SOFTWARE ENGINEERING — BASICS

- Software = Instructions + Data Structures + Documentation
- Software doesnot "wear out"

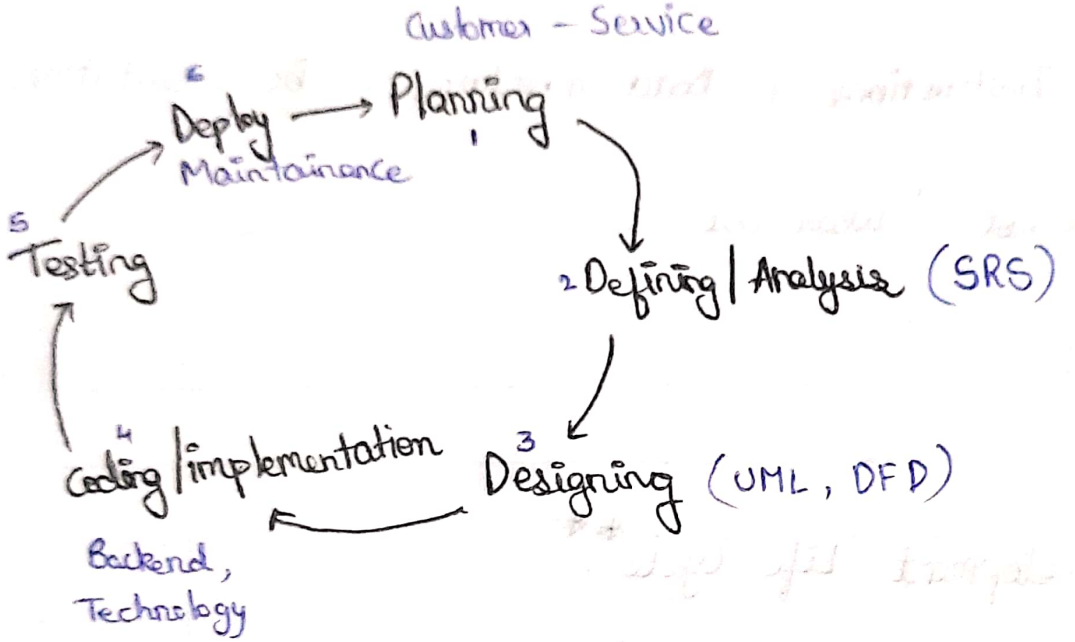
*. SYLLABUS :

1. Introduction.
2. Software Development Life Cycle**.
3. Requirement Analysis (SRS).
4. Software Project Management* (COCOMO).
5. Software Design (Coupling, Cohesion, UML, DFD, class Diag).
6. Coding and Testing**.
7. Maintenance.
8. Quality Management & Reuse.

*. Definition & Evolution :

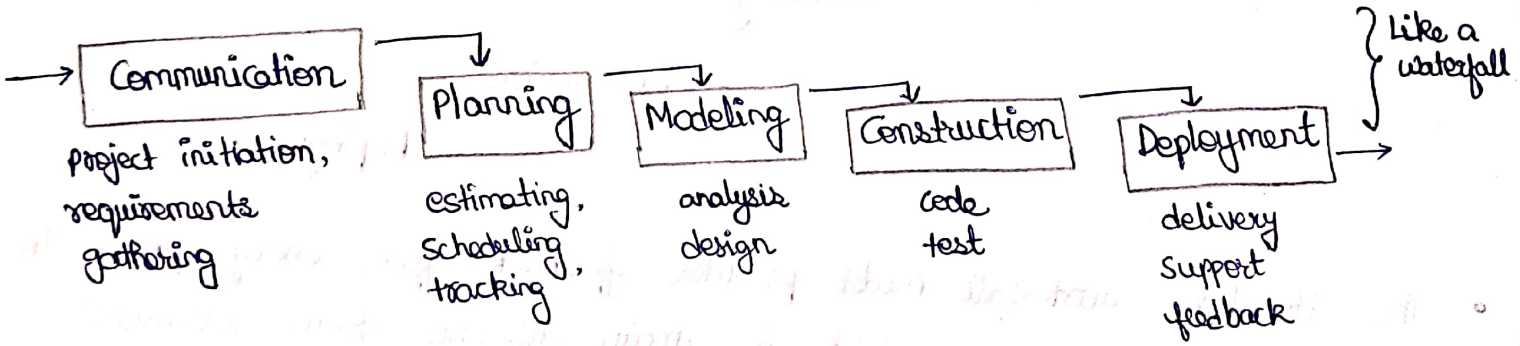
- It is the systematic, disciplined, cost-effective techniques for software development.
- Engineering approach to develop a software.

*. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) :



WATERFALL MODEL :

- There are times when the requirements for a problem are well understood — when work flows from communication through deployment in a reasonably linear fashion.
- Winston Royce introduced Waterfall Model in 1970. This model has 5 phases : Communication, Planning, Modeling, Construction, Deployment
- The waterfall model, sometimes called the "classic life cycle", suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling and construction, deployment, culminating in ongoing support of the completed software.



Usage of SDLC Waterfall Model :

- When requirements are well known, constant and not changed regularly.
- When a project is short & simple. The situation is calm.
- Where tools & technologies used is consistent & dont change regularly.
- All the resources are well prepared and are available to use.

Advantages

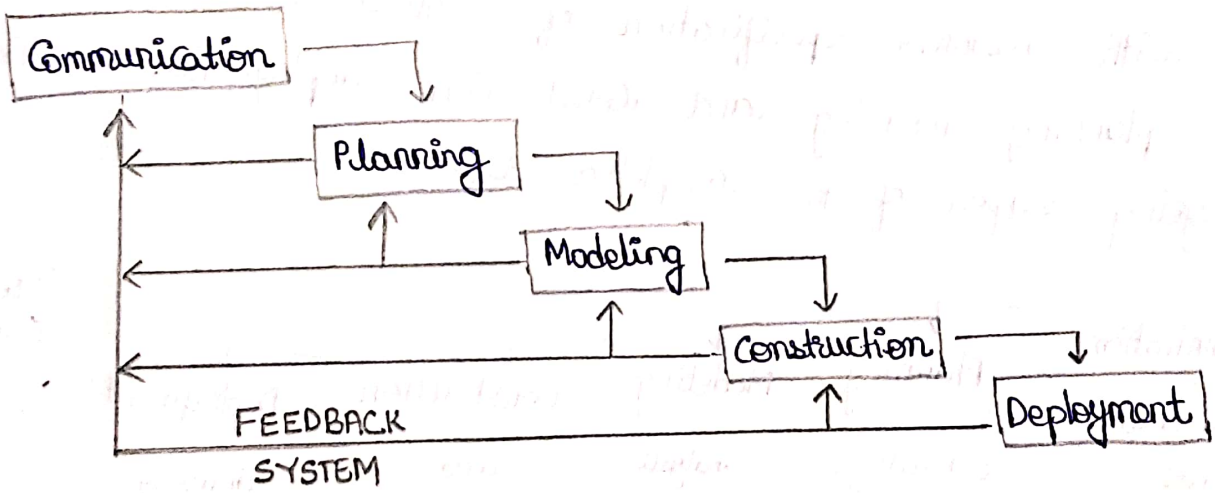
+ Base Model for development.
+ Easy to implement.
+ Resources required is minimum.
+ Easy to cover/upgrade progress.
+ Easy in control, clarity on project.

Disadvantages

- Higher risk factor
- Not suitable for complex projects
- Tough to go back to phase.
- Identifying the challenges and risks earlier is not possible.
- Maximum efforts in Maintenance
- No feedback to previous step.

ITERATIVE WATERFALL MODEL :

- It is updated Waterfall Model with Feedback system.
- Feasibility Model has no backtrack / Feedback.



- The iterative waterfall model provides feedback from every phase to its preceeding phases, which is main difference from classical waterfall model.

Advantages

+ Feedback Path.
+ Simple, Easy implementation.
+ Small Projects application.

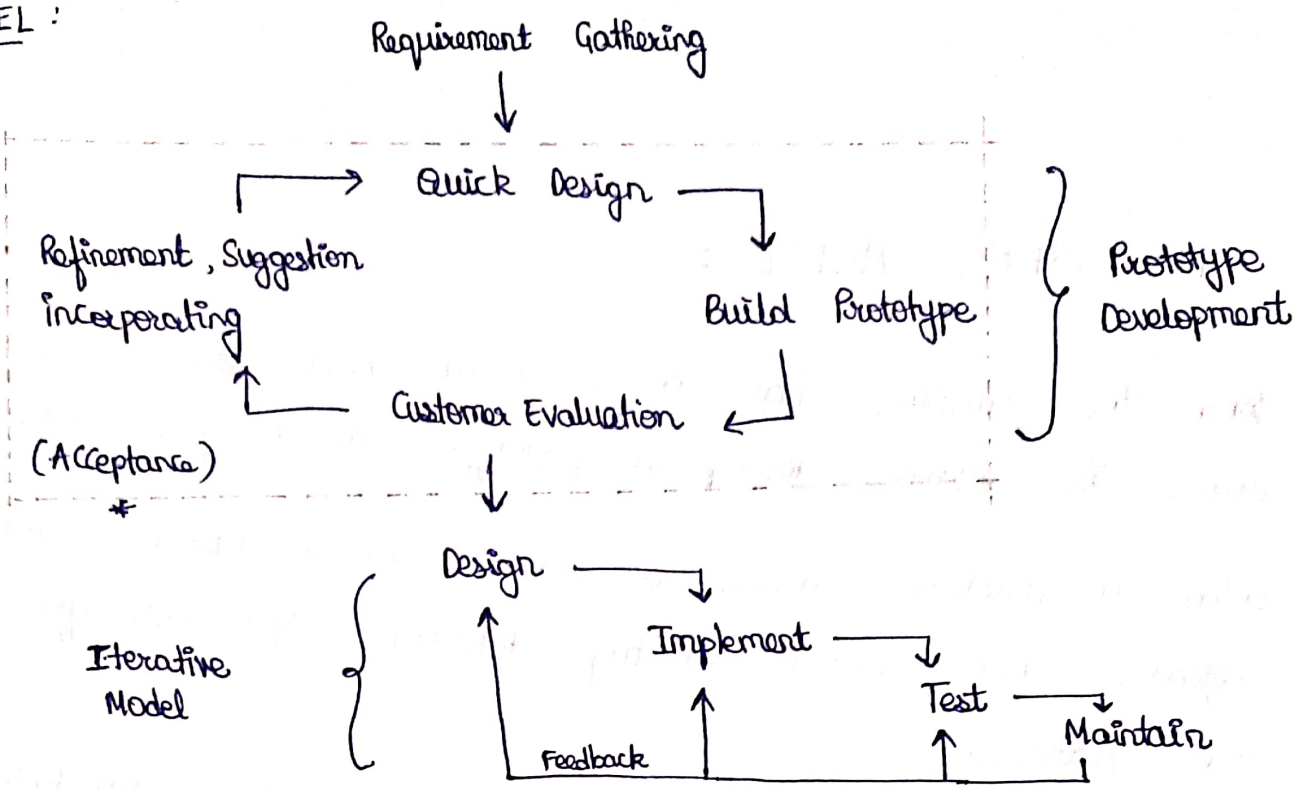
Disadvantages

- Not flexible to changes.
- Risk handling not supported.
- Limited customer interaction.
- Rigid (No change).
- No phase overlapping.

PROTOTYPING MODEL :

- When the customer has a legitimate need, but is clueless about the details, develop a prototype as a first step.
- Often, a customer defines a set of general objectives for software, but doesnot identify detailed requirements for functions and features.
- The prototyping paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.
- The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A Prototyping iteration is planned quickly, and modeling (in form of "quick design") occurs.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- The prototype can serve as "first system". Although some prototypes are built as "throwaways", others are evolutionary in the sense that the prototype slowly evolves into "actual system".

*. MODEL :



THE PROTOTYPE PARADIGM

*. Types : Rapid Throwaway Prototypes, Incremental Prototype, Evolutionary Prototype, Extreme Prototype

*. Usage : (Effective paradigm for software Engineering).

- When requirements are unclear.
- It is important to perform planned & controlled Prototyping.

Advantages

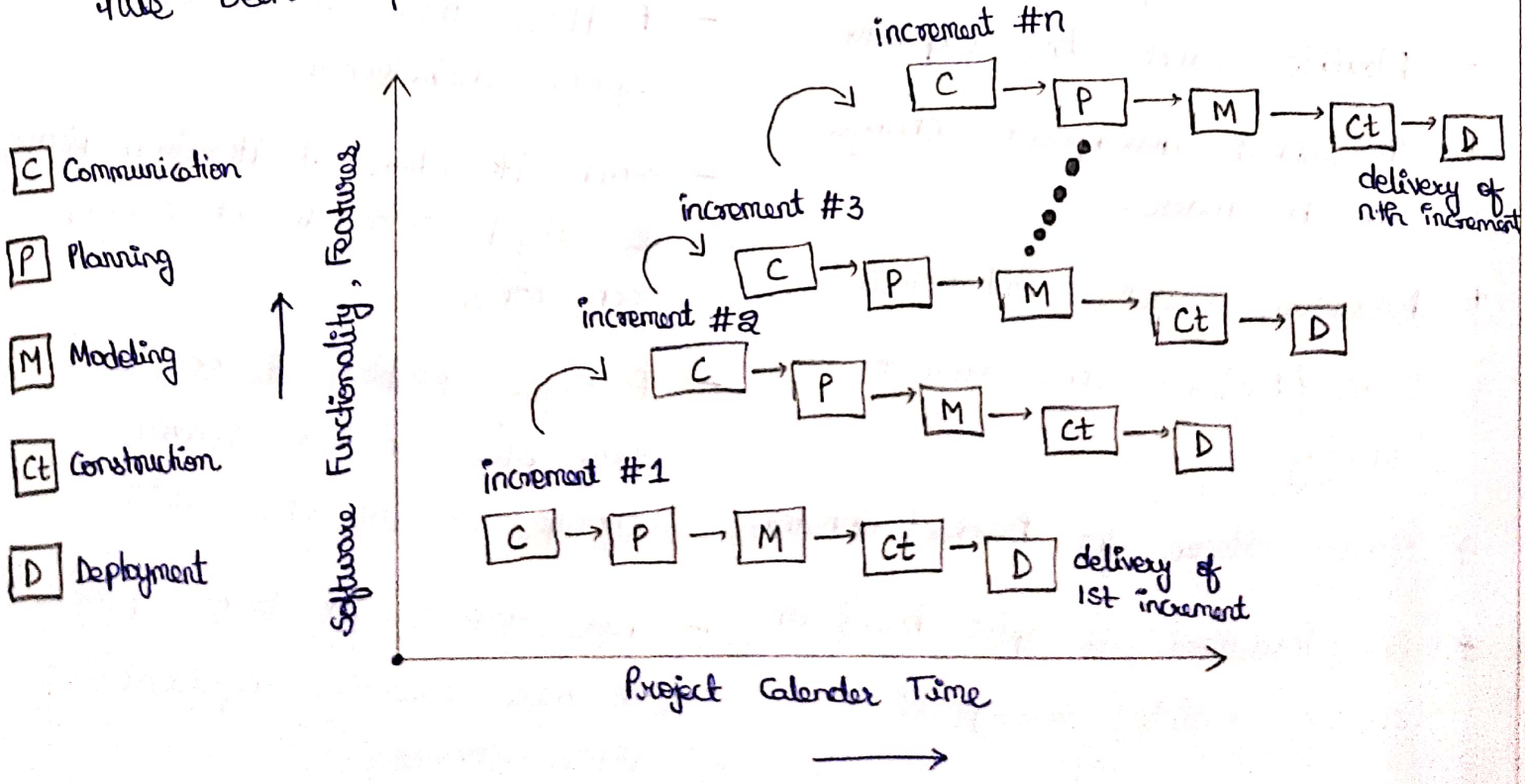
+ Errors are easily cleared.
+ Maximum Customer interaction.
+ Straightforward, easy to understand.
+ Well supports for Technical / Requirement risks.
+ Less failure / chances to fail.
+ clear understanding

Disadvantages

- Slow & time taking.
- Cost of development is more.
- Poor Documentation because of requirements change.
- If idea changes, throwaway prototype & build new one.
- The less-than-ideal choices may become integral part of system.

INCREMENTAL MODEL :

- The incremental Model delivers a series of releases, called increments, that provide progressively more functionality for the customer as each increment is delivered.
- There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process.
- Here the requirements are broken down into multiple standalone modules of software development cycle.
- Each iteration passes through the requirements, design, coding and testing phases. And subsequent release of the system adds function to the previous release until all designed functionality has been implemented.



Characteristics

* System development is broken down into many mini development projects.
* Partial systems are successively built to produce a final total system.
* Highest Priority requirement is tackled first.
* Once the requirement is developed, requirement for that increment are frozen.

Use of Incremental Model

- Requirements of the system are clearly understood.
- When demand for an early release of a product arises.
- When Software Engineering team are not very well skilled or trained.
- When high risk features and goals are involved.
- Such methodology is more in use of web application and product based companies.

Advantages

+ Quick development is possible
+ Flexible and less expensive
+ Throughout development, changes can be made.
+ Maximum Customer Interaction
+ Errors / Problems are easy to identify
+ Early release for Product demand.
+ Cost / investment is split based on current module development.

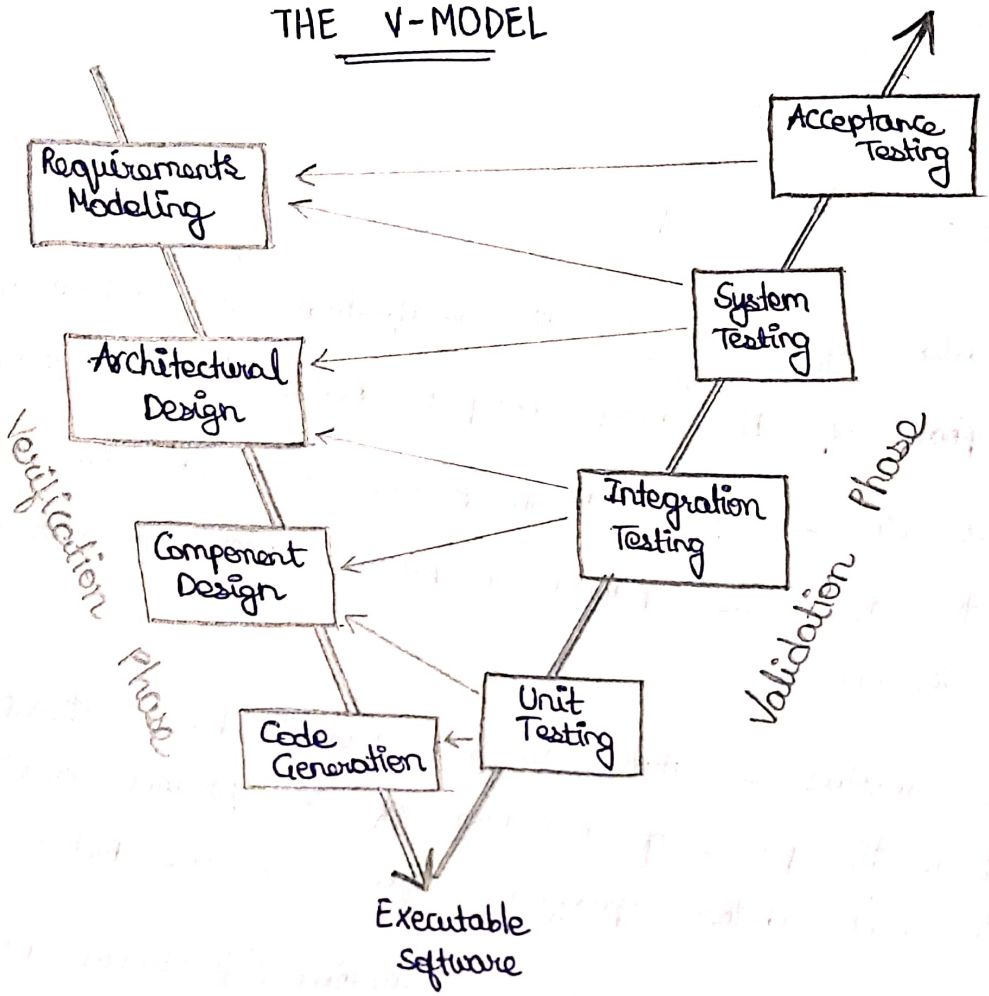
Disadvantages

- It requires good planning designing
- Problems might cause due to system architecture.
- Each iteraction at iteration phase is rigid and doesnot overlap each other.
- Rectifying problem in one unit may effect in corresponding changes in all other units.
- Time taking for larger projects ie more customer requirements / more changes.

V- Model :

- V Model also referred to as the Verification and Validation Model.
- In each phase of SDLC, must complete before the next phase starts.
- It follows a sequential design process same as the waterfall model.
- Testing of the device is planned in parallel with a corresponding stage of development.
- Verification involves - static analysis method done without executing code. It is the process of evaluation of the product development process to find whether specified requirements are met.
- Validation involves dynamic analysis method (functional, non-functional) Testing is done by executing the code. This process is to classify the software after the completion of development process to determine whether the software meets the customer expectations and requirements.
- These two phases are joined by coding phase in a "V-shape". Thus it is known as "V-model".
- Phases of Verification : Requirements modeling, Architectural design, Component design, Code Generation
- Phases of Validation : Unit Testing, Integration Testing, System Testing, Acceptance Testing
- Every phase in downward sequence, there is a corresponding testing phase in the following sequence

THE V-MODEL



Usage of V-Model :

- When the requirements are well defined and not ambiguous.
- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantages of V-Model

+ Easy to understand.
+ Testing happens before coding.
+ Higher chance of success than Waterfall.
+ Saves lot of time.
+ Avoids downward flow of defects.
+ Works for small plans.

Disadvantages of V-Model

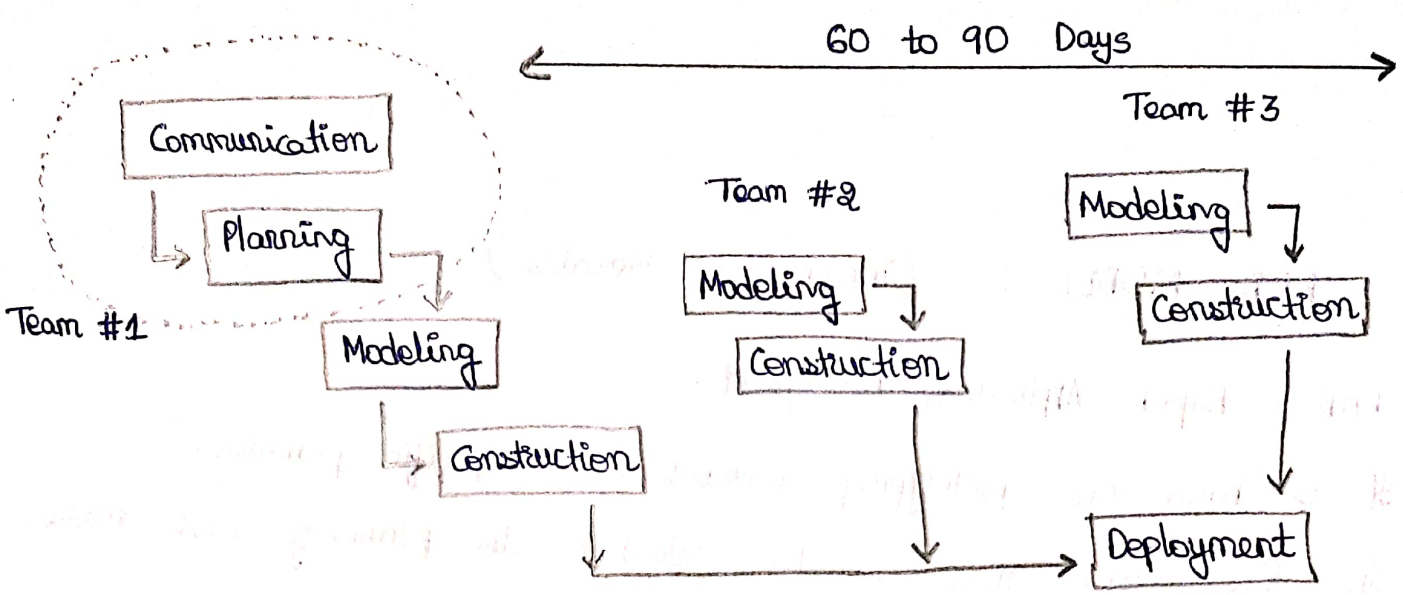
- Very rigid & least flexible.
- Not a good for complex project.
- No prototypes are produced
- Doesnot function for midway changes
- No feedback / interaction with customers.

RAD MODEL : (incremental + iterative)

- RAD - Rapid Application Development.
- It is based on prototyping without any specific planning.
- In this model, there is less attention to planning and more priority is given to development tasks.
- This is an incremental process model that emphasizes on extremely short development cycle.
- It is provided that requirements are well understood and project scope is constrained. encompass following phases:

1. Communication : Works to understand business problems.
2. Planning : works in parallel on various system functions.
3. Modeling : Business Modeling + Data Modeling + Process Modeling.
4. Construction : Assumes 4th Generation Techniques (Reuse, Automation)
5. Deployment : prototypes are individually tested & deployed.

- RAD focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of existing prototype.
- RAD uses minimal planning in favour of rapid prototyping (Prototypes developed are reusable).



THE RAD MODEL

Usage of RAD Model :

- When a system/ software is to be developed within short span.
- When the requirements are clear, well known.
- When the user will be involved all through the life cycle.
- When technical risk is less / More human source is available.
- If there's a need of modeling along with cost of automated tools of code generation.

Advantages

+ Flexible & Adabtable to changes.
+ Reduction of Manual coding.
+ Reduced Development time.
+ More Customer Feedback.
+ More Productivity in short span.
+ Applicable for larger system developments when scarcity.

Disadvantages

- Not suitable for small Projects.
- If technical risk is high, it fails.
- More human resource, limited application.
- More modelling, planning skills needed (more Man intelligence).
- Lack of Documentation time where maximum time is spent on development.

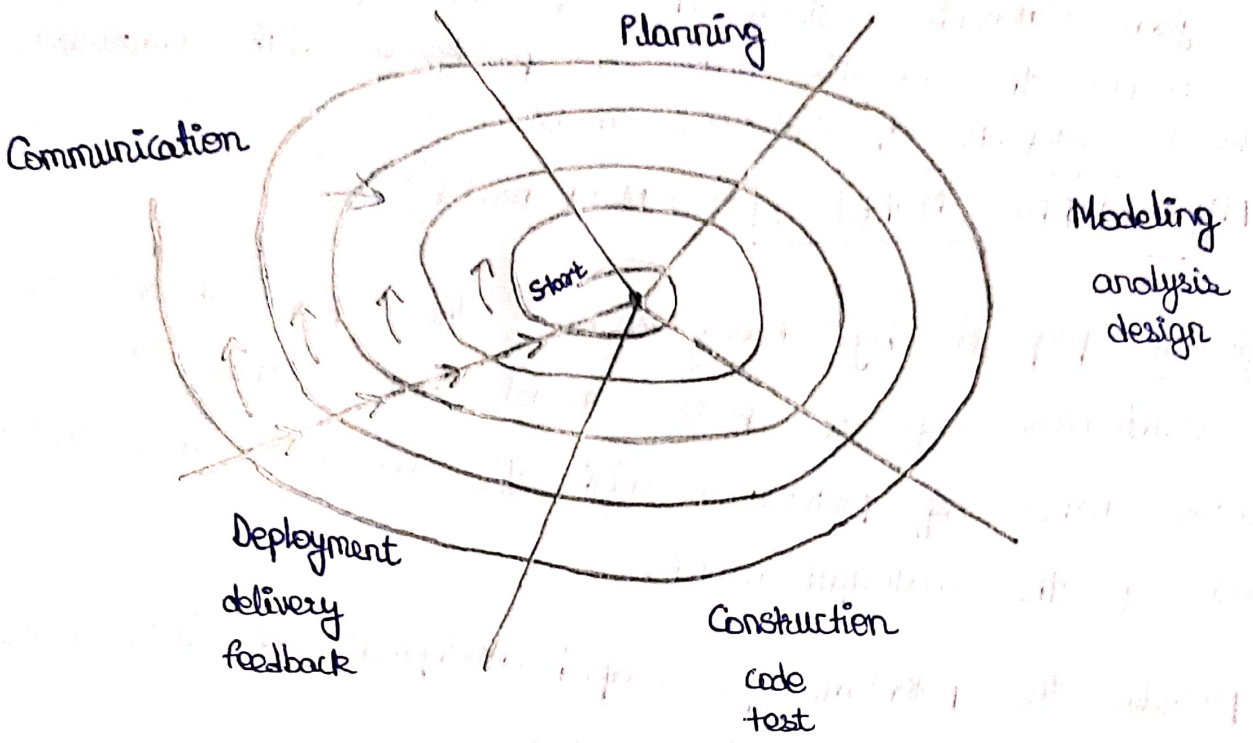
* The given statement " It is a risk - driver process model generator that couples the iterative nature of prototyping with controlled & systematic aspects of traditional model".

IMP * **THE SPIRAL MODEL :** (Meta model)

- Originally proposed by Barry Boehm, the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- It provides the potential for rapid development of increasingly more complete versions of the software.
- The spiral model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- It has two main distinguishing features. One is cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance.

  during early iterations ⟶ model or prototype
  during later iterations ⟶ complete risk-free versions

A TYPICAL SPIRAL MODEL



- Phases : Planning, Risk Analysis, Engineering, Evaluation.
- Use of Spiral Model :

→ When project is large, requirements are more.
→ When risk & cost evaluation is important.
→ When requirements are unclear and complex.
→ When changes may require any time.
→ When Less experience development Team is present.

