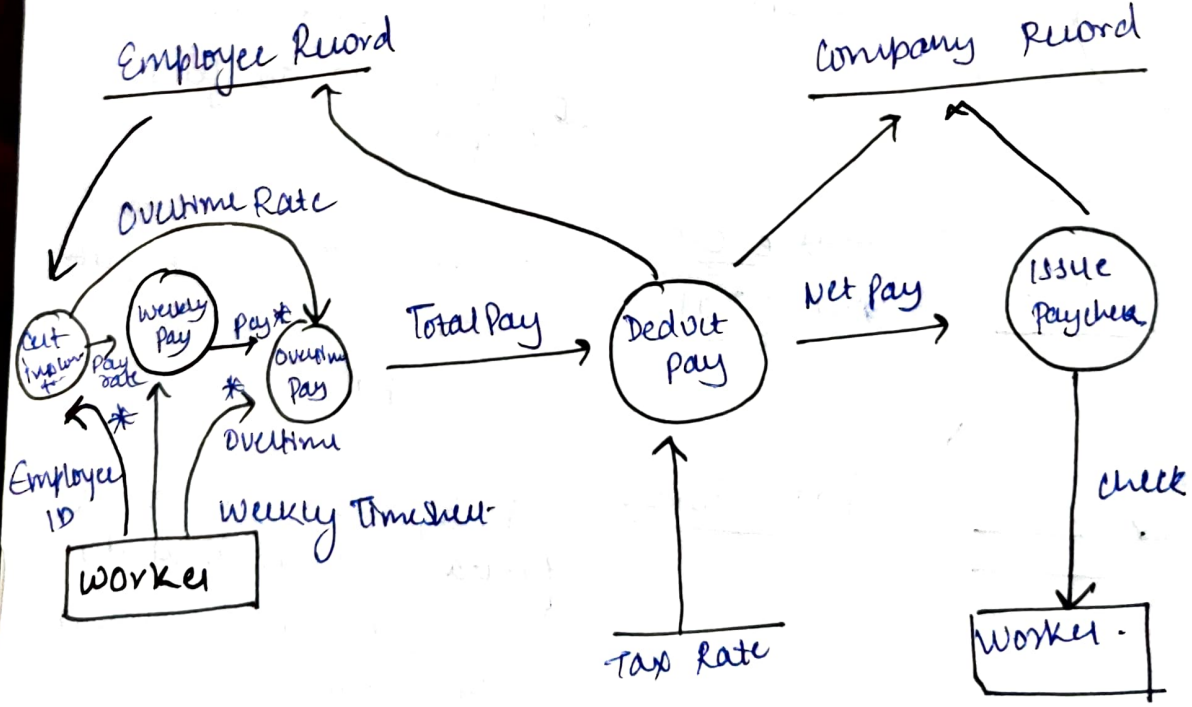
A dfd may be used to represent the system at any level of abstraction

→ DFD may be partioned into levels that represent increasing information flow and functional details.

At level 0 DFD: also called a fundamental system model or context diagram represents the entire software element as a single bubble with incoming and outgoing arrows representing input and output



Level 0 DFD



Level 1 DFD

Data Dictionaries

→ They are the repositories to store information about all data items defined in DFDs

Typical information stored:

Name of data item: Name should be self explanatory

Aliases: Name by which data item is called like D.E.O

Description: Textual detail of why data item is used.

Related data items: Relationship among data item.

Range of values: All possible values.

Data structure definition: If data is primitive then data structure captures physical structure of the data item If data is aggregate then it captures the composition of data items in term of other data items.

Data dictionary notations

$x = a + b$	x consist of a and b
$x = [a/b]$	x consist of a or b
$x = a$	x consist of optional data item a
$x = y\{a\}$	x has y or more occurrences of a
$x = \{a\}_2$	x has 2 or less occurrences of a
$x = y\{a\}z$	x consist of occurrences of a lying between y and z

Data dictionary can be used for → ordering listing of all data item, subset of data item

→ Design the software and test cases.

→ Find a data item name from a description.

For timesheet DFD

Weekly-timesheet = employee-name + id + [regular-hrs + overtimehrs]*

Pay_rate = [hourly | daily | weekly] + dollar-amt

Employeename = last + first + middle

id = digit + digit + digit + digit.

Requirement Documentation

This is the way of representing requirement in a consistent format.

SRS serves many purpose depending upon who is writing it

→ written by developer
→ written by customer.] → Serves as a contract

SRS Should:- Correctly define all requirement
→ not discribe any design details
→ not impose any additional constraints

SRS should be correct, Unambiguous, Complete, Consistent, Verifiable, Modifiable, Traceable.

Requirement Documentation:-

Correct:- An SRS is correct if and only if every requirement stated shall be there in the software.

Unambigous An SRS is unambigious if and only if every requirement stated has only one interpretation.

Complete: An SRS is complete if and only if it includes the following elements:

(i) Responses to both valid and invalid inputs.

(ii) All significant requirements whether related to functionality, performance, design constraints, attributes or external interfaces.

(iii) Full label and references to all figures, tables and diagrams in SRS and definition of all terms and unit of measure.

Consistent: An SRS is consistent if the requirement described is of no conflict.

Stability: If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement.

