Software characteristics:

Different software characteristics decide whether the software is good or bad. These attributes reflect the quality of a software product. And these are depended on the application of the software also.

For example, a banking system must be secure while a telephone switching system must be reliable.

Following are the characteristics of good software: (Qualities of good software).

Understandability: Software should be easy to understand, even to novice users. It should be efficient to use.

Cost: Software should be cost effective as per its usage.

Maintainability: Software should be easily maintainable and modifiable in future.

Modularity: Software should have modular approach so it can be handled easily for testing.

Functionality: Software should be functionally capable to meet user's requirements.

Reliability: It should have the capability to provide failure-free service.

Portability: Software should have the capability to be adapted for different environments.

Correctness: Software should be correct as per its requirements.

Documentation: Software should be properly documented so that we can re-refer it in future.

Reusability: It should be reusable, or its code or logic should be reusable in future.

Interoperability: Software should be able to communicate with various devices using standard bus structure protocol.

Verifiability: Software should be verifiable with its properties and functionalities with its planning and analysis in previous phase.

Along with these characteristics, software has some special characteristics which are explained below.

(IEEE Definition) Software Engineering is the application of a systematic, disciplined and quantifiable approach to the development, operation and maintenance of software.

Software Engineering - A layered approach:

Software engineering can be viewed as a layered technology.

It contains process, methods and tools that enable software product to be built in a timely manner.

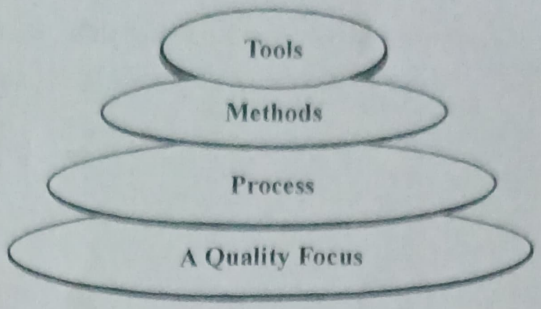


Figure: Software Engineering Layers

As we can see in the above figure, this layered approach contains mainly four layers.

1. **A quality focus layer:**
   - Software engineering mainly focuses on quality product.
   - It checks whether the output meets with its requirement specifications or not.
   - Every organization should maintain its total quality management (TQM).
   - This layer supports software engineering.
2. **Process layer:**
   - Software process is a set of activities together if ordered and performed properly, the desired result would be produced.
   - Main objective of this layer is to develop software in time.
   - This layer is the heart of software engineering.
   - It holds all the technology layers together like GLUE.
   - It is also working as foundation layer.
   - It defines the framework activities.
3. **Method layer:**
   - It provides technical knowledge for developing software. It describes 'how-to' build software product.
   - It creates software engineering environment to software product using CASE tools. (CASE tools combines software, hardware and software engineering database).
   - This layer includes requirements analysis, design, program construction, testing, and support.
4. **Tools layer:**
   - It provides support to below layers using automated or semi-automated tools.
   - Due to this layer, process is executed in proper manner.

- **Need of software engineering:**

The reasons why software engineering is needed to develop software products are given below:

- To help developers to obtain high quality software product.
- To develop the product in appropriate manner using life cycle models.
- To acquire skills to develop large programs.

- To acquire skills to be a better programmer.
- To provide a software product in a timely manner.
- To provide a quality software product.
- To provide a software product at an agreed cost.
- To develop ability to solve complex programming problems.

Also learn techniques of specification, design, user interface development, testing, project manageme

- Shortly, it can be said that software is important to make things easier, faster, more reliable and safer.

Generic view of software engineering

- The work associated with software engineering can be categorized into three generic phases → Definition phase, Development phase and Support phase.

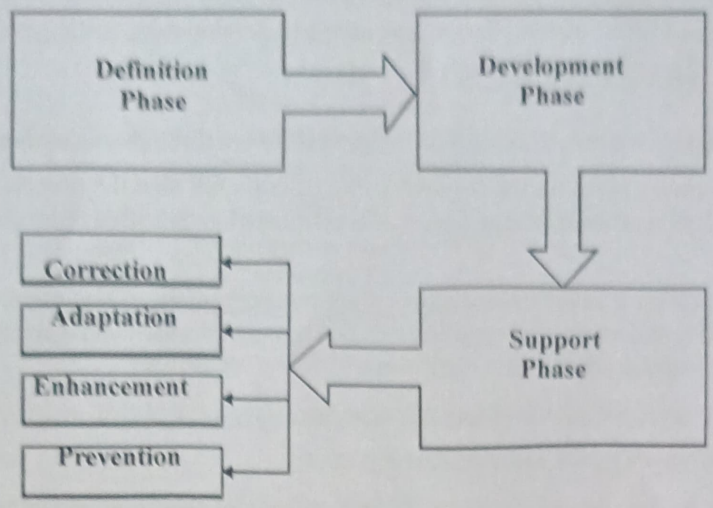


Figure: Software generic view

1. The Definition Phase:

- It focuses on *"what part"*. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behaviour can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system.
- The key requirements of the system and the software are identified.
- Three main activities performed during this phase: system or information engineering, software project planning and requirement analysis.

2. The development phase:

- It focuses on *"how part"* of the development. During development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized, how the design will be translated into a programming language, and how testing will be performed.
- Main activities are performed under this phase are: software design, code generation and software testing.

3. The support phase:

- The support phase focuses on "change" associated with error correction.
- Four types of change are encountered during the support phase:
  - → **Correction:** corrective maintenance changes the software to correct defects.
  - → **Adaption:** Adaptive maintenance results in modification to the software to accommodate changes to its external environment.
  - → **Enhancement/Perfection:** Perfective maintenance extends the software beyond its original functional requirements.

→ **Prevention:** Preventive maintenance to enable the software to serve the needs of its end users.

The definition phase focuses on **'what'**
The development phase focuses on **'how'**
The support phase focuses on **'change'**

GENERIC (NOT SPECIFIC TO A GROUP) FRAMEWORK ACTIVITIES

A software framework provides a standard way to build deploy software product. And a software process is the set of activities and associated results that produce a software product.

Any standard software process model would primarily consist of two types of activities: **A set of framework activities,** which are always applicable to all the projects and **A set of umbrella activities** which are non SDLC activities that are applicable throughout the process.

It provides common process framework for all projects.

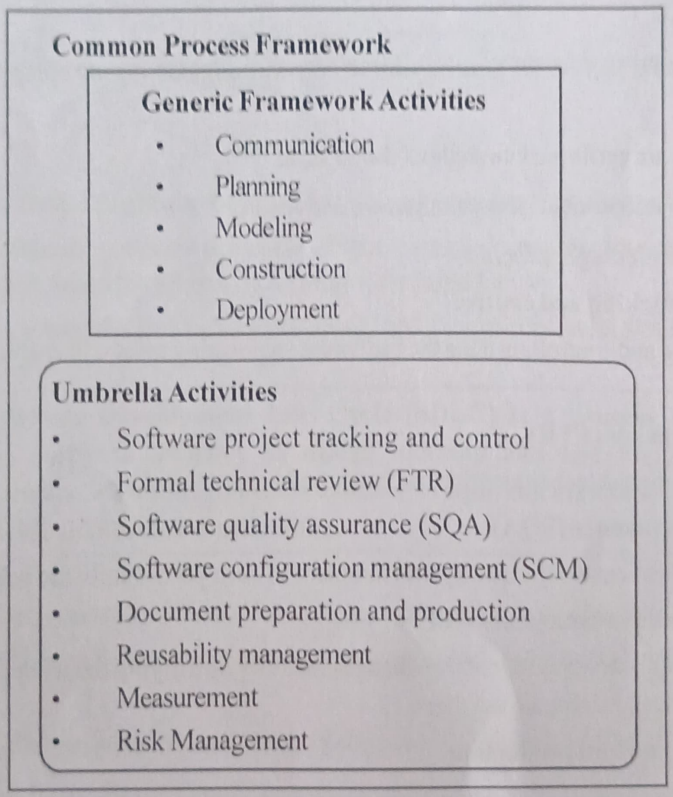


Figure: Generic framework and Umbrella activities

The Adaptable Process Model (APM) defines the following set of framework activities.

Communication

The software development starts with the communication between customer and developer.

- If the customer wants some corrections or demands for the additional capabilities, then the change is [illegible] improvement in the quality of the software.

Umbrella activities:

- The phases and related steps of the generic view of software engineering are complemented by a number of umbrella activities.
- Umbrella activities are performed throughout the process.
- These activities are independent of any framework activity.

Typical activities in this category include:

1. **Software project tracking and control**

   When project tracking and controlling done then software engineering tasks will enable to get the job done on time.

2. **Formal technical review (FTR)**

   This includes reviewing the techniques that has been used in the project.

3. **Software quality assurance (SQA)**

   This is very important to ensure the quality measurement of each part of software being developed.

4. **Software configuration management (SCM)**

   SCM is a set of activities designed to control changes made by identifying the work products that are likely to change, establishing relationships among them.

5. **Document preparation and production**

   All the project planning and other activities should be hardly copied and the production gets started here.

6. **Reusability management**

   This includes the backing up of each part of the software project they can be corrected or any kind of support can be given to them later to update or upgrade the software at user/time demand.

7. **Measurement (estimation)**

   This will include all the measurement or estimation of every aspects of the software project like: time estimation, cost estimation etc.

8. **Risk management**
   - As we know that 'tomorrow's problem is today's risk'. Risk management is very important activity for any type of software development.
   - It identifies potential problems and deal with them when they are easier to handle before they become critical.
   - Risk management allows early identification of risks and provide management decisions to the solutions, and improve quality of the product.

SOFTWARE DEVELOPMENT MODELS / LIFE CYCLE MODELS

- Every system has a life cycle. It begins when a problem is recognized, after then system is developed, grows until maturity and then maintenance needed due to change in the nature of the system. So it died and new system or replacement of it taken place. (This can be shown in below figure)
- Software process models describe the sequence of activities needed to develop a software product.

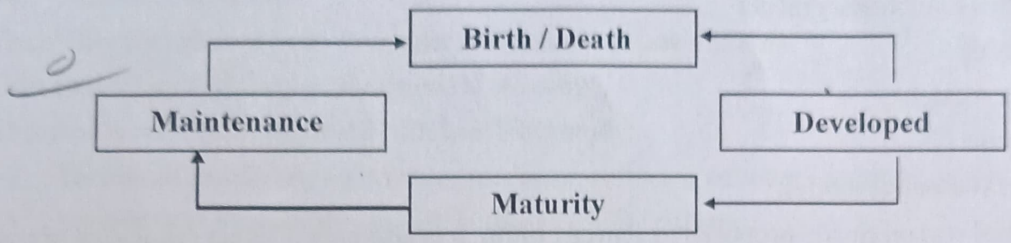


Figure: System Life Cycle

- The goal of the software development process is to produce high quality software product.
- As per IEEE Standards, software life cycle is: "the period of time that starts when software product is conceived and ends when the product is no longer available for use."
- A software life cycle model is also called a Software Development Life Cycle (SDLC). More suitable definition of SDLC is given below:

> **Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software, which meets exact customer's requirements and completion within time and cost estimation.**

- Software life cycle is the series of identifiable stages that a software product undergoes during its lifetime.
- Software life cycle model (process model) is a descriptive and diagrammatic representation of the software life cycle.
- A life cycle model represents all the activities required to make a software product transit through its life cycle phases.
- General stages of software development life cycle are → feasibility study, requirement analysis and specification, design, coding, testing and maintenance.