Advantages

+ Changes can be done later stage.
+ Cost Estimation is easy.
+ Perfect for Risk Assesment/Management.
+ Development is fast.
+ Customer Feedback is considered.
- Radius of Spiral = Total Cost
- Angular Dimension = Progress

Disadvantages

- Risk of not meeting schedule or budget.
- Needs to be followed strictly
- Applicable for large projects only.
- The spiral keeps on rotating until satified.
- Complex & sometimes Expensive
- More Documentation (more Intermediate stages)

Activities in Spiral Model :

Each cycle in spiral is divided into 4 parts. Until the team satisfied, the spiral keeps on rotating.

(1.) Planning –

- It includes estimating the cost, schedule and resources for the iteration.
- It also involves understanding the system requirements for continuous communication between system analyst & customer.

(2.) Risk Analysis –

Identification of potential risk is done while risk mitigation strategy is planned and finalized.

(3.) Engineering –

It includes testing, coding and deploying the produced software at the customer site.

(4.) Evaluation –

Evaluation of software by the customer. Also includes identifying & monitoring risks such as schedule slippage & cost overrun.

EXAMPLES :

- The best suited example for spiral Model is "MS-Excel" because MS-Excel sheet having several cells, which are components of excel sheet.

- Since we have to create cells first, then we can perform operation on the cells like split cells, into half, merge cells into two, and then we can draw graphs on the excel sheet.
- In spiral Model we can perform 2 types of changes

1. Major changes
2. Minor changes

} Can be applied to following example in various stages.

*. Comparision of Various SDLC Models : [IMPORTANT]

→ classical Waterfall :

- Basic, Rigid, Inflexible - Not for Real Project.
- No changes, no parallelism, No user interaction.

→ Iterative waterfall :

- Basic problem is well defined.
- Feedback system is implemented.

→ Prototype Model :

- User Requirement not clear, costly, No early lock of requirements, High user involvement, Reusability.

→ Incremental Model :

- Module by module Delivery, Easy to test & Debug.

→ Evolutionary Model :

- Incremental model + No lock on requirements + Large Projects.

→ RAD Model :

- Time, cost constraint, User interaction at all levels, Reusability.

→ Spiral Model :

- Not for small projects, Risk handling, Analysis, less Experience Development team can work.

⑧ Component Based Development : (CBSE)

- In CBSE, we develop software using already available components.
- In this kind of development there is no concept of building any software from scratch.
- Components communicate with each other by their well defined interfaces.

Properties : Independent, standardized, Deployable, Documented, Reusability, Extensibility, Maintenance

Goals : Save time, money when building complex projects.
Enhance software quality (component quality)
Detect defects within systems (fault detection)

Advantages : Minimized delivery, Improved efficiency, Improved quality, Minimized expenditures.

Routines : Component development, publishing, lookup, retrieval, analysis, assembly.

- This is much advanced than OOP models.

*. 4th Gen Techniques :

Adv
- Report Generation, Data manipulation, Screen Interaction, Code Generation, spread sheet Capability.
- Includes tools, takes specifications from user, generate corresponding codes.

FRAMEWORK ACTIVITIES :

- A process framework establishes the foundation for a complete software engineering process by identifying a small no. of framework activities that are applicable to all software projects, regardless of their size or complexity.
- In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.
- A generic process framework for software engineering encompasses five activities :

1) Communication
2) Planning
3) Modeling
4) Construction
5) Deployment

- For many software projects, framework activities remain same, are iteratively applied as a project progresses.
- Each iteration produces a software increment that provides stakeholders with a subset of overall software features and functionality.
- As each increment is produced, the software becomes more and more complete.

UMBRELLA ACTIVITIES :

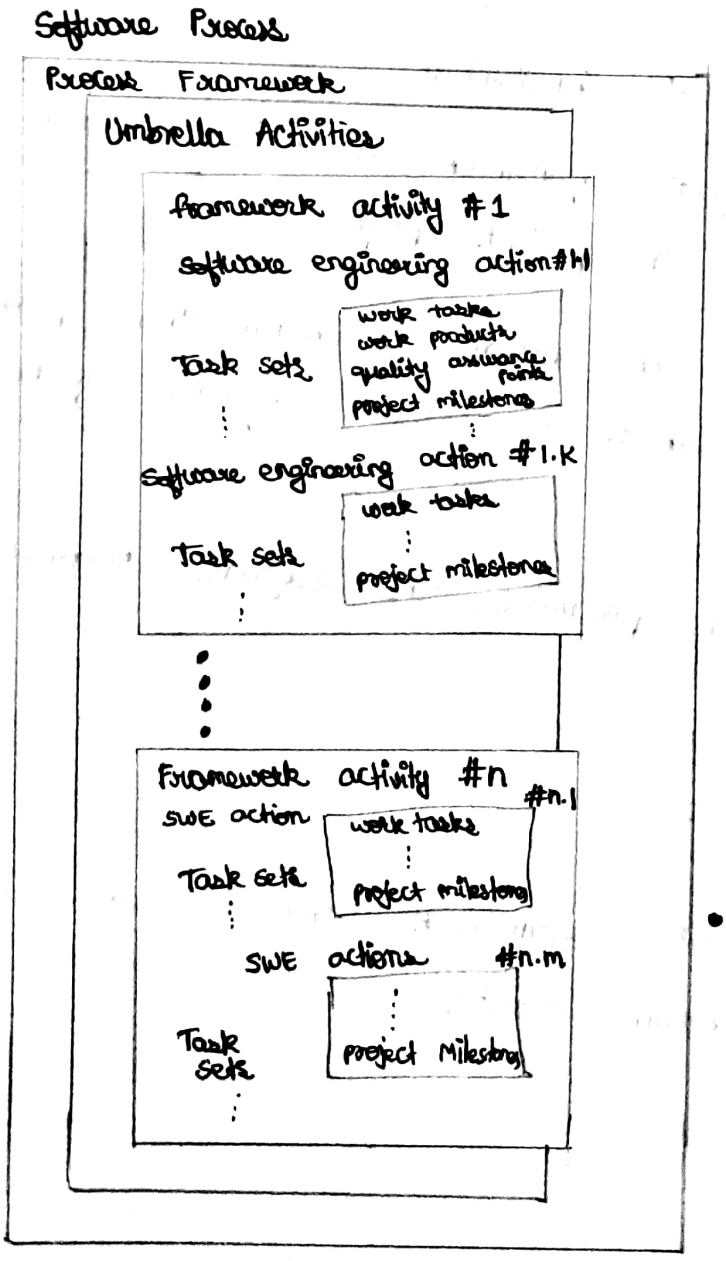
- Software engineering process framework activities are complemented by a number of umbrella activities

- In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.

- Typical Umbrella activities include :

1) Software project tracking and Control
2) Risk Management
3) Software quality assurance
4) Technical reviews
5) Measurements
6) Software configuration & Management
7) Reusability management
8) Work product preparation and production

SOFTWARE PROCESS FRAMEWORK



- The software process is represented schematically here.
- Each framework activity is populated by a set of SWE actions.
- Each SWE action is defined by a task set (of SWE) that identifies the work tasks that are to be completed, products that are to be produced.
- Quality assurance points that will be required and the milestones that will be used to indicate progress.

Framework Activities (in detail) :

① Communication :

- Before any technical work can commence, it is critically important to communicate and collaborate with customer (and other stakeholders).
- The intent is to understand stakeholders objectives for the project and to gather requirements that help define software features and functions.

② Planning :

- Any complicated journey can be simplified if a map exists.
- A software project is a complicated journey, and the planning activities create a "map" that helps guide the team as it makes the journey.
- The map - called a software project plan - defines the software engineering work by describing the technical (risks) tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

③ Modeling :

- Create a sketch of thing to understand big picture. — what it will look like architecturally.
- An SWE does the same thing by creating models to better understand software requirements & design that will achieve those requirements.

④ Construction :

- what you design must be built.
- The activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

⑤ Deployment :

- The software (as a complete entity or as a partially completed increment) is delivered (to be) the customer who evaluates the delivery product and provides feedback based on the evaluation.
- These 5 generic framework activities can be used during the development of small, simple programs, the creation of web applications, and for the engineering of large, complex computer-based systems.

UMBRELLA ACTIVITIES (In detailed) :

① Software project tracking and control — allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

② Risk Management — assesses risks that may affect the outcome of the project ensure software quality.

③ Software quality assurance : defines and conducts the activities required to ensure software quality.

④ Technical reviews - assess software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

⑤ Measurement - defines and collects process, project and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.

⑥ Software Configuration and Management - manages the effect of changes throughout the software process.

⑦ Reusability management - defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

⑧ Work Product preperation and production - encompasses the activities required to create work products such as models, documents, logs, forms and lists.

Framework Activities (E-Book Management):

① Communication :

- First the company will collect all the user requirements and then apply them repeatedly until the project keeps progressing.
- At each step of iteration, it provides stakeholders that there should be some features like login page, signup page, register, etc.. for functionalities like create, update, delete, read, edit, ...

② Planning :

- The developers plan the flow of the E-book Management system.
- Like at which stage what to be developed, and what functionalities to be added.

③ Modeling :

- Now, they create a sketch of thing to understand big picture — what is what and how this look like.
- Putting entire coding on a paper with a computer aided software design tools like Wireframes, Figma, Canva, Lucidcharts, ...

④ Construction :

- Now the technical team takes the design/models built in previous stage as reference & started to code in IDEs.

⑤ Deployment :

- Finally the developed E-Book Management are hosted on hosting sites like GITHUB, Netlify, AWS cloud, HerokuApp, etc... & further tends to manage it.

MODULE - 2

*. REQUIREMENT ENGINEERING : (STEP 1)

- Understanding requirements (Base)
- Identify & understand Potential Problem → Inception
  Need to define nature & scope of problem statement

① Inception - make idea clear.

② Elicitation - what is required?

- problems of scope - model ill defined.
- problems of understanding - (take requirements for obvious/granted).
- problems of volatility - change in requirements over the time.

③ Elaboration - detailed discussion with stakeholders to make, refine, expand, elaborate, modify the information.

- Elaboration helps in develop Refined Requirement Model. Trying to create, refine, describe user scenarios. Trying to describe how users will interact with system.

④ Negotiation - Given a limited buisness resources they expect lot of requirements.
↓
Modify some of requirements, discussion aimed at reaching an agreement

⑤ Specification - Specify functional/non-functional, constraints over requirements through written doc, Graphical models, User Scenarios, SRS, UML diagrams, Mathematical models, etc...

- Large Projects - Specify through Graphical Models.
- Smaller Projects - Specify by UML, User Case Scenarios.

⑥ Validation - Quality / Examination of requirements.

→ Ambiguity checks, inconsistent reviews,
→ identify, uncover errors, check for unrealistic requirements, misunderstandings in requirements
→ Get exact proper requirements.

⑦ Requirements Management - Specified requirement MATCH Customer Expectations

- Needs to identify changes in requirements based on time / other reasons.
- Trying to monitor, track, control the requirements and the changes to the requirements at any time as the time proceeds.

→ The process of gathering, analyzing, documentating, validating, managing requirements. This process has 4 main steps:

1) Feasibility study
2) Requirement Gathering
3) Software Requirement Specification
4) Software requirement validation.

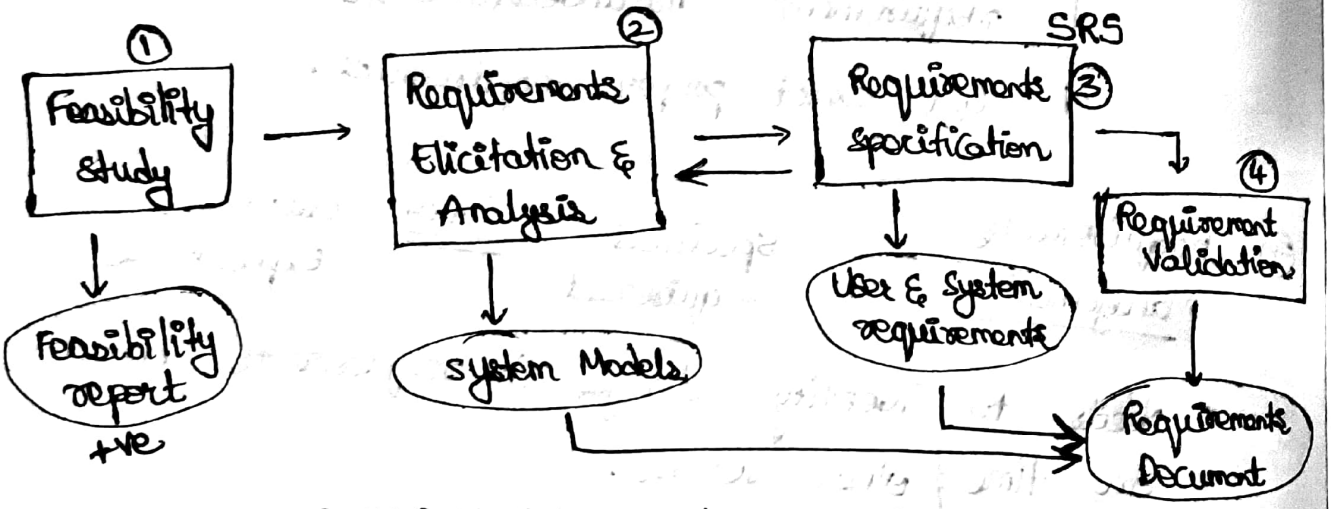
→ participants in this process are :

1) Software Engineer
2) Managers
3) Users
4) Customers

} outside organization

* Feasibility study - Budget Report

- client approaches organization.
- Rough idea about what all functions that software must perform & which all factors are expected from software.
- Feasibility study Report (✓) → goes to step 2



REQUIREMENT ENGINEERING PROCESS

*. Requirements Gathering :

- Both Analyst & Engineer gathers requirements from customers.
- SRS - software requirement specification. System Analysts creates SRS documents.
  - requirement - natural language → Design
  - Technical - structural language → GUI screen points
  - Design descriptions - Pseudocodes → DataFlow diagrams (Math)

*. Requirement Validation :

- practically implement ?
- If they are valid (based on functionality & domain of software) ?
- Ambiguity, inconsistence checks.
- Demonstrate the Model → Requirement Document.

*. UML (Unified Modeling Language) :

- This is a visual language.
- To define a standard way to visualize the way the system has designed / to be designed.
- Just like making Blueprint. (Also called General purpose modeling language).
- Reduces time to understand.

*. 4+1 Architecture View Model :

① Logical View

- static Modeling view
- deals with functionalities
- Sequence, class & communication diagrams

② Development View

- Component design view
- Developer's view
- Component & Package diagram
- subsystem & system development

③ Process View

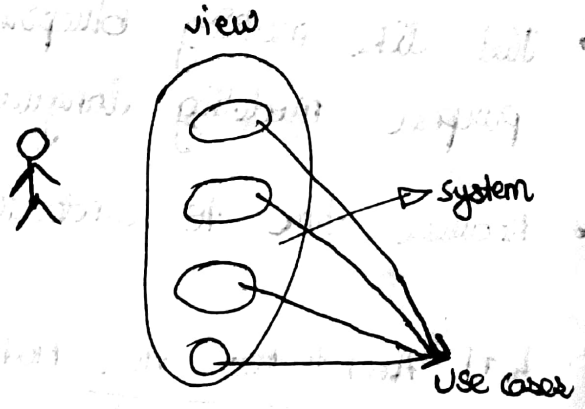
- Task view.
- Deals with runtime aspects of the system.
- Scalability & performance.
- Runtime description of system structure.
- Activity diagram.

④ Physical view :

- deployment of system.
- Tools & environment of system.
- System engineers' point of view.
- Deployment diagram.

(4+1) Scenario :

- Use Cases view



*. Building blocks of UML :

① Things :

- Modeling Elements
- Structural Things : Static Components like nouns (class, usecase, interface).
- Behavioural Things : Represent activity between structual things verbs of UML (Interaction).
- Grouping Things : Logically grouping similar elements. Packages to group usecases.

- Annotation Things : Represent extra info element. Sticky notes in diagrams.

② RELATIONSHIP :

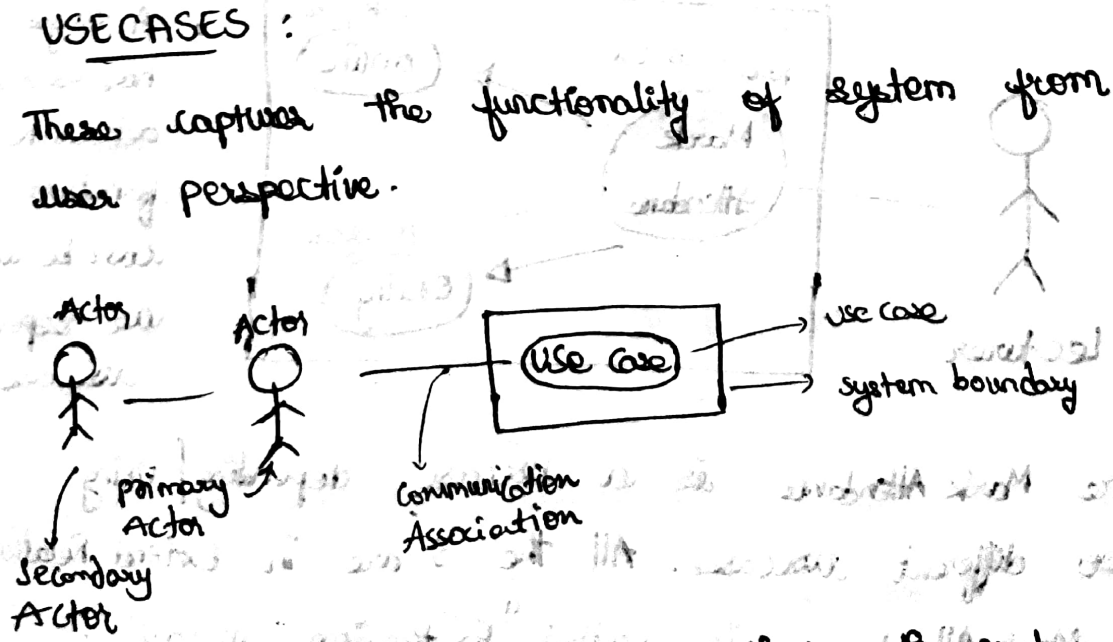
- Represent Relation among things.
- Association - Link between thing/objects (——)
- Dependency : (⟶) like e.g., X ⟶ Y
- Generalization : General or specific thing (⟶▷)
- Realization : Relation between classifier (- - - - -▷)

③ Diagrams :

- Pictorial Representation of System..
- Dynamic model : object, usecase, sequence, Activity, collaboration, state chart.
  (↓ behaviour of system)
- Static model : class, component, deployment.
  (↓ structure of system)

*. USECASES :

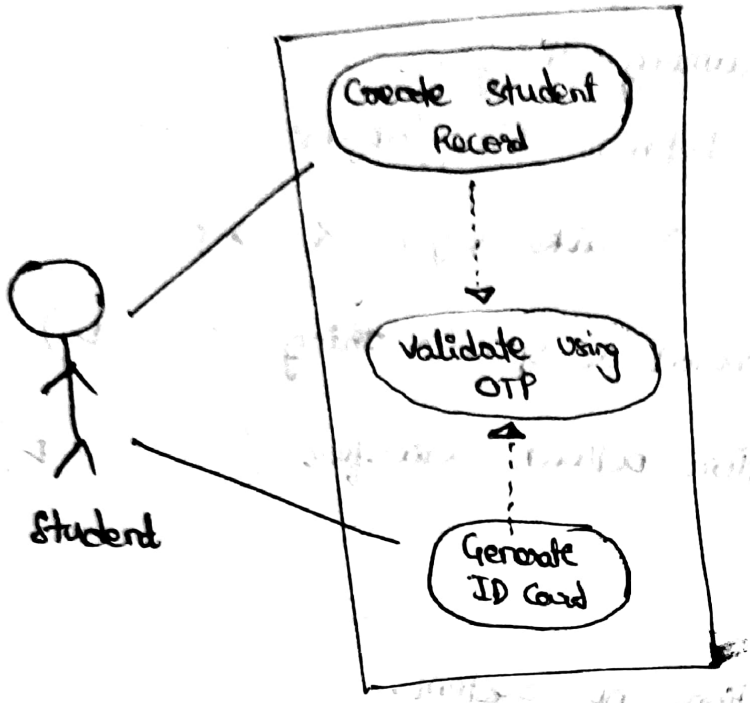
- These capture the functionality of system from user perspective.



- Primary Actor : First input & activates the system.
- Secondary Actor : Input after system activates

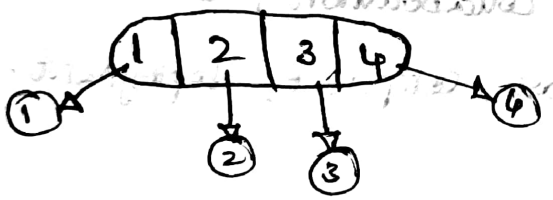
*. USECASE RELATIONSHIPS : [IMP]

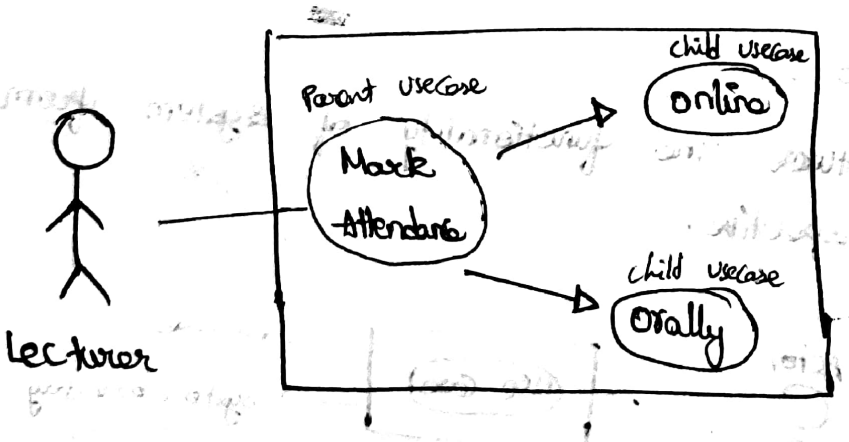
1. Inclusion Usecase : (Include Relationship)



- 2 usecases are using same usecase
- Validate using OTP is here called Inclusion usecase.
- This type of relation between given 3 usecases follows Include Relation

2. Extends Relationship : (Extension Usecase)

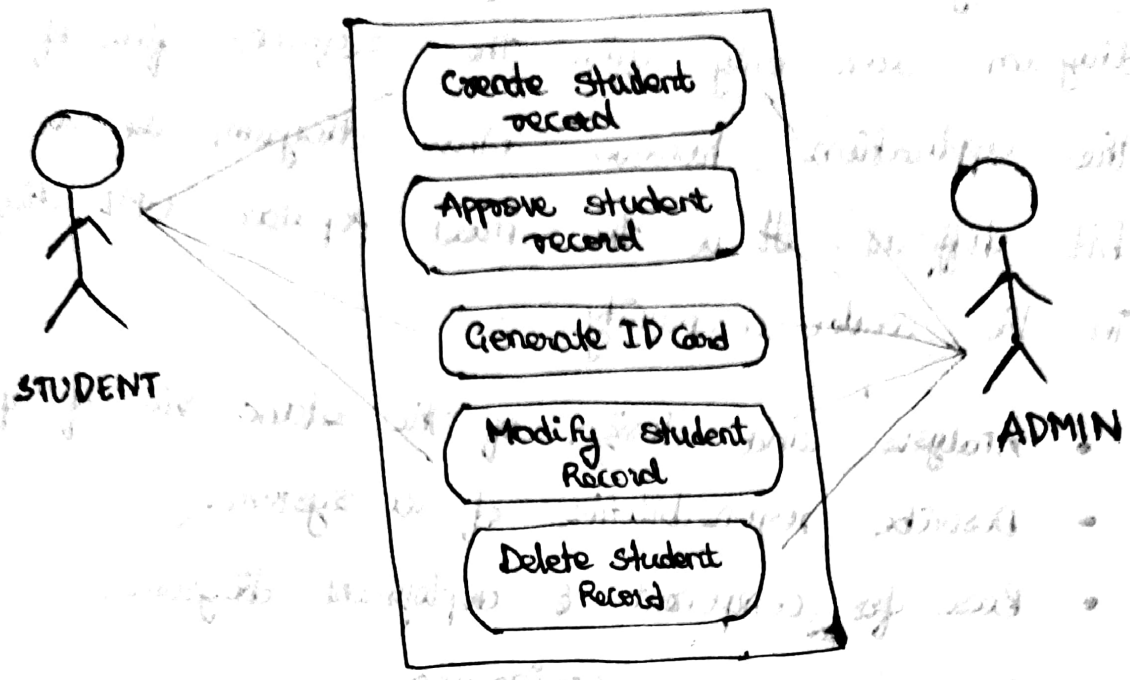




- A single usecase has several alternate functionalities can be used as seperate usecase.

- Here Mark Attendance is a usecase depending/using two different usecases. All the 3 are in Extend Relationship & Mark Attendance is called "Extension usecase".

*. USECASE PACKAGES :



- Package - A logical group of similar usecase.
- common usecase - student record management

*. CLASS DIAGRAM :

- class Diagram is a static diagram. It represents the static view of an application.
- class Diagram is not only used for visualizing, describing, and documenting different aspects of the system but also for constructing executable code of the software application.
- class diagram shows a collection of classes, interfaces, associations, collaborations & constraints. It is also known as "structural diagram".
- The basic purpose of class diagram is to model the static view of an application. class diagrams are the only diagrams which can be directly mapped with object oriented languages.

- UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

→
- Analysis and design of the static view of application
- Describe responsibilities of a system.
- Base for, Component & deployment diagrams.
- Forward & reverse engineering.

*. How to draw Class Diagram ?

- class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application.
- Important Points to Consider:

1) The name of the class diagram should be meaningful to describe the aspect of the system.

2) Each element and their relationships should be identified in advance.

3) Responsibility (attributes & methods) of each class should be clearly identified

4) For each class, minimum no. of properties should be specified, as unnecessary properties will make the diagram complicated.

5) Use notes wherever required to describe some aspects of the diagram. At the end of the diagram drawing, it should be understandable to developer/coder.

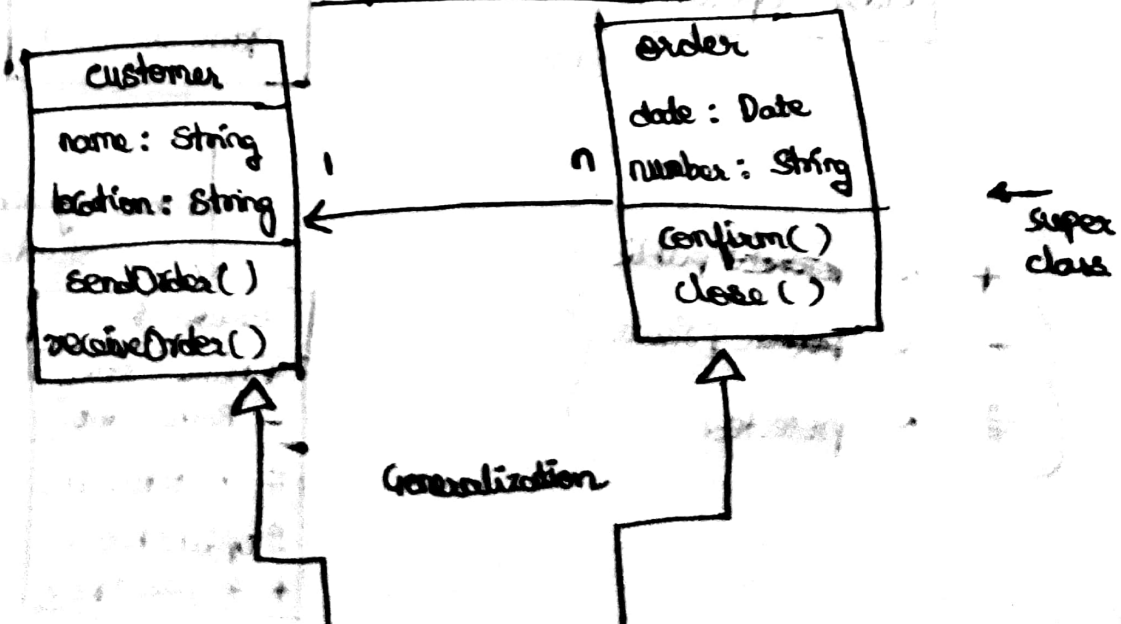
6) Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

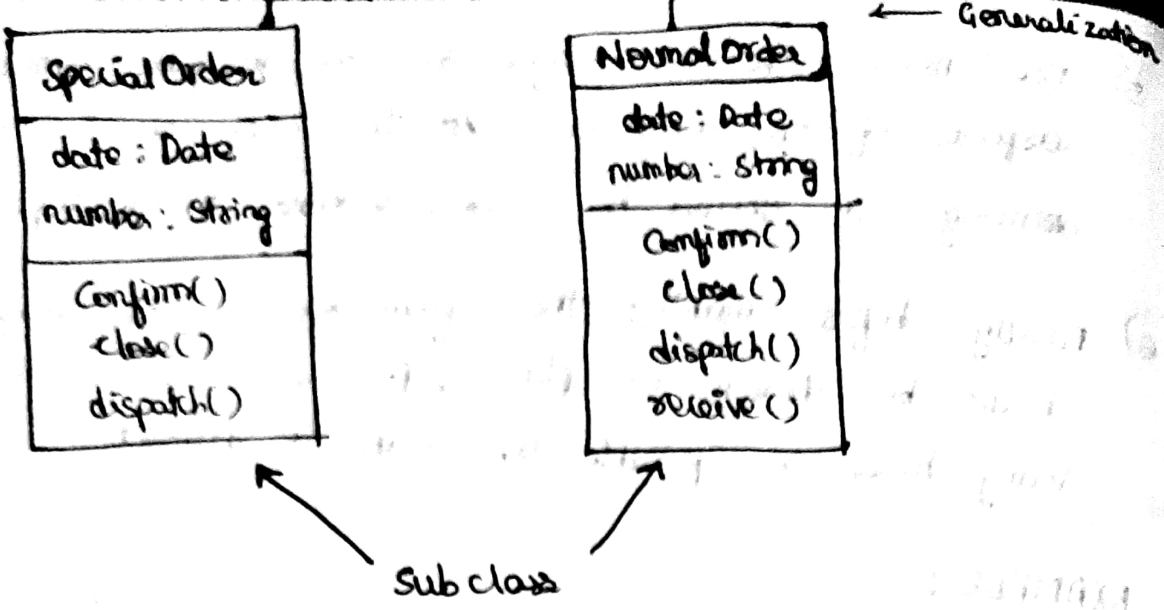
EXAMPLE:

The following diagram is an example of an "Order System" of an application.