Verifiable:- An SRS is verifiable if and only if every requirement stated can be verified.

Modifiable: An SRS is modifiable if and only if its structure and style are such that can adapt to any change.

Organisation of SRS:

IEEE has published guidlines to Organize an SRS.

1 Introduction

1.1 Purpose: Describe capabilities in 1 line, constraints.

1.2 Scope: functionalities of system in brief

1.3 Definitions: Acronyms and abbreviations.

1.4 References

1.5 Overview: The describes the rest of requirements, interfaces, features.

2 Overall Description Explain the existing system

2.1 Product Perspective: Whether the system is independent or a part of a larger system. A block diagram showing the major components, interconnections etc

2.1.1 System Interfaces

2.1.2 User Interfaces

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

2.1.5 Communication Interfaces

2.1.6 Memory constraints

2.1.7 Operations

2.1.8 Site Adaption Requirement

2.2 Product functions: Summarize major function with roles

2.3 User characteristic: The technical skill required by user to use the system.

2.4 Constraints: Description of items that will limit developer options

2.5 Assumption for dependencies: Any change may affect the system.

2.6 Apportioning of requirements: Identify requirement that may be delayed until future version of system.

3 Specific Requirements:

This section will specify requirements to a level of detail to enable designers and testers.

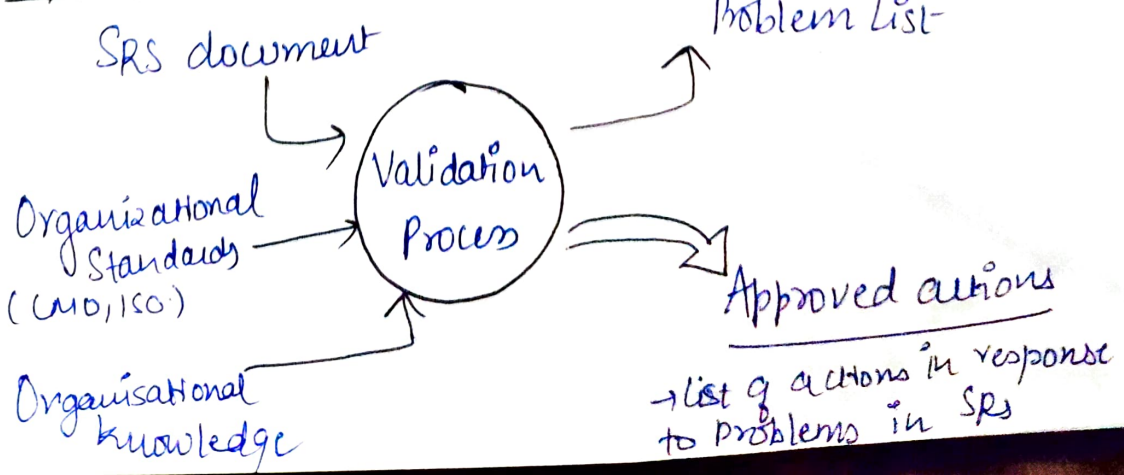
3.1 External Interfaces
3.2 Hardware and Software Interfaces
3.3 Functions
3.4 Logical Database
3.5 Design constraints
3.6 Software System attribute
3.7 Organization of Specific Requirement
3.8 Additional Comments

4 Change Management Process
5 Document Approval Approvers of SRS document
6 Supporting Information Indexes, TOC, Appendices.

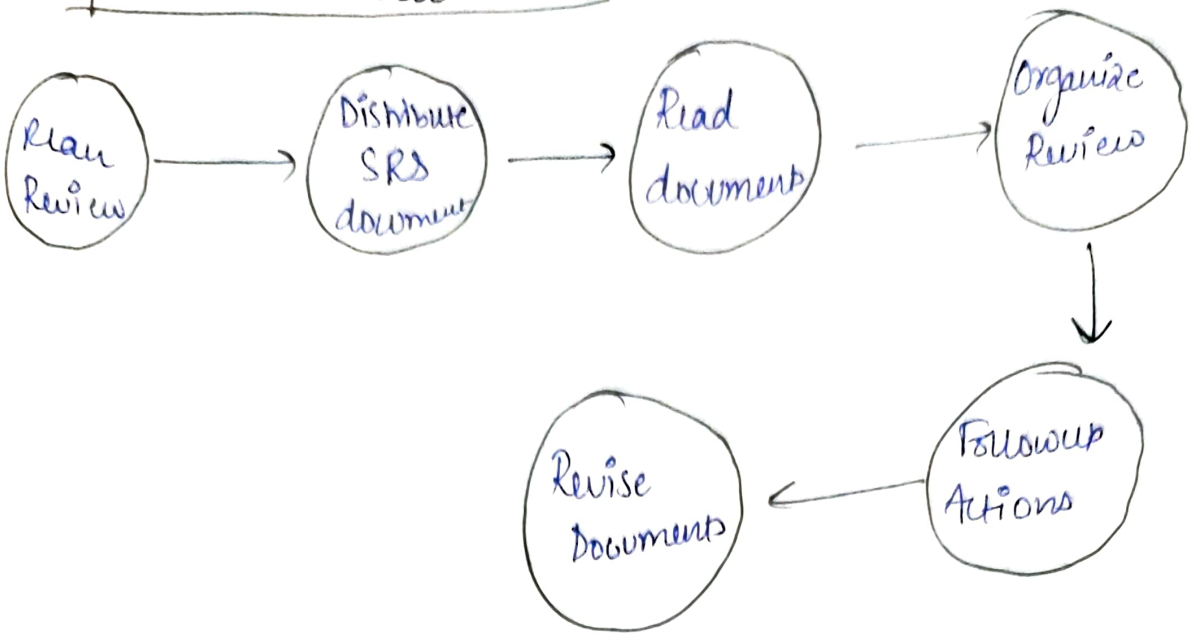
Requirements Validation

Check the SRS document for-
→ Completeness and Consistency
→ Conformance to Standards
→ Requirement conflicts
→ Technical errors
→ Ambiguous requirements.