Need of life cycle models:

- The documentation of life cycle models enhances the understanding between developers and ...

Different Software Development Life Cycle (SDLC) models OR Software Development Models:

Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

(1) Classical waterfall model
(2) Iterative waterfall model
(3) Incremental Evolutionary model
(4) RAD model
(5) Prototype model
(6) Spiral model

(1) Classical Waterfall model:

- This model was originally proposed by Royce (1970). It is also called 'linear sequential life cycle model'.
- It is also called 'traditional waterfall model' or 'conventional waterfall model'.
- It's just a theoretical way of developing software All other life cycle models are essentially derived from it.
- This model breaks down the life cycle into set of phases like:
  - Feasibility study
  - Requirements analysis and specification
  - Design
  - Coding and unit testing
  - Integration and system testing
  - Maintenance

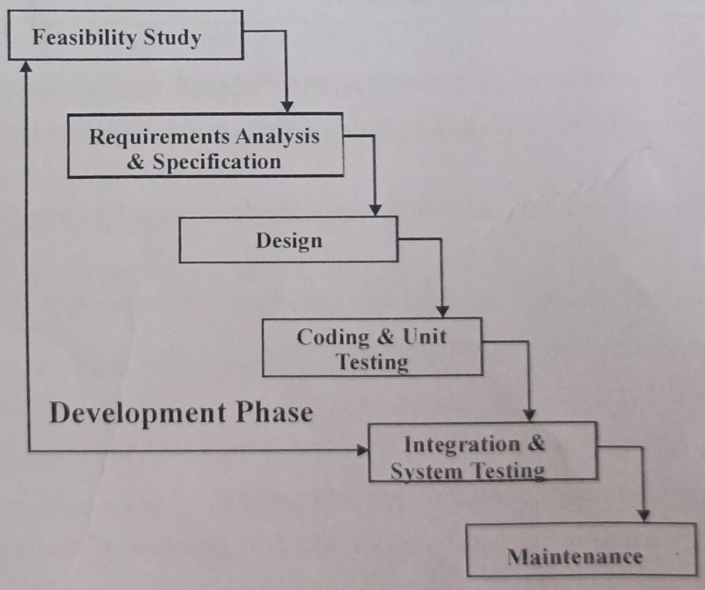


Figure: Classical Waterfall model

Now let's discuss the characteristics and working of every phase one by one.

1. Feasibility Study:

- Aim of this phase is to determine whether the system would be financially and technically feasible to develop the product.
- This is an abstract definition of the system.
- It includes the analysis of the problem definition and collection of relevant information of input, processing and output data.
- Collected data are analysed for:
  - → abstract definition Formulation of
  - → different solutions Analysis of
  - → alternative solutions
- Feasibility is evaluated from developer and customer's point of view.
- Cost/Benefit analysis is also performed at this stage.
- Three main issues are concerned with feasibility study:
  - → **Technical Feasibility** → contains hardware, software, network capability, reliability and availability.
  - → **Economical Feasibility** → contains cost/benefit analysis.
  - → **Operational Feasibility** → checks the usability. Concerned with technical performance and acceptance within the organization.

2. Requirement Analysis and Specification:

- Aim of this phase is to understand the exact requirements of the customer and to document them properly.
- It also reduces communication gap between developers and customers.
- Two different activities are performed during this phase:
  1. Requirements gathering and analysis
  2. Requirements specification
- **Requirement gathering:** The goal of this activity is to collect all relevant information from the customer regarding the product to be developed.
- **Requirements analysis:** Analyse all gathered requirements to determine exact needs of the customers/clients and hence improve the quality of the product.
- **Requirements specification:** During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.
- The important components of SRS are → the functional requirements, the non-functional requirements and the constraints of the system.
- Output of this phase is → SRS document, which is also called Black box specification of the problem.
- This phase concentrates on "what" part, not "how".

> ☐ Requirement gathering and requirement analysis & specification collectively called **'Requirement Engineering'.**

3. Design:

- The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
- This phase affecting the quality of the product.
- Two main approaches are concerned with this phase:
  1. Traditional design approach
  2. Object oriented design approach

Traditional design approach:

- This approach divided into two parts: structure analysis and structure design.

Structural Analysis:

- Where the detailed structure of the problem is examined.
- Identify the processes and data flow among these processes.
- DFD is used to perform the structure analysis and to produce result.
- In structure analysis functional requirement specified in SRS are decomposed and analysis of data flow is represented diagrammatically by DFD.

Structure Design:

- During structured design, the results of structured analysis are transformed into the software design.
- Two main activities are associated with it :
  - → **Architectural design (High-level design)** - decomposing the system into modules and build relationship among them.
  - → **Detailed design (Low-level design)** - identified individual modules are design with data structure and Algorithms.

Object oriented design approach:

- In object-oriented approach, objects available in the system and relationships between them are identified.
- This approach provides lower development time and effort.
- Several tools and techniques are used in designing like:

Flow chart	DFD	Data Dictionary
Decision Table	Decision Tree	Structured English

4. Coding and Unit testing:

- It is also called the implementation phase.
- The aim of this phase is to translate the software design into source code and unit testing is done module wise.
- Each component of the design is implemented as a program module, and each unit is tested to determine the correct working of the system.
- The system is being operational (working conditions) at this phase.
- This phase affects testing and maintenance.
- Simplicity and clarity should be maintained.
- Output of this phase is → set of program modules that have been individually tested.

5. **Integration and system testing:**

- Once all the modules are coded and tested individually, integration of different modules is undertaken.
- The modules are integrated in a planned manner.
- This phase is carried out incrementally over a number of steps and during each integration step, the partially integrated system is tested and a set of previously planned modules are added to it.
- Finally, when all the modules have been successfully integrated and tested, system testing is carried out.
- Goal of this phase is → to ensure that the developed system works well to its requirements described in the SRS document.
- Basic function of testing is to detect errors. So, it is also called 'quality control measure'.
- Testing procedure is carried out using test data: Program test and System test. Program test is done on test data and system test is done actual data.
- Proper test cases should be selected for successful testing.
- It consists of three different kinds of testing activities.
  - → **α- testing:** It is the system testing performed by the development team.
  - → **β-testing:** It is the system testing performed by a friendly set of customers.
  - → **Acceptance testing:** It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Output of this phase is → system test plan and test report (error report).

6. **Maintenance:**

- It requires maximum efforts among all the phases to develop software product.
- This phase is needed to keep system operational.
- Generally, maintenance is needed due to change in the environment or the requirement of the system.
- Maintenance involves performing following four kinds of activities:
  - → **Corrective maintenance** - Correcting errors that were not discovered during the product development phase. It is done after product development.
  - → **Perfective maintenance** - Improving and enhancing the functionalities of the system according to the customer's requirements.

    It mainly deals with implementing new or changed user requirements and improving system performance.
  - → **Adaptive maintenance** - Porting the software to work in a new environment and ensure working. It also consists of adaption of software in the changes of environment.
  - → **Preventive maintenance** - is scheduled, routine maintenance to keep equipment running as well as prevent downtime and expensive repair cost. It reduces the likelihood of failure.

In this phase, the system is reviewing for studying the performance and knowing the full capabilities of the system.

- **Advantages of waterfall model:**
  - It is simple and easy to understand and use.
  - Clearly defined stages. As each phase has well defined input and outputs.
  - It helps project personnel in planning.
  - Waterfall model works well for smaller projects where requirements are very well understood.

- which technology is understood.

→ When the project is small.

(2) Iterative Waterfall model:

- Classical waterfall model is idealistic: It assumes that no defect is introduced during any development activity.
- But in practice defects do get introduced in almost every phase of the life cycle. Even defects may get at much later stage of the life cycle. So, solution of this problem is iterative waterfall model.
- Iterative waterfall model is by far the most widely used model. Almost every other model is derived from the waterfall model.
- The principle of detecting errors as close to its point of introduction as possible - is known as "phase containment of errors."
- Phase containment of errors can be achieved by reviewing after every milestone.

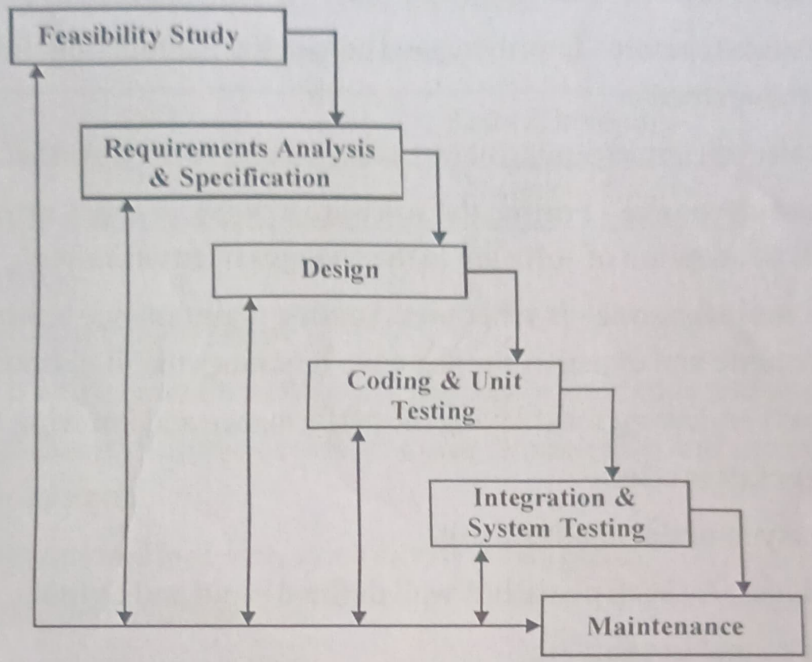


Figure: Iterative Waterfall Model

(3) Incremental Model:

- It is also referred as the successive version of waterfall model using incremental approach and evolutionary model.
- In this model, the system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core product of the system. The core product is used by customers to evaluate the system.
- The initial product skeleton is refined into increasing levels of capability : by adding new functionalities in successive versions.