- First of all, Order & Customer are identified as the two elements of the system. They have 1 to many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the order class. In addition, they have additional functions like, dispatch() and receive().

Sample class Diagram

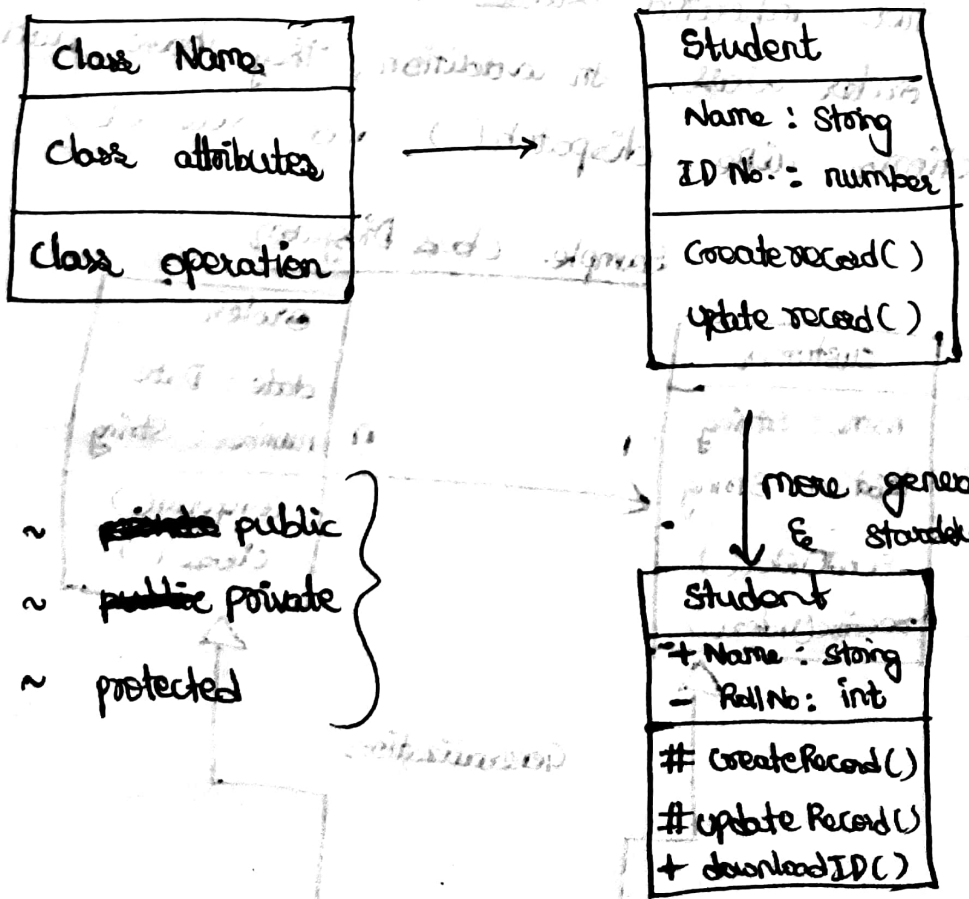




*. Class diagrams are used for :

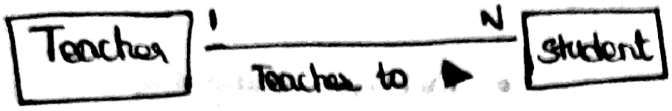
- Describing the static view of the system.
- Showing the collaboration among the elements of the static view
- Describing the functionalities performed by the system.
- Construction of software applications using OOP languages.

Structure of class Diagram

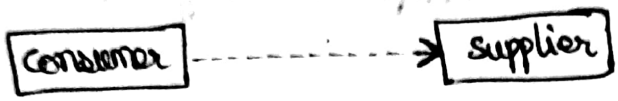


Class Diagram Relationships :

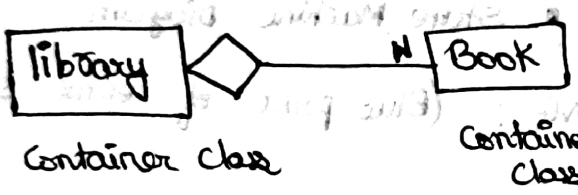
① **Association** - static relationship between classes



② **Dependency** - one class depends on another class



③ **Aggregation** - "has a relationship"

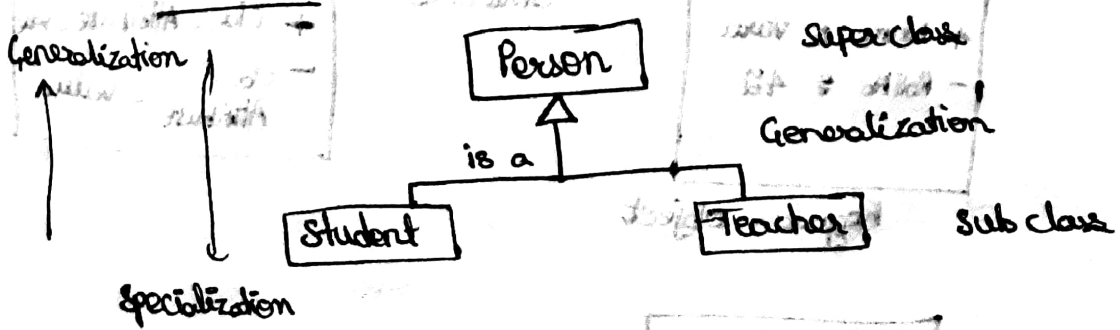


- Contained class doesnot have a strong dependency on the life cycle of Container class.

④ **Composition** -

Contained class have a strong dependency on the life cycle of container class.

⑤ **Generalization** - "is a relationship"



*. UML - Modelling Tyes :

I. STRUCTURAL MODELING :

- classes diagrams
- object diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagrams
- Component diagram

II. BEHAVIORAL MODELING :

- Activity diagrams
- Interaction Diagrams
- State Machine Diagram
- UseCase

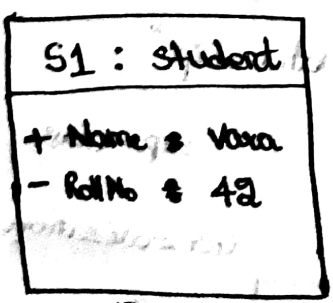
II. ARCHITECTURAL MODELING : (Blue print of entire system)

- Package diagrams

*. OBJECT DIAGRAM :

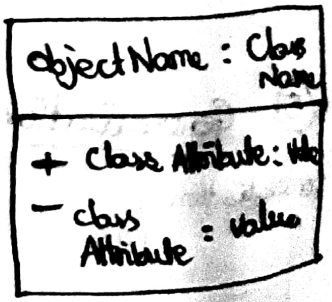
- It is like a snapshot of your instances or objects of the system and the relationship that exists between these objects is represented by object Diagram. (at a particular instance)**

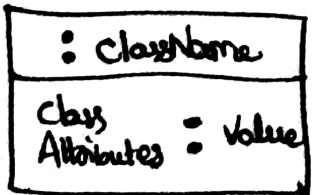
Ex :



Regular object

Structure ≈





→ Anonymous objects

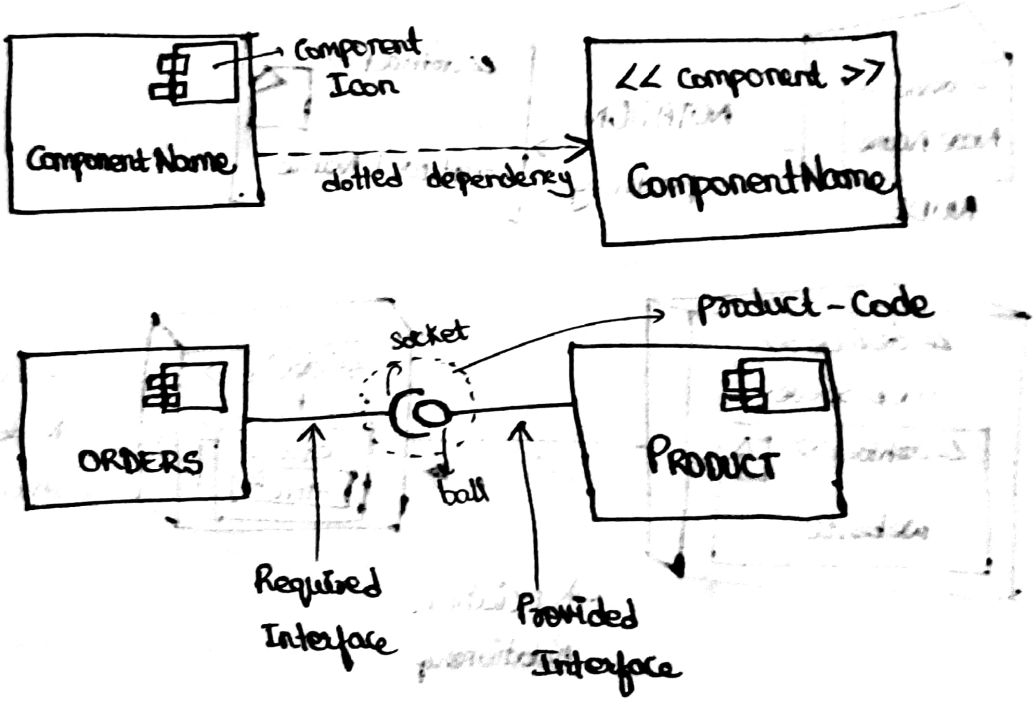
*. COMPONENT DIAGRAM :

⇒ The purpose of the component diagram can be :

- Visualize the components of system
- construct executables by using forward & reverse engg.
- Describe the organization & relationships of the components.

⇒ Component diagrams can be used to :

- Model the components of system
- Model the database schema
- Model the executables of an application.
- Model the system's source code.



*. DEPLOYMENT DIAGRAM :

⇒ The purpose of deployment diagram is :

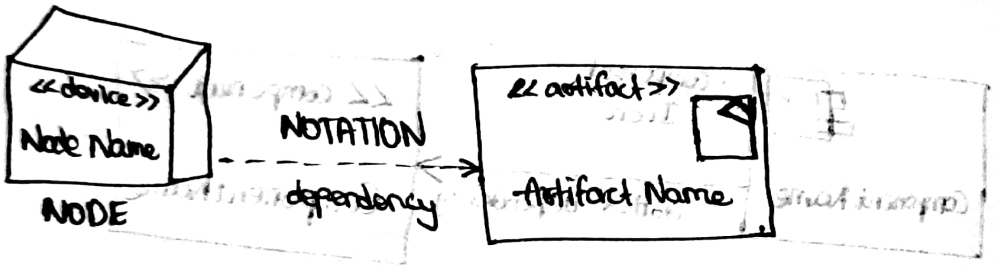
- Visualize the hardware topology of system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing models.

→ Uses of Deployment Diagram :

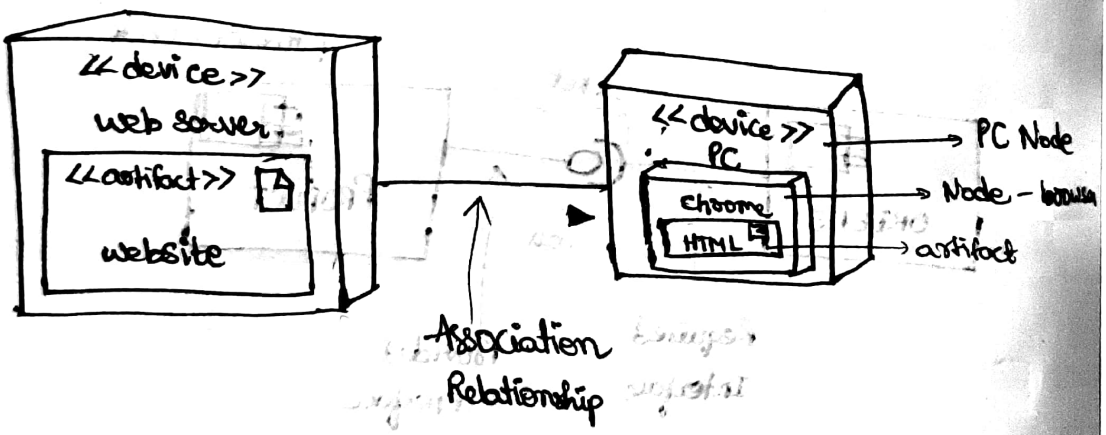
- To model the hardware topology of a system.
- To model the embedded system.
- To model the hardware details for client/server sys
- To model the hardware details of distributed application
- for Forward & Reverse engineering.

→ & Basic building blocks in Deployment diagram

1) Artifacts - which are deployed on Nodes (Tables, files
2) Node - powerful computational unit to execute Artifacts.



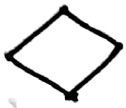
Ex :



II. BEHAVIOURAL MODELLING :

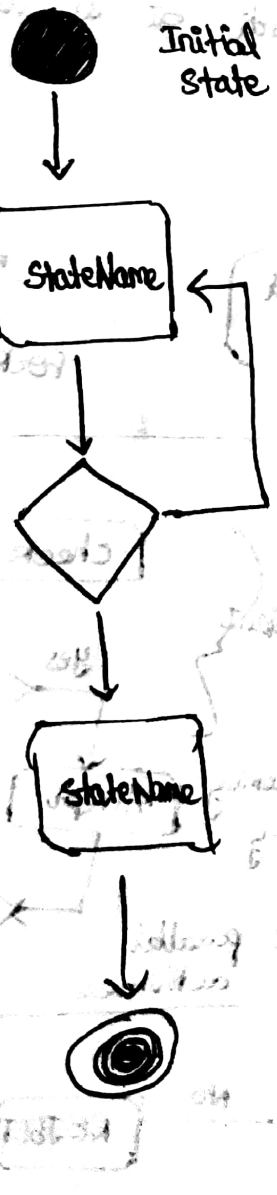
*. STATE MACHINE DIAGRAM: (also called Statechart/State dig)

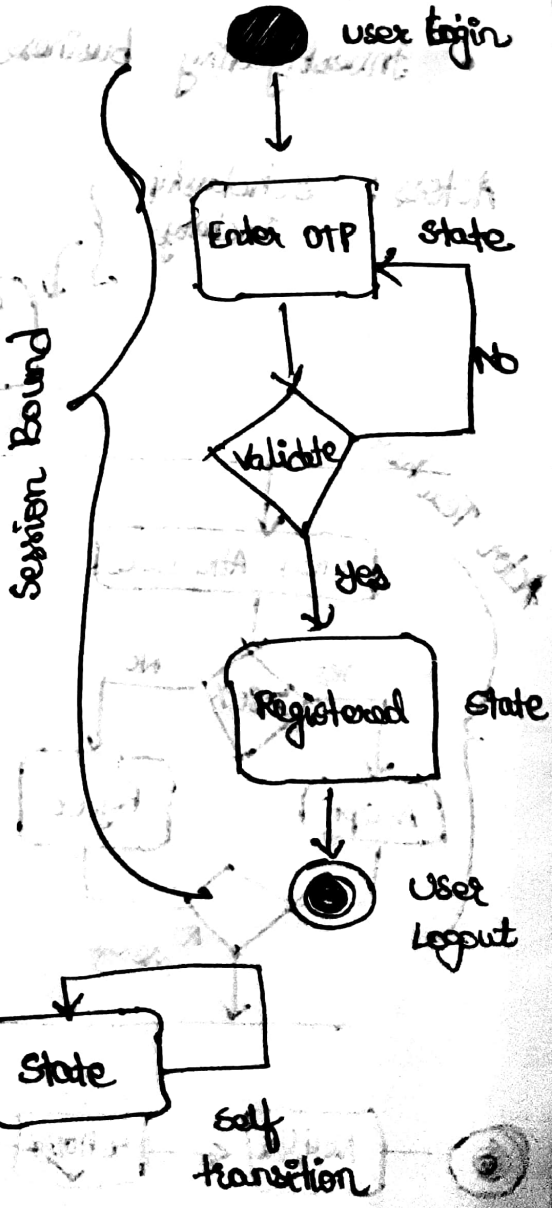
- Dynamical aspects are (visualized, represented, depicted of your system) captured of a system through this diagram.

- State - a movement in life cycle of object.
  Represented as  .
- Event - change of states when triggred
- Transition - change of state (direction)
- Initial state -  : starting point of everything.
- Final/End state -  : end point of everything.
- Decision Box -  : works like if/else

{ Notation of Sample state Machine Diagram }

{ Example }





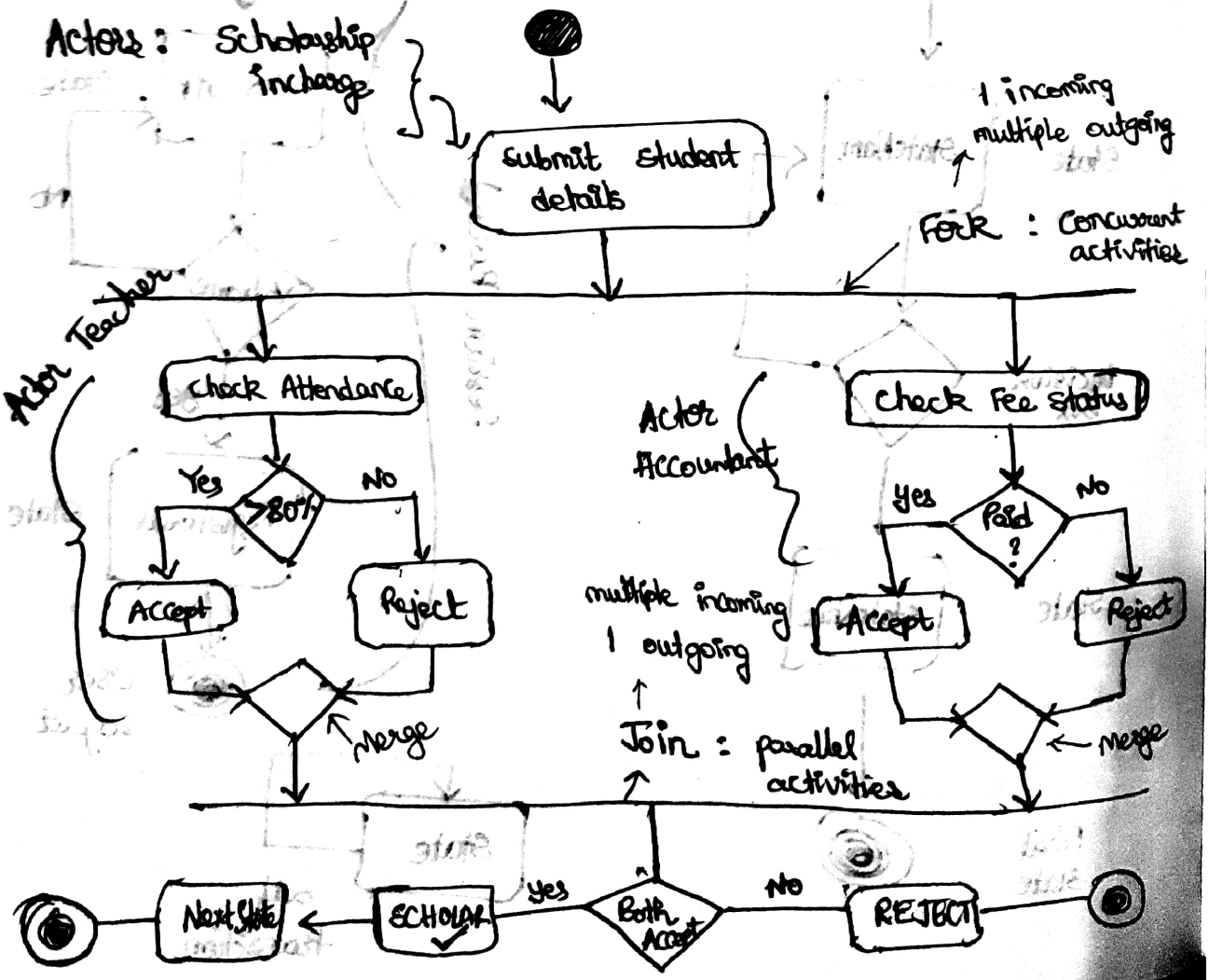
*. ACTIVITY DIAGRAM :

→ The purpose of activity diagram is :

- Draw the activity flow of system.
- Describe sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

→ Use of Activity Diagram :

- Modelling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.



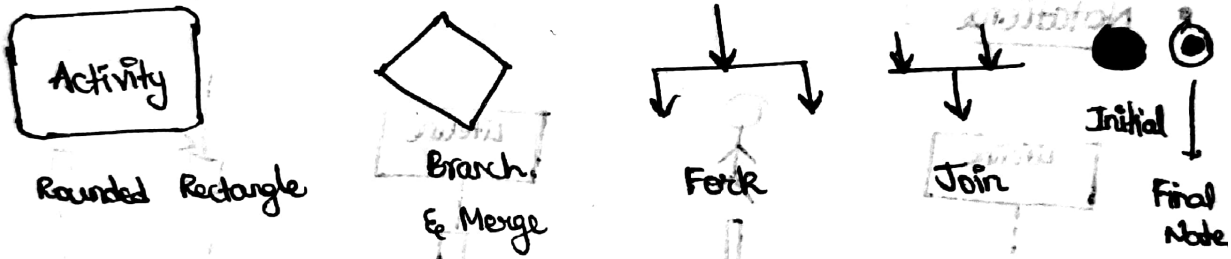
List of Diagrams in Syllabus :

1. Scenario Modelling → Use Case, Activity Diagram, Swimlane Diagram
2. Class Modelling → Class diagram
3. Flow oriented Modelling → Data Flow Diagram
4. Behavioural Modelling → State chart Diagram, Sequence Diagram

Analysis Models
UML Rules

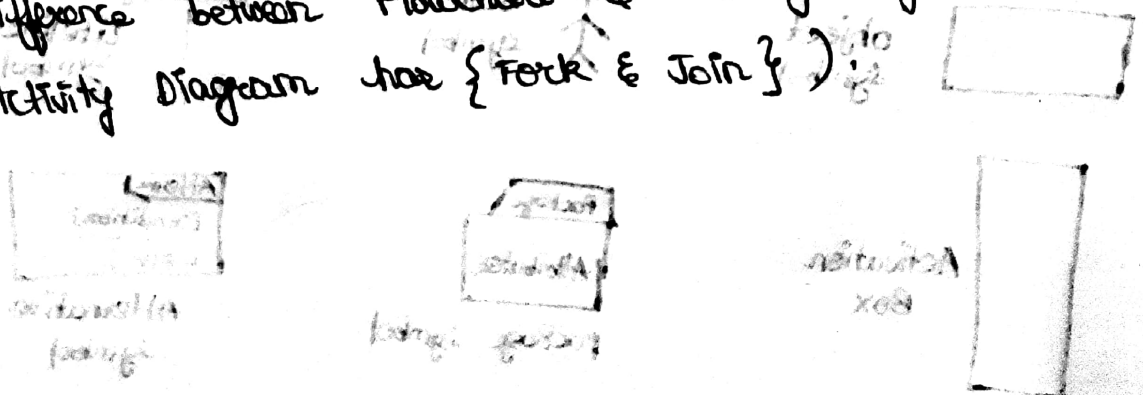
*. SWIMLANE ACTIVITY DIAGRAM :

(Syntax in Activity)



- Swimlane Activity Diagram is the flavour of Activity Diagram, which also give information about which Role is performing the underlying Activity.
- Activity Diagram is divided into multiple columns and each column is dedicated to a particular Role.

(Difference between Flowchart & Activity diagram is that Activity Diagram has {Fork & Join}) :

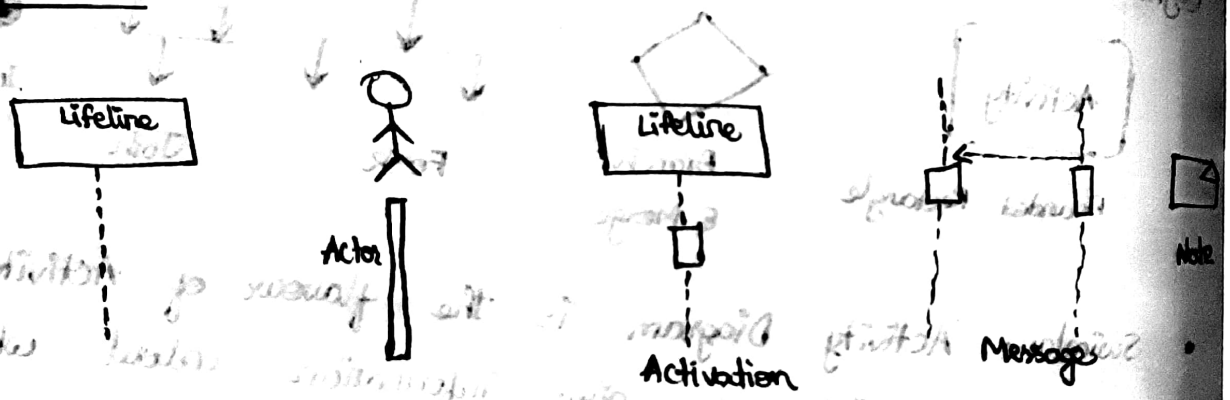


*- SEQUENCE Diagram :

- The sequence Diagram represents the flow of messages in the system and it is also termed as event diagram.
- Purpose of sequence Diagram :

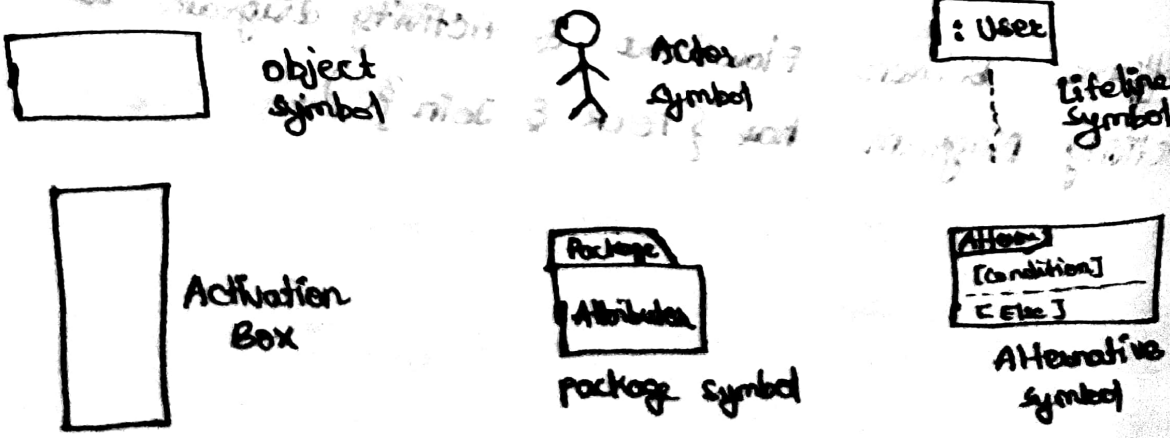
1) To model high-level interaction among active objects within a system.
2) To model interactions among objects inside a collaboration realizing a use case.
3) It either models generic interactions or some certain instances of interaction.

- Notations :



Example : (High-level sequence diagram of online Bookshop)

- Any online customer can send/search a book catalog, view a description of a particular book, add a book to its shopping cart, and do checkout.



- **Advantages :**

+ It explores real-time application.
+ Depicts message flows between objects.
+ Easy maintenance / Easy to Generate.
+ Implement both Forward, Reverse Engg.,
+ It can easily update as per change.

- **Disadvantages :**

- Too many lifelines
- If order of flow change
- More Diagram complex
- Varies for type of sequence
- Too time taking

- **Use Cases of Sequence Diagram**

- Usage scenario, Method Logic, Service Logic, Sequence diagram Visio.

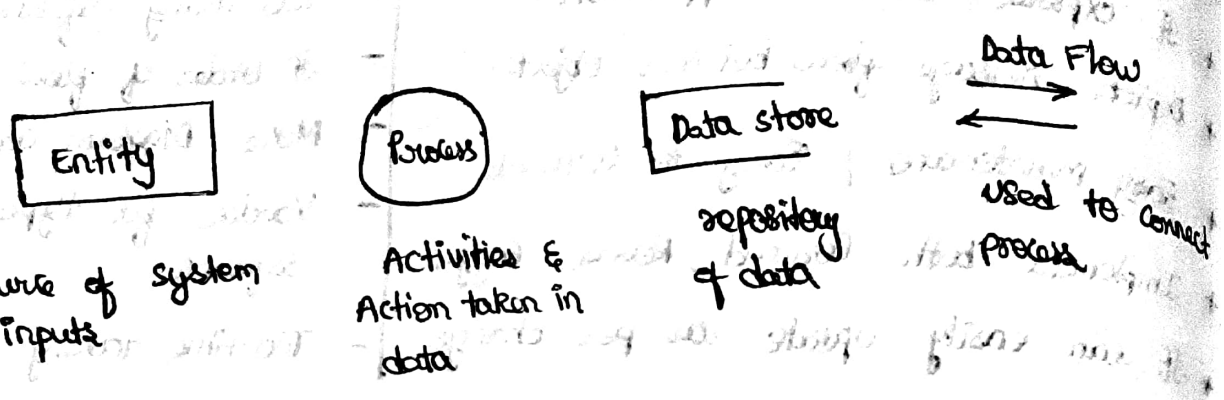
*. DATAFLOW DIAGRAM : (IMP)

- A Dataflow Diagram is a traditional visual representation of information flows within a system.
- A neat and clear DFD can depict the right amount of the system requirement graphically.
- There is a prominent difference between DFD & Flowchart.
- Flowchart depicts flow of control in program modules.
- DFD depicts flow of data in the system at various levels. DFD doesnot contain any control / branch elements.