Requirement Validation Process



- Requirement Review Process



Problem Action after the Review of SRS

Requirements clarification: Requirements may be badly expressed

Missing information → from stakeholders

Requirement conflicts
→ negotiate with stakeholders

Unrealistic Requirements
→ Stakeholders must be consulted

Security Issues
→ Review the system in accordance with Security standards

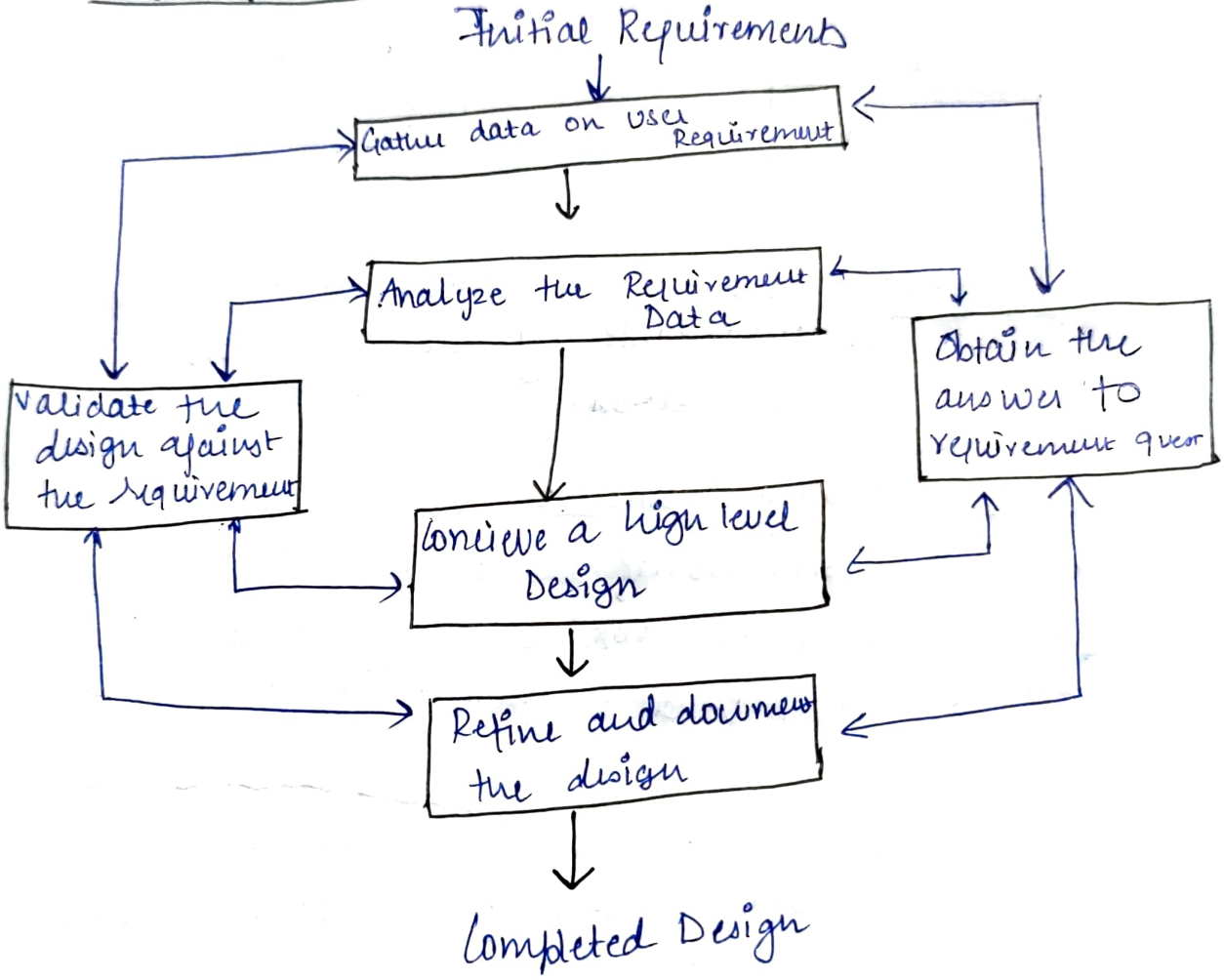
Software Design

→ SRS is the input to the design Process
↓
which tells us how a system works

→ Purpose of design phase is to produce a solution to a problem given in SRS document.

→ Designing Software System means determining how requirements are realized and result is a software design document.