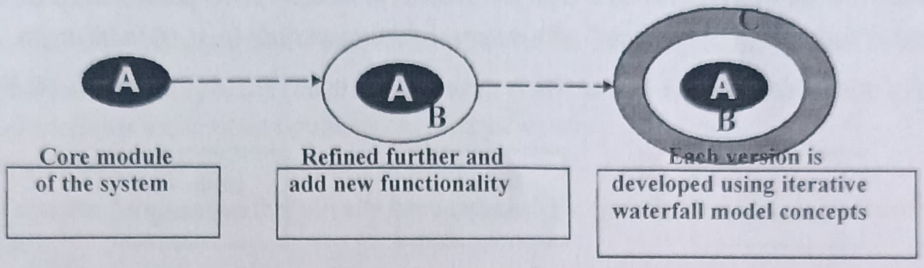


Figure: Incremental model

- From above figure, we can analyse that at the initial stage, core module/product is developed. Then by adding customer requirements or new functionalities the incremental version is built. Each version is developed using iterative waterfall model concepts.

Advantages:

→ Each successive version performing more useful work than previous versions.

→ Initial product deliver is faster.

→ The core modules get tested thoroughly, thereby reducing chance of errors in final product.

→ The model is more flexible and less costly to change the scope and requirements.

→ User gets a chance to experiment with partially developed software.

→ This model helps finding exact user requirements.

→ Feedback providing at each increment is useful for determining the better final product.

Disadvantages:

→ Sometimes it is difficult to subdivide problems into functional units.

→ Model can be used for very large problems.

→ It needs good planning and design.

→ Resulting product cost may exceed.

Applications:

→ When the problem is very large and user requirements are not well specified at initial stage.

→ When new technology is used in development.

(4) RAD model:

- Full form of RAD is - Rapid Application Development. This model was proposed by IBM in 1980.
- Rapid application development (RAD) is an incremental software development process model that emphasize on extremely short development cycle.
- It emphasizes on reuse.
- The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- If requirements are well understood and project scope is limited, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days).
- Figure of this model is shown below. Many development teams are working parallel to complete the task.

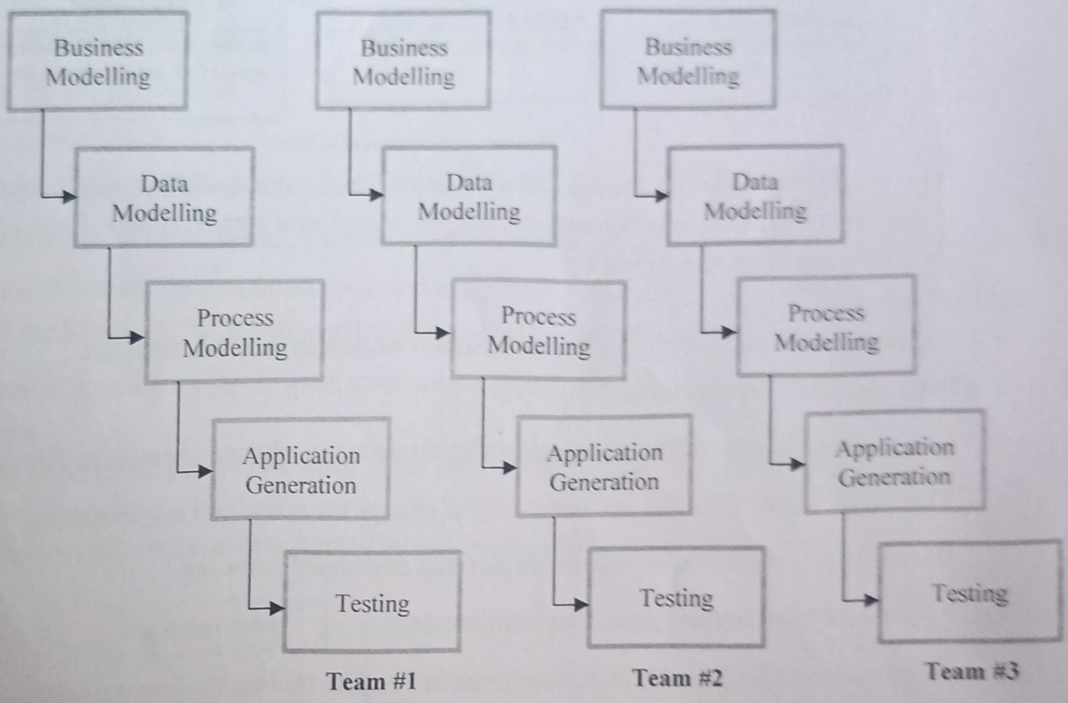


Figure: RAD Model

- **Phases of RAD model are:**

1. **Business modeling**

- The information flow among business functions is carried out in this phase. Business functions are modelled.

2. **Data modeling**

- The characteristics of objects (attributes) and their relationships are defined.

3. **Process modeling**

- The data objects defined in data modeling are transformed to implement business functions.

4. **Application generation**

- Automated tools are used to convert process models into code and the actual system. Tools like VB, VC++ and JAVA etc. are used.
- RAD works to reuse existing components or create new ones.

5. **Testing and turn over**

- As RAD uses the components that already been tested so the time for developing and testing is reduced.

Advantages:

→ Application can be developed in a quick time.

→ This model highly makes use of reusable components.

→ Reduce time for developing and testing.

→ Due to full customer involvement, customer satisfaction is improved.

Disadvantages:

→ Requirements must be cleared and well understood for this model.

→ It is not well suited where technical risk is high.

→ In it, highly skilled and expert developers are needed.

→ User involvement is necessary.

Applications:

→ The RAD model is mostly used when the system is modularized and all the requirements are well defined.

→ When a system needs to be produced in a short span of time (2-3 months).

→ RAD SDLC model should be chosen only if resources with high business knowledge are available.

→ When the technical risk is less.

(5) Prototype model:

- Prototype is a working physical system or sub system. Prototype is nothing but a 'toy implementation of a system'.
- In this model, before starting actual development, a working prototype of the system should be built first.
- A prototype is actually a partial developed product.
- A prototype usually turns out to be a very crude version of the actual system.
- Compared to the actual software, a prototype usually has:
  - limited functional capabilities
  - low reliability
  - inefficient performance
- Prototype usually built using several shortcuts, and these shortcuts might be inefficient, inaccurate or dummy functions.
- Prototype model is very useful in developing GUI part of system. The prototype model is shown in below figure.

In working of the prototype model, product development starts with initial requirements gathering phase.

- Then, quick design is carried out and prototype is built.
- The developed prototype is then submitted to the customer for his evaluation.
- Based on customer feedback, the requirements are refined and prototype is modified.
- This cycle of obtaining customer feedback and modifying the prototype continues till the customers approve the prototype.
- Then the actual system is developed using the different phases of iterative waterfall model.
- There are two types of prototype model.

I. **Exploratory development prototype model:** in which the development work is done with the users to explore their requirements and deliver a final system.

II. **Throw away prototype model:** in which a small part of the system is developed and given to the customer to try out and evaluate. Then feedback is collected from customer and accordingly main system is modified. The prototype is then discarded.

Advantages:

→ A partial product is built at initial stage, so customers can get a chance to have a look of the product.
→ New requirements can be accommodated easily.
→ Missing functionalities identified quickly.
→ It provides better flexibility in design and development.
→ As the partial product is evaluated by the end users, more chance of user satisfaction.
→ Quick user feedback is available for better solution.

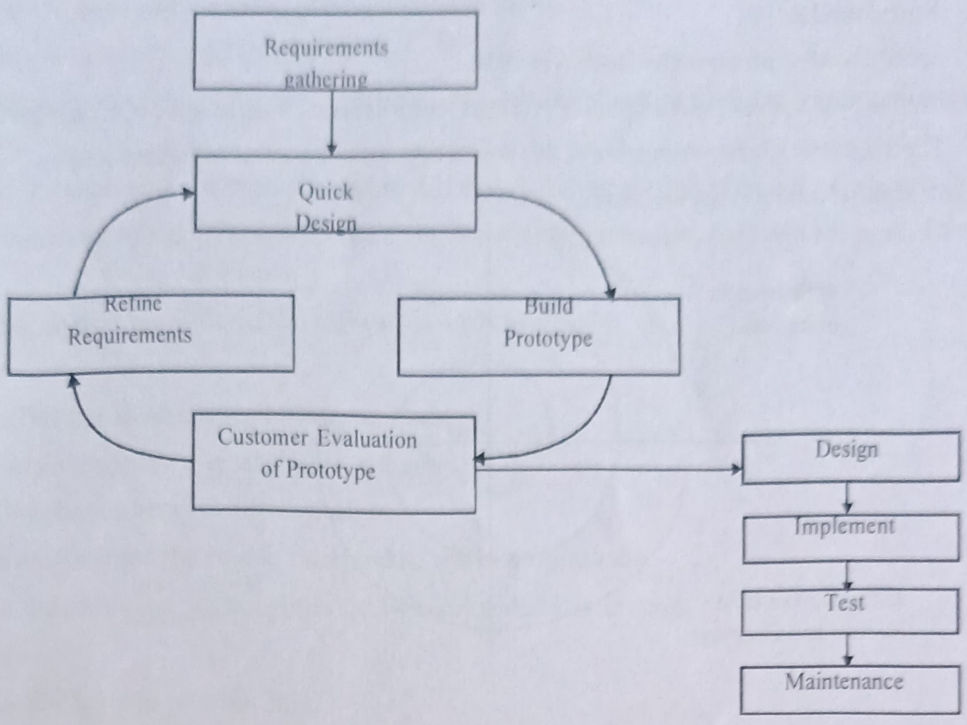


Figure: The Prototype Model

Disadvantages:

→ The code for prototype model is usually thrown away. So, wasting of time is there.

→ The construction cost of developing the prototype is very high.

→ If end user is not satisfied with the initial prototype, he may lose interest in the final product.

→ This model requires extensive participation and involvement of the customers that is not possible every time.

Applications:

→ This model used when desired system needs to have a lot of interactions with end users.

→ This type of model generally used in GUI (Graphical User Interface) type of development.

(6) Spiral model:

- Spiral model is proposed by Boehm in 1986.
- In application development, spiral model uses fourth generation languages (4GL) and developments tools.
- The diagrammatic representation of this model appears like a spiral with many loops.

Figure of spiral model is shown below.

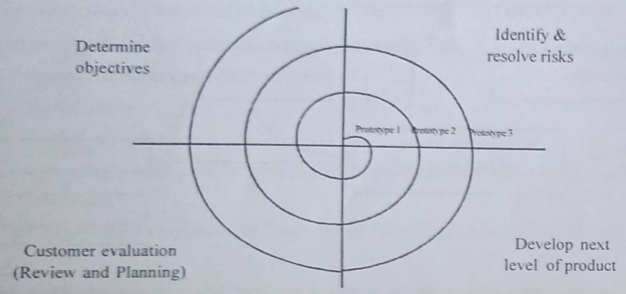


Figure: Spiral Model

Each loop of the spiral represents a phase of the software process:

- The innermost loop might be concerned with system feasibility,
- The next loop with system requirements definition,
- The next one with system design, and so on
- Number of phases is not fixed in this model.
- This model is more flexible compared with other models.
- Each loop in the spiral is split into four sectors (quadrants).