- **Types of DFD :**

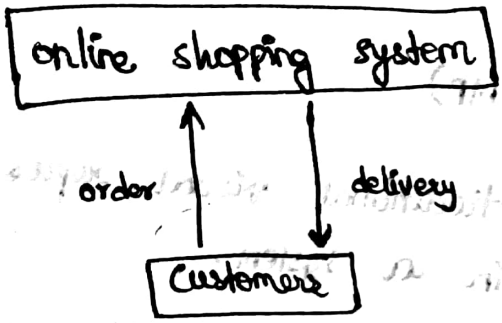
1) Logical DFD - System process + flow of data in system.
2) physical DFD - data implementation in system.

- DFD Components :

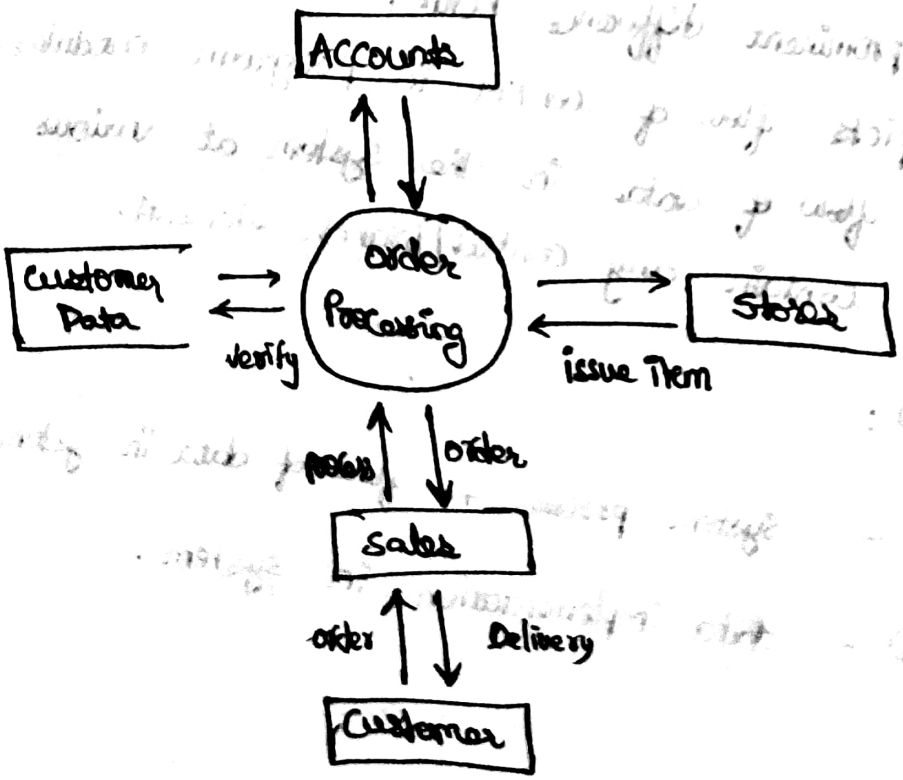


- Levels of DFD :

+ Level 0 : highest abstraction level (represents entire info of one diagram)



+ Level 1 : flow of data among various modules. Mentions basic process & source of information.



+ Level 2 : At this level, DFD shows how data flows inside the modules mentioned in level 1.

- Structure Charts :

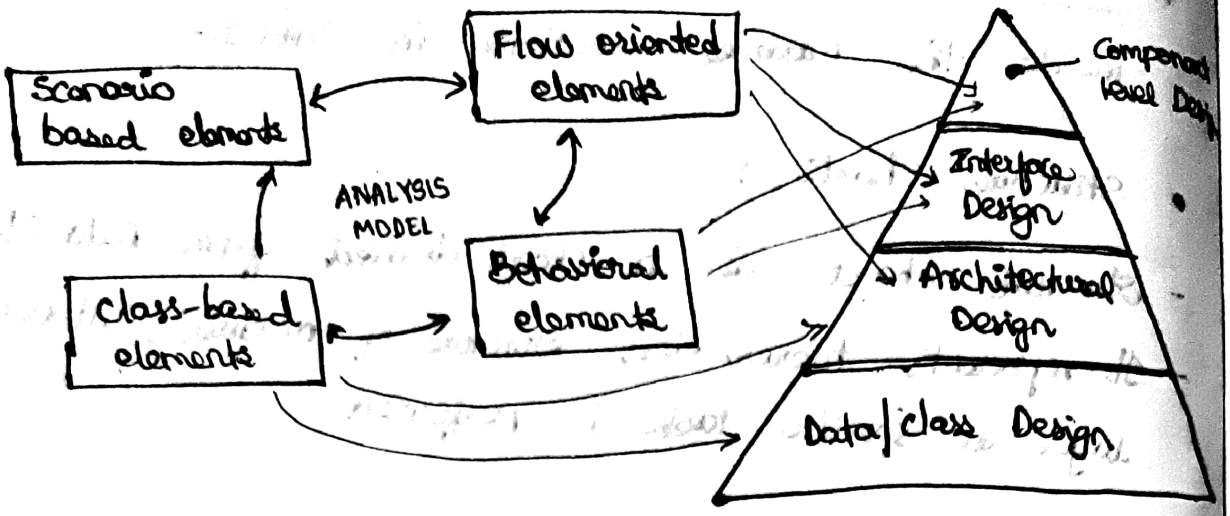
- Structure chart is a chart derived from Data Flow.
- It represents hierarchical structure of modules. At each layer a specific task is performed.

} Not in Syllabus

---

* DESIGN CONCEPTS :

- Software design encompasses the set of principles, concepts and practices that lead to the development of high-quality system or product.
- The goal of design is, to produce a model or representation that exhibits : Firmness, Commodity and Delight. Software design changes continually, as new methods, better analysis, and broader understanding evolve.
- Software design is the last software engineering action within the modeling activity and sets the stage for construction (coding & testing).
- The requirements models, manifested by :
  ① Scenario based ② class-based ③ Flow-oriented ④ behavioural elements feed the design task.



DESIGN MODEL

- Software design is transforming requirements into "blueprint" for constructing the software.
- The design is represented as high level of abstraction.
- Software Quality Guidelines :

1) The design must implement all explicit requirements contained in the requirements model.

2) The design must be a readable, understandable guide for those who generate code.

3) The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

*. Quality Attributes :

- Hewlett-Packard developed a set of software quality attributes: {FURPS}.
- Functionality, Usability, Reliability, Performance and Supportability.

CSE1005 : SOFTWARE ENGINEERING (Slot F+TF)

G. VARAPRASAD
19BCE7048

ASSIGNMENT – 1

*. Consider the process of stock Maintenance system is that customer login to the particular site to place the order for the customer product. The stock maintenance system is described sequentially through steps. The customer login to the particular site.

- They fill the customer details
- They place the orders for their product.
- The vender login and views the customer details and orders.
- Outline an object - oriented analysis & design process based on this scenario
- How this scenario will be used in Bottom-up implementation.
- Admin can enter their stock details purchase, sales, maintaining report
- After customer chooses the demands and lists them, the application interface will change into a real-time camera which is capable of recognizing the interesting objects such as crayon labels, including the desired products, products themselves or employees of market.
- Customers should select the item from the shop.
- Vender makes the bill for the selected item.
- Customer gives the credit card to vender to swap the card.
- They required amount transactions is done by the card reader.
- Vender will issue the balance statement to the customer.

Consider all the above requirements, sketch its UML with all possible diagrams and design a solution using one or more design patterns.

UML :

- UML (Unified Modeling Language) is a general purpose graphical modeling language in the field of Software Engineering.
- UML is used to specify, visualize, construct and document the artifacts (major elements) of the software system.
- UML has following features like :
  + It is a generalized modeling language.
  + It is distinct from other programming languages.
  + It is interrelated to object-oriented analysis & design.
  + It is used to visualize the workflow of the system.
  + It is a pictorial Modeling language, useful to generate artifacts.

- Analysis Models (UML Rules) :

1. Scenario Modeling → Usecase.
   → Activity Diagram.
   → Swimlane Diagram.
2. Class Modeling → Class diagram.
3. Flow oriented Modeling → Data Flow diagram.
4. Behavioural Modeling → Statechart Diagram.
   → Sequence Diagram.

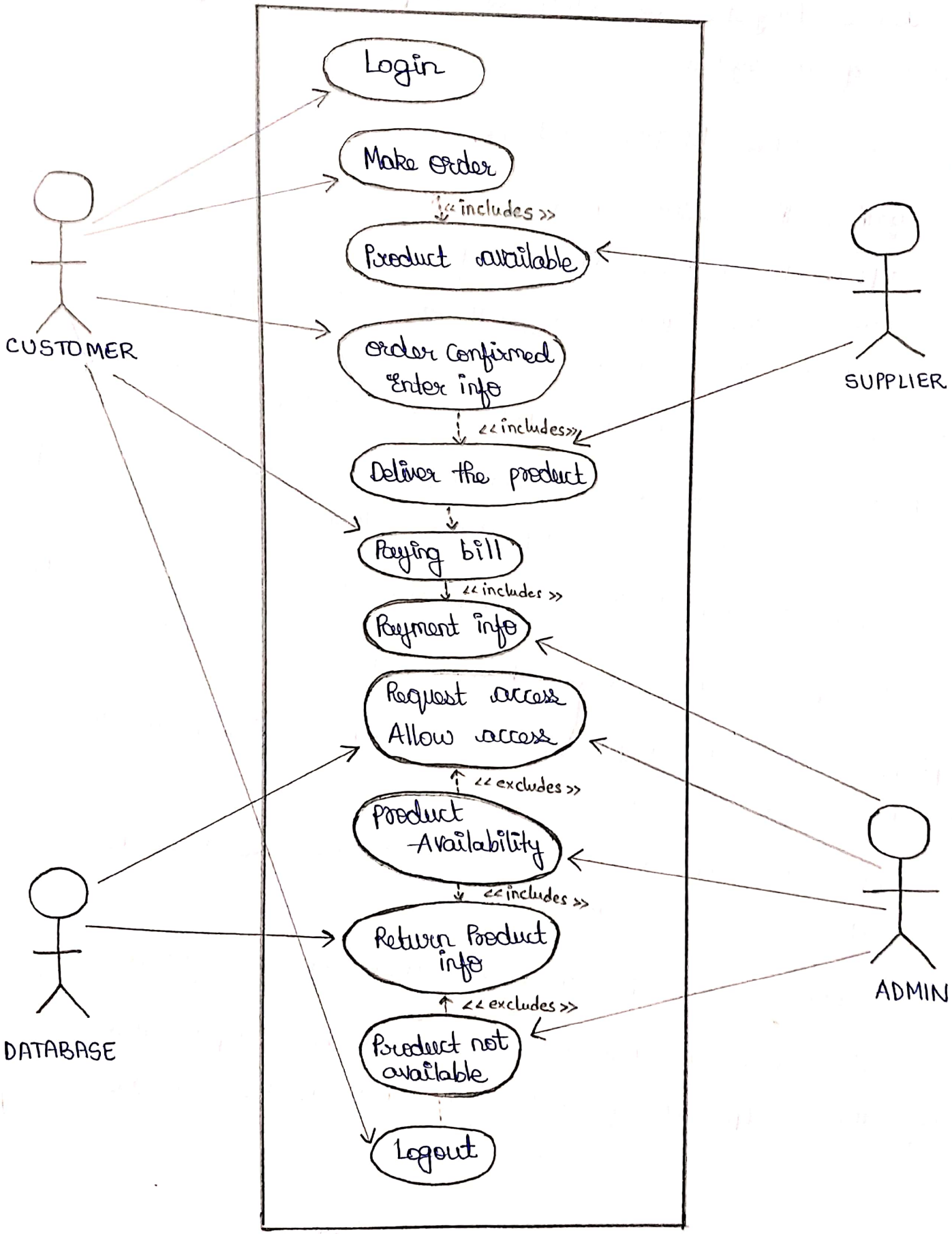
Now, considering an Example - "STOCK MAINTENANCE SYSTEM", we will try to implement some of the UML diagrams.

USECASE DIAGRAM :

- UML Usecase Diagram is used to represent the dynamic behaviour of a system.
- Purpose of Use Case Diagram includes :
  1. It gathers the system's needs.
  2. It depicts the external view of the system.
  3. It recognizes the internal as well as external factors that influence the system.
  4. It represents the interaction between the actors.
- UML provide Use case diagram notation to illustrate the names of use case and author relationship between them.
- Here actors involved in this system are customer, supplier, etc...
- Rules to draw UML UseCase Diagram includes :

1. A pertinent and meaningful name should be assigned to the actor or a usecase of a system.
2. The communication of an actor with a use case must be defined in an understandable way.
3. Specified notations to be used as and when required.
4. The most significant interactions should be represented among the multiple no. of interactions between the usecase & actors.

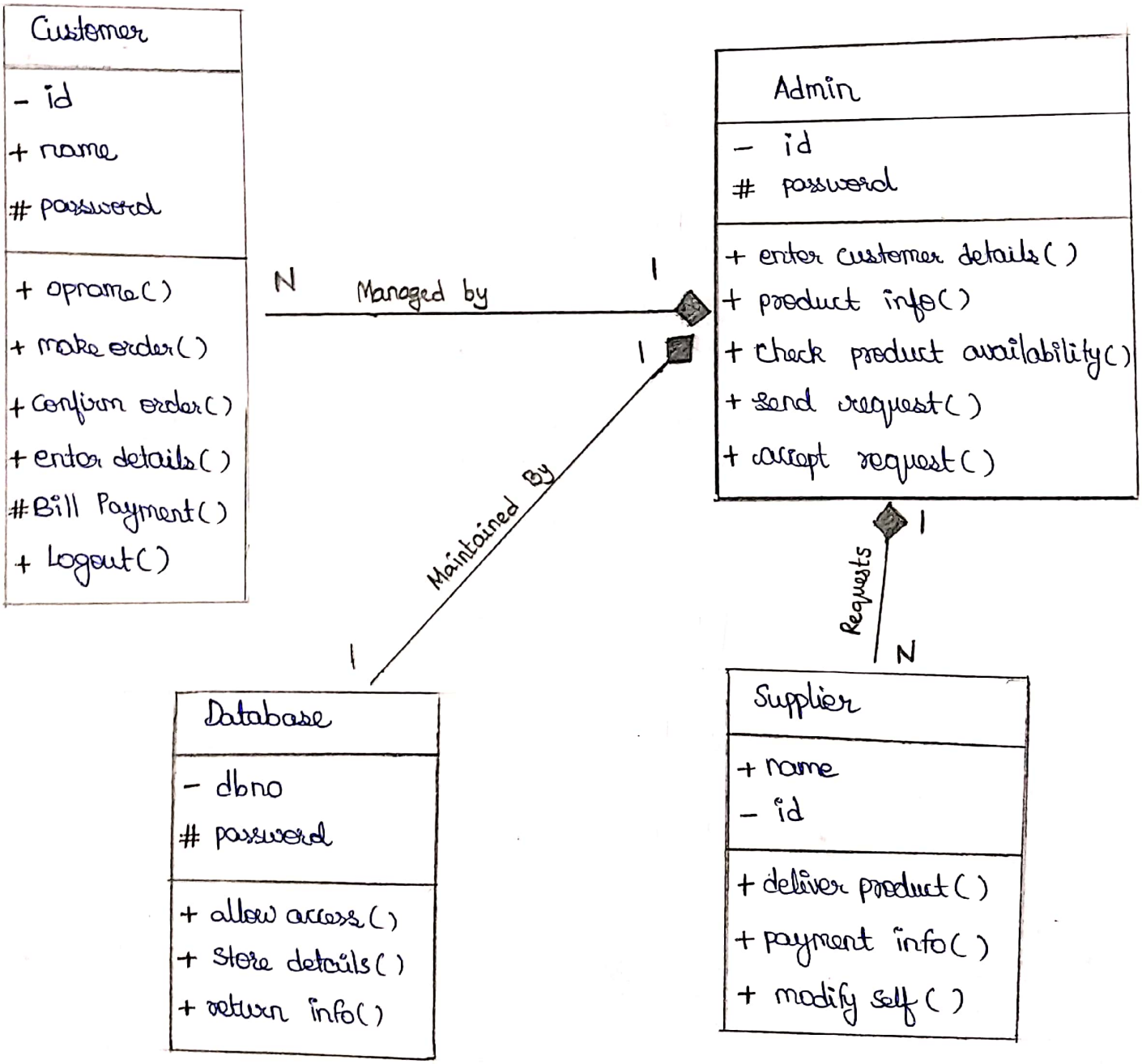
STOCK MAINTENANCE SYSTEM – USECASE



CLASS DIAGRAM :

- A Class Diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and relationships between the classes.
- It is represented using a rectangle with three compartments. Top compartment have the class name, middle compartment has the attributes and the bottom compartment with operations.

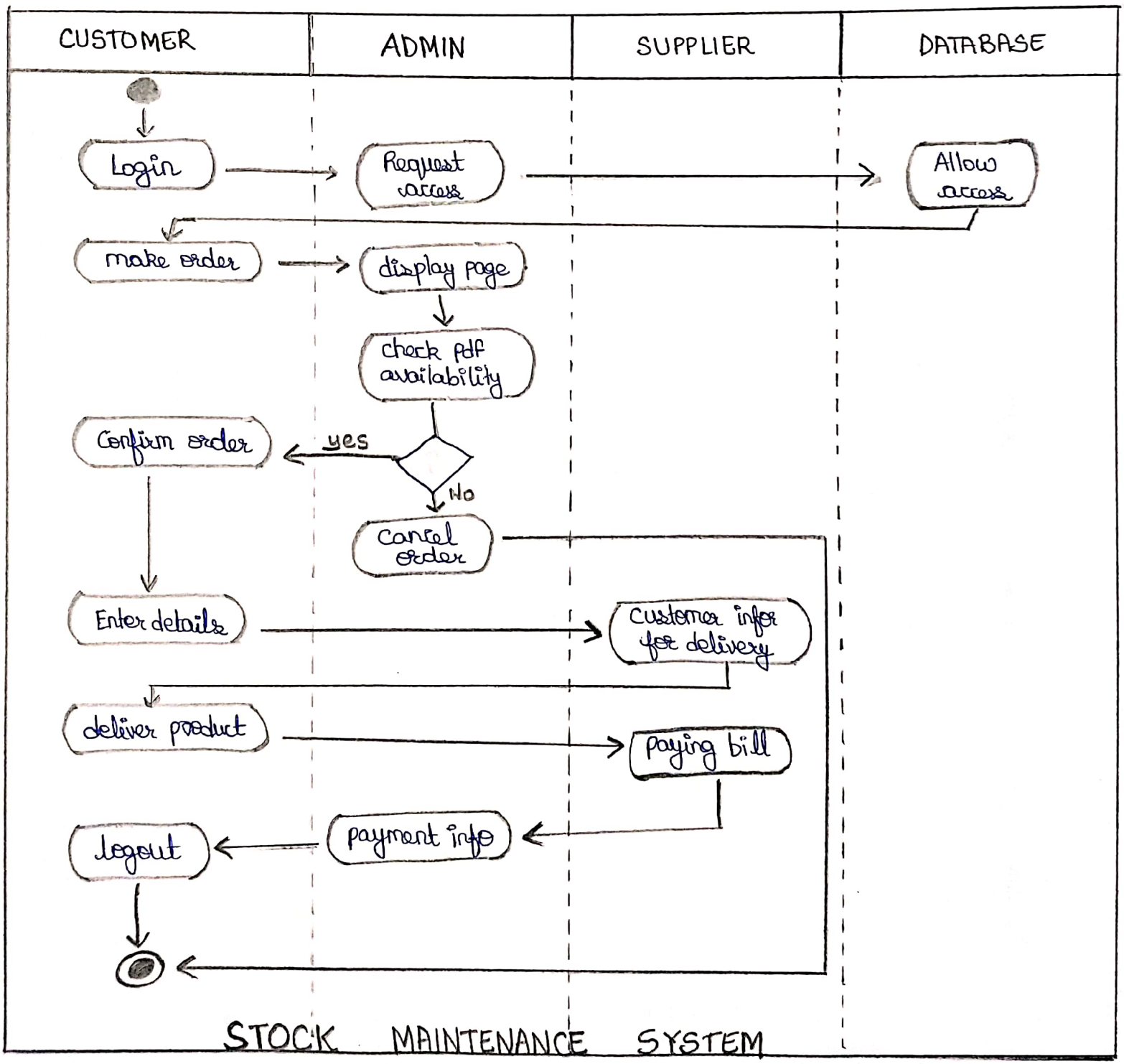
STOCK MAINTENANCE SYSTEM



CLASS DIAGRAM

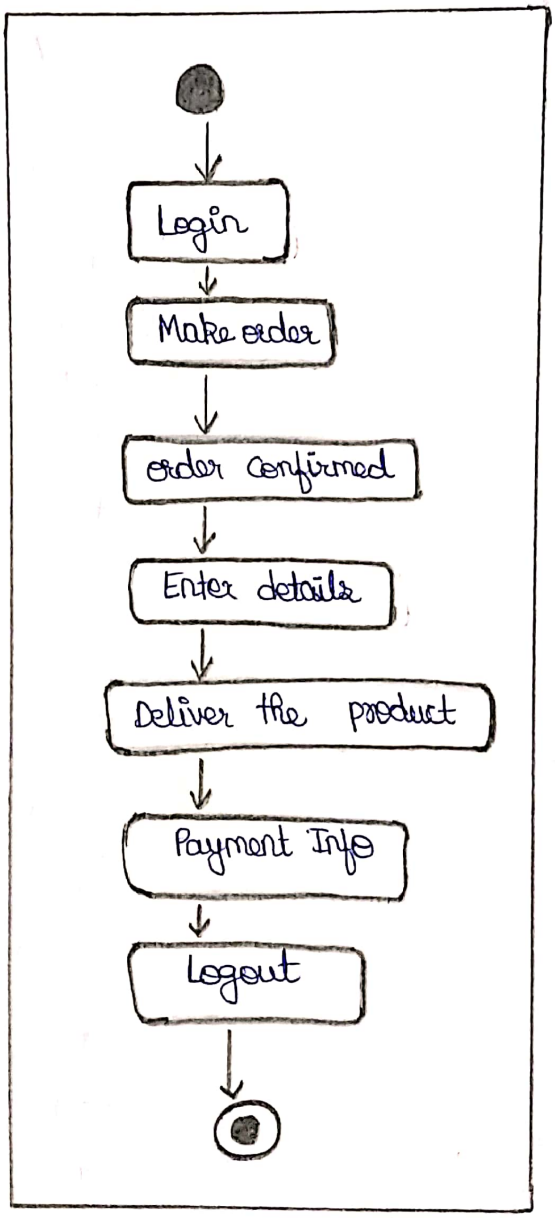
ACTIVITY DIAGRAM : (Swimlane Diagram)

- The activity Diagram shows the activity of the process.
- An activity diagram is a variation or a special case of a state machine diagram in which the states or activity representing the performance of operation and transition are triggered by the completion of operation.
- The purpose is to provide view of flows and what is going on inside a use case.



STATE CHART DIAGRAM :

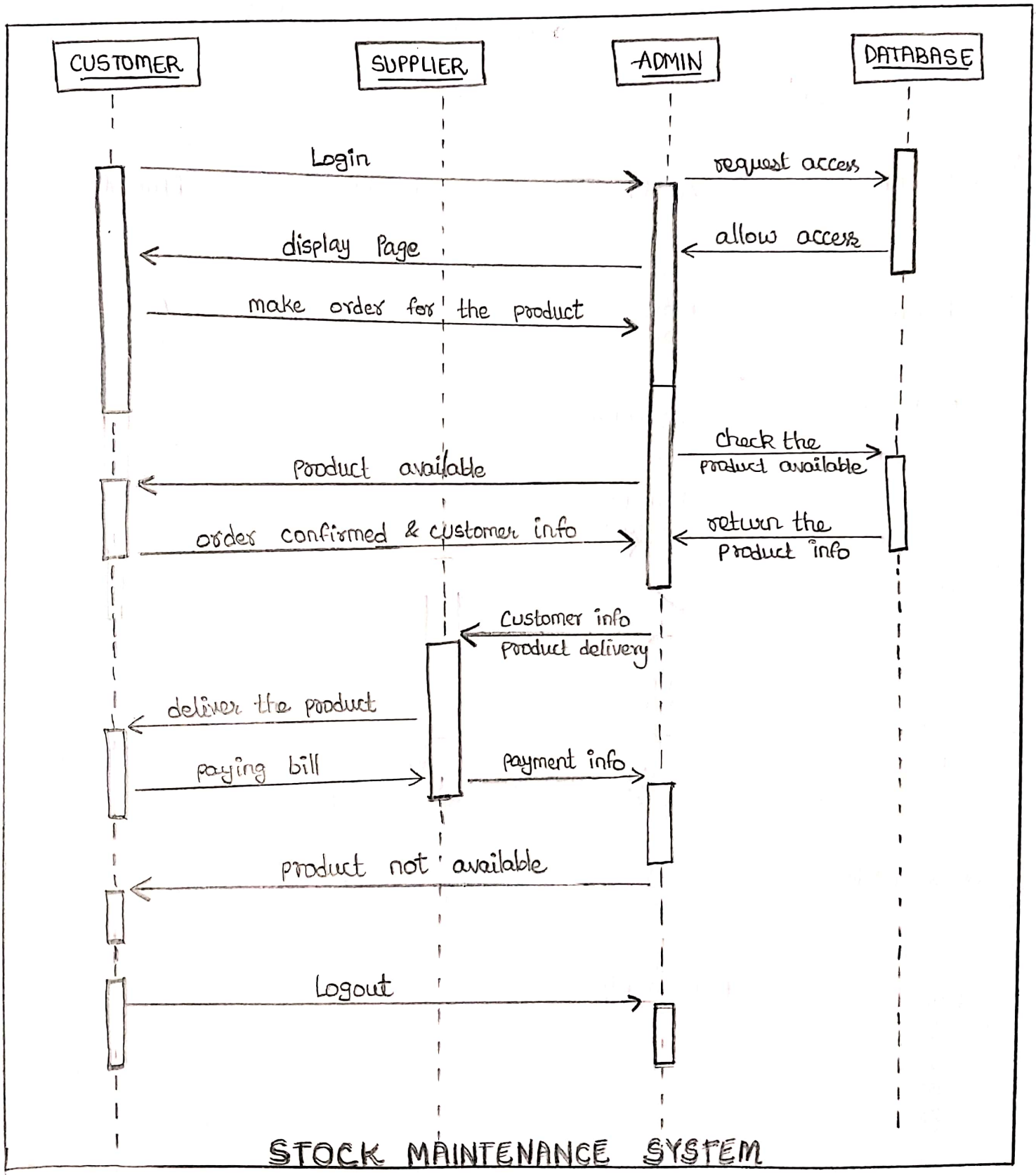
- The state chart diagram in UML diagram defines the states available in the system. It has various states in the system.
- It has one starting state & one ending state. State chart diagram is a curved rectangle box diagram.
- The various states in stock maintenance system are login, make order, confirm order, enter details, deliver the product, payment info and logout.



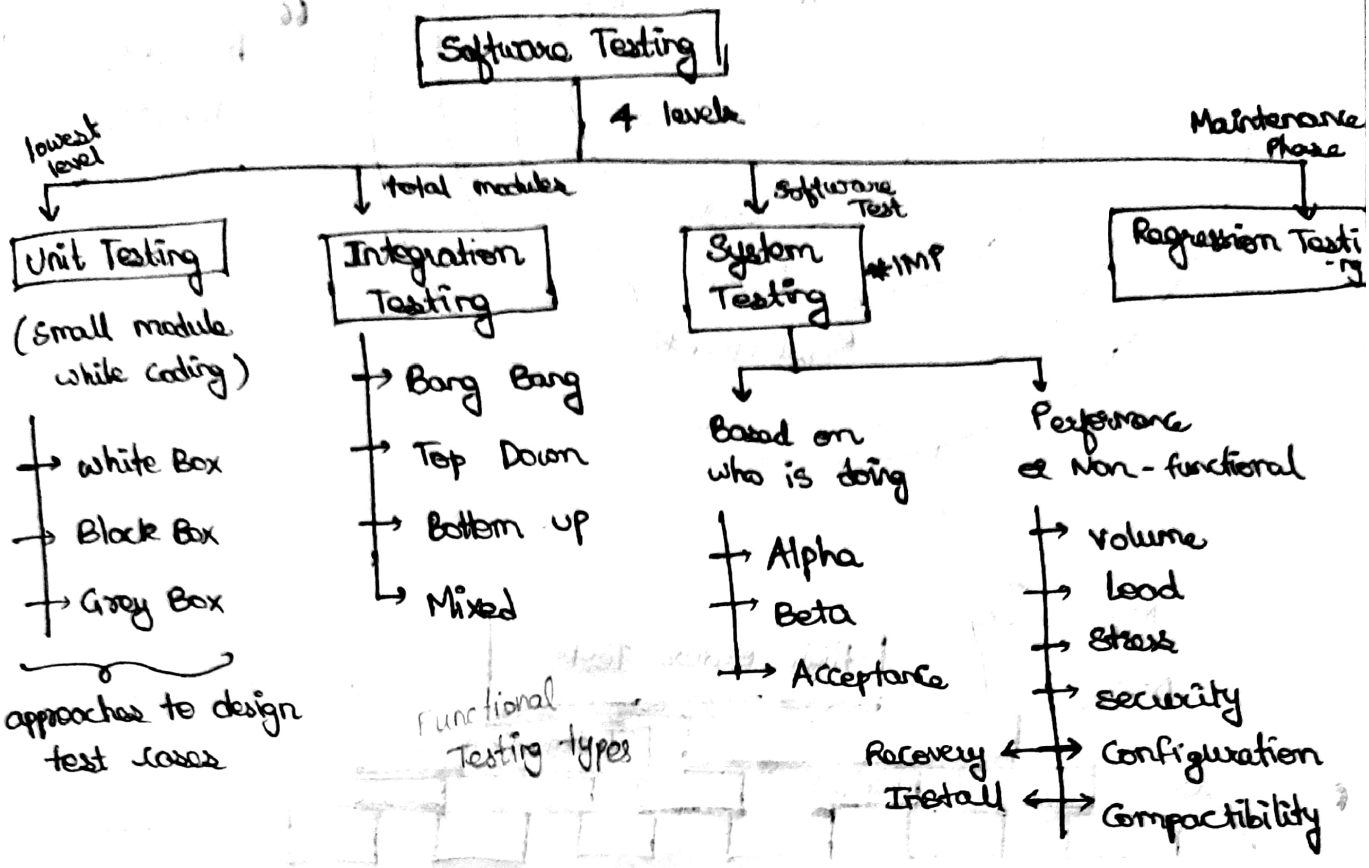
STOCK MAINTENANCE SYSTEM – STATE CHART

SEQUENCE DIAGRAM :

- A sequence diagram shows an interaction arranged in time sequence.
- It shows object participating in interaction by their lifeline by the message they exchange arranged in time sequence.
- Vertical dimension represent time and horizontal dimension represent object.