Design framework :-



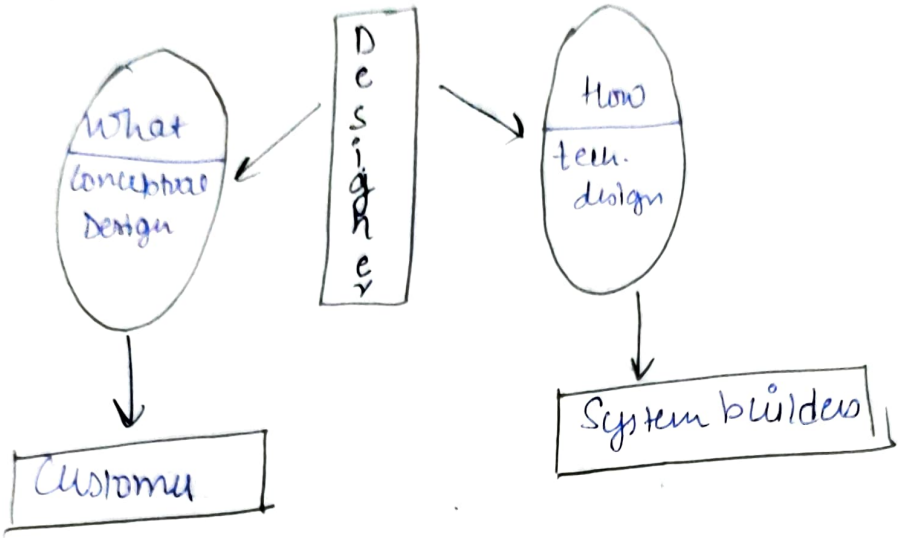
Process of Software Design transform the ideas into detailed implementation descriptions

Design Satisfy both → Customer → what sys. does
→ Programmer → How it does.

Design is a two part iterative Process.

(1) Conceptual Design describe the customers of what system will do.

(2) Technical design: Technical design allows system builders to understand hardware and software needed to solve customers problems.



Conceptual Design:- It describes the system in a language understandable by the customers and is independent of implementation.

It answers the following questions:-

Where will data come from? what will happen to data in system? How will system look to users? What choices will be offered to users? What is the timing of event?

Technical Design:- It describes Hardware configuration, Software needs, Communication interfaces, I/O of the system, Network architecture and any other thing that translates the requirements into a Solution to the customer's problem.

Objectives of Design:-

→ Design fills the gap between requirements specification and coding.
→ Taking specifications, deciding how program will be organised and methods it will use in sufficient details as to be directly codiable

The design needs to be:-

Correct and complete
understandable
At right level
Maintainable

Software Design iteratively goes through a number of different phases. The starting point ~~design~~ in Informal design is refined by adding information to make it consistent and complete.

Why design is Important:-

A good design is key to successful product as well as a well designed system is easy to implement, understandable, and reliable.
→ The software design should contain a complete, accurate and precise solution to a problem to ensure its quality implementation.

Characteristics:

- The design must implement all explicit requirements contained in Analysis Model and must accomodate implicit requirements.
- Design must be readable, understandable test and subsequently support the system.
- Design should provide a complete picture of software addressing Data,

functional and Behavioural domain from an implementation prospective

⇒ Modularity :- There are many definition of modules from fortan subroutine, java class, C++ class, java package etc.

→ A modular system consist of well defined units (managers) with well defined systems interfaces among the units.

Properties of Modular System:-

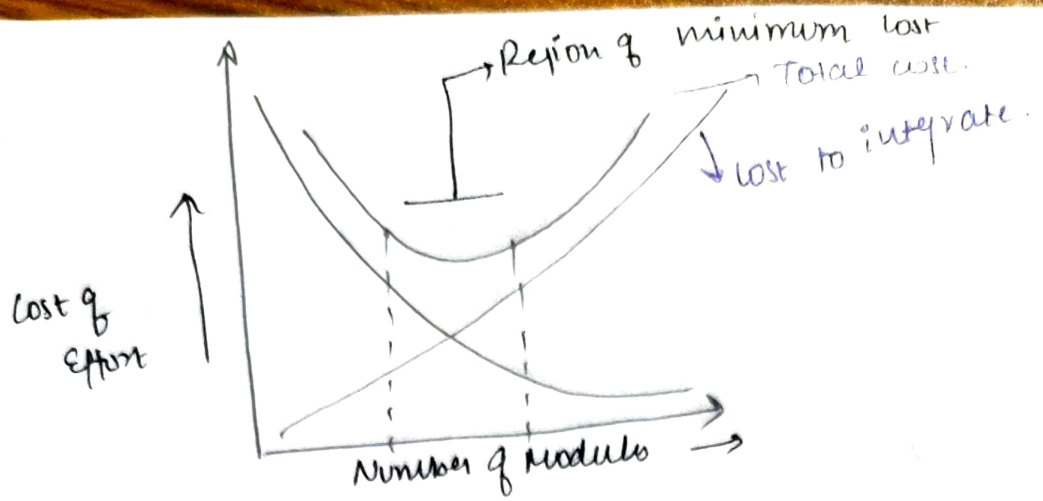
→ Each module is well defined subsystem useful in other application.
→ Each module has a single well defined purpose
→ Module can be seperatly compiled and stored in a library.
→ Modules can use other Modules
→ It should be easier to use than to build.
→ Modules should be simpler from outside than from inside

→ A system is considered as modular if each component could be implemented seperately and a change to one has minimal effect on other components.