1st Quadrant- Determine objectives:

- Identifying the objectives, identifying their relationships and find the possible alternative solutions. Objectives like: performance, functionality, hardware software interface etc.
- Also examine the risks associated with these objectives.

2nd Quadrant- Identify and resolve risks:

- In this phase, detailed analysis is carried out of each identified risk
- Alternate solutions are evaluated and risks are reduces at this quadrant.

3rd Quadrant - Develop next level product:

- Develop and validate the next level of the product after resolving the identified risks.
- Activities like design, code development, code inspection, testing and packaging are performed.
- Resolution of critical operations and technical issues of next level product are performed.

4th Quadrant - Review and Planning (Customer evaluation):

- In this part, review the results achieved so far.
- Plan the next iteration around the spiral. Different plans like development plan, test plan, installation plan are performed.
- With each iteration around the spiral; progressively more complete version of the software gets built.
- In spiral model at any point, Radius represents cost and Angular dimension represents progress of the current phase.
- At the end, all risks are resolved and software is ready to use.

Advantages:

- It is more flexible, as we can easily deal with changes.
- Due to user involvement, user satisfaction is improved.
- It provides cohesion between different stages.
- Risks are analyzed and resolved so final product will be more reliable.
- New idea and additional functionalities can be easily added at later stage.

Disadvantages:

- It is applicable for large problem only.
- It can be more costly to use.
- It is more complex to understand.
- More number of documents are needed as a greater number of spirals.
- Risk analysis phase requires much higher expertise.

Applications (when to use spiral model):

- Used when medium to high-risk projects.
- When users are unsure for their needs.
- When requirements are complex.

Spiral model can be viewed as a Meta model.

- Spiral model subsumes almost all the life cycle models.
- Single loop of spiral represents Waterfall Model.
- Spiral model uses a prototyping approach by first building the prototype before developing actual product.
- The iterations along the spiral model can be considered as supporting the Evolutionary Model.
- The spiral model retains the step-wise approach of the waterfall model.

ambiguities and inconsistencies from customer perception.

- Mainly two activities are concerned with this task.

Requirement gathering	Requirement analysis

Requirement gathering:

- It is usually the first part of any software product.
- This is the base for the whole development effort.
- The goal of the requirement gathering activity is → to collect all related information from the customer regarding the product to be developed.
- This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.
- In this phase, meeting with customers, analyzing market demand and features of the product are mainly focused.
- So, activity of market research (for competitive analysis) is done.
- It involves interviewing the end-users and studying the existing documents to collect all possible information.

Requirement gathering activities are:

- Studying the existing documents
- Interview with end-users or customers
- Task analysis
- Scenario analysis
- Form analysis
- Brainstorming
- Questionnaires
- Group discussion

Requirement analysis:

- The goal of the requirement analysis activity is → to clearly understand the exact requirements of the customers.
- **IEEE** defines requirement analysis as (1) the process of studying user needs and (2) The process of studying and refining system hardware or software requirements.
- Requirement analysis helps to understand, interpret, classify, and organize the software requirements in order to assess the feasibility, completeness, and consistency of the requirements.

Requirement Engineering'.

❖ SOFTWARE REQUIREMENT SPECIFICATION (SRS)

- SRS is the output of requirement gathering and analysis activity.
- SRS is a document created by system analyst after the requirements are collected from various sources.
- SRS is a detailed description of the software that is to be developed. It describes the complete behavior the system.
- SRS describes 'what' the proposed system should do without describing 'how' the software will do (what part, not how).
- It is working as a reference document to the developer.
- It provides guideline for project development, so minimizes the time and efforts for software development.
- SRS is actually a contract between developer and end user. That helps to dissolve the disagreement.

- The SRS translates the ideas of the customers (input) into the formal documents (output).
- The SRS document is known as black-box specification, because:
  - In SRS, internal details of the system are not known (as SRS doesn't specify how the system will work).
  - Only its visible external (i.e., input/output) behavior is documented.
- SRS documents serves as contract between customer and developer, so it should be carefully written. (Sometimes SRS is also written by the customers also).
- The organization of SRS is done by the system analyst.

Benefits of SRS (Features of SRS):

- SRS provides foundation for design work. Because it works as an input to the design phase.
- It enhances communication between customer and developer because user requirements are expressed in natural language.
- Developers can get the idea what exactly the customer wants.
- It enables project planning and helps in verification and validation process.
- Format of forms and rough screen prints can also be represented in SRS.
- High quality SRS reduces the development cost and time efforts.
- As it is working as an agreement between user and developer, we can get the partial satisfaction of the end user for the final product.
- SRS is also useful during the maintenance phase.

Contents of the SRS document:

- An SRS should clearly document the following three things:

(i) Functional requirements of the system

- Functional requirements are those which are related to the technical functionality of the system.
- These are the services which the end users expect from the final product. And these are the services which a system provides to the end users.
- It clearly describes each of the function that the system needs to perform along with the input and output data set.

(ii) Non-functional requirements of the system

- The non-functional requirements describe the characteristics of the system that can't be expressed functionally. For example, portability, maintainability, reliability, usability, security, performance etc.
- Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system in particular conditions, rather than specific behaviors.
- Sometimes these requirements are also called quality attributes.

(iii) Constraints (restrictions) on the system

- That describes what the system should do or should not do. These are some general suggestions regarding development.
- A constraint can be classified as:
  - Performance constraint
  - Operating system constraint

- Economic constraint
- Life cycle constraint
- Interface constraint

- **Characteristics of a good**

SRS: Concise

SRS should contain brief and concise information regarding the project; no more detailed description of the system should be there.

Complete

It should be complete regarding the project, so that can be completely understood by the analyst and developers as well as customers.

Consistent

An SRS should be consistent through the project development. Requirements may not be conflict at the later stage.

Conceptual integrity

SRS should clearly provide the concepts of the system, so that can be read easily.

Structured

SRS should be well structured to understand and to implement.

Black box view

SRS should have black box view means; there should not be much detailing of the project in it (only describe what part, not how).

Verifiable

It should be verifiable by the clients or the customers for whom the project is being made.

Adaptable

It should be adaptable in both sides from the clients as well as from the developers.

Maintainable

SRS should be maintainable so in future changes can be made easily.

Portable

It should be portable as if we can use the contents of it for the same types of developments.

Unambiguous

There should not be any alternates of SRS that creates ambiguity.

Traceable

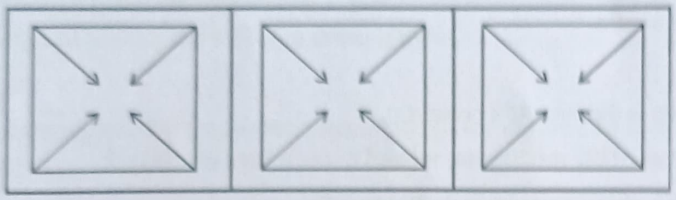
Each of the requirements should be clear and refer to the future development.

COHESION AND COUPLING

- Modularity is clearly a desirable property of any software development.
- In software development, modularity is →decomposition of a program into smaller programs (or modules).
- A system is considered modular if it consists of multiple modules so that each module can be implemented separately and debugged separately.
- Modular system provides advantages like:
  - Easy to understand the system.
  - System maintenance is easy.
  - Provide reusability.

- Modularity is successful because developers use prewritten code, which saves resources. Overall, modularity provides greater software development manageability.
- Cohesion and coupling are two modularization criteria that are often used together.
- Most researchers and developers are agreed that for good software design neat decomposition is highly needed, and the primary characteristic of neat decomposition is 'high cohesion and low coupling'.

- **Cohesion:**
  - Cohesion is →a measure of functional strength of a module.
  - Cohesion keeps the internal modules together, and represents the functional strength.
  - Cohesion of a module represents how tightly bound the internal elements of a module are to one another.



Cohesion = strengths of relations within modules

Classification of cohesion:

Coincidental	Logical	Temporal	Procedural	Communicational	Sequential	Functional

Worst (Low) → Best (High)

Coincidental cohesion

- It is the lowest cohesion. Coincidental cohesion occurs when there are no meaningful relationships between the elements.
- A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely.
- It is also called random or unplanned cohesion.

Logical cohesion

- A module is said to be logically cohesive if there is some logical relationships between the elements of module, and the elements perform functions that fall into same logical class.
- For example: the tasks of error handling, input and output of data.

Temporal cohesion

- Temporal cohesion is same as logical cohesion except that the elements are also related in time and they are executed together.
- A module is in temporal cohesion when a module contains functions that must be executed in the same time span.
- Example: modules that perform activities like initialization, cleanup, and start-up, shut down are usually having temporal cohesion.

Procedural cohesion

- A module has procedural cohesion when it contains elements that belong to common procedural unit.
- A module is said to have procedural cohesion, if the set of the modules are all part of a procedure (algorithm) in which certain sequence of steps are carried out to achieve an objective.

- Example: the algorithm for decoding a message

Communicational cohesion

- A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, for example the set of functions defined on an array or a stack.
- These modules may perform more than one function together.

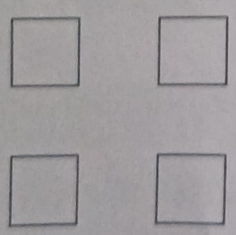
Sequential cohesion

- When the output of one element in a module forms the input to another, we get sequential cohesion.
- Sequential cohesion does not provide any guideline how to combine these elements into modules.
- For example, in a TPS (transaction processing system), the get-input, validate-input, sort-input functions are grouped into one module.

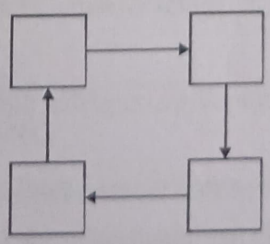
Functional cohesion

- Functional cohesion is the strongest cohesion.
- In it, all the elements of the module are related to perform a single task.
- All elements are achieving a single goal of a module.
- Functions like: compute square root and sort the array are examples of these modules.

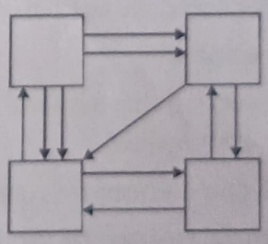
- **Coupling:**
  - Coupling between two modules is →a measure of the degree of interdependence or interaction between these two modules.
  - Coupling refers to the number of connections between 'calling' and a 'called' module. There must be at least one connection between them.
  - It refers to the strengths of relationship between modules in a system. It indicates how closely two modules interact and how they are interdependent.
  - As modules become more interdependent, the coupling increases. And loose coupling minimize interdependency that is better for any system development.
  - If two modules interchange large amount of data, then they are highly interdependent or we can say they are highly coupled.
  - High coupling between modules makes the system difficult to understand and increase the development efforts. So low (OR loose) coupling is the best.