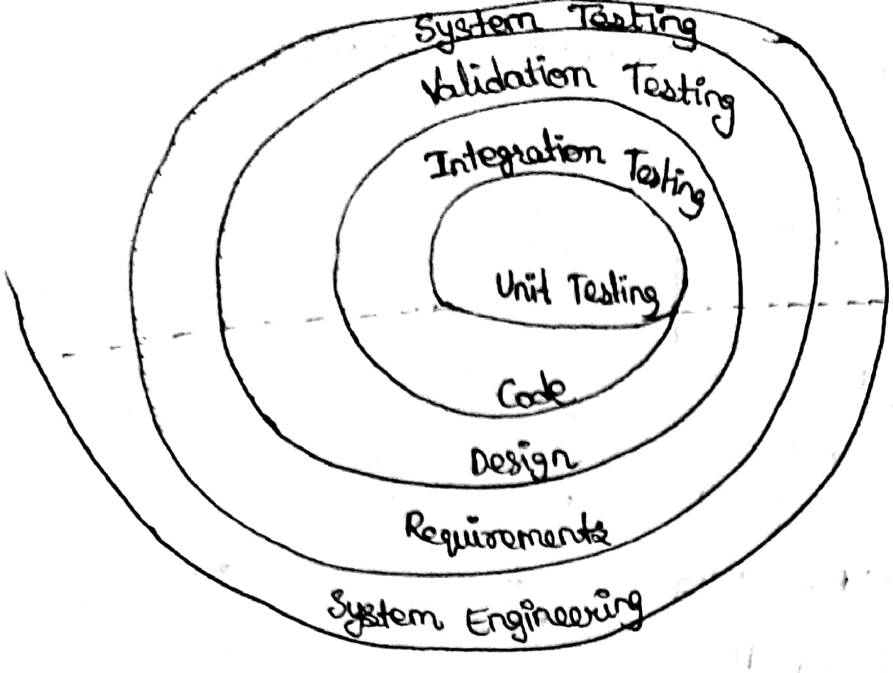


MODULE - 3 : SOFTWARE TESTING

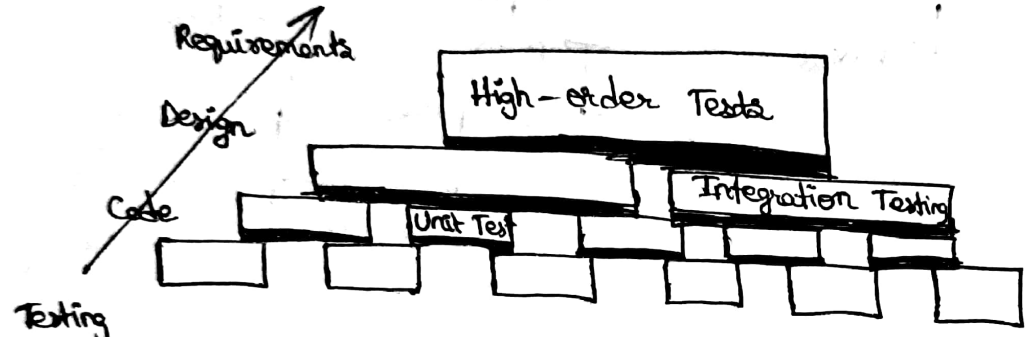


*. Verification (vs) Validation :

VERIFICATION	VALIDATION
1. Are you building it right ?	1. Have you built the right thing?
2. Check whether an artifact confirms its' previous artifact	2. Check the final product against specification. (SRS)
3. Done by developer	3. Done by Tester.
4. Concerned with phase containment of errors.	4. Aim is to make final product error free.
5. Method involves Review, Inspection, integration Testing	5. Involves System Testing.
6. Static & Dynamic Activities	6. Only Dynamic Activities.
7. In process / In every phase	7. After entire process / phase.



" TESTING STRATEGY



SOFTWARE TESTING STEPS

- The clean room software engineering approach suggests statistical use techniques that execute a series of tests derived from a statistical sample of all possible program executions by all users from a targeted population.

* . STRATEGIC ISSUES :

1. Specify product requirements in quantifiable manner long before testing commences.

2. state testing objectives explicitly.

3. Understand the users of the software and develop a profile for each user category.

4. Develop a testing plan that emphasizes "rapid cycle testing".

5. Build "robust" software that is designed to test itself.

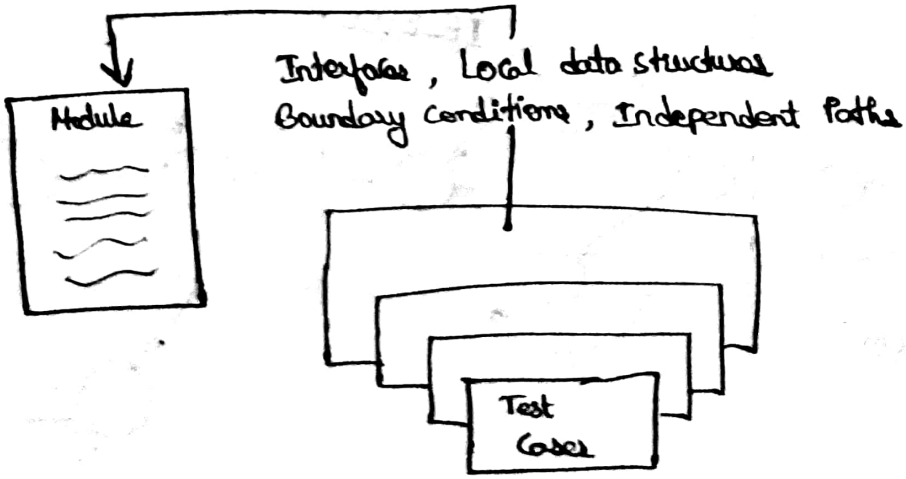
6. Use effective technical reviews as a filter prior to testing.

7. Conduct technical reviews to assess the test strategy and test cases themselves.

8. Develop a continuous improvement approach for the testing process.

*. UNIT TESTING :

- Unit Testing focuses verification effort on the smallest unit of software design.



TESTING in Syllabus :

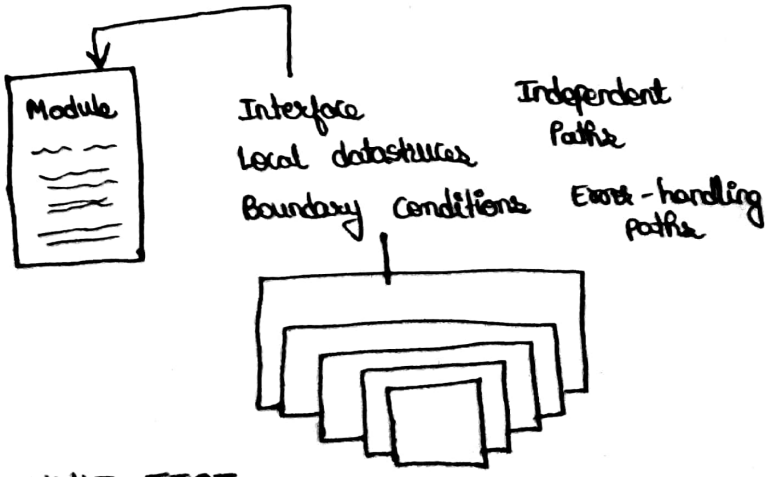
1. Unit Testing – Behavioural Testing (Black box, White Box)
2. Integration Testing – Top down, Bottom up, Smoke Testing
3. Regression Testing
4. Black Box – Graph-based, Equivalence Partioning
   Boundary value Analysis, Orthogonal Array
5. White Box
   - Basic Path Testing (exam)
     - Flow Graph Notation
     - Independent Program Paths
     - Deriving Test Cases
     - Graph Matrices

     } exam
   - Control Structure Testing
     - conditional Testing
     - Data Flow Testing
     - Loop Testing

UNIT TESTING :

- Unit Testing focuses verification effort on the smallest unit of software design. The unit Test focuses on the internal processing logic and data structures within the boundaries of a component.
- This type of testing can be conducted in parallel for multiple components.
- Unit-Test considerations: Unit Tests are illustrated schematically in following figure. The module interface is tested to ensure to that information schematically (in following figure) properly flows into and out of the program out unit under test.
- Local data structures are examined to ensure the data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.

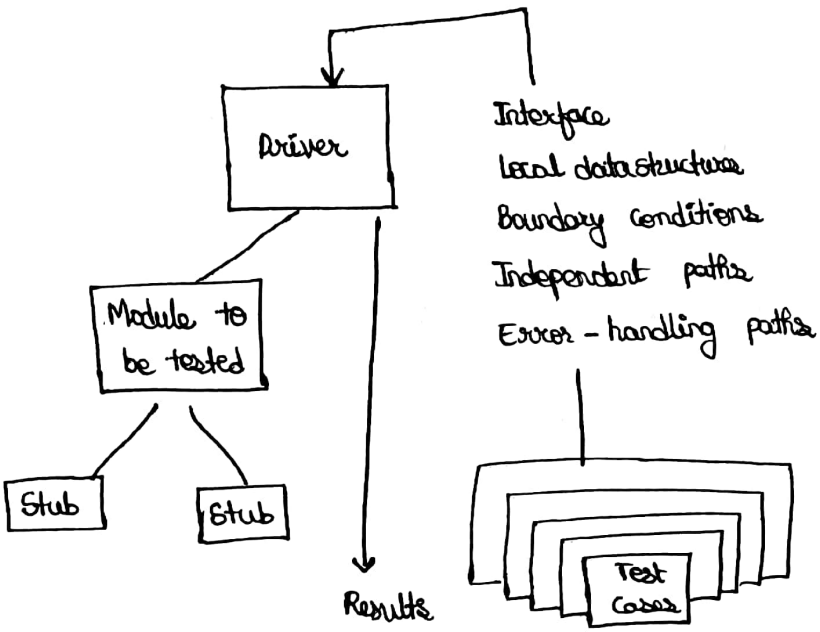
And finally, all error-handling paths are tested.



UNIT TEST

- Selective testing of execution paths is an essential task during the unit Test.
- Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisions, or improper control flow.
- Boundary testing is one of the most important unit testing tasks. Software often fails at its Boundaries. That is, errors often occur when the nth element of an n-dimensional array is processed, when the ith repetition of a loop with ith pass is invoked, when the maximum or minimum allowable value is encountered.
- A good design anticipates error conditions and establishes error-handling paths to reroute to cleanly terminate processing when an error does occur. This approach is "antibugging."
- Among the potential errors that should be tested when error handling is evaluated are:
  i. error description is unintelligible.
  ii. error noted doesnot correspond to error encountered.
  iii. error condition causes system intervention prior to error handling.
  iv. exception - condition processing is incorrect, (or)
  v. error description doesnot provide enough information to assist all in the location of the cause of error.

- UNIT TEST PROCEDURES :
- Unit Testing is normally considered as an adjunct to the coding step. The design of coding unit Tests can occur before coding begins or after source code has been generated.



UNIT TEST ENVIRONMENT

- The unit Test environments is illustrated in above figure. In most applications a driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate (invoked by) the component to be tested.
- Unit Testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the no. of test cases is reduced and errors can be more easily predicted and uncovered.
- Drivers and stubs represent testing "overhead". That is, both are software that must be coded (formal design is not commonly applied) but that is not delivered with the final software product.
- If drivers and stubs are kept simple, actual overhead is relatively simple / low.

Advantages of Unit Testing:

- It eliminates the dependencies on other modules.
- Allows testing of individual parts of software
- Validates the quality and accuracy of the program code.
- Team can execute unit testing before the culmination of the development process.
- In Unit Testing the code is modular, which makes them more reusable.
- Reduces the cost of testing and increases the reliability of code.

Disadvantages of Unit Testing:

- Unit Testing cannot be used for a comprehensive detection of errors.
- It is not possible for the team to analyze and evaluate every execution path in the software.
- There can be difference in platform, as the product is not developed and tested on the platform where it will be developed / deployed.

BLACK-BOX TESTING :

- Any engineered product can be tested in one of two ways.
- The first test approach takes an external view and is called black-box testing.
- Black-box testing alludes to tests that are conducted at the software interface.
- A black-box test examines some fundamental aspect of the system with little regard for the internal logical structure of the software.
- Black-box testing also called behavioral testing, focuses on the functional requirements of the software.
- That is, "black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a program."
- Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.
- Black-box testing attempts to find errors in the following categories : (1) incorrect or missing functions,
  (2) interface errors.
  (3) errors in data structures or external database access.
  (4) behavior or performance errors, and
  (5) initialization and termination errors.

- Tests are designed to answer the following questions:

i. How is functional validity tested?

ii. How are system behavior and performance tested?

iii. What classes of input will make good test cases?

iv. Is the system particularly sensitive to certain input values?

v. How are the boundaries of data class isolated?

vi. What data rates and data volumes can the system tolerate?

vii. What effect will specific combinations of data have on system operation?

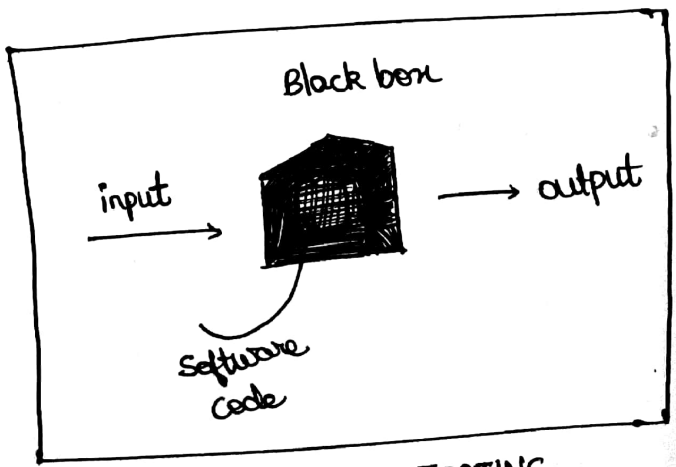
- By applying black-box techniques, you derive a set of test cases that satisfy the following criteria:

(1) test cases that reduce, by a count that is greater than one, the no. of additional test cases that must be designed to achieve reasonable testing, and

(2) test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

- Black Box Testing techniques:

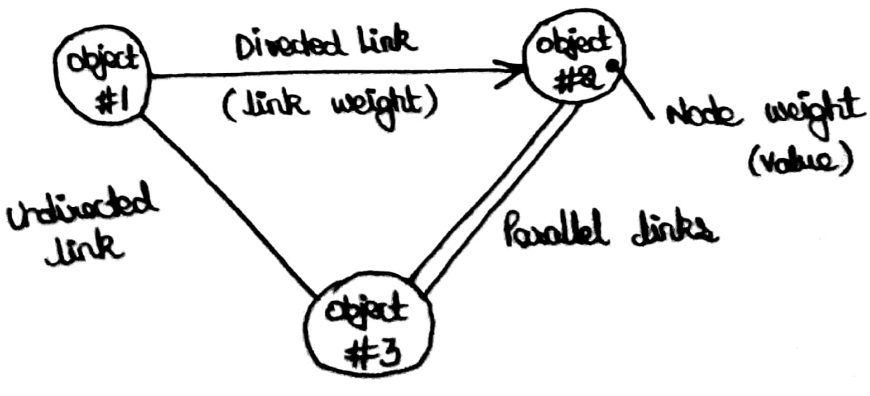
1. Graph-based Testing Methods.
2. Equivalence Partitioning.
3. Boundary Value Analysis.
4. Orthogonal Array Testing.



BLACK BOX TESTING

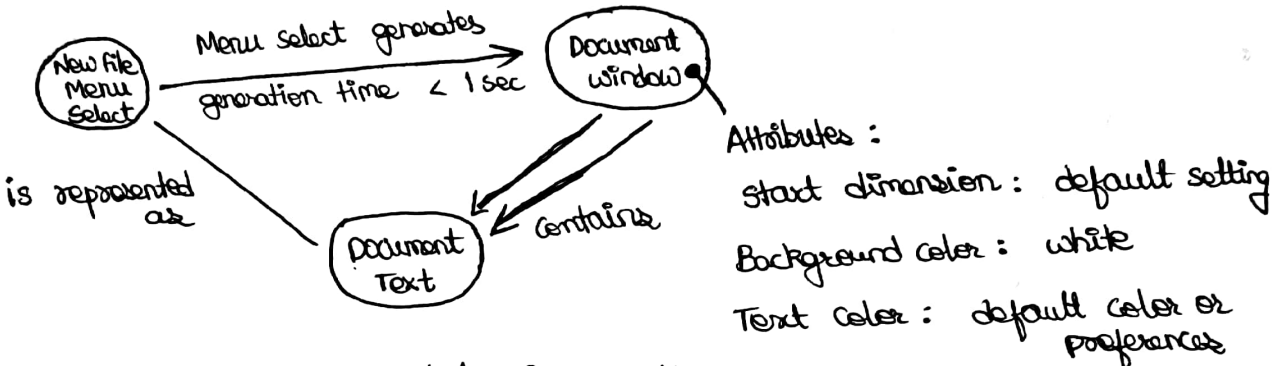
GRAPH - BASED TESTING :

- The first step in black-box testing is to understand the objects that are modeled in software and the relationships that connect these objects.
- Once this has been accomplished, the next step is to define a series of tests that verify "all objects have the expected relationship to one another".
- Stated in another way, software testing begins by creating a graph of important objects and their relationships and then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.
- To accomplish these steps, you begin by creating a graph, it is a collection of nodes that represent objects, links that represent the relationships between objects, node weights that describe the properties of a node, and link weights that describe some characteristic of a link.
- The symbolic representation of graph is shown in the below figure. Nodes are represented as circles connected by links that take a no. of different forms.



- A directed link (represented by an arrow) indicates that a relationship moves in only one direction.
- A bi-directional link, is also called symmetric link, implies that the relationship applies in both directions.
- Parallel links are used when a no. of different relationships are established between graph nodes.

Example :



- Beizer describes a no. of behavior testing methods that can make use of graphs :

1. Transaction Flow Modeling.
2. Finite state modeling
3. Data flow modeling.
4. Timing modeling.

- Each and every application is a bilild-up of some objects. All such objects are identified and the graph is prepared.
- From this object graph, each object relationship is indentified and test cases are written according to discover the errors.

EQUIVALENCE PARTITIONING :

- This technique is also known as Equivalence class Partitioning (ECP). In this technique, input values to the system or application are divided into different classes or groups based on its similarity in the outcome.
- Here, instead of using each and every input value, we can now use any one value from the group/class to test the outcome.
- This way, we can maintain test coverage while we can reduce the amount of rework and most importantly the time spent.
- For Example : (Voting Eligibility)

As present in the below, the "AGE" text field accepts only numbers from 18 to 60. There will be three sets of classes or groups

AGE * [ ] (accepts value between 18 to 60)

Equivalence class Partitioning

Invalid	valid	Invalid
< = 17	18 to 60	> = 61

- Two invalid classes will be
  - Less than or equal to 17.
  - Greater than or equal to 61.

  No. of invalid classes = 2

+ A valid class will be anything between 18 to 60. No. of valid classes/groups here = 1.

- We have thus reduced the test case to only 3 test cases based on the formed classes thereby covering all the possibilities
- So, testing with any one value from each set of the class is sufficient to test above voting eligibility scenario.

BOUNDARY VALUE ANALYSIS : (BVA)

- The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.
- Boundary refers to values near the limit where the behavior of the system change.
- In Boundary value analysis, both valid and invalid inputs are being testing to verify the issues.

- Guidelines for BVA :

i. RANGE : between a and b then test case should be designed beyond a and b.

Ex :

1
2
3
4
5
6
7

range (3 - 5)
a b

Produce correct results

} also tested at 2, 6 they should give invalid/incorrect results.

ii. If input selects minimum & maximum then values just above and just below are also tested.

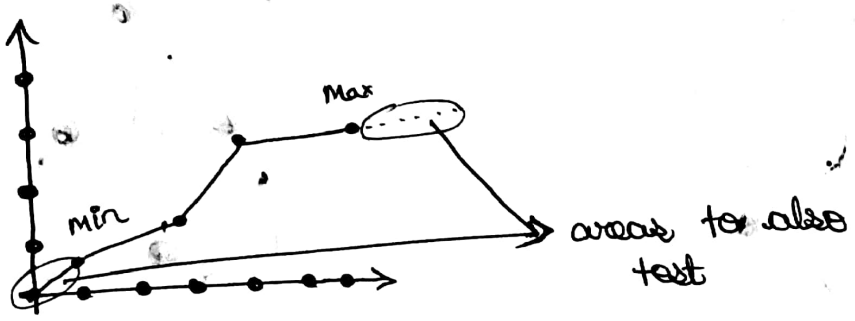
Ex : set = {18, 20, 21, 24}

min = 18 (just above is 17)
max = 24 (just below is 25)

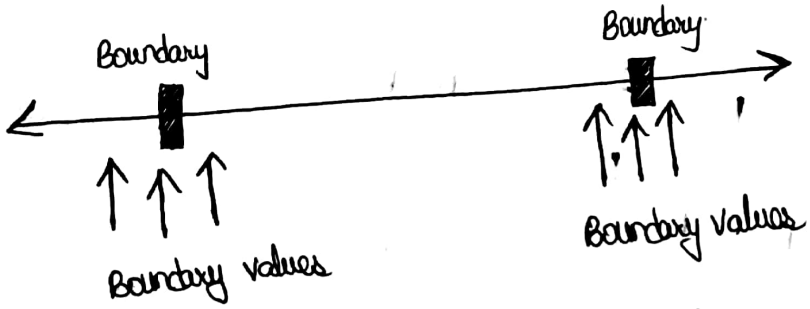
} these are also tested

iii. The above two guidelines are applied to output conditions

Ex:



iv. If internal program or its Data structure prescribed some boundary values like an array of 100 no.'s then it should use test cases that are upto 100 entries only.

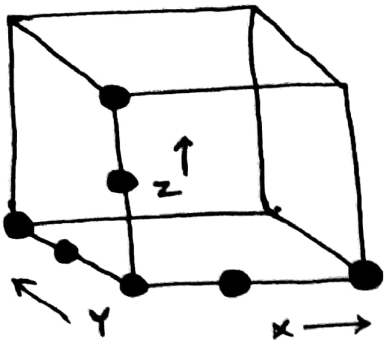


- If we want to test a field where values from 1 to 100 should be accepted, then we choose the array boundary values: 1-1, 1+1, 100-1, 100+1. Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, 101.

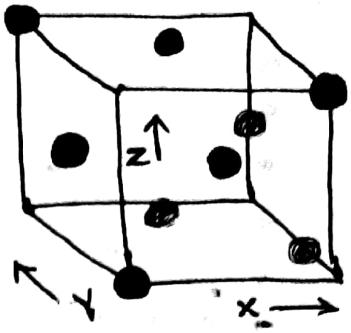
ORTHOGONAL ARRAY TESTING:

- orthogonal Array testing can be applied to problems in which the input domain is relatively small but too large to accomodate exhaustive testing.
- The orthogonal array testing method is particularly useful in finding region faults - an error category associated with faulty logic within a software component.

Example: (SEND function of FAX app)



one input item at a time



L9 orthogonal array

- Phadke assess the results of test using L9 orthogonal array in the following manner:

Test case	Test parameters			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

1. Detect and isolate all single mode faults
2. Detect all double mode faults
3. Multimode faults

- Consider 3 input values x, Y, Z All these input values have three discrete values associate with it.

  Test cases $\longrightarrow 3^3 = 27$

- P1 1, send it now
  P1 2, send it one hour late
  P1 3, send it after midnight
  P2, P3, P4 would take on values of 1, 2, 3, signifying other send functions.

} P1, P2, P3, P4 are passed to send function. (Fax App)

WHITE BOX TESTING :

- White-box Testing of software is predicated on close examination of procedural detail.
- Logical paths through the software and collaborations between components are tested by exercising specific sets of conditions and/or loops.
- White-box testing, sometimes called glass-box testing, is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases.
- Using white-box testing methods, you can derive test cases that
  (1) guarantee that all independent paths within a module have been exercised at least once,
  (2) exercise all logical decisions on their true and false sides,
  (3) execute all loops at their boundaries and within their operational bounds, and
  (4) exercise internal data structures to ensure their validity.
- The term "white-box" is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outershell into it's inner workings.
- Two different White Box testing methods are :
  1) Basis Path Testing
  2) Control Structure Testing.

BASIS PATH TESTING :

- Basis Path testing is a white-box technique first proposed by Tom McCabe.
- The basis path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program atleast one time during testing.

STEPS of BASIS PATH TESTING :

To conduct basis path testing of a program, follow the steps below:

1. Draw the control flow graph of the program.
2. Calculate the cyclomatic complexity of the control flow graph. This will be the maximum no. of independent paths in the graph.
3. Identify independent paths in the control flow graph.
4. Design test cases based on the independent paths identified so that the test cases execute all independent paths.

ADVANTAGES :

+ Reduces the no. of redundant tests.
+ All program statements are executed and tested once.
+ Complete branch coverage.

Basis Path Testing has following steps :

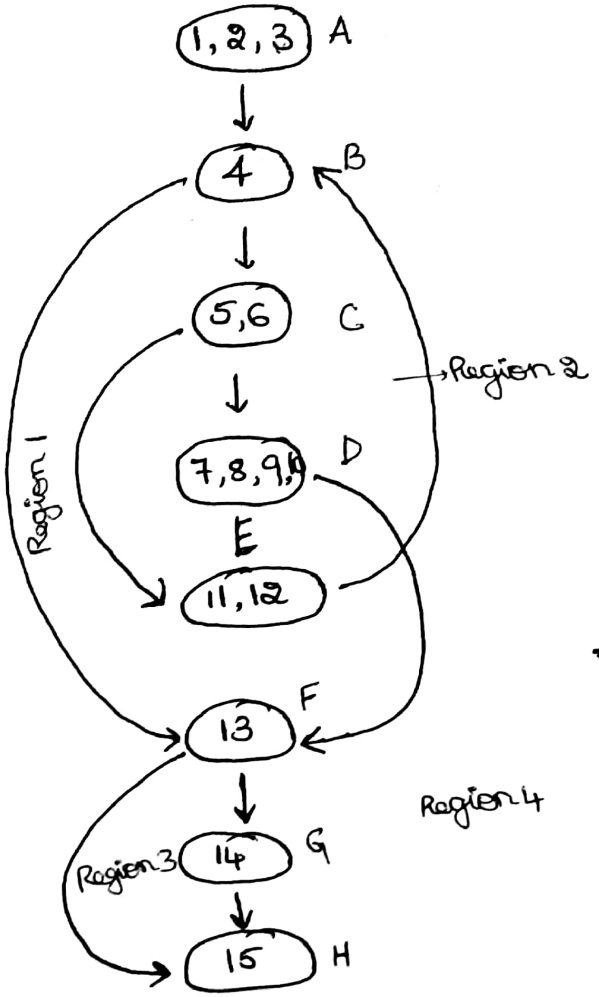
1. Flow Graph Notation :

Step 1 : Start numbering the statements after declaration of the variables (if no variables have been initialized in statement). For the given code snippet, the numbering is as follows :

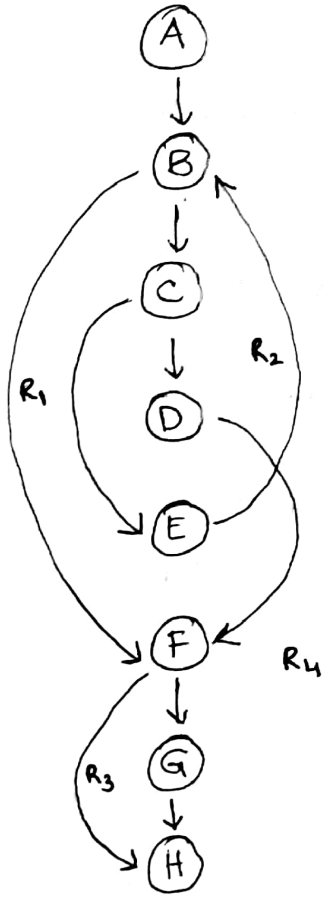
```
int main()
{
    int n, i;
    cout << "Please Enter a number : " << endl;  (1)
    cin >> n;  (2)
    i = 2;  (3)          (4)
    while (index <= n-1) {  (5)
        if (n%i (6) == 0) {  (7)
            cout << "Not a Prime Number" << endl;  (8)
            break;  (9)
    (10) }
        i++;  (11)
(12) }
    if (i == n) (13)
        cout << "Prime Number" << endl;  (14)
}  //end main  (15)
```

Step 2 : Put the sequential statements into one node, here the statements 1, 2, 3 are all sequential statements and hence they should be combined into single node.