→ Modularity enhances design clarity, that in turn eases implementation debugging, testing, documenting maintenance of software product.

To what extent can we modularise

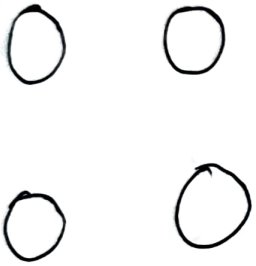
As number of module grows the effort associated with integrating the module also grows so under and over modularity should be avoided.



Module Coupling

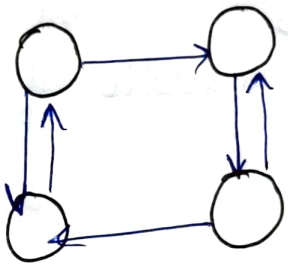
Coupling is the measure of the degree of interdependence between modules. It is measured by the number of interconnection between the modules.

Coupling increases as the number of calls between module increase or the amount of shared data increases.



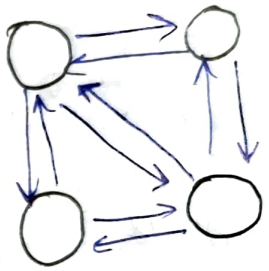
Uncoupled

(no dependencies)



Loosely coupled

(Some dependencies)



Highly coupled

(many dependencies)

High coupling means more errors. How coupling can be achieved by:-

- Controlling the number of parameters passed amongst modules
- Avoid passing undesired data to calling module
- Maintain parent child relationship between calling and called module
- Pass data not the control information (flag variables

Types of coupling:- (1) Data coupling (best) Only communicate through the data only.

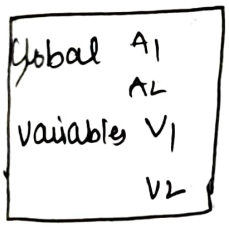
(2) Stamp coupling:- When complete data structure is passed from one module to another.

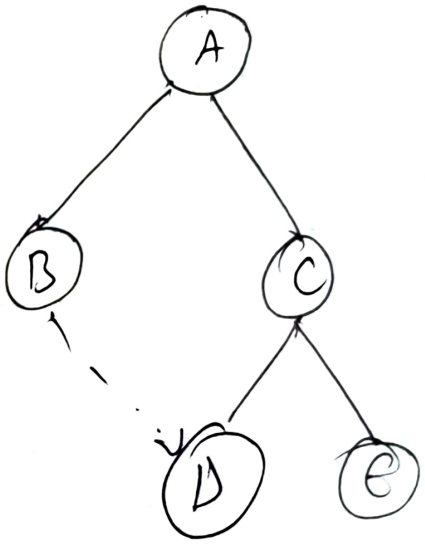
(3) Control coupling:- If two moduls communicate by passing controling information i.e usualy through the flag variables.

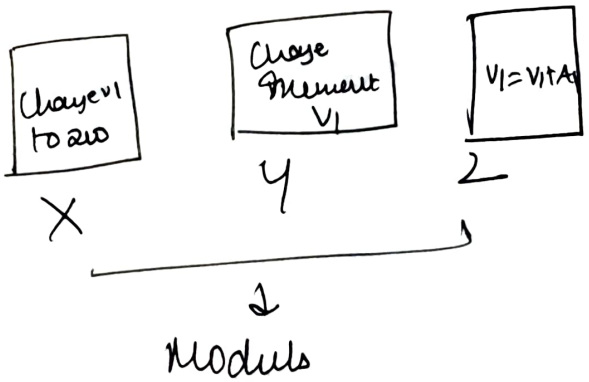
(4) External coupling:- When module A has dependency over hardware or related to communication tools or devices.

(5) Common coupling:- By exchanging data. Global data areas are commonly found in programs Layout changes in them will affect all the moduls.

(6) Content coupling (worst) :- When module A changes data of module B or when control is passed from one module to the middle of another.







Content Coupling.

Common Coupling:

Module Cohesion:- Cohesion is a measure of the degree to which the elements of a module are functionally related.

Cohesion = Strength of relations within modules

Types of Cohesion:- (Best) ↑ (worst)

Functional Cohesion: A and B are part of a single functional task.

Sequential Cohesion:- Modules A output some data which form input to B. This is the reason for them to be contained in same procedure.

Communicational Cohesion: If they operate on same input data or contribute to same data output.

Procedural Cohesion: Occurs in modules whose instructions although accomplish different task yet have been combined. because they must occur in certain order.

Temporal Cohesion: They all are related by the fact that all task that are related by the fact that all task must be executed in same time span.

Logical Cohesion: Contains instructions that appear to be related because they fall into same logical class of functions.

Coincidental Cohesion: that contain instructions that have little or no relationship to another.

Relationship between Cohesion and Coupling :-

→ High cohesion low coupling

↓

Design goals for Engineer.

→ Strategy of Design:-

- good Strategy is to organise the modules in such a way that are easier to develop and later to change

Structured design technique helps developers to deal with size and complexity of programs.