No coupling



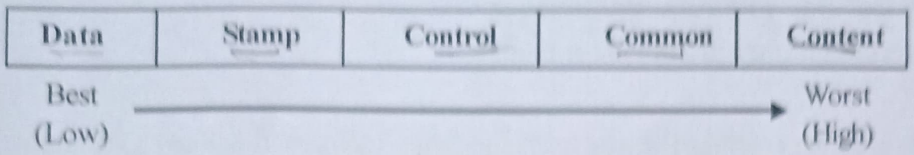
Loose coupling



High coupling

Classification of coupling:

Five different types of coupling can occur between two modules.



Data coupling

- Two modules are data coupled, if they communicate using an elementary data item that is passed as a parameter between these two.
- For example → an int, a char, a float etc.
- It is lowest coupling and best for the software development.

Stamp coupling

- Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling

- Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another module.
- An example of control coupling is a flag set in one module and tested in another module.

Common coupling

- Two modules are common coupled, if they share data through some global data items. It means two or more modules are communicating using common data.

Content coupling

- It is the highest coupling and creates more problems in software development.
- Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.
- It is also known as 'pathological coupling'.

- **Functional independence:**
  - A module having high cohesion and low coupling is said to be functionally independent of other modules.
  - So, that a cohesive module performs a single task or function.
  - A functionally independent module has minimal interaction with other modules.
  - For good software design neat decomposition is highly needed, and the primary characteristic of neat decomposition is **'high cohesion and low coupling'.**

> **Intra dependency (Cohesion) between modules should be high and inter dependency (Coupling) should be low.**

Need of functional independence:

- Functional independence is a good key to any software design process due to following reasons :

1. **Error isolation:**
   - It reduces error propagation.

- The reason behind this is → if a module is functionally independent, its degree of interaction with the other modules is less.
- So, the error of one module can't affect another module.

2. **Scope of reuse:**

- Reuse of a module becomes possible. Because each module does some well-defined and precise function, and the interaction of the module with the other modules is simple and minimal.
- Therefore, a cohesive module can be easily taken out and reused in a different program.

3. **Understandability:**

- Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

Difference between functional and non-functional requirements:

Functional requirements	Non-functional requirements
These describe what the system should do.	These describe how the system should behave.
These describe features, functionality and usage of the system.	They describe various quality factors, attributes which affect the system's functionality.
Describe the actions with which the work is concerned.	Describe the experience of the user while doing the work.
Characterized by verbs.	Characterized by adjectives.
Ex: business requirements, SRS etc.	Ex: portability, quality, reliability, robustness, efficiency etc.

Difference between Cohesion & Coupling:

Cohesion	Coupling
Cohesion is the indication of the relationship within module.	Coupling is the indication of the relationships between modules.
Cohesion shows the module's relative functional strength.	Coupling shows the relative interdependence among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the single thing.	Coupling is a degree to which a component / module is connected to the other modules.
While designing you should go for high cohesion. i.e. a cohesive component/ module focus on a single task with little interaction with other modules of the system.	While designing you should go for low coupling i.e., dependency between modules should be less.
Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.	Making private fields, private methods and nonpublic classes provides loose coupling.
Cohesion is Intra - Module Concept.	Coupling is Inter -Module Concept.

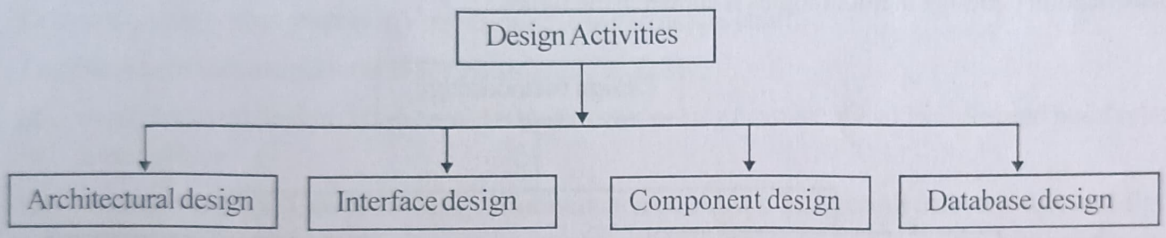
Unit-3
Software Design with UML

❖ DESIGN PROCESS

- The design process is a sequence of steps to describe all aspects of the software.
- Design process specifies how aspect of the system (means how the system will be implemented and how it will work).
- The purpose of the design process is → to plan a solution of problems specified in SRS.
- Design process includes user interface design, input output design, data design, process and program design and technical specification etc.
- It transforming the customer requirements (described in SRS) into appropriate form that is suitable to implement using any of programming languages.
- Output of design process is →design documents.

Classification of design activities:

- Design activities are depending on the type of the software being developed.
- Design activities can be classified like:



Design Activities

1. **Architectural design:**
   - Where you can identify the overall structure of the system, sub-systems, modules and their relationships.
   - It defines the framework of the computer based system.
   - It can be derived from DFD (data flow diagram).
2. **Interface design:**
   - Where you can define the interface between system components.
   - It describes how system communicates with itself and with the user also.
   - It can be derived from DFD and State transition diagram.
3. **Component design:**
   - That defines each system component and show how they operate.
   - It can be derived from State transition diagram.

4. **Database design:**
   - Where you can define the system data structure and show how they are represented in a database.
   - Existing database can be reused or a new database to be created.
   - The data objects and their relationships are defined in ERD (entity relationship diagram) and detailed content described in data dictionary (DD).
   - It can be derived from ERD and DD.

Classification of design methodologies:

- Design methodologies are followed in software development from beginning up to the completion of the product.
- Design methodologies use to provide guidelines for the design activity.
- The nature of the design methodologies are dependent on the following factors:
  - → The software development environment.
  - → The type of the system being developed.
  - → User requirements.
  - → Qualification and training of the development team.
  - → Available software and hardware resources.
- There are large numbers of software design methodologies. Different methodologies are used to solve different type of problems.

Classification of design methodologies is shown in the figure.

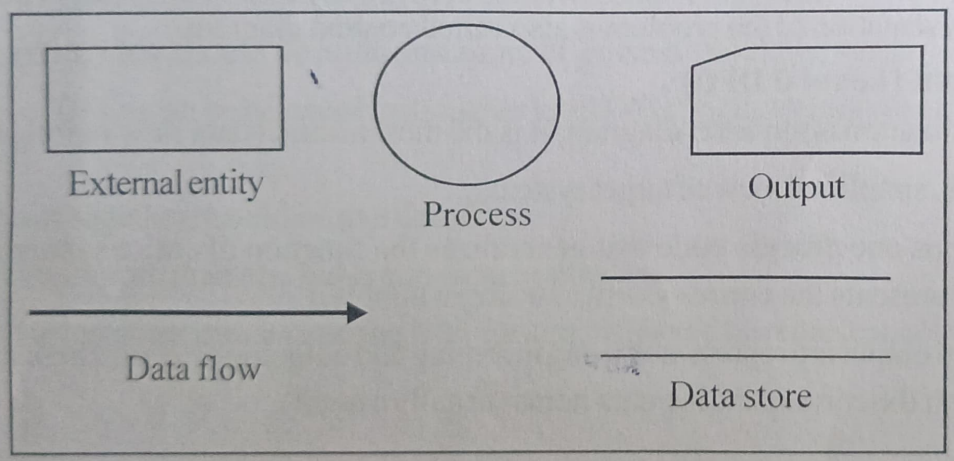
(**Note:** There are many different notations available for cardinality and modality, like Chen's notations, Crow's foot notation etc. So it may be possible that you may see different notations in various books or websites.)

DATA FLOW DIAGRAMS

- DFD (Data Flow Diagram) is also known as bubble chart or data flow graph.
- DFDs are very useful in understanding the system and can be effectively used during analysis. It shows flow of data through a system visually.
- The DFD is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions.
- It views a system as a function that transforms the inputs into desired outputs.
- Each function is considered as a process that consumes some input data and produces some output data.
- The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system.
- Functional model can be represented using DFD.

➢ Primitive symbols used in construction of DFD model:

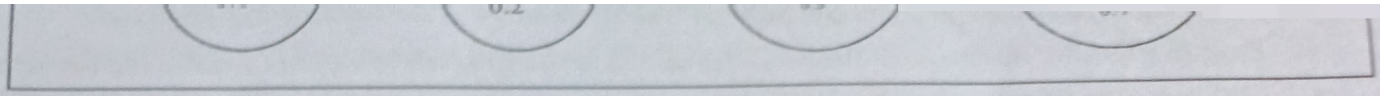
- The DFD model uses a very limited number of primitive symbols.



DFD symbols

1. **Process (function)**
   - Process or function is represented by circle or bubble.
   - Circles are annotated with names of the corresponding functions.
   - A process shows the part of the system that transforms inputs into outputs.
     - The process is named using a single word that describes what the system does functionally. Generally processis named using 'verb'.
2. **External entity**
   - Entity is represented by a rectangle. Entities are external to the system which interacts by inputting data to the system or by consuming data produced by the system.
   - It can also define source (originator) or destination (terminator) of the system.
3. **Data flow**
   - Data flow is represented by an arc or by an arrow.
   - It used to describe the movement of the data.
   - It represents the data flow occurring between two processes, or between an external entity and a process. It passes data from one part of the system to another part.
   - Data flow arrows usually annotated with the corresponding data names. Generally data flow named using 'noun'.
4. **Data store**
   - Data store is represented by two parallel lines.
   - It is generally a logical file or database.
   - It can be either a data structure or a physical file on the disk.
5. **Output**
   - Output is used when a hardcopy is produced.
   - It is graphically represented by a rectangle cut either a side.