The flowgraph obtained will be,

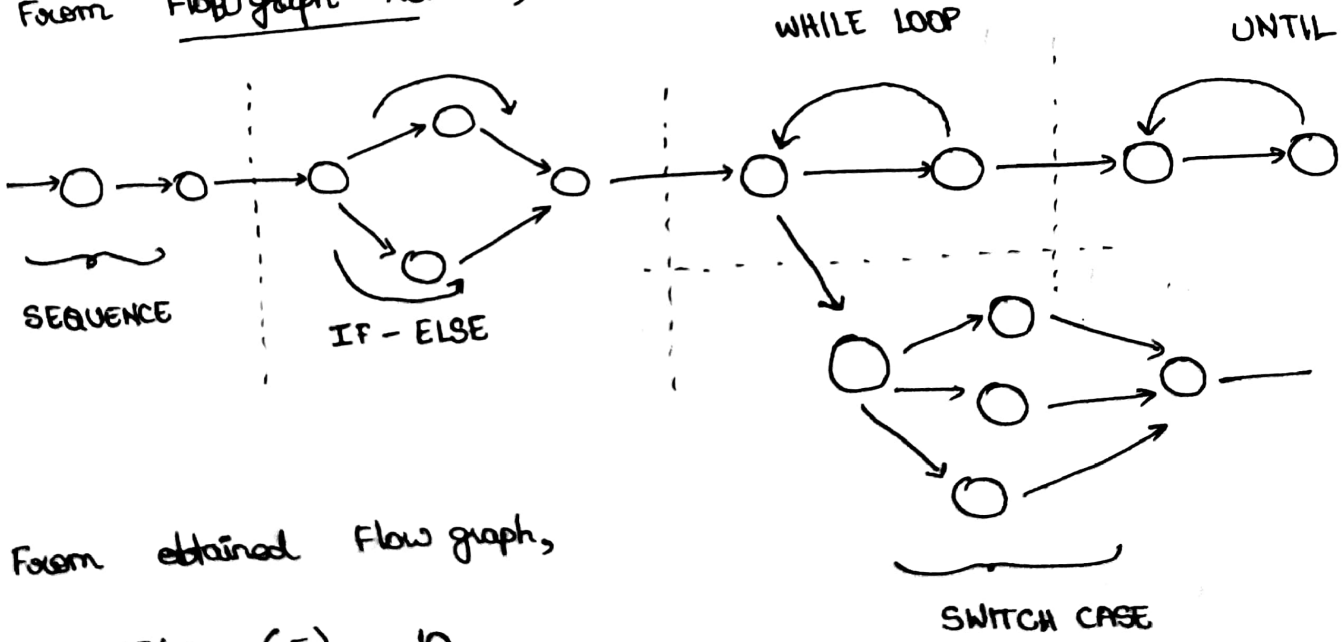


Simplified version :



8 nodes
10 edges

From Flow graph Notation,



From obtained Flow graph,

No. of Edges (E) = 10

No. of Nodes (N) = 8

No of Predicate Nodes (P_n) = 3 { Node B, C, F }

No of Regions = 4 { R_1, R_2, R_3, R_4 }

② Calculate the cyclomatic complexity : V(G)

Cyclomatic complexity is given by three methods :

i. V(G) = No. of Edges − No. of Nodes + 2

ii. V(G) = No of Regions

iii. V(G) = No. of Predicate Nodes + 1

So, V(G) = E − N + 2 = 10 − 8 + 2 = 2 + 2 = 4

(or) V(G) = 4 (no. of regions)

(or) V(G) = Pn + 1 = 3 + 1 = 4

- All the three methods result same value, **Cyclomatic Complexity = 4** *

③ Independant Paths :

As the cyclomatic complexity V(G) for graph has come out to be 4, therefore there are 4 independent paths.

Path 1 : A → B → F → G → H

Path 2 : A → B → F → H

Path 3 : A → B → C → E → B → F → G → H

Path 4 : A → B → C → D → F → H

Each of these paths have atleast derived one new node edge which has not been traversed in other paths.

④ Deriving Test Cases :

- Lets Prepare test cases that will force the execution of each path in the basis set.
- We need to provide input to the program such that each independent path is executed.
- For given code snippet, the following will be the test cases :

#	Independent Path Covered	Test Case	output
1)	A – B – F – H	n = 1	No output
2)	A – B – F – G – H	n = 2	Prime Number
3)	A – B – C – E – B – F – G – H	n = 5	Prime Number
4)	A – B – C – D – F – H	n = 9	Not a Prime Number

- other one would be Graph Matrices : To store the flow graph traced in the computer memory, we use this data structure. Works like Adjacency Matrix (Graph Datastructure).

CONTROL STRUCTURE TESTING :

- Basis path testing is simple and highly effective, but it is not alone sufficient.
- These broaden control structure testing coverage and improve quality of white-box testing.
- It includes following methods :

  1) Conditional Testing 2) Data Flow testing 3) Loop Testing

① Condition Testing :

- Condition testing is a test-case design method that exercises the logical conditions contained in a program module.
- A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (~) operator.
- A relational expression takes the form : E1 <relational-operator> E2 where E1 & E2 are arithmetic expressions and <relational-operator> is one of the following : <, <=, =, ≠, >, >=.
- A compound condition is composed of two or more simple conditions Boolean operator, and parentheses.
- Boolean operators allowed in a compound condition include OR (|), AND (&), and NOT (~). A condition without relational expression is referred to as Boolean expression.
- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include Boolean operator errors, Boolean variable errors,

Boolean parenthesis errors, relational operators errors, and arithmetic expression errors.

- The condition testing method focuses on testing each condition in the program to ensure that it doesn't contain errors.

② Data Flow Testing :

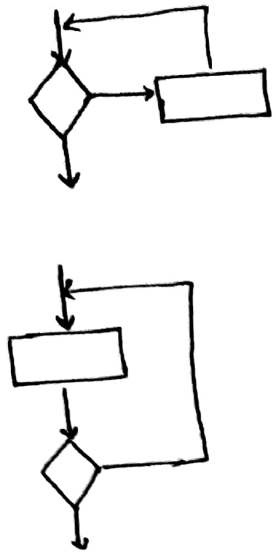
- The data flow testing method selects test paths of a program according to the locations of the definitions and uses of variables in the program.
- To illustrate the data flow testing approach, assume that each statement in a program is assigned a unique statement number and that each function doesn't modify its parameters or global variables.
- For a statement with S as its statement number,

  DEF (S) = { X | statement S contains a definition of X }

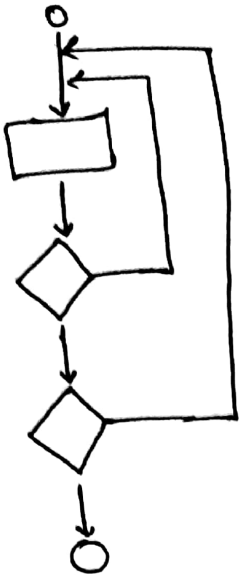
  USE (S) = { X | statement S contains a use of X }

③ Loop Testing :

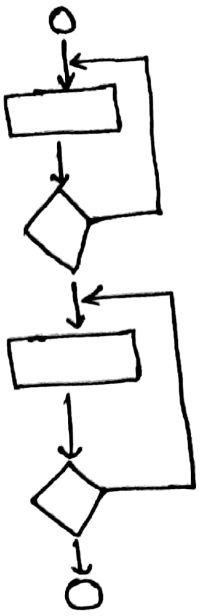
- Loops are cornerstone for the vast majority of all algorithms implemented in software.
- Loop Testing is a white box testing technique that focuses exclusively on the validity of loop constructs.
- Four different classes of loops can be defined : simple loops, concatenated loops, nested loops, and unstructured loops.



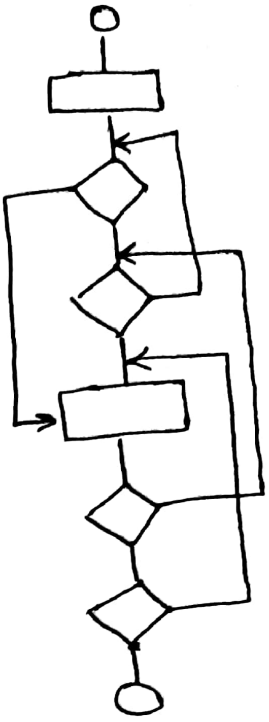
Simple Loops



Nested Loops



Concatenated Loops



Unstructured Loops

- Simple Loops : The following set of tests can be applied to simple loops, where n is the maximum no. of allowable passes through the loop.

i. Skip the loop entirely
ii. only one pass through the loop.
iii. Two passes through the loop.
iv. m passes through the loop where $m < n$.
v. $n-1$, n, $n+1$ passes through the loop.

- Nested Loops : If were to extend the test approach for simple loops, the no. of possible tests would go grow geometrically as the level of nesting increases. This would result in an impractical no. of tests.

Beizer suggests an approach that will help to reduce the no. of tests :

i. Start at the innermost loop. Set all other loops to minimum values.

ii. Conduct simple loop tests for the innermost loops while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.

iii. Work outward, conducting tests for the next loop, but keeping all other loops at minimum values and other nested loops to typical values.

iv. Continue until all loops have been tested.

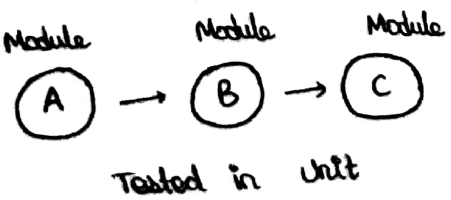
- Concatenated loops : Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop1 is used as the initial value for loop2, then the loops are not independent.

- Unstructured loops : Whenever possible, this class of loops should be redesigned to reflect the use of the structured loops programming constructs.

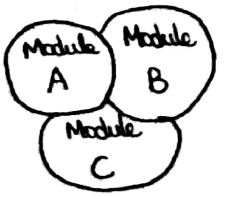
- Loop Testing reveals loops initialization problems.
- By going through the loops once, the uninitialized variables in the loop can be determined.
- Testing can also fix loop repetition issues.
- Loops can also reveal capacity / performance bottlenecks.

INTEGRATION TESTING :

- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- The objective is to take unit-testing components and build a program structure that has been dictated by design.
- There is often a tendency to attempt non incremental integration that is, to construct the program using a "big bang" approach.
- All components are combined in advance. The entire program is tested as a "whole". If a set of errors is encountered.
- Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.
- Incremental loop integration is the antithesis of the big bang approach The program is constructed and tested in small increments, where errors are easier to correct;
- Interfaces are more likely to be tested completely; and a systematic test approach may be applied.
- There are two different incremental integration strategies:
  1. Top-down integration
  2. Bottom-up integration.



Tested in Unit Testing



Under Integration testing

Features of Integration Testing :

1. Done after Unit Testing is complete
2. Integration as per test plan
3. Checks Functionality.

Importance of Integration Testing :

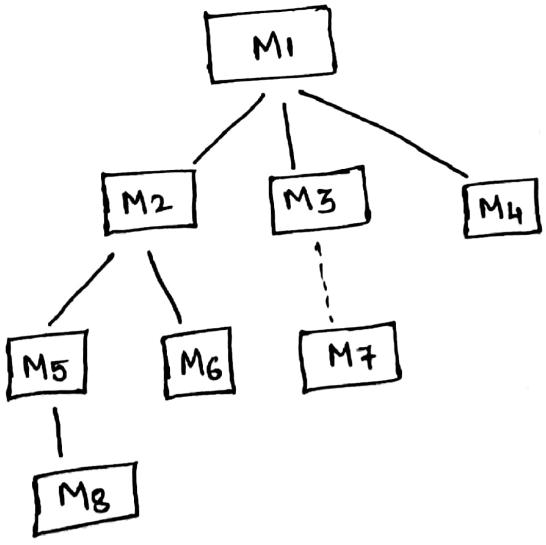
- Faster development, a confident level is high.
- Easy to integrate different modules.
- It is easy to test.
- Code coverage is high and easy to track.
- Top-down or Bottom-up integration testing starts at early stages of software development and bugs are caught easily.
- Provides Major assistance in creating real-time use cases.
- It easily finds system level issue like, broken database, integration mistakes and more.

Drawbacks of Integration Testing :

- The main flaw in Integration Testing is that any scenario that is not covered in the test plan will be left untouched and as such, critical sources of errors and malfunctions might be overlooked.
- To be precise the success of Integration Testing lies in the perfection of the test plan.
- Since test plan is made by humans one needs to cater for its inefficiencies.

TOP-DOWN INTEGRATION :

- Top-down integration is an incremental approach to construction of the software architecture.
- Modules are integrated by moving downward through the control hierarchy, begining with the main control module (main program).
- Modules subordinate to the Main control module are incorporated into the structure in either a depth-first or breadth-first manner.



- Referring to left figure, dept first integration integrates all components on a major control path of program structure.
- For example, selecting the left-hand path, components M1, M2, M5 would be integrated first.
- Next M8 or M6 would be integrated
- Then the central and right-hand control paths are built.
- Breadth-first integration incorporates all components directly subordinate at each level, moving across all the structure horizontally.
- From the figure, components M2, M3, M4 would be integrated first. The next control level, M5, M6 and so on follows.
- The integration process is performed in a series of 5 steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.

2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.

3. Tests are conducted as each component is integrated.

4. On completion of each set of tests, another stub is replaced with the real component.

5. Regression testing (may be) conducted to ensure that new errors have not been introduced.

- The top-down integration strategy verifies major control or decision points early in the test process.

- Advantages :

+ Well-known management style
+ Greater clarity
+ More accountability
+ Quicker implementation

- Disadvantages

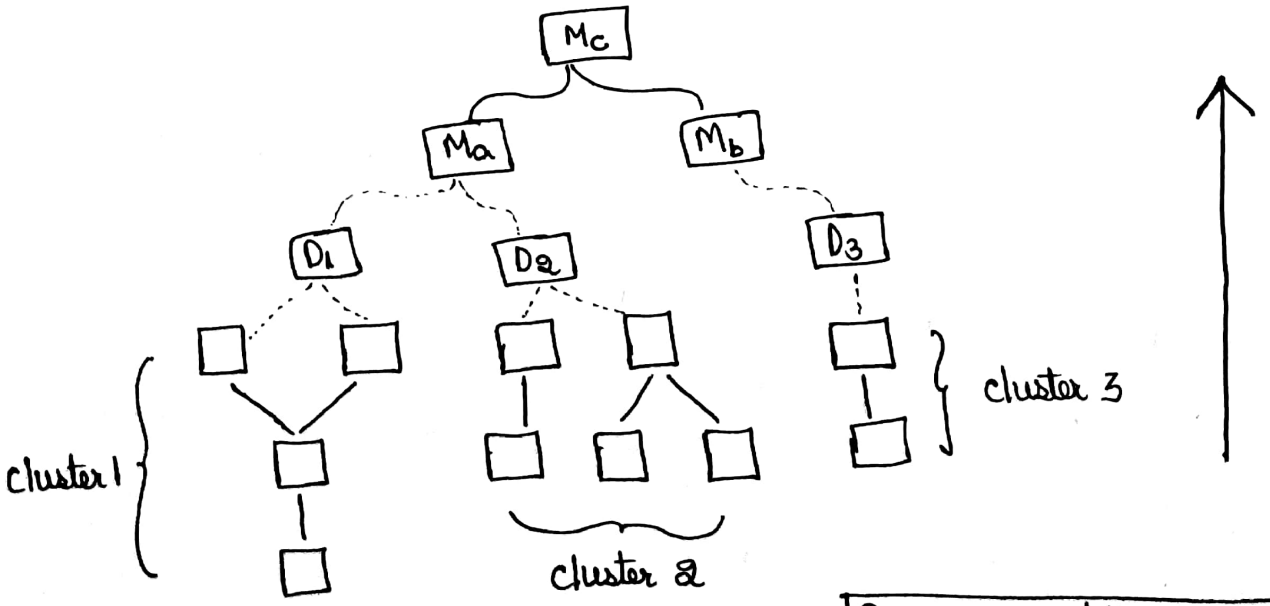
– More of strain on leadship.
– Less creativity.
– Team disengagement.
– Great distance between decision-makers, decisions.

- When approaching a project from the top down, higher-level decision makers start which a big picture goal and work backward to determine what actions different goals groups and individuals will need to take in order to reach the goal.

BOTTOM - UP INTEGRATION :

- Bottom-up integration testing, as its name implies, begins construction and testing with atomic module (i.e. components at the lowest levels in the program structure).
- Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated.
- A Bottom-up integration strategy may be implemented with the following steps :

1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub function
2. A driver (a control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.



BOTTOM - UP INTEGRATION

- Integration follows the pattern illustrated in the above figure. Components are combined to form clusters 1, 2, 3.
- Each of the clusters is tested using a driver (shown as a dashed block).
- Components in cluster 1, 2 are subordinate to Ma. Drivers D1, D2 are removed and the clusters are interfaced directly to Ma. Similarly, driver D3 for cluster 3 is removed prior to integration with Module Mb.
- Both Ma & Mb will be ultimately be integrated with component Mc, and so forth.
- As integration moves upward, the need for seperate test drivers lessens. In fact, if the top two levels of program structure are integrated top down, the no. of drivers can be reduced substantially and integration of clusters is greatly simplified.

- Advantages
  + Fault localization is easy in this approach.
  + More informed decisions.
  + Better team morale.
  + More room for creativity.

- Disadvantages
  - Low momentum
  - Shift in leadership dynamics
  - Lack of high-level insight
  - Early prototype is not possible.

- Top-Down Testing is beneficial if the significant defect occurs toward the top of the program.
- Bottom up Testing is beneficial if the crucial flaws encounters towards the bottom of the program.

SMOKE TESTING :

- Smoke Testing is an integration Testing approach that is commonly used when product software is developed.
- It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis.
- In essence, the smoke testing approach encompasses the following activities :

1. Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

2. A series of tests is designed to expose errors that will keep the build from properly performing its functions. The intent should be uncover 'showstopper errors that have the highest likelihood of throwing the software project behind schedule.

3. The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.

- McConnal describes the smoke testing in following manner :
  - Smoke Test should exercise the entire system from end to end.
  - It doesnot have to be exhaustive, but it should be capable of exposing major problems.

- The smoke test should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly.
- Smoke testing provides a no. of benefits when it is applied on complex, time critical software projects.

- "Integration risk is minimized." Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered.
- "The quality of the end product is improved." Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better quality will result.
- "Error diagnosis and correction are simplified". Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with "new software increments" — that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.
- "Progress is easier to assess" With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

REGRESSION TESTING :

- Regression Testing is the re execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Regression Testing helps to ensure that changes donot introduce unintended behavior or a additional errors.
- Regression Testing may be conducted manually, by reexcuting a subset of all testcases or using automated capture / playback tools.
- Capture / playback tools enable the software engineering to capture test cases and results for subsequent playback and comparision.
- The regression test suite (the subset of tests to be executed) contains three different classes of test cases:

  i. A representative sample of tests that will exercise all software functions.

  ii. Additional tests that focus on software functions that are likely to be affected by the change.

  iii. Tests that focus on the software components that have been changed.

- As integration testing proceeds, the no. of regression tests can grow quite large.

IDENTIFICATION OF BUGS

Program Failure → Debugging → Identification of bugs

MODIFICATION

Modification of source code to make it bug free

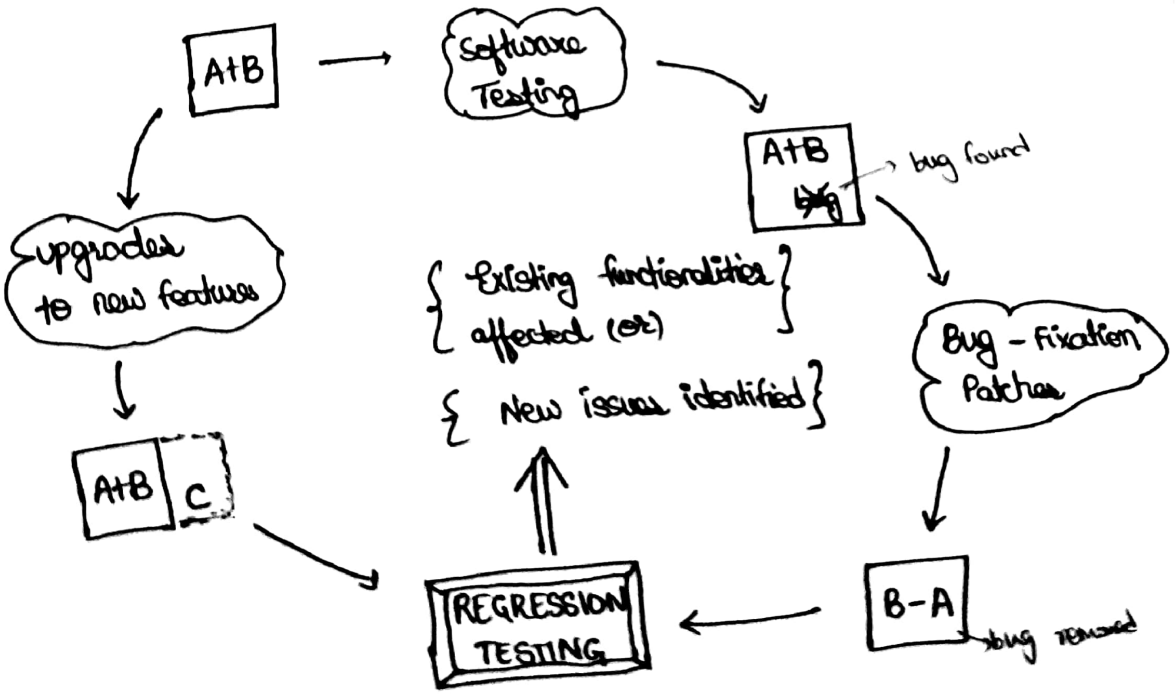
Selecting the existing test cases covering the modified and affected parts of code

Generate new test cases if required

Execute test cases to perform regression testing

SELECTION & EXECUTION OF TEST CASES

Example :



- Functional Point Analysis
- Architectural Design Metrics
- Metrics for Software Quality
- Interpretation of Different Metrics

} MODULE - 4

SOFTWARE QUALITY :

- Goal of software engineering is to produce a high-quality system, application or product within a time frame that satisfies a market need.
- The quality of system, application or product is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program, and the tests that exercise the software to uncover errors.
- You can use measurement to assess the quality of the requirements and design models, the source code, and the test cases that have been created as software is engineered.

METRICS :

1) Measuring Quality - Correctness, Maintainability, Integrity, Usability
2) Defect Removal Efficiency (DRE)

- Project Manager must also evaluate quality as the project progresses. Private metrics collected by individual Software Engineers are combined to provide project level results.

IMP
- Metrics such as work product errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric.

- Error data can also be used to compute the defect removal efficiency (DRE) for each process framework activity.

1) MEASURING QUALITY

Ex : Consider building a Library Management System, you should measure the Quality of the software by correctness, maintainability integrity and usability.

A- Correctness :

- The Library Management System (LMS) must operate correctly or it provides little value to its users.
- Correctness is the degree to which the software performs its required function.
- The most common measure is defects per K-Lines of Code, in the LMS where the defect is verified lack of conformance to requirements like login Page, Issue Book, Generate Report, Manage student Profile, etc...
- For Quality assessments purpose, defects are countered over a standard period of time, typically a year.

B. Maintainability :

- Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, enhanced if the customer desires a change in requirements.

- The LMS software in being Maintained & when a new user arrives, it should adapted to user & if the user is using it in an diffrent OS like Android, Linux, Mac OS, ... it should get adapted or if the customer quickly wants to change some UI, functionality then it should be enhanced to adapt to changes.

C. Integrity :

- Suppose if an attacker is trying to bypass login page / access student details in LMS, the system ability to detect it & withstand to the attacks (both accidental or intentials) just like Session Hijacking, SQL injection, XX Scripting, ... The system should be enough to authenticate & validate every user input field with proper error messages to users.

$$\boxed{\text{Integrity} = \sum \left[1 - (\text{threat} \times (1 - \text{security}))\right]} *$$

.D. Usability :

- If the LMS software is not easy to use, it is often doomed to failure, even if the functions that is performs are valuable.
- Usability is an attempt to quantify ease of use and can be measured in terms of the characteristics.

② Defect Removal Efficiency (DRE).

$$DRE = \frac{E}{E+D} \quad *$$

where E – no. of errors found before deliver

D – No. of defects found after delivery.

Suppose, the LMS has found there are 40 errors in code before it has been deployed then E = 40. And when the user is using the app, it has some defects like bad error message, failed to authenticate, Access control not used, screen stuck, etc.. it is around 20 defects. then D = 20

$$DRE = \frac{40}{40+20} = \frac{40}{60} = 0.667$$

- Ideal value of DRE is 1

DEFECT REMOVAL EFFICIENCY (DRE):

"A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE)"

- DRE is a measure of the filtering ability of quality assurance and control actions as they are applied throughout all process framework activities.
- When considered for a project as a whole, DRE is defined in following manner:

$$DRE = \frac{E}{E+D} \quad *$$

where E - the no. of errors found before delivery of the software to the end user and

D - the no. of defects found after delivery.

- The ideal value of DRE is 1. That is no defects are found in the software.
- Realistically, $D > 0$ ⟹ DRE can still approach 1.
- As E increases (for given value of D), the overall value of DRE begins to approach 1.
- In fact E increases, D will decrease. That means errors are filtered out before they become defects.
- DRE used as metric, it provides an indicator of the filtering ability of quality control and assurance activities.

EXAMPLE :

⇒ Suppose a Company is developing a web application software. In each case say, it identified errors during software testing phase are 100. And later on when the project is released into market, it get about 50 bugs more from people after being used.

Case ①

- So, the Deficiency of not testing the software. The Defect Removal Efficiency (DRE) is

No. of Errors before delivery, $E = 100$

No. of bugs after delivery, $D = 50$

$$DRE = \frac{E}{E+D} = \frac{100}{100+50} = 0.6667$$

Case ②

- And also assume if $D=0$, then E may be any +ve number,

$$DRE = \frac{E}{E+D} = \frac{E}{E+0} = \frac{E}{E} = 1 \text{ (ideal value)}$$

means All the errors are fixed before delivery itself and the project is bug free.

Case ③

- Similarly, If $E=0$, D is any positive value, then

$$DRE = \frac{0}{0+D} = 0 \Rightarrow$$ No bugs were fixed during testing & project is Buggy.

FUNCTION POINT METRICS :

- The function point metrics (FP), first proposed by Albrecht can be used effectively as a means of measuring the functionality delivered by system.
- Information Domain values are defined in the following manner :

i. No. of External inputs (EIs) = 2 (username, Password)

ii. No. of External outputs (EOs) = 2 (Payment, Food details)

iii. No. of External inquiries (EQs) = 2 (location, status)

iv. No. of Internal logical files (ILFs) = 1 (Food details)

v. No. of External interface files (EIFs) = 2 (Payment, Location Map)

- Function Points :

$$FP = Count_{total} \times [0.65 + 0.01 \times \Sigma F_i]$$ where

FP - Function Points of requirement model

$Count_{total}$ - sum of all FP entries obtained

ΣF_i - The F_i ($i = 1$ to 14) are value adjustment factors (VAF) based on responses to following questions :

Questions :

1. Does the system require reliable backup & recovery ? (rating = 4)
2. Are specialized data communications required to transfer information to or from application ? (rating = 3)

3. Are the distributed communications processing functions ? (rating = 3)

4. Is performance critical ? (rating = 2)

5. Will the system run in existing, heavily utilized operational environments ? (rating = 5)

6. Does the system require online data entry ? (rating = 5)

7. Does the online data entry require input transactions to be built over multiple screens or operations ? (rating = 4)

8. Are the ILFS updated online ? (rating = 4)

9. Are the inputs, outputs, inquiries complex ? (rating = 1)

10. Is the internal processing complex ? (rating = 0)

11. Is the code designed to be reusable ? (rating = 3)

12. Are conversion, installation included in design ? (rating = 2)

13. Is the system designed for multiple installations in different organizations ? (rating = 1)

14. Is application designed to facilitate change & ease use of user by the users ? (rating = 3)

$F = 14 * Scale$ where Scale is the mean of all ratings

- 0 - No influence
- 1 - Incidental
- 2 - Moderate
- 3 - Average
- 4 - Significant
- 5 - Essential

} If used these notics to rate these 14 Questions.

∴ So, Scale = $\frac{4+3+3+2+5+5+4+4+1+0+3+2+1+3}{14}$

Scale = $\frac{40}{14}$ = 2.8571

Σfi = 14 * Scale = 14 * 2.8571 = 40

Now,

Information Domain Value	Count	Weighting Factor Simple	Average	Complex		
External inputs	2	(3)	4	6	=	6
External outputs	2	(4)	5	7	=	8
External inquiries	2	(3)	4	6	=	6
Internal Logical files	1	(7)	10	15	=	7
External interface files	2	(5)	7	10	=	10
					=	37

Count total = 6+8+6+7+10 =

Now, FP = Count total × [0.65 + 0.01 × ΣFi]

= 37 × [0.65 * 0.01 × 40] = 37 (0.65 + 0.4) = 1.05 × 37

= 37 × 1.05 = 38.85

∴ Total Functional Points = 38.85 ≈ 39

MANAGEMENT SPECTRUM :

- Effective software project management focuses on the four Ps – People, Product, Process, Project.
- The order is not arbitrary. The manager who forgets that software engineering work is an intensely human endeavour will never have success in project management.

① PEOPLE :

- The most important component of a product and its successful implementation is human resources.
- In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product.
- We need to have a good team inorder to save our time, cost and effort.
- Some assigned roles in software project planning are project manager, team leaders, stakeholders, Software Teams, Agile Teams, (Coordination and Communication Issues)
- Organizations that achieve high levels of People-CMM maturity have a higher likelihood of implementing effective software project management practices.

ⓐ Stakeholders :

- The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies.

→ Senior managers who define the business issues that often have a significant influence on the project

→ Project (technical) managers who must plan, motivate, organize and control the practitioners who do software work.

→ Practitioners who deliver the technical skills that are necessary to engineer a product or application.

→ Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.

→ End users who interact with the software once it is released for production use.