Analyst create instructions for the developers about how should code be written and how pieces of code should fit together to form a program. It is important for two reasons:

First if any preexisting code needs to organized and pieced together

Second it is still for project team to have to write some code that support the application logic of the system.

Bottom Up design:-

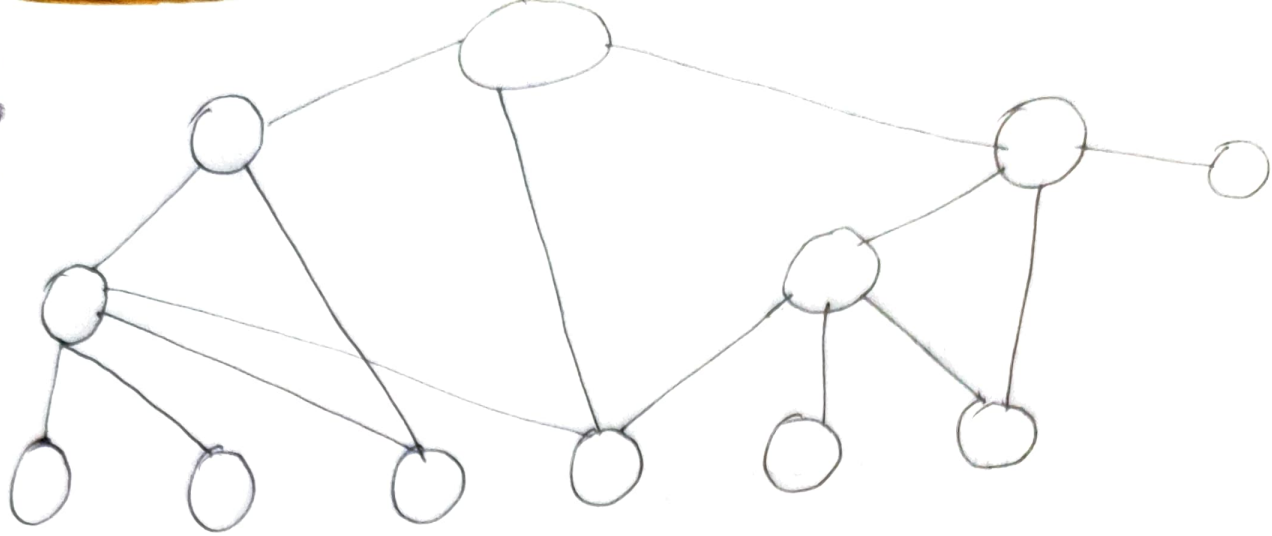
The approach combines modules to provide layer ons and so on till we arrived at a desired system

These modules are collected together in the form of a library.

Since the design progress from bottom layer upwards method is known as bottom up design.

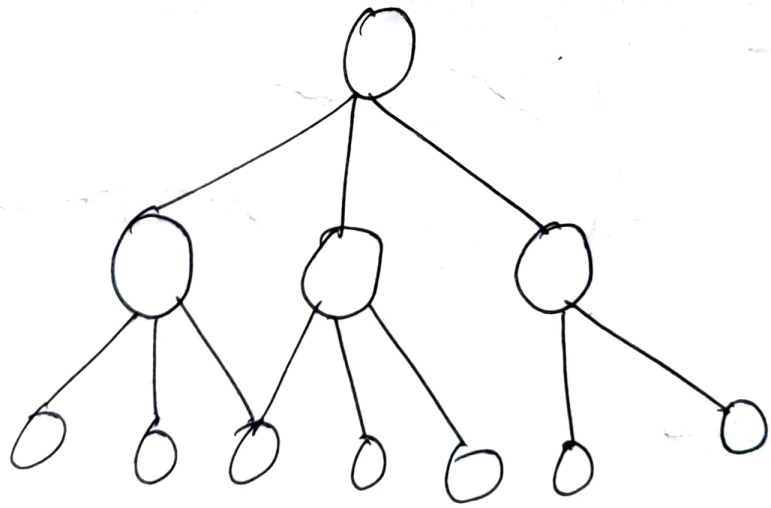
A lot of intution is required to decide what functionality a module should provide

If a system is to be built from an existing system the approach is more suitable.



Bottom up design

Top down Approach:- This is a step wise refinement It starts by identifying the major modules and then decomposing them into lower level modules and iterating until the disired level of detail is achieved. It starts from an abstract design in each step the refinement to a more concrete level until we reach a level where no more refinement is needed and the design can be implemented directly.



#Hybrid design:- for top-down approach to be effective, some bottom up approach is essential for the following reasons.

To permit common submodule

→ Near the bottom up hierarchy where interface is simpler and the need for bottom up testing is higher

→ On the use of prewritten library modules in particular reuse of modules.

Function Oriented Design:-

→ Here the design is decomposed into a set of interacting units where each unit has clearly defined function. Thus system is designed from a functional viewpoint.

→ we continue the refinement of each module until we reach the statement level of our programming language.

For a module we must require that several other module as in design reusable structure.