➢ **Developing DFD model of the system:**



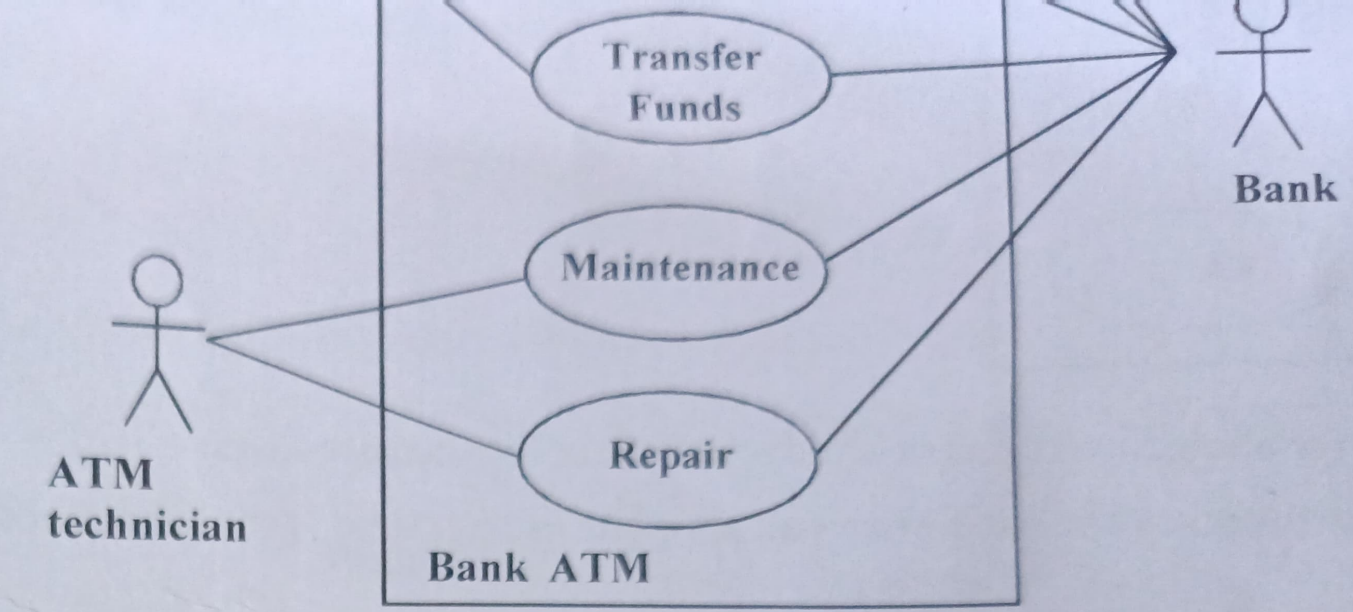
Note: Level1 DFD can be drawn for each high level functional requirement shown in the SRS.

➢ **Advantages of DFD model:**

- DFD is a simple graphical technique which is very simple to understand and easy to use.
- It can be used as a part of system documentation.
- DFD can provide detailed description of the system components.
- It provides clear understanding to the developers about the system boundaries and analysis of the system.
- It explains the logic behind the data flow within system.
- It provides structure analysis of the system.
- Symbols used in DFD model are very less.
- It does not provide any time dependent behavior like we cannot consider at which time we have to do particular process.

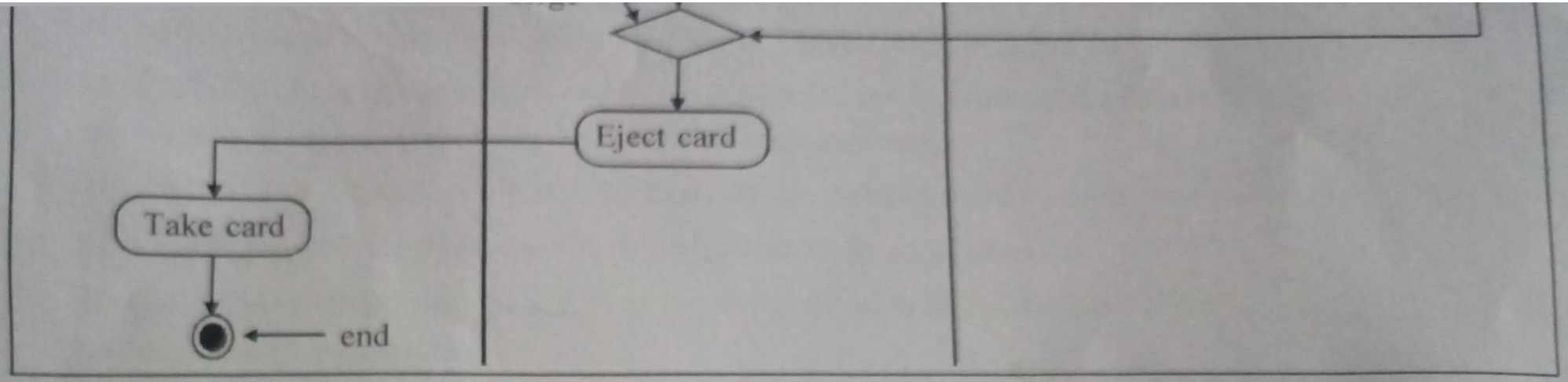
➢ **Disadvantages of DFD model:**

- Control information is not defined by a DFD.
- DFD does not provide any specific guidance as how exactly decompose a given function into its sub functions we have to use subjective judgment to carry out decomposition.
- Sometimes it puts programmers in little confusing state.
- Different DFD models have different symbols, e.g. in Gane and Sarson notations → process is represented rectangle while in Demarco and Yordan notations→it is ellipse, so making confusion at the time of referrin (In some of the notations you can see the process symbol is rectangle while in some other notations, proce symbol is ellipse and that is confusing).
- Physical considerations are left out in DFD.



- **Advantages of use case diagram**
  - It is easy to understand and draw.
  - It is used to capture the functional requirements of the system.
  - It is used as basis for scheduling effort.
  - It is used to verify whether all the requirements are captured.
- **Disadvantages of use case diagram**
  - Use case diagrams are not fully object oriented.
  - It does not provide any guideline when to stop.
- **Applications of use case diagram**
  - Requirement analysis
  - High level design
  - Reverse engineering
  - Forward engineering

Developing an Activity diagram:



Advantages of activity diagram

- Activity diagrams can be very useful to understand complex processing activities.
- Different activities are grouped together based on actor. That is represented by swimlanes.
- It can be useful for analyzing a use case and understating workflow of system.

- Activity Diagrams are good for describing synchronization and concurrency between activities.
- Partitioning can be helpful in investigating responsibilities for interactions and associations between objects andactors.

➢ **Disadvantages of activity diagram**

- The activity diagram does not provide message part. Means do not show any message flow from one activityto another.
- It can't describe how objects collaborate.
- It takes time to implement.
- Complex conditional logics (like Truth table) can't be represented by activity diagram.
- There are lot many symbols compare to other UML diagrams, so sometimes make it confusing to the developer.

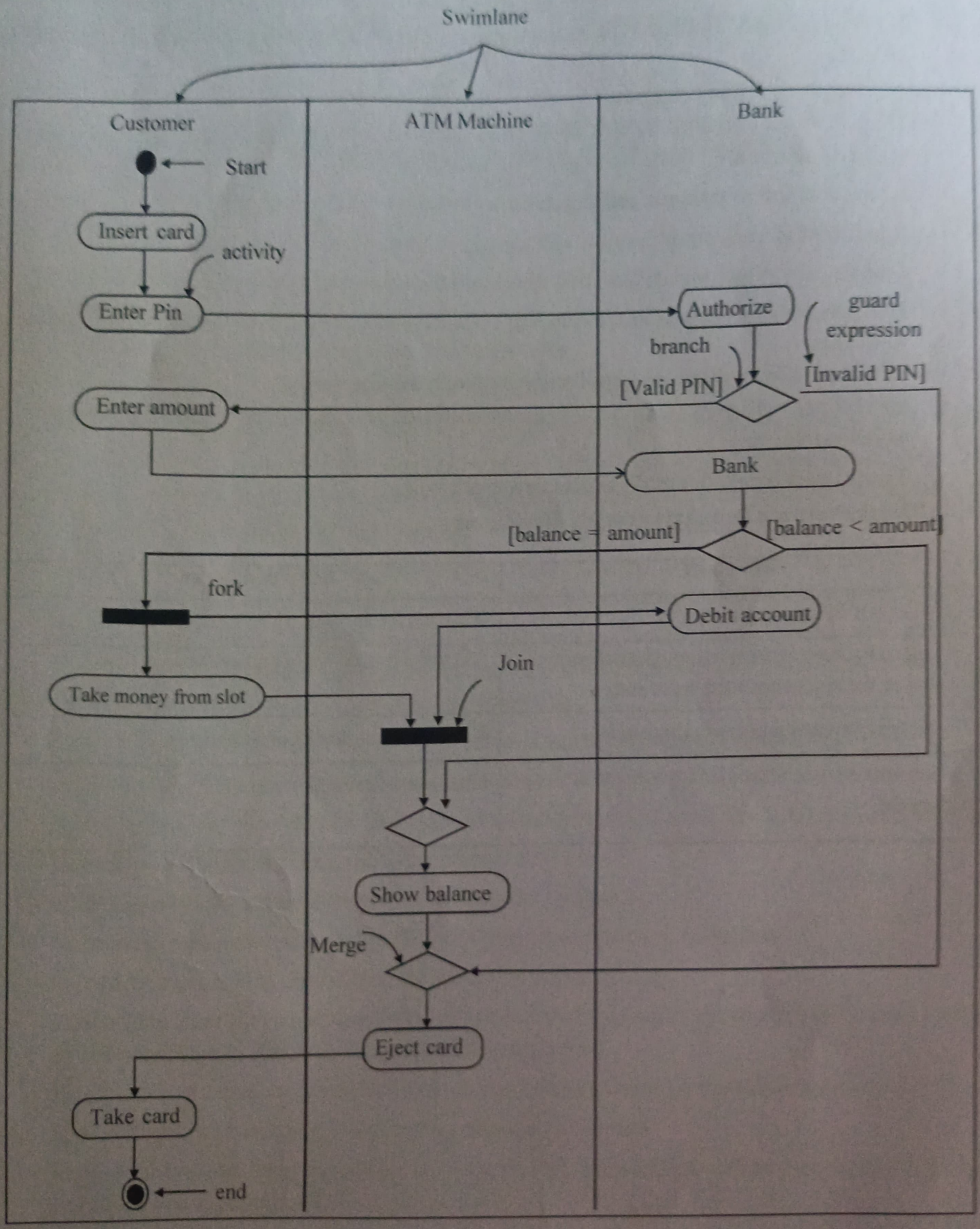
➢ **When to use Activity diagram (Applications of activity diagram)**

- Mainly activity diagram used to describe the parallel behavior of the system. It makes a great tool for workflowmodeling.
- It is also used in multithreaded programming application.

➢ **Difference between flowchart and activity diagram**

Flow chart	Activity diagram
It is limited for sequential access.	It is used for parallel and concurrent processing.
It is used for flow of control through an algorithm, not used for object oriented procedure.	It is usually used for object oriented systems.
Concept of swimlanes is not there in it.	It has the functionality of swimlanes.
It has limited functionalities compare to activity diagram.	It has more functionality.

➢ **Another example showing swimlanes**



➢ **Advantages of activity diagram**

- Activity diagrams can be very useful to understand complex processing activities.
- Different activities are grouped together based on actor. That is represented by swimlanes.
- It can be useful for analyzing a use case and understating workflow of system.

There are several metrics to measure problem size. Each of them has its own advantages and disadvantages.

- We will consider two important metrics to estimate size : *Lines of code (LOC) and Function point (FP).*

4.2.1 Lines of code (LOC) :

- The simplest measure of problem size is Lines Of Code (LOC) or Source Lines Of Code (SLOC).
- It is very popular because of its simplicity.
- LOC is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. Comments line and headers lines are ignored at counting the source code.
- In other term, it is software metric that measures the size of software in terms of lines in the program.
- In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted.
- According to Boehm, every line of source text should be calculated as one LOC.
- By using the estimation of the lowest level modules, project managers arrive at the total size estimation.
- One main advantage of LOC is that it is very simple to understand and directly relates to end product.