⑥ Team Leaders :

- Project management is a people-intensive activity and for this reason, competent practitioners often make poor team leaders. They simply dont have the right mix of people skills. And yet, as Edgemon states.
- MOI model of leadership :
  - → Motivation – The ability to encourage (by push or pull) technical people to produce to their best ability.
  - → Organization – The ability to mold existing processes (or invent new ones) that will enable it to transfer into final product.
  - → Ideas or innovation – The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

- Another view of the characteristics that define an effective project manager emphasizes four key traits :

→ Problem solving : An effective software project manager must diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations and remain flexible enough to change direction if initial attempts at problem solution are fruitless.

→ Managerial Identity : A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.

→ Achievement : A competent manager must reward initiative and accomplishment to optimize the productivity of a project team.

→ Influence and team building : An effective project manager must be able to read people; must be able to understand verbal and non verbal signals and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

© The Software Team :

- The "best" team structure depends on the Management style of your organization, the no. of people who will populate the team and their skill levels, and the overall problem difficulty.

- Mantei describes seven project factors that should be considered when planning the structure of software engineering teams :

  (1) difficulty of the problem to be solved.

  (2) Size of resultant programs in lines of code or Function Points.

  (3) Time that the team will stay together.

  (4) Degree to which the problem can be modularized.

  (5) Quality & Reliability of System to be built.

  (6) Rigidity of the delivery date.

  (7) degree of sociability (communication) required for the project.

- Constantine suggests four organizational paradigms for software engineering teams :

  (1) A closed Paradigm

  (2) A random Paradigm

  (3) An open paradigm

  (4) A synchronous paradigm

④ Agile Teams :

- Many software organizations advocate agile software development as an antidote to many of the problems that have plagued software project work.
- An agile Team is a group of employees, contractors, or freelancers responsible for executing an Agile Project. For instance, when using a Scrum Framework, an Agile Team should have a Scrum master, a product owner, and any other required team members.
- Agile teams conduct meetings and based on information obtained, the teams adopt the approaches for an increment of the work.

② THE PRODUCT :

- As the name infers, this is the deliverable or the result of project. The project manager should clearly define the product scope to ensure a successful result, control the team members as well technical hurdles that encounter during building of product.
- The product can consist of both tangible or intangible such as shifting the company to a new place or getting a new software in a company.

ⓐ Software Scope :

- The first software project management activity is the determination of software scope. Scope is defined by answering the following questions:

(i) Context – How does the software to be built fit into a larger system, product or busines context, and what constraints are imposed as a result of the context?

(ii) Information objectives – What customer-visible data objects are produced as output from the software? What data objects are required for input?

(iii) Function and performance – What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

- Software scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded.

Ex: No. of simultaneous users, size of mailing list, maximum allowable response time are stated explicitly, product cost restricts memory size and desired algorithms available in Java.

⑥ Problem Decomposition :

- Problem Decomposition sometimes called partitioning or problem elaboration is an activity that sits at the core of software requirements analysis.
- During the scoping activity, no attempt is made to fully decompose the problem. Rather decomposition is applied in two major areas :

  (1) the functionality and content (information) that must be delivered

  (2) the process that will be used to deliver it.

- A complex problem is partitioned into smaller problems that are more manageable. This is the strategy that applies as project planning begins.
- Software functions, described in the statement of scope, are evaluated and refined to provide more detail prior to the begining of estimation.
- Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful.

③ THE PROCESS :

- In every planning, a clearly defined process is the key to the success of any product. It regulates how the team will go about its development in respective time period.
- The process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.
- Project planning begins with the melding of the product and the process. Each function to be engineered by your team must pass through the set of framework activities that have been defined for your software organization.

- The team must decide which process model is most appropriate for :

(1) the customers who have requested the product and the product and the people with who will do the work.

(2) The characteristics of the product itself

(3) The project environment in which the software team works.

@ Melding the Product and the Process :

Project planning begins with the melding of the product and the process. A matrix as shown below is constructed from the framework

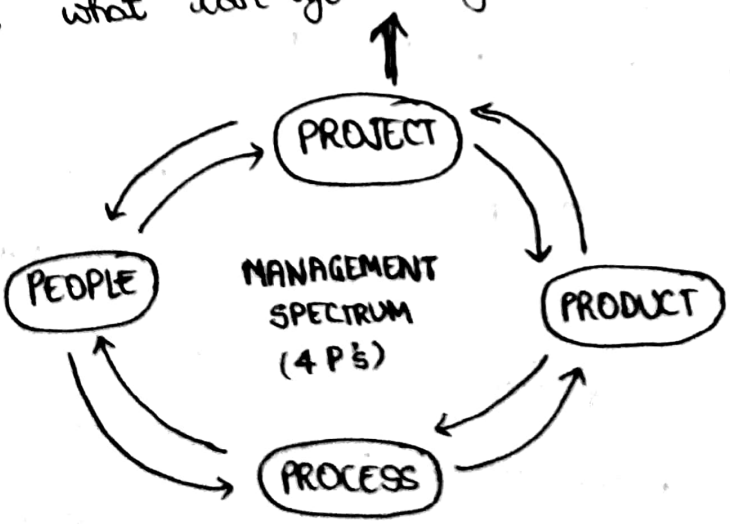
FRAMEWORK ACTIVITIES	COMMON PROCESSES	① Communication	② Planning	③ Modelling	④	⑤
	Software Engineering Tasks					
	Product Functions					
①	Text Input					
②	Edit and Formatting					
③	Automatic Copy Edit					
④	Page Layout Capability					
⑤	Automatic Indexing, TOC					
	File management					
	Document Production					

MELDING PROBLEM AND PROCESS

- The job of the project manager is to estimate resource requirements for each matrix cell, start and end dates for the tasks associated with each cell, and work products to be produced as a consequence of each task.

ii) ④ THE PROJECT : (FROM MANAGEMENT SPECTRUM)

- The last and final P in software project planning is Project. In this phase, the project manager plays a critical role.
- They are responsible to guide the team members, to achieve the projects' target and objectives, helping & assisting them with issues, checking on cost and budget, and making sure that the project stays on track with the given deadlines.
- Reel suggests a five-part commonsense approach to software projects:
  - (1) Start on Right foot
  - (2) Maintain momentum
  - (3) Track Progress
  - (4) Make smart decisions
  - (5) Conduct a postmortem analysis.
- "In order to manage a succesful software project, you have to understand what can go wrong so that problems can be avioded."



Software Project Planning

- Joh Reels defines signs that indicate that an information systems project is in jeopardy.
- In somecases, software people don't understand their customer needs
- This leads to a project with a poorly defined scope. In other projects, changes are managed poorly.
- Jaded industry professionals often refer to "90-90" rule when discussing particularly difficult software projects : The first 90 percent 90% of a system absorbs 90% of allotted effort & time. That last 10% takes another 90% of alloted effort, time.
- How does the manager act to avoid the problems just noted? So, Reel suggested a five-part commonsense approach to software projects :

① Start on the right foot : This is accomplished by working hard to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project. It is reinforced by building the right team and giving the team the autonomy, authority and technology needed to do the job.

② Maintain momentum : Many projects get off to a good start and then slowly disintegrate. To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.

③. Track Progress - For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using technical review) as part of quality assurance activity. In addition, software process & project measures can be collected and used to assess progress against averages developed for the software development organizatio

④ Make Smart Decisions - In essence, the decisions of the project manager and the software team should be to "keep it simple". Wherever possible, decide to use commercial off-the shelf software or existing software components or patterns, decide to avoid custom interfaces when standard approaches are available, decide to identify and then avoid obvious risks, and decide to allocate more time than you think is needed to complex or risky tasks (you'll need every minute).

⑤ Conduct a postmortem analysis - Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from from team members, customers and record findings in written form.

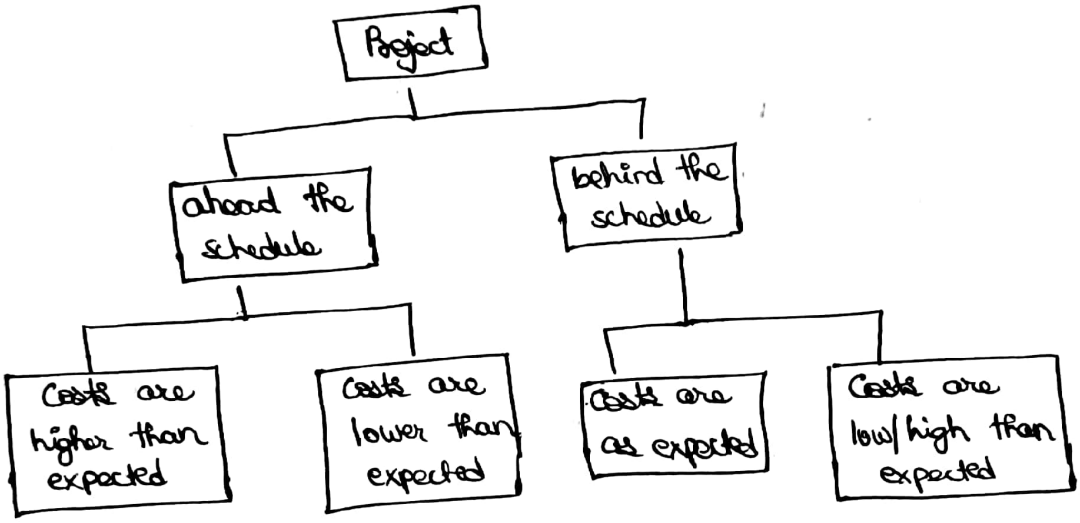
EARNED VALUE ANALYSIS :

- While working in the project management and IT project development team, you will realize that a lot of time is required to measure the project performance progress.
- Three important factors in project management from the project management triangle i.e. scope, time, and cost.
- Planning and control have significantly impacted the way the project has evolved. This is the reason for which Earned Value Management was introduced.
- EVM is a methodical project management process for measuring project performance and progress. The basic idea is to find discrepancies in projects based on the comparision of work done and work planned.
- EVM is used on project schedule and project cost control. It is a very useful tool in project forecasting. The project baseline is the main part of EVM and is also the starting point of all EVM related, activities.
- EVM - Earned Value Management is :
  - is used on the project cost.
  - is used in project schedule control.
  - offers the possibility to provide forecasts of project performance problems.
  - Has the reference point of all related activities, the project baseline.
- The main feature of any EVM implementation include :
  - A project plan, to identify work to be accomplished.
  - Evaluation of planned work called Planned Value (PV) or Budgeted Cost of Work scheduled (BCWS).
  - The earning metrics to quantify the achievements of work called Earned Value (EV) or Budget Cost of Work Performed (BCWP).

- other features, in case the project is more complex.

Earned Value Management Metrics :

- We need to know if project is ahead or behind the schedule?
- We need to know if the project is over budget or is under budget?
- What will be the total cost of project if the project is ahead of schedule but the costs are highest than expected?
- If the project is behind schedule but the costs are lower?



- We will get answers to all these questions by performing analysis using EVM.

1) Initial Metrics :

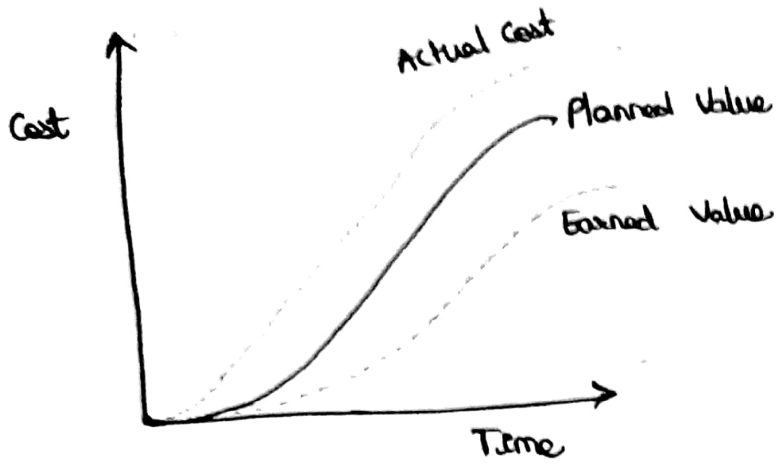
- For calculating Earned Value of a project, we need the following key factors (according to PMBOK :) :

(a) Budget at Completion (BAC) – The total Planned Value of the Project.

(b) Duration (D) – Project Duration.

(c) Planned Value (PV) – is that part of the approved cost estimate planned to be spent on the activity during a specified period.

(d) Actual Cost (AC) – total of costs in achieving work on activity.



EARNED VALUE ANALYSIS CHART

- It is a way to measure and monitor the level of work completed on a project against the plan.

EVM INDICATORS :

1) Schedule Variance (SV) : It is the variance between Earned Value and Planned Value. It let us identify how much you are ahead or behind schedule in terms of costs.

$$SV = EV - PV$$

If $SV < 0$ – the project is behind the schedule.
If $SV = 0$ – the project is on the schedule.
If $SV > 0$ – the project is ahead of the schedule.

2) Schedule Performance Index (SPI): It is the measurement of progress achieved against progress planned.

$$SPI = \frac{EV}{PV}$$

If $SPI < 1$ – the project is running behind the schedule.
If $SPI = 1$ – the project is progressing exactly as planned.
If $SPI > 1$ – the project is progressing well against the schedule.

3) Cost Variance (CV): It is calculated by subtracting the Actual cost (AC) from Earned Value (EV). It lets us know whether you are under or over budget

$$CV = EV - AC$$

If $CV < 0$ – the project is over budget.

If $CV = 0$ – the project is on budget.

If $CV > 0$ – the project is under the budget.

4) Cost Performance Index (CPI): It is the measurement of the value of work completed against the actual cost

$$CPI = \frac{EV}{AC}$$

If $CPI < 1$ – project is over budget

If $CPI = 1$ – project is on budget

If $CPI > 1$ – project is under budget

5) Estimated at Completion (EAC): It is an indicator for forecasting how much the total project will cost.

$$EAC = \frac{BAC}{CPI}$$

6) Estimate to Complete (ETC): It is an estimation of funds required to complete the remaining work in a project. This EVM metric is used for forecasting the budget needed for the remaining project work.

$$ETC = \frac{BAC - EV}{CPI}$$

EVM ANALYSIS :

- To perform Earned Value Analysis at one predefined status point, we will have to perform the following steps :

1) Collect the inputs :
   - Budget at Completion (BAC)
   - Planned Value (PV)
   - Earned Value (EV)
   - Actual Cost (AC)

2) Analyse the schedule status :
   - Schedule Variance (SV)
   - Schedule Performance Index (SPI)

3) Analyse the cost status :
   - Cost Variance (CV)
   - Cost Performance Index (CPI)

4) Forecasting the project status :
   - Estimate to Complete (ETC)
   - Estimate at Completion (EAC)

5) Prepare some reports with the analysis and plans :

- In case we want to analyze each task in the project at a certain point in time, we will have to perform the following steps for each task :
  + Collect the % complete of each task.
  + Collect Planned Value (PV) for each task.
  + Calculate Earned Value (EV) for each task.

- obtain Actual Cost (AC) for each task.
- Perform schedule status for each task.
- Perform cost status, forecasting for each task.
- Compile the results and Create a global report.

+ Here MS Project & MS Excel can be succesfully used for calculating and tracking the EVM Analysis.

EXAMPLES :

Let us consider a very simple example of a software Project. A which is to be completed in one year (12 months) and the total cost is $300000. The start date of the project is 01 October 2019 and end date of the project is 30 September 2020. Assuming that the budget is same for each month. Let us consider that the analysis of the project is performed after 6 months (31 March 2020). The review performed on the project has been shown that 40% of the work has been completed after 6 months and the actual cost is $100000.

- The first step for EVM is to collect the inputs (initial data):

Given.

Budget at Completion, BAC = $300000

Earned Value, EV = 0.4 * BAC = $120000

Planned Value for 6 months = $150000

Actual Cost = $100000.

- Based on the formulas discussed above, we will now calculate the indicators and make Earned Value Analysis (EVA):

1) Scheduled Variance (SV) :

$$SV = EV - PV = \$120000 - \$150000 = -\$30000$$

- Since the result is negative, the project is behind the schedule. It is also easy to observe that the Earned value is lower than the Planned Value.

2) Scheduled Performance Index (SPI):

$$SPI = \frac{EV}{PV} = 0.8$$

- SPI = 0.8 < 1 means the project is performed 80% of the work, it was supposed to at this status point. Also, the project is behind the schedule. For every estimated hour of work of the project, the project team is completing only 0.8 hours, which means 48 minutes. Hence the project is running behind the schedule.

3) Cost Variance (CV):

$$CV = EV - AC = \$120000 - \$10000 = \$20000$$

- The value of project A at the current state is greater than the money spent on it, which means the project is under budget.

4) Cost Performance Index (CPI):

$$CPI = \frac{EV}{AC} = \frac{\$120000}{\$100000} = 1.2$$

- The CPI for the analyzed project is greater than 1, so that the project is performing well against the budget.

5) Estimate at Completion (EAC):

$$EAC = \frac{BAC}{CPI} = \frac{\$300000}{1.2} = \$250000$$

- According to the result, based on the existing analysis, if the project will continue in the same conditions, then at the end of the project the budget will be $250000.
- Estimated at Completion is an indicator for forecasting of how much the total project will cost. So, the total cost will be less than estimated in case the project will go in this direction.

6) Estimate to Complete (ETC):

$$ETC = \frac{BAC - EV}{CPI} = \frac{\$300000 - \$120000}{1.2} = \$150000$$

- Estimate to complete (ETC) is a forecast of how much more money we will need to be spent to complete the project. To finalize the project, if the work will continue like this until the first evaluation for after 6 months, the total amount of money to finalize the project will be $150000.

ADVANTAGES:

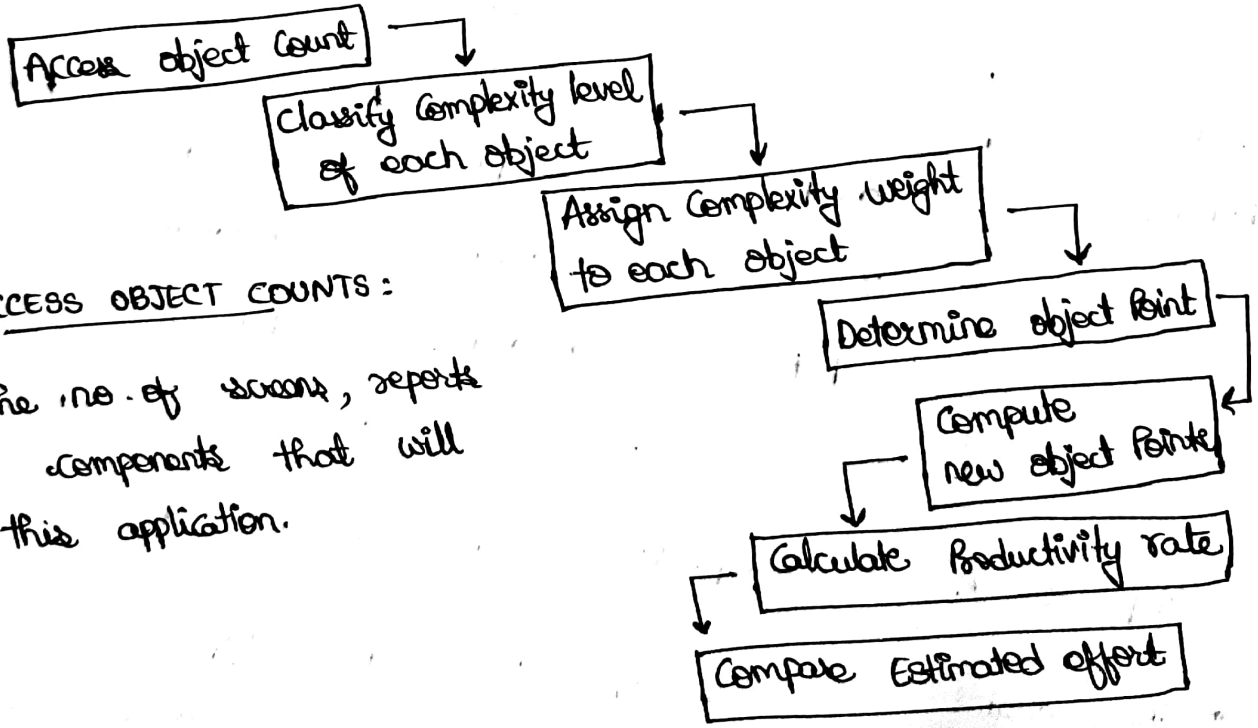
+ Greater a performance framework.
+ Budget needed in future estimation.
+ Measure progress of the project.
+ Provide solid Risk Management.

OBJECT POINT - COCOMO II Application :

- COCOMO-II is the revised version of original COCOMO, is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.
- Application of Composition Estimation Model allows one to estimate the cost, effort at the stage 1 of COCOMO II MODEL.
- In this model size is the first estimated using object Points.
- object Points are easy to identify and count. object Points defines screen, reports, third generation (3GL) modules as objects.
- object Point estimation is a new size estimation technique but it is well • suited in Application Composition Sector.

ESTIMATION OF EFFORTS :

- Following steps are taken to estimate effort to develop a project.



Step1 : ACCESS OBJECT COUNTS :

- Estimate the no. of screens, reports and 3GL components that will comprise this application.

Step 2 : CLASSIFY COMPLEXITY LEVELS OF EACH OBJECT :

- We have to classify each object instance into simple, medium and difficult complexity level depending on values of its complexity characteristics.
- complexity levels are assigned according to the given table :

FOR SCREENS

No. of views Contain	Source of data tables		
	Total < 4 < 2 servers < 3 clients	Total < 8 2-3 servers 3-5 clients	Total 8 + > 3 servers > 5 clients
< 3	Simple	Simple	Medium
3-7	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult

FOR REPORTS

No. of section Contain	Source of data tables		
	Total < 4 < 2 servers < 3 clients	Total < 8 2-3 servers 3-5 clients	Total 8+ >3 servers >5 clients
0-1	Simple	Simple	Medium
2-3	Simple	Medium	Difficult
4+	Medium	Difficult	Difficult

Step 3 : Assign complexity weights to each object

- The weights are used for three object types i.e. screens, reports and 3GL components. Complexity weight are assigned according to object's complexity level using following table :

Object Type ↓	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Components	—	—	10

COMPLEXITY WEIGHT

Step 4 : DETERMINE OBJECT POINTS :

- Add all the weighted object instance to get one number and this is known as "OBJECT POINT COUNT".

$$\text{object Point} = \sum (\text{No. of object instances}) * (\text{Complexity weight of each object instance}) \quad *$$

Step 5 : Compute New object Points (NOP)

- We have to estimate the % reuse to be achieved in a project :

Depending on % reuse

$$NOP = \frac{(\text{object points}) * (100 - \%\text{reuse})}{100} \quad *$$

- NOP are the object point that will need to be developed and differ from the object Point count because there may be reuse of some object instance in project.

Step 6 : Calculate Productivity rate (PROD) :

- Productivity rate is calculated on the basis of informing information given about developer's experience and capability.

For calculating it, we use the following table,

Developers experience and capability	Productivity (PROD)
Very Low	4
Low	7
Normal	13
High	25
Very High	50

PRODUCTIVITY RATE

Step 7 : COMPUTE THE ESTIMATED EFFORT :

- Effort to develop a project can be calculated as :

$$\text{Effort} = \frac{\text{NOP}}{\text{PROD}} *$$

Efforts is measured in person-month
where NOP – New object Points
PROD – productivity rate

EXAMPLE :

Consider a database application project with :

1) The application has four screens with four views each and seven data tables for three servers and four clients.

2) Application may generate two reports of six section each from seven data tables for two servers and three clients.

10% reuse of objects points.

- Developers experience and capability in similar environment is Low.

Calculate the object Point count, NOP, effort to develop such project.

Step1 :

No. of screens = 4

No. of records = 2

Step2 :

For Screens :

No. of views = 4

No. of data tables = 7

No. of Servers = 3

No. of Clients = 4

⇓

Complexity Level = MEDIUM

For Reports :

No. of sections = 6

No. of data tables = 7

No. of Servers = 2

No. of clients = 3

⇓

Complexity Level for each report = DIFFICULT

Step3 :

By using complexity weight table we can assign complexity weight to each object instance depending upon their complexity level.

Complexity weight for each screen = 2

Complexity weight for each report = 8

Step 4 :

$$\text{object Point Count} = \sum \left(\text{no. of object instances}\right) * \left(\text{Complexity weight of corresponding object}\right)$$

$$\text{Object Point Count} = (4 \times 2) + (2 \times 8) = 24$$

Step5 :

% reuse of object points = 10 % (given)

$$NOP = \frac{(\text{object points}) * (100 - \%\text{reuse})}{100} = \frac{24\,(100-10)}{100} = 21.6$$

Step 6 :

Developer's experience and capability is low (given)

Using information given,

Productivity rate, PROD of given project = 7

Step 7 :

$$\text{Effort} = \frac{\text{NOP}}{\text{PROD}} = \frac{21.6}{7} = 3.086 \text{ person-month.}$$

Conclusion :

Therefore, effort to develop the given project = 3.086 person-month

QUALITY FACTORS :

- Even the most jaded software developers will agree that high-quality software is an important goal.
- It is defined as : An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.
- It emphasizes software Quality with three points :
  a) An effective software process establishes the infrastructure that support any effort at building a high quality software product.
  b) A useful product delivers the content, functions, and features that the end user desires, but the delivery is error-free way.
  c) By adding value for both the producer & user of product, high quality software provides benefits for software organization and end-user community.
- The end result is (1) Greater software product revenue, (2) better profitable when an application supports a business process, (3) improved availability of information that is crucial for business.
- Various Quality Factors List includes :
  (A) David Garvin's Quality Dimensions - Performance Quality, Feature Quality, Reliability, Conformance, Durability, Serviceability, Aesthetics, Perception.

(B) McCall's Quality Factors –

i. Product operation – Correctness, Efficiency, Integrity, Reliability, Usability.

ii. Product Revision – Maintainability, Flexibility, Testability.

iii. Product Transition – Portability, Re-usability, Interoperability.

(C) ISO 9126 Quality Factors – Functionality, Reliability, Usability, Efficiency Maintainability, Portability.

(d) Targeted Quality Factors – Intuitiveness, Efficiency, Robustness, Richness.

(A) DAVID GARVIN'S QUALITY DIMENSIONS :

- According to him, the quality should be considered by taking a multidimensional viewpoint that begins with an assessment of conformance and terminates with transcendental view.

1) Performance Quality – Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provide value to end user?

2) Feature Quality – Does the product provides features that surprise and delight first-time end users?

3) Reliability – Does the software deliver all features and capability without failure? Is it available to when it is needed? Does it deliver functionality that is error free?

4) Conformance – Does the software conform to local and external and local software standards that are relevant to the application?

5) Durability - Can the software be maintability (changed) or corrected (debugged) without, the inadvertent generation of unintened side effects ?

6) Serviceability - Can the software be mainted or corrected in an acceptably short time period ?

7) Aesthetics - There's no question that each of us has a different and very subjective vision of what is aesthetic. An aesthetic entity has a certain elegance, a unique flow, and an obvious "presence" that are hard to quantify but are evident nonetheless.

8) Perception - In some situations, you have a set of prejuidices that will influence your perception of quantity, quality.

EX: For a bad quality vendor's software product, you will have negative perception. If a vendor has an excellent reputation, you may perceive quality, even if it doesnot really exist.

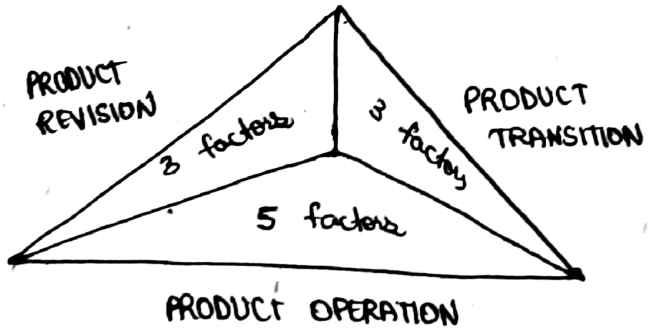
- Gravin's quality dimensions provide you with a "soft" look at software quality.
- You also need "hard" quality factors that are categorized into 2 categories :

(1) Factors that can be directly measured (defects uncovered during testing)

(2) Factors that can indirectly measured (usability, maintainability)

(B) McCALL'S QUALITY FACTORS :

- McCall, Richards, and Walters propose a useful categorization of factors that affect software quality.
- The 3 categories focuses on : its operational characteristics, its ability to undergo change, and its adaptability to new environments.



1) Reliability - The extent to which a program can be expected to perform its intended function with required precision.

2) Efficiency - The amount of computing resources and code required by a program to perform its function.

3) Integrity - Extent to which access to software or data by unauthorized persons can be controlled.

4) Usability - Effort required to learn, operate, prepare input for and interpret output of program.

5) Maintainability - Effort required to locate and fix an error in a program.

6) Flexibility - Effort required to modify an operational program.

7) Testability - Effort required to test a program to ensure that it performs its intended function.

8) Portability - Effort required to transfer the program from one hardware and / or software system environment to another.

9) Reusability - Extent to which a program can be reused in other applications - related to packaging, scoping of functions that the program performs.

10) Interoperability - Effort required to couple one system to another.

(C) ISO 9126 QUALITY FACTORS :

- The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software.
- The standard identifies six key quality attributes :

1) Functionality - The degree to which the software satisfies stated needs as indicated by the following subattributes : suitability, accuracy, interoperability, compliance and security.

2) Reliability - The amount of time that the software is available for use as indicated by the following subattributes : maturity, fault tolerance, recoverability.

3) Usability - The degree to which the software is easy to use as indicated by the following subattributes : understandability, learnability, operability.

4) Efficiency - The degree to which the software makes optimal use of system resources as indicated by the following subattributes: time behavior, resource behavior.

5) Maintainability - The ease with which repair may be made to the software as indicated by : analyzability, changeability, stability testability.

6) Portability - The ease with which the software can be transposed from one environment to another as indicated by the following subattributes : adaptability, installability, conformance, replacability.

- ISO 9126 factors don't lend for direct measurements. But, they provide basis for indirect measurements and excellent checklist for assessing the quality of system.