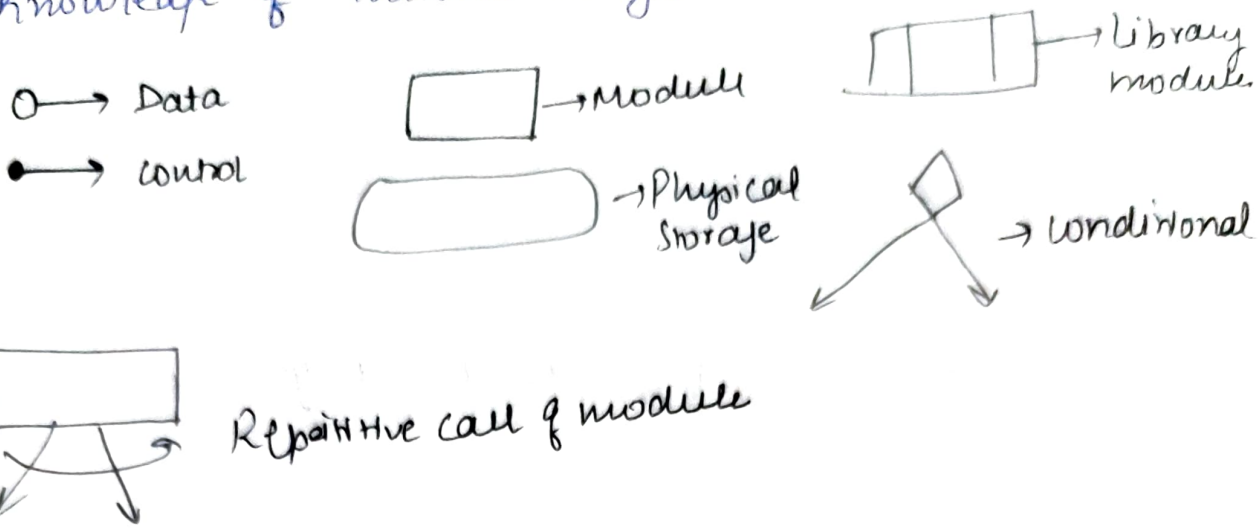
Design notations:- largely used in design process to represent design or design decisions

for a function oriented design the design can be represented by

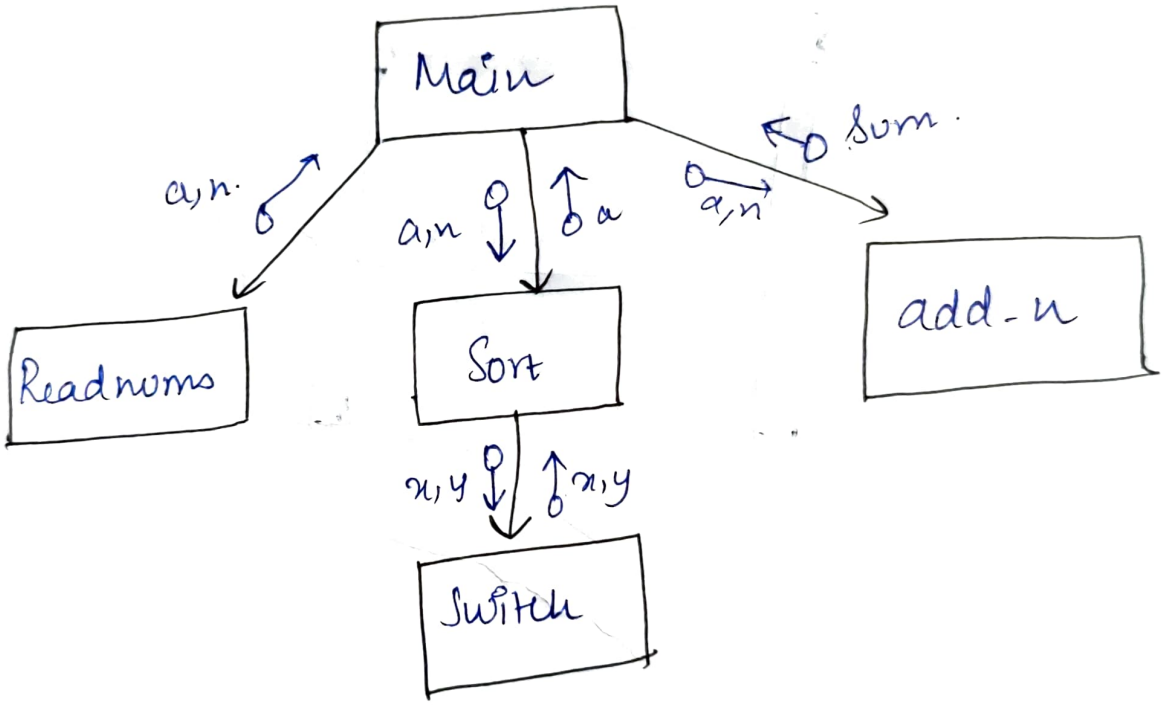
→ Data flow Diagrams
→ Data Dictionaries
→ Structure Charts
→ Pseudo code.

Structure Chart:-

It partition a system into block boxes. A block box means that functionality is known to users without the knowledge of internal design.



→ Structure chart for "Sorting and Adding Number".