However, LOC has several disadvantages.

- LOC is language dependent. LOC focuses on coding activity only, while a good problem size measure should consider the overall complexity of the problem.
- LOC gives only numerical values of problem size, that is depend on coding style. It can't measure the size of specification.
- LOC does not measure with the quality and efficiency of the code.
- LOC metric suffer from accuracy when use of high level languages, code reuse, library subroutine etc.
- LOC doesn't issue logical and structural complexities.
- It is very difficult to accurately estimate the LOC in the final product from the problem specification. Accurate LOC can be computed only after code has been fully developed.
- LOC doesn't work well with non-procedural languages.

4.2.2 Function Point metric (FP) :

LOC can be computed only after code has been fully developed.

- LOC doesn't work well with non-procedural languages.

4.2.2 Function Point metric (FP) :

- Function point metric was first proposed by Albrecht in 1979 and internationally approve modified model in 1983. This metric overcomes many of the shortcomings of the LOC metric.
- Function point metrics, measure functionality from the users' point of view, that is, on the basis of what the user requests and receives in return.

- One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification.
- In LOC metric, the project size can be accurately determined only after the product has fully been developed. While in case of FP, the size of a software product is directly dependent on the number of different functions or features it supports.
- Software supporting many features or functions should have larger FP in size.
- Each function when invoked reads some input data and transforms it to the corresponding output data.
- By using the number of input and output data values, function point metric computes the size of a software product.
- FP considers five different characteristics of the product to calculate the size. The function point (FP) of given software is the weighted sum of these five items, and it will give unadjusted function point (UFP).

  UFP = (Number of inputs)*4 +
  (Number of outputs)*5 +
  (Number of inquiries)*4 +
  (Number of files)*10 +
  (Number of interfaces)*10

- The meaning of each parameter is as follow :

(1) Number of inputs

- Each data item input by the user is counted.
- Group of related inputs are considered as a single input.
- For example, while entering the data concerning student to student information system software; the data items name, age, gender, address, phone number, etc. are together considered as a single input.

(2) Number of outputs

- It refer to reports printed, screen outputs, error messages produced, etc.
- The set of related data items is counted as one input.

(3) Number of inquiries

- It is the number of distinct interactive queries which can be made by the users.
- These should be user commands for specific actions.

(4) Number of files

- Each logical file is counted. A logical file means groups of logically related data is counted as a file.
- The files can be data structures or physical files.

(5) Number of interfaces

- The interfaces used to exchange information with other systems.
- Examples of such interfaces are data files on tapes, disks, communication links with other systems etc.
- Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next.
- TCF refines the UFP by considering 14 factors which assigns 0 (no influence) to 6 (strong influence). These numbers are summed and yielding DI (Degree of Influence).
- Now TCF is computed as

$$\mathbf{TCF = (0.65 + 0.01 * DI)}$$

- Determine the critical path.
- Among all of the above activities: identifying tasks and breaking down them into small activities done through work breakdown structure (WBS). PERT and CPM used to sequence the activity and estimating time and project schedule is developed through Microsoft project tool.

4.4.1 Work breakdown structure (WBS) :

- Work Breakdown Structure (WBS) is used to decompose a given task set recursively into small activities.
- The WBS is a uniform, consistent, and logical method for dividing the project into small, manageable components for purposes of planning, estimating, and monitoring.
- An effective WBS encourages a systematic planning process, reduces the omission of key project elements, and simplifies the project by dividing it into manageable units.
- A WBS will provide a roadmap for planning, monitoring, and managing all factors of the project like ' resource allocation, scheduling, budgeting, productivity, performance etc.
- WBS can be shown graphically in a hierarchical tree structure and developed top to bottom manner.
- The root of the tree is labelled by the problem name.
- Each node of the tree is broken down into smaller activities that are made the children of the node.
- Each activity is recursively decomposed into smaller sub-activities until at the leaf level.
- WBS can be done by the decision of the project manager.

Types of WBS :

- There are three types of WBS as follows :

(i) **Process WBS :** it decomposes large processes into smaller ones. Each process finally decomposed in the task.

(ii) **Product WBS :** it decomposes large entity into smaller components. It is used by system engineers.

(iii) **Hybrid WBS :** in includes both process and product elements into single WBS.

- There are two methods of WBS presentation :

4.4.5 Project Monitoring and Control (PMC) :

- Planning is one of the most important project activities. And without proper planning, project monitoring and control is not possible.
- **Monitoring** - collecting, recording, and reporting information concerning project performance that project manager and others wish to know.
- **Controlling** - it uses data from monitoring activity to bring actual performance from planned performance.
- Project monitoring and planning (PMC) activities take place in parallel with project execution, so the implementation should be corrective at appropriate level.
- The main purpose is to - *to ensure quality of final product.*

➡ **Why there is a need of PMC**

- Simply because we know that things don't always go according to plan.
- To detect and react appropriately to deviations and changes to plans.

➡ **What do we monitor and control**

- We need to monitor → men, machine, money, material, space, time, tasks, quality, performance and we need to control' time, cost and performance.
- PERT chart is mainly used for project monitoring and control.

➡ **Monitoring and controlling project work includes following activities :**

- Comparing the work that is occurring to the project management plan.
- Assessing work performance information to determine if any corrective or preventative actions are necessary.
- Analyzing, tracking, monitoring, and reporting on project risks.
- Providing status reports, accomplishments, and issue reports.
- Monitoring the implementation of approved changes.
- Do scope verification process, schedule control, quality control and cost control.
- Making sure that approved defect repairs have been made.
- Take corrective actions when needed.
- Monitoring and controlling outputs related to risk management include updating the risk register.
- Monitoring and controlling outputs related to communications management include performance reports, forecasts, and resolved issues.
- Techniques used for project monitoring and control activity are: Earned Value Analysis (EVA) and Critical ratio.
- A way of measuring overall performance (not individual task) is using an aggregate performance measure - Earned Value.
- **Earned Value Analysis** (EVA) is an industry standard method of measuring project's progress at any given point of time.
- Earned value of work performed (value completed) for those tasks in progress, found by multiplying the estimated percent physical completion of work for each task by the planned cost for those tasks. The result is amount that should be spent on the task so far. This can be compared with actual amount spent.
- Especially for large projects, it may be worthwhile calculating a set of critical ratios for all project activities

- The critical ratio is :

$$\frac{\text{Actual progress}}{\text{Scheduled progree}} * \frac{\text{Budgeted cost}}{\text{Actual cost}}$$

- If ratio is 1 everything is probably on target.
- The further away from 1 the ratio is, the more we may need to investigate.
- Continuous monitoring gives the project management team insight the health of the project, and identifies any areas that can require special attention.

4.5 RISK MANAGEMENT

- Tomorrow's problem is today's risk.
- Software risk is a problem that could cause some loss or threaten the success of software project, but which hasn't happened yet.
- This risk may affect negatively to the cost, schedule, technical success or quality of the project.
- Risk management is the process of identifying, addressing and eliminating the problems of risks before they can damage the project.

✢ Objectives of the risk management

- Identify potential problems and deal with them when they are easier to handle before they become critical.
- Focus on the project's objectives and consciously look for things that may affect project quality.
- Allow early identification of risks and provide management decisions to the solutions.
- Increase the chance of project success.

Risk management activities :

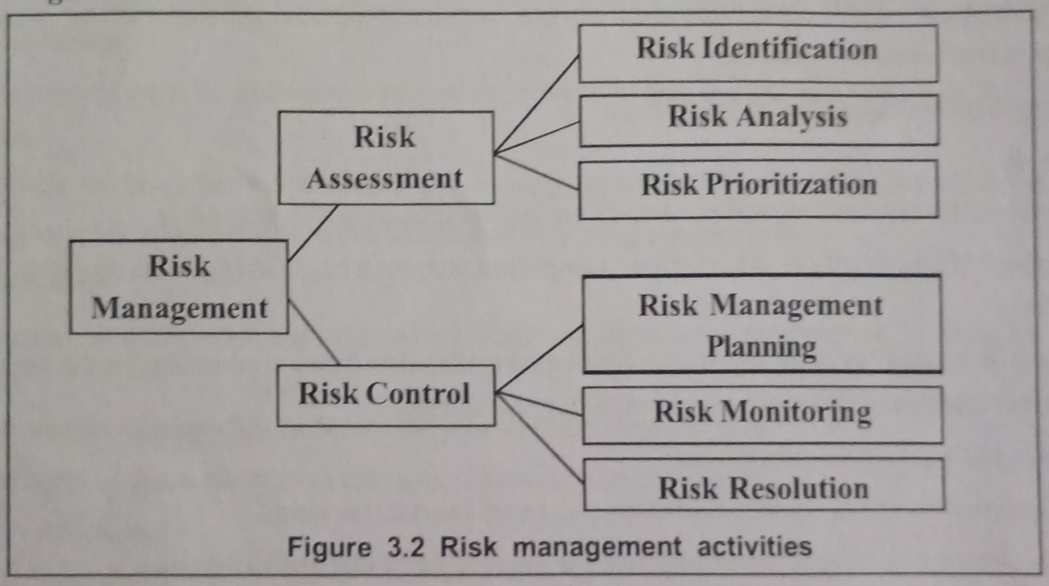


Figure 3.2 Risk management activities

4.5.1 Risk assessment :

- It is the process of examining a project and identifying areas of potential risk.
- It includes the following activities :
  - (i) Risk identification
  - (ii) Risk analysis
  - (iii) Risk prioritization

Unit-5

Software Coding & Testing

5.1 CODE REVIEW (CODING CONCEPTS)

- In general, coding means set of guidelines for a specific programming language. Coding is what makes it possible for us to crate computer software, applications, websites etc.
- In the simple term, coding is → telling a computer what you want it to do, which involves typing in step-by-step commands for the computer to follow.
- Objective of coding → transform the design document into high level language code and unit test this code.
- Input to the coding phase → design document (module structure and data structure algorithms).
- Good software development organizations normally require their programmers to have some well-defined and standard style of coding that is coding standards.
- Coding standards, sometimes referred to as programming styles or coding convention.
- The purpose of coding standards :
  - ➠ It gives a uniform appearance to the codes written by different engineers.
  - ➠ It enhances code understanding.
  - ➠ It encourages good programming practices.
- Coding standards list several rules to be followed and coding guidelines provide general suggestions.
- Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

.1 Coding standards :

standards are language-specific programming rules that greatly reduce the probability of

(i) Code walkthrough

- Code walk through is an informal code analysis technique proposed by Fagan.
- This technique can be used throughout the software lifecycle to assess and improve the quality of the software products.
- A Code Walkthrough is an informal meeting where the programmer leads the review team through his/her code and the reviewers try to identify faults.
- In the process of code walkthrough, after a module has been coded, successfully compiled and all syntax errors eliminated, a few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates (અનુકરણ) execution of the code by hand.
- Members note down their findings and discuss them in the meeting.
- Several guidelines are produced in the meetings and accepted as examples.
- Some of the prerequisite guidelines are the following.
  - → The team performing code walk through should not be either too big or too small. Ideally, it should consist of between three to seven members.
  - → Discussion should focus on discovery of errors and not on how to fix the discovered errors.

✢ Advantages of code walkthrough

- It is useful for the people if they are not from the software discipline; they are not used to or cannot easily understand software development process.
- It improves project team communication and morale of team members.
- It's an excellent educational medium for new team members.
- If it is done right, it can save time and improve quality over the project lifecycle.

✢ Limitations of code walkthrough

- The main disadvantage is that it takes more time.
- As this technique is informal, the documentation of this process is not done.

(ii) Code inspection

Code inspection is a formal, efficient and economical method of finding faults in design and code proposed by

✢ **Limitations of code walkthrough**

- The main disadvantage is that it takes more time.
- As this technique is informal, the documentation of this process is not done.

(ii) Code inspection

- Code inspection is a formal, efficient and economical method of finding faults in design and code proposed by Fagan [1976].
- The goal of code inspection is → to identify and remove bugs before testing the code and to discover the algorithmic and logical errors.
- During code inspection, the code is examined in the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk through.
- Coding standards are also checked during code inspection.
- Good software companies list some common types of errors and these are discussed during code inspection to look out for possible errors.
- An inspection team consists of four persons who play the role of moderator, reader, recorder and author.
  - **Moderator :** he is a technically knowledgeable person. He leads the inspection process.
  - **Reader :** the reader takes the team through the code.

- **Recorder :** he notes each error on a standard form.
- **Authors :** he understands the errors found and try to solve unclear areas.

✢ **Following is a list of some general programming errors which can be checked during code inspection**

- Jumps into loop (in case of nested loop)
- Non terminating loops.
- Array includes out of bound.
- Improper storage allocation.
- Use of uninitialized variables.
- Mismatch between actual and formal parameters.
- Use of incorrect logical operations.
- Control flows and computational expressions.

✢ **Advantages of code inspection**

- List all potential design flows. That makes software code maintainable and less costly.
- A detailed error feedback is provided to individual parameters.
- It makes easier to change in the code.
- As this technique is formal, proper documentation of this method is done.

- *Code walkthrough is informal technique lead by an author while code inspection is formal technique lead by moderator.*
- *Code walkthrough and code inspection both are static testing techniques.*

5.2 SOFTWARE DOCUMENTATION

- Software documentation is an important aspect of both software projects and software engineering in general.
- It could be paper document (manuals, broacher etc) or an electronic document (text written inside code). Paper documents are called external documents and electronic documents are called internal documents.
- Software documentation can be defined as, an artefact (માનવ સર્જિત) whose purpose is to communicate information about the software system to which it belongs.
- Software documents work as an information repository for software engineers.
- It could be the part of the software (internal) and it can be available offline (external).
- When various kinds of software products are developed then not only the executable files and the source code are developed but also various kinds of documents such as users' manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process.
- Two main requirements for good documentation are → it is complete and up-to-date.

✢ **Advantages of good software documentation**

- Good documents enhance understandability and maintainability.
- Reduce effort and time for maintenance.
- Provides helps to users for effectively use of system.
- Good software documents helps in handling manpower turn over.
- Good software documents help the manager effectively track the progress of the system.
- Different types of software documents can be classified into two parts
  1. Internal documentation
  2. External documentation

1. Internal documentation

- It's a part of the source code itself.
- Internal documents are included in the syntax of the programming languages.
- Internal documentation is provided through appropriate module headers and comments embedded in the source code.
- The main objective of the internal documentation is to provide help to the user and the programmer to get a quick understanding of the program and the problem to modify the program as early as possible.
- It is also provided through the useful variable names, module and function headers, code indentation, code structuring, use of enumerated types and constant identifiers, use of user-defined data types, etc.
- Internal documents not only explain the programs, or program statements, but also help programmers to know before any action is taken for modification.
- Good internal documentation appropriately formulating by coding standards and coding guidelines.

2. External documentation

- These documents take place outside of the source code. It is provided through various types of supporting documents such as users' manual, software requirements specification document, design document, test documents, etc.
- External documents which focuses on general description of the software code and is not concerned with its detail.
- It includes information such as algorithms used in software code, dependencies of code in libraries, format of the output produced etc.
- External documents have two types : one for the users and one for those who wants to understand how the program works.
- It makes the user aware of the errors that occur while running the software code.
- All external documents are produced in orderly manner.

- **Test suite :** This is the set of all test cases with which a given software product is to be tested.
- Testing is four stage process :

  Unit testing → subsystem testing → system testing → acceptance testing.
- Initially all the modules are tested individually by their programmers, then interactions between these modules are tested (integration testing), then whole system as a single component is tested and finally the system is tested against users' data for final approval from the users.

+ **Difference between verification and validation**

Verification	Validation
→ Verification is the process of confirming that software meets its specification.	→ Validation is the process of confirming that software meets customers' requirements.
→ Verification is the process of determining whether the output of one phase of software development confirms to that of its previous phase.	→ Validation is the process of determining whether a fully developed system confirms to its requirements specification.
→ Verification is concerned with phase containment of errors	→ Validation is concerned with final product be error free.
→ Verification → "are we doing right ?"	→ And Validation → "have we done right ?"
→ Verification comes before validation.	→ Validation comes before verification.
→ It is static testing.	→ It is dynamic testing.
→ Cost of verification is less.	→ Cost of validation is more.
→ Verification does not include the execution of code.	→ Validation includes the execution of code.

Both strategies are used to finds defects in software project, but in different way, Verification is used to identify the errors in requirement specifications & Validation is used to find the defects in the implemented software application

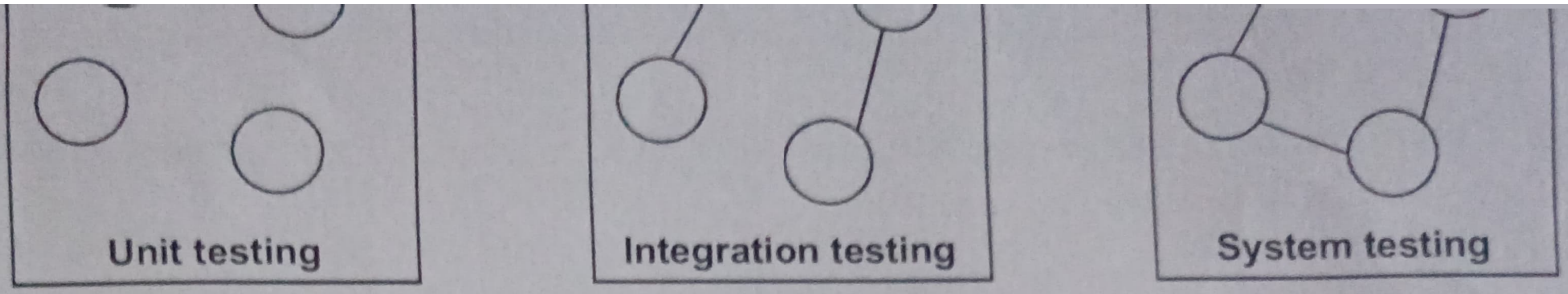
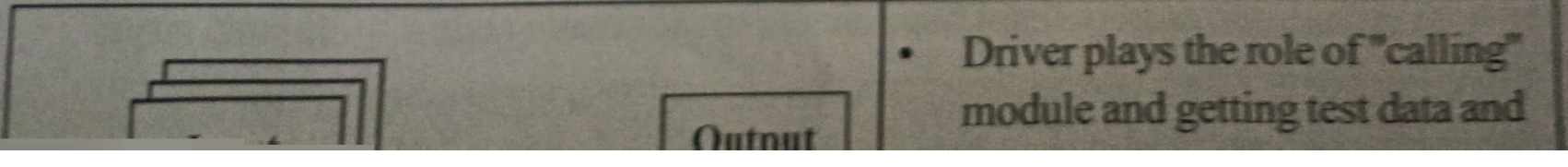


Figure 5.1 Level of testing

5.3.1 Unit testing :

- Unit testing is the first level of testing.
- It is the process of taking a program module and running it in isolation form from the rest of the product.
- Unit testing is undertaken after a module has been coded and successfully reviewed.
- Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation.

- This testing is a white box testing methodology.
- Purpose of unit testing is to validate that each unit of the software code performs as expected.
- Unit testing is done during development phase.
- A complete environment is needed to execute the unit testing on the module.
- Following steps are needed in order to test the module
  - o The procedures belonging to other driver module which contains non-local data structure.
  - o A procedure belonging to the module which is working as stub module.
  - o A procedure belonging to the module which is under test.
- The calling module and called module should be unit tested.
- An issue with the unit testing is that unit testing is done as a part, not the whole system. But in execution, one module may use other modules that have not been developed yet. For that dummy modules are needed.
- Due to this, unit testing often require driver and/or stub modules. The role of driver and stub modules is described in below figure.



✣ **Advantages of unit testing :**

- Improve the quality of the code.
- Finds the software bugs early.
- One of the main benefits of unit testing is that it makes the coding process more agile. (ઝડપી, સરળ અને સુવ્યવસ્થિત)
- It simplifies integration testing.

5.3.2 Black box testing :

- This method is also called behavioural testing or functional testing.
- It is a testing method where test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software.

- Functionality of the black box is understood completely in terms of its inputs and output.

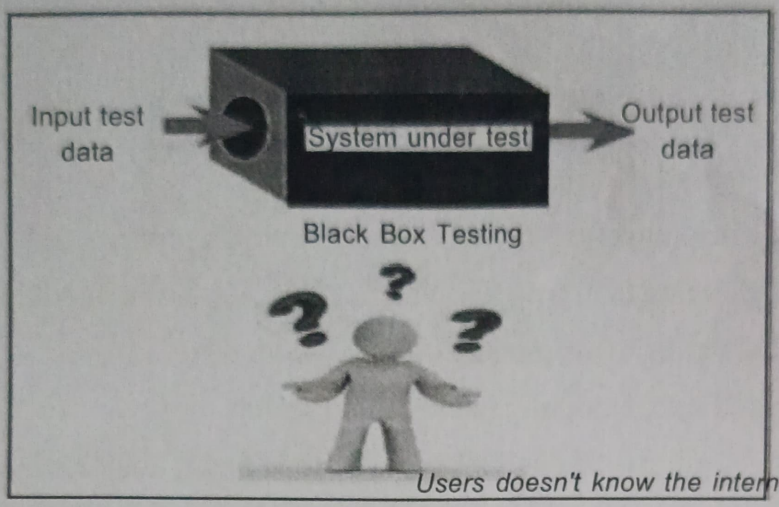