(D) TARGETED QUALITY FACTORS :

- It focuses on software as a whole and can be used as generic indication of quality of application.
- To conduct more assessments, you'll need to address specific, measurable attributes of interface :

1) Intuitiveness - The degree to which the interface follows / expected usage patterns so that even a novice can use it without significant training.

2) Efficiency - The degree of locating / initiating operations. UI

3) Robustness - The degree of handling bad input data or inappropriate

4) Richness - The degree of providing rich feature set to interface.

SOFTWARE QUALITY ASSURANCE (SQA) :

- Software Quality Assurance is a set of activities followed by software engineers to ensure that the software meets the specified requirements of the user, and it doesnot have any error. Along with this the software is able to manage risk and secures the data of user at each level.

- Though maintaining the quality of the software is looked after by the software developing organization. But software quality assurance (SQA) organization ensure that activities involved in maintaining the quality of the software are conducted efficiently by the software organization.

- Software Quality Assurance (SQA) is the set of actions performed by the SQA group to ensure the quality of a software. Software quality assurance is incorporates :

  1) SQA focus on the software.
  2) Assuring and controlling the quality of software which include technical evaluation and testing.
  3) Applying software engineering practices effectively.
  4) Managing software work products like an architectural model, document with a description of the software, source code. Controlling changes made to software work product.
  5) Measuring and reporting the quality of the software.

ELEMENTS OF SQA :

- SQA encompases a wide range of activities that can be performed to ensure the quality of the software.

(1) STANDARDS :

- We all know that organizations like IEEE, ISO etc has lined up a lot of software engineering standards which should be imposed by the customer and even embraced by software engineers while developing software.
- SQA team must ensure that standards established by the authorized organization are followed by the software engineers.

(2) REVIEWS & AUDIT :

- The software is evaluated technically by the SQA team in order to discover an error in the software. While auditing a software the SQA team confirms that good quality guidelines are followed by the software engineers while developing the software.

(3) TESTING :

- The elemental goal of software is to identify the bug in the software. SQA team has the responsibility of planning the testing systematically and conducting it efficiently. This would raise the possibility of finding the bug in the software if any.

④ ANALYZING ERROR :

- Software testing reveals bugs and errors in the software which are analysed by the SQA team in order to discover how the bugs are introduced in the software and also discover the possible methods required to eliminate these errors or bugs.

⑤ CHANGE MANAGEMENT :

- The customer can ask for modifications between the development of the software. The changes are the most distracting aspect of any software project.
- If the implementation of the changes is not properly managed, it will cause confusion which will affect the quality of the software. The SQA team takes care that change management is practiced while developing the software.

⑥ EDUCATION :

- It is the responsibility of SQA organization to enlighten the software organizations by improving their software engineering practices.

⑦ VENDOR MANAGEMENT :

- It is the responsibility of SQA team to suggest software vendors, to accept the quality practices while developing the software. SQA must also incorporate these quality instructions in a contract with software vendors.

⑧ SECURITY MANAGEMENT :

- With the increase in cybercrime, the government has regulated the software organizations to incorporate the policies to protect data at all level. It is the responsibility of SQA to verify whether the software organizations are using appropriate technology to acquire software security.

⑨ SAFETY :

- The hidden defects of any human-rated software (aircraft applications) can lead to catastrophic events. So, it is the responsibility of SQA to evaluate the effect of software failure and introduce steps to eliminate the risk.

⑩ RISK MANAGEMENT :

- Though it is the job of software organization to analyze and reduce the risk in the software. But, it is the responsibility of SQA to make sure that risk management actions are properly conducted and the backup plan has been confirmed.

GOALS, ATTRIBUTES AND METRICS OF SQA :

@ Requirements quality :

- SQA must ensure that the requirements specified by the user are properly reviewed by the software developers. Software developers have properly analyzed the correctness, completeness of specified requirements as it has a strong influence on the quality of software.

i. Attribute : Metric

Ambiguity : There can be a no. of ambiguous modifiers in the specified requirement.

Completeness : There can be a no. of requirements that are neither announced not determined yet.

Understandability : There can be a no. of sections or subsections in the specified requirements that are not understood by developers.

Volatility : Requirements can be volatile as the user can request a no. of changes in the requirements. As the requirements are volatile and the user can request changes in it, the time requires to complete the process will also be changed.

Traceability : There can be a no. of requirements that cannot be traced to design code.

Model Clarity : Developers can design a no. of UML models for the specified requirements. Each model would have a no. of descriptive pages.

(b) Design Quality :

- After analyzing the requirements specified by the user UML models are designed to visualize the way the system will be designed. The job of SQA is to evaluate whether the design model conforms the specified requirements to achieve quality.

ii. Attribute : Metric

Architectual Integrity : Developers must be able to assess the architectural elements to confirm its quality.

Component Completeness : The components are completely defined or not.

Interface Complexity : The no. moves required to get the desired function.

Patterns : The no. of patterns used to design architectural model.

(c) Code Quality :

- The code of the software and its' related descriptive information must obey the local coding standards. So, that it can be maintained for long in future. It is the job of SQA to observe the quality of code.

iii. Attribute : Metric

Complexity : The no. of cyclomatic complexity in code.

Understandability : The understandability of code relies on the percent of internal comments in code and the naming convention used while naming the variables.

Reusability : The no. of components in the code that can be reused.

(d) Quality Control Effectiveness :

- Software developers should use minimal resources and try to achieve high quality. The job of SQA is to analyze whether the resources are allocated in an effective manner.

iv. Attributes are - Resource Allocation, Completion rate.

SQA PLAN :

- The SQA plan is the guideline which provides the direction for software quality assurance. SQA plan act as template which can be implemented for each software project. Some standards have been established by IEEE. According to these standards, the structure of the plan should be able to identify :

1) The motive and scope of plan.
2) Explain all the SE work product that is within scope of SQA.
3) All the standards & practices applied.
4) SQA activities and tasks.
5) Tools and methods assisting SQA activities & tasks.
6) Software configuration Management.
7) The technique of organizing and maintaining SQA related info.

- Software Quality Assurance is the set of actions that are applied to all the software engineering process while developing the software to ensure the quality of the software.

- Software Quality Assurance (SQA) can be performed by the software developer team or another set of staff can be appointed for this.

CAPABILITY MATURITY MODEL :

- Capability Maturity Model (CMM) is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- Capability Maturity Model Integration (CMMI) is a successor of CMM and is a more evolved model that incorporate best components of individual disciples of CMM like software CMM, Systems Engineering CMM, People CMM, etc..
- Since CMM is a reference model of matured practices in specific disciple, so it becomes difficult to integrate these disciples as per the requirement
- This is why CMMI is used as it allows the integration of multiple disciples as and when needed.

*. objectives of CMMI :

1) Fulfilling customer needs and expectations.
2) Value creation for investors | stockholders.
3) Market growth is increased.
4) Improve quality of products and services.
5) Enhanced reputation in industry.

*. CMMI Representation – staged and continuous :

- A representation allows an organization to pursue a different set of improvement objectives. There are two representations of CMMI
  1) Staged Representation
  2) Continuous Representation

① Staged Representation :

- uses a pre-defined set of process areas to define improvement path
- provides a sequence of improvement where each part in the sequence serves as a foundation for the next.
- an improved path is defined by maturity level.
- maturity level describes the maturity of process in organization.
- staged CMMI representation allows comparision between different organizations for multiple maturity levels.

② Continuous Representations :

- allows selection of specific process areas
- uses capability levels that measures improvement of an individual process area
- Continuous CMMI representation allows comparision between different organizations on a process-area-by-process-area basis.
- allows organizations to select processes which require more improvement.
- In this representation, order of improvement of various processes can be selected with which allows the organizations to meet their objectives and eliminate risks.

*. CMMI Model – Maturity levels :

- In CMMI with staged representation, there are five maturity levels described as follows :

1) Maturity level 1 : INITIAL

- process are poorly managed or controlled.
- unpredictable outcomes of process involved.
- ad hoc and chaotic approach used.

- No KPAs (Key Process Areas) defined.
- Lowest quality and highest risk.

2. Maturity Level 2 : MANAGED

- requirements are managed.
- processes are planned and controlled.
- projects are managed and implemented according to their documented plans
- This risks involved is lower than Initial level, but still exists.
- Quality is better than Initial level.

3. Maturity Level 3 : DEFINED

- processes are well characterised and described using standards, proper procedures, and methods, tools, etc..
- Medium quality and medium risk involved.
- Focus is process standardization.

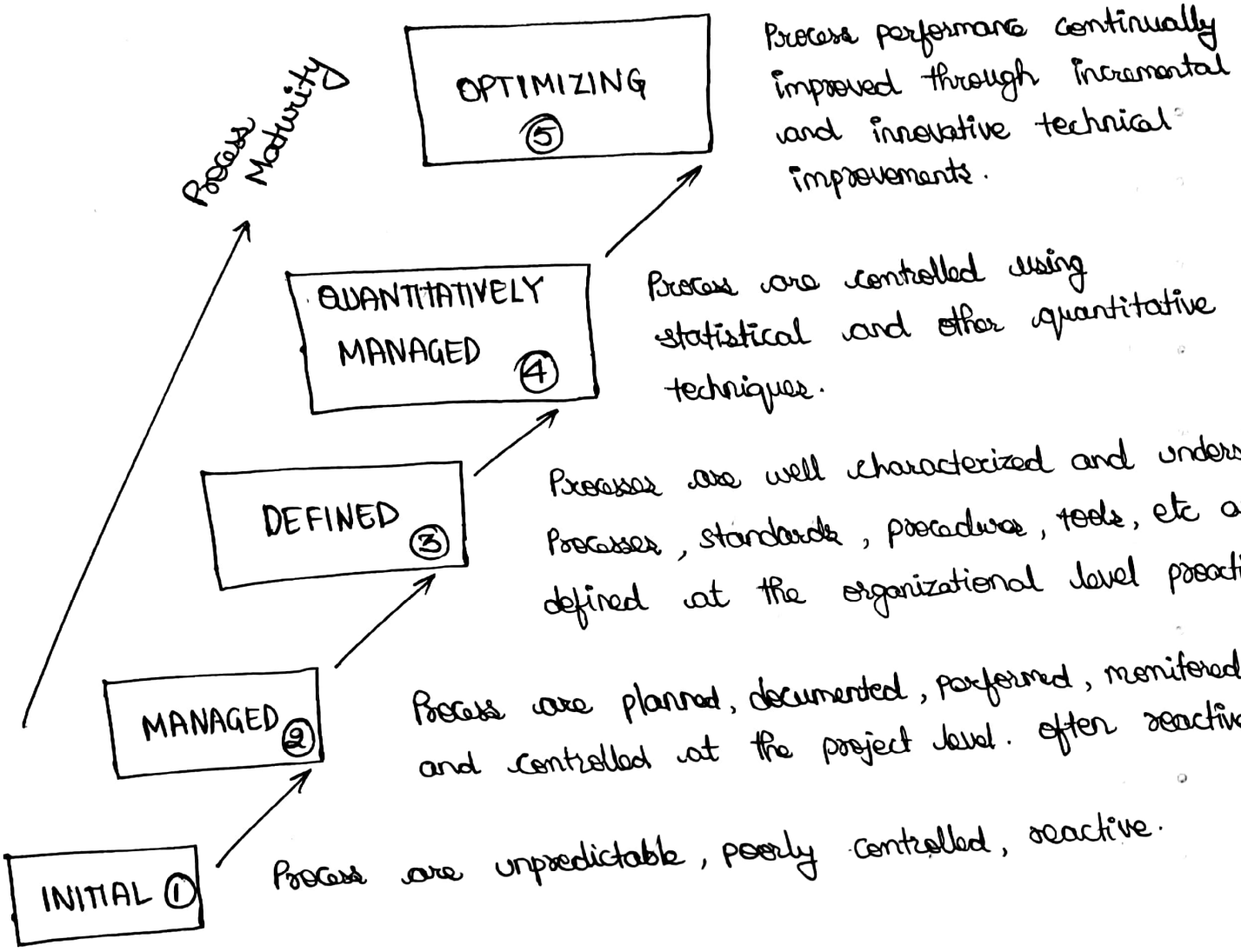
4. Maturity Level 4 : QUANTITATIVELY MANAGED

- quantitative objectives for process performance and quality are set.
- quantitative objectives are based on customer requirements, organization needs, etc..
- process performance measures are analyzed quantitatively.
- higher quality of processes is achieved.
- Lower risk.

5. Maturity Level 5 : OPTIMIZING

- continuous improvement in processes and their performance.
- improvement has to be both incremental and innovative.

- highest quality of processes.
- lowest risk in process and their performance.



*. CMMI Model - Capability levels :

- A capability level includes relevant specific and generic practices for a specific process area that can improve the organizations processes associated with that process area.
- For CMMI models with continuous representation, there are six capability levels as described below:

1. Capability level 0 : INCOMPLETE

- incomplete process - partially or not performed.

- one for or more generic specific goals of process area are not met.
- No generic goals are specified for this level.
- this capability level is same as maturity level 1 – INITIAL.

2. Capability level 1: PERFORMED

- process performance may not be stable.
- objectives of quality, cost and schedule may not be met.
- a capability level 1 process is expected to perform all specific and generic practices for this level.
- only a start-step for process improvement.

3. Capability Level 2: MANAGED

- process is planned, monitored and controlled.
- managing the process by ensuring that objectives are achieved.
- objectives are both model and other including cost, quality, schedule.
- actively managing processing with the help of metrics.

4. Capability Level 3: DEFINED

- a defined process is managed and meets the organization's set of guidelines and standards.
- focus is process standardization.

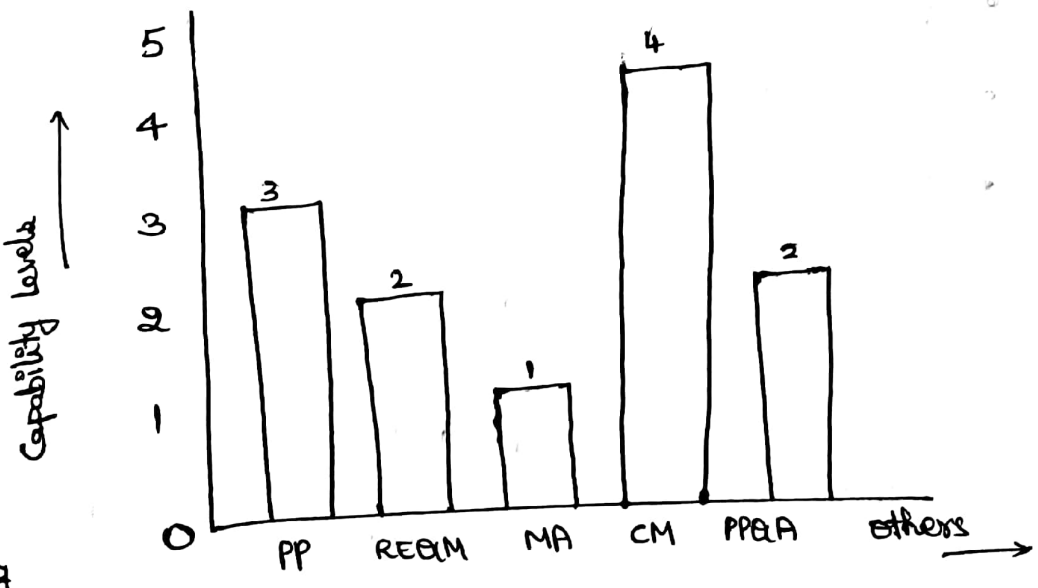
5. Capability Level 4 : Quantitatively Managed

- process is controlled using statistical and quantitative techniques
- process performance and quality is understood in statistical terms and metrics.
- quantitative objectives for process quality and performance are established.

6. Capability Level 5 : OPTIMIZING

- focuses on continually improving process performance.
- performance is improved in both ways – incremental, innovation.
- emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

CMMI Process Area Capability Profile



PP – Project Planning
REQM – Requirements Management
MA – Measurement & Analysis
CM – Configuration Management
PPQA – process and product QA

- The continuous CMMI meta-model describes a process in 2D in above graph.
- All processes are rated according to the capability levels.

- For Example, Project Planning is one of eight process areas defined by the CMMI for "project management" category. The specific goals (SG) are the associated specific practices (SP) defined for project planning are :

SG 1 : Establish Estimates :

1.1 Estimate the scope of project.
1.2 Establish Estimates of Work Product and Task Attributes.
1.3 Define Project Life Cycle.
1.4 Determine Estimate of Effort and Cost.

SG 2 : Develop a Project Plan :

2.1 Establish the budget and schedule.
2.2 Identify the project risks.
2.3 Plan for Data Management.
2.4 Plan for Project Resources.
2.5 Plan for needed knowledge & skills.
2.6 Plan Stakeholder Involvement.
2.7 Establish the project plan.

SG 3 : obtain Commitment to the Plan :

3.1 Review Plans that affect the project.
3.2 Reconcile Work and Resource Levels.
3.3 obtain Plan Commitment.

- In addition to specific goals and practices, the CMMI also defines a set of five generic goals and related practices for each process area.

- To achieve a maturity level, the specific goals and practices associated with a set of process areas must be achieved.
- The relationship between maturity levels and process area is shown below:

⇓ PROCESS AREAS REQUIRED TO ACHIEVE MATURITY LEVEL

Level	Focus	Process Areas
⑤ Optimizing	Continuous process improvement	organizational innovation and deployment causal analysis and resolution.
④ Quantitatively Managed	Quantitative Management	organizational process performance Quantitative project management
③ Defined	Process standardization	Requirements development Technical solution Product Integration Verification & validation organizational process focus organizational process definition organizational training Integrated Project Management Risk Management
② Managed	Basic Project Management	Requirements management Project planning Project Monitoring & Control Supplier Agreement Management Measurement & Analysis Process & Product Quality Assurance
① Performed		

THE COST OF QUALITY :

- There is a cost of activity in every project, it should have business value and software testing is no exception.
- To optimize its business value, test managers should optimize testing approximately and ends up. There should not be unnecessary testing otherwise it causes unnecessary delays and ends up incuring more costs.
- Also, there should not be incomplete or too less testing otherwise, there may be a chance of defective products to be handed to the end users. Therefore, software testing should be done approximately.

Cost of Quality :

- It is the most established, effective measure of quantifying and calculating the business value of testing. There are four categories to measure cost of quality : Prevention costs, Detection costs, Internal failure costs, and External failure costs.
- These are explained as follows below:

1) Prevention Costs include cost of training development / developers on writing secure and easily maintainability code.
2) Detection Costs include the cost of creating test cases, setting up testing environments, revisiting testing requirements.
3) Internal failure costs include costs incurred in fixing defects just before delivery.
4) External failure Costs includes product support costs incurred by delivering poor quality software.

- Major parts of total costs are detecting defects and internal failure cost. But, these are less costs than external failure costs. That's why testing provides good business value.

Business value of Software Testing :

- Test manager has responsibility to identify the business value and provide communications between teams and senior management. It concerns the cost of quality.
- To deliver business value, these are some of the measured ways, like :

1) Detecting defects.
2) Documenting defects.
3) Status reports & test metrics on project progress.
4) Status reports on process implementation and product development.
5) Increasing confidence in product quality.
6) Legal Liabilities.
7) Decreasing risks.
8) Ensuring predictable product.
9) Enhancing reputation for product.
10) Enhancing reputation for process quality.
11) Testing can prevent mission failure.

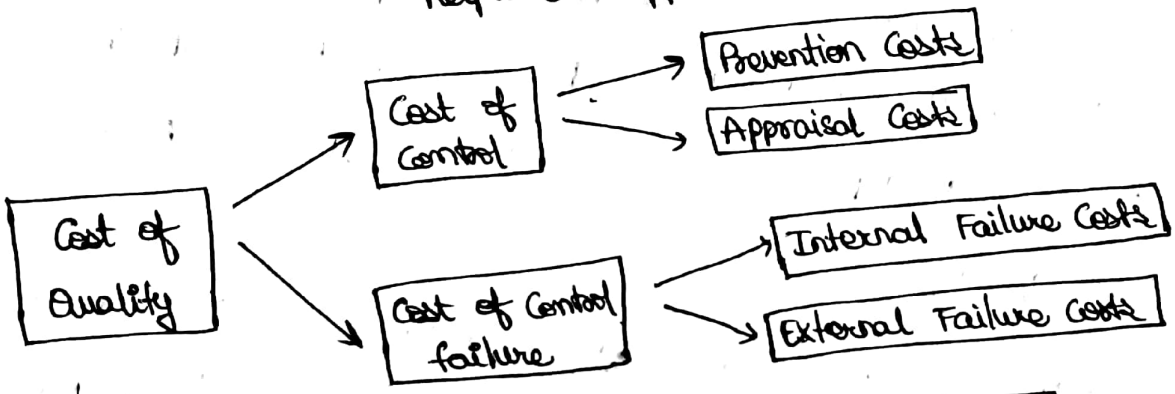
Examples :

1) Prevention Costs : quality planning, project management, feature review, QA product review, Agile & process review, team training

2) Appraisal Costs : Technical reviews cost, cost of data collection, metrics evaluation, testing & debugging, Quality Control

3) Internal Failure Cost : Cost of rework, re-testing, side effects of it.

4) External Failure Cost : Product service, liability, recall, external defects complaint resolution, product return, replacement helpline support, labour charges, etc...



COQ = Cost of Control + Cost of failure of Control *

EXAMPLE (CASE) :

- Say, we are developing a mobile app with 2 scenarios : with and without quality management. In each case, we are dealing with 800 errors (bugs) total, assume a $80 price to fix a bug found internally, while a $100 price to fix a bug found externally.
- Consider following 2 cases to understand about it :

1) NO Quality Management

2) 100 H for Quality Management

1) First Case – NO QUALITY MANAGEMENT :

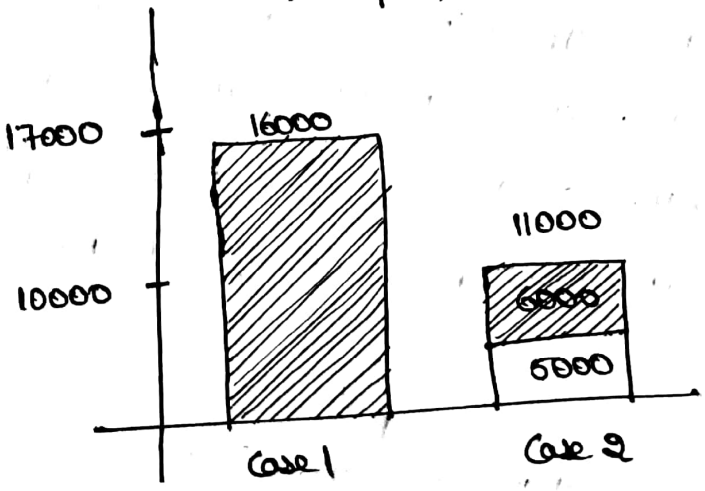
- In the first case, without quality management in place, COQ investment is zero, and we only spend money to fix bugs.
- Say, we found 50 bugs internally, and 150 are reported by customers after they have used the app.

$$\text{Total COQ} = (50 \times \$90) + (150 \times \$100) = \$16000.$$

2) Second Case – 100 H FOR QUALITY MANAGEMENT :

- In the second case, lets assume, we spend 100 additional hours on quality management procedures. Thus, at the average $50 hourly developer rate, we invest about $5,000 in software quality.
- As a result, we detect more bugs internally – 175, lower external bugs to 25 (after the app has been delivered to public).

$$\text{Total COQ} = \$5000 + ((175 \times \$80) + (25 \times \$100))$$
$$= \$11,000$$



- We see a favour in case (2)

□ – Investment

▨ – COQ

- If you invest in essential features of a product and you bind and ensure real quality there, then COQ in SDE is really worth considering.

Advantages :

- Provides cost - benefit justification for needed Corrective Actions & Improvement projects.
- Assists in quantifying the costs and associated with inefficient or incapable process that result in unwanted variation & waste.
- Highlights Strengths & Weakness of Quality & Manufacturing System.
- Reframes improvement opportunities into financial benefits for ROI.
- Finally it saves time, money, effort, reputation of the company.

Disadvantages :

- Doesnot lead to improvement.
- Showcases scorebased of only current performance.
- Cannot determine Root Cause Analysis.
- Inability to Quantify the hidden Quality costs.

Conclusion :

- It is noted that most IT-Companies end-up with 15-80% quality-related costs out of total sales and revenue, few of them spend even more.
- A rule of thumb for efficient and profitable workflow would be 10 to 15%.

BUSINESS PROCESS REENGINEERING (BPR):

- Business Process reengineering (BPR) extends far beyond the scope of information technologies and software engineering.
- Among the many definitions that have been suggested for BPR is one published in magazines.
- "A search for, and the implementation of, radical change in business process to achieve breakthrough results"
- A business process is "a set of logically related tasks performed to achieve a defined business outcome"

Ex: Designing a new product, purchasing services and supplies, hiring a new employee, and paying suppliers.

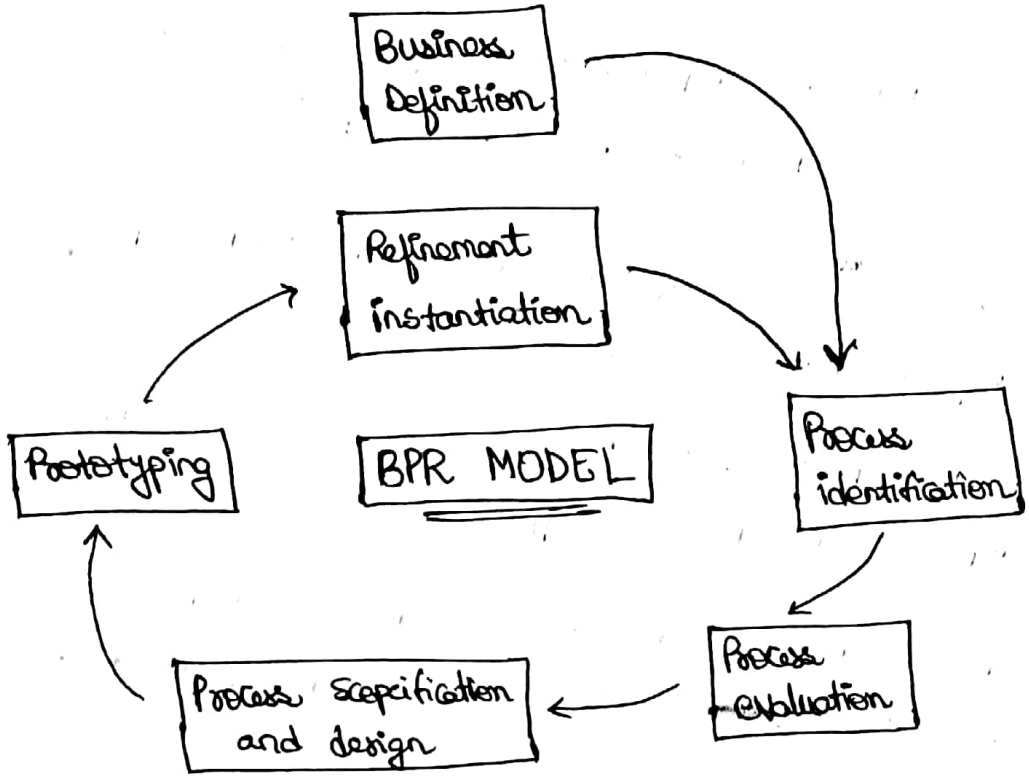
- Every business process has a defined customer - a person or group that receives the outcome (e.g., an idea, a report, a design, a service, a product).
- Every system is actually a hierarchy of subsystems. A business is no exception. The overall business is segmented in:

The Business → Business systems → Business Process → Business sub processes

*. A BPR MODEL:

IMP
- "Like most engineering activities, business process reengineering is iterative."

- Business Goals and the process that achieve them must be adapted to changing business environment.
- For this reason, there is no start and end to BPR – it is an evolutionary process. A model for business process reengineering is in below figure.



- The model defines six activities :

1) Business definition – Business goals are identified within the context of four key drivers : cost reduction, time reduction, quality improvement and personal development and empowerment. Goals may be defined at the business level or for a specific component of business.

2) Process identification – Process that are critical to achieving the goals defined in the business definition are identified. They may be ranked by importance, by need for change, or in any other way that is appropriate for the reengineering activity.

3) Process evaluation - The existing process is thoroughly analyzed and measured. Process tasks are identified : the costs, and time consumed by process tasks are noted ; and quality / performance problems are isolated.

4) Process specification and design - Based on information obtained during the first three BPR activities, use cases are prepared for each process that is to be redesigned. Within the context of BPR, we use cases identify a scenario that delivers some outcome to a customer. Within the use case as the specification of the process, a new set of tasks are designed for the process.

5) Prototyping - A redesigned business process must be prototyped before it is fully integrated into the business. This activity "tests" the process so that refinements can be made.

6) Refinement and instantiation - Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

- These BPR activities are sometimes used in conjunction with workflow analysis tools. The intent to these tools to build a model of existing workflow in an effort to better analyze existing processes.

- The objective of BPR tools is to support the analysis of assessments of existing business process and the specification of design of new ones.

- In general, BPR tools allow a business analyst to model existing business process in an effort to assess workflow inefficiencies or functional problems. Once existing problems are identified, tools allow the analysis to prototype and/or simulate the revised business process.

Conclusions: (What I think):

- BPR can work, if it is applied by motivated, trained people who recognize that process reengineering is continuous activity.
- If BPR is conducted effectively, information systems are better integrated into business process.
- But even if Business reengineering is a strategy that is rejected by a company, software reengineering is something that must be done.
- Ten of thousands of legacy systems – applications that are crucial to the success of business large & small – are in dire need of refurbishing and rebuilding.

EXAMPLE: "FAST FOOD COMPANY". Completely redesigning the delivery of products can give you unexpected results. In restaurant, the process goes like all stress, the customer orders, the order goes to the kitchen, prepares the meal and delivers it to the consumer.

CSE1005

SOFTWARE ENGINEERING

NOTES

THE END