Functional Procedural Layers: Function are built in layers

Level 0
- function name
- Relationship with other component
- Author, date

Level 1
- function parameter
- Global variable
- Routine called
- I/O assertions

Level 2
- Local data structure
- Exception handling
- → Any other limitation

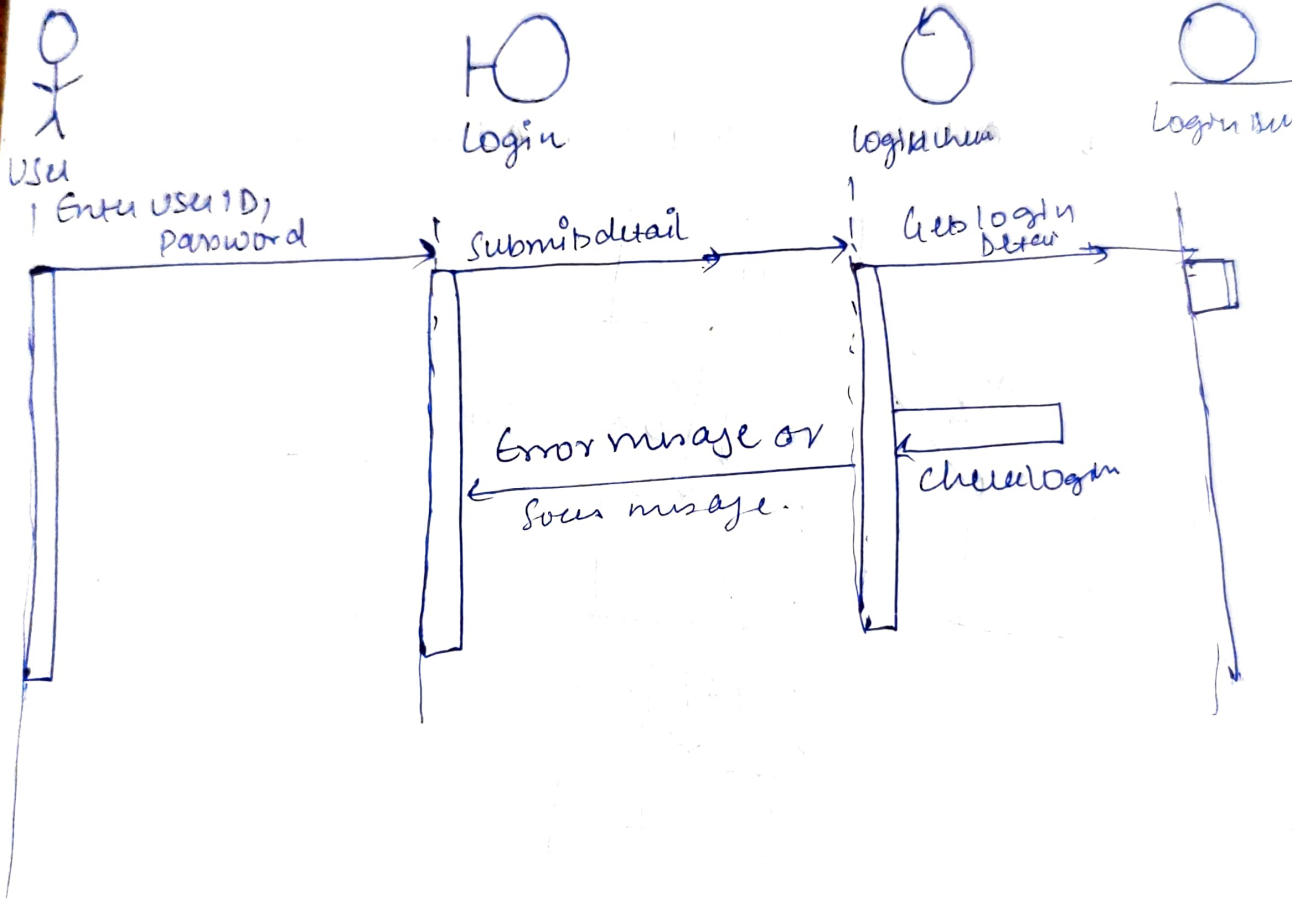
Sequence Diagram The object type used in this analysis model are entity objects, interface objects and control objects

Entity Object

Interface Object

Control object

Sequence Diagram for Login (Result Management) Module



Software Testing

→ It is an important discipline and consumes significant amount of effort and a properg strategy has to be implemen to carry out testing activities systematically and effectively.

Software Testing:- Testing of Software Product

↓

is a specialised discipline requiring unique skills.

What do you mean by Testing?

→ Testing is a process that demonstrates that errors are not present

→ The purpose of testing is to show that a program performs the intended functions correctly.

→ It is a process of establishing a connection/confidence that program is doing what it is supposed to do.

→ These definitions are opposite to what the testing actually does

Testing:- Testing is a process of executing a program with intent of finding errors.

Thus by testing we are increasing the reliability.

Why Should we Test?

→ Testing is an expensive activity yet launching Software without testing may lead to cost exponentially much higher than that of testing, specially in the systems where Human safety is involved

→ In software life cycle the earlier the errors are detected and removed lower is the cost of their removal

- What Should we Test → we should test the programs response to every possible input i.e we should test the program for all valid or invalid inputs

→ we should test those areas where chances of getting a fault is maximum.

→ Organizations should develop strategies and techniques for effective testing.

↓

A Strategy should be developed for testing small portion of program also to develop test cases for whole system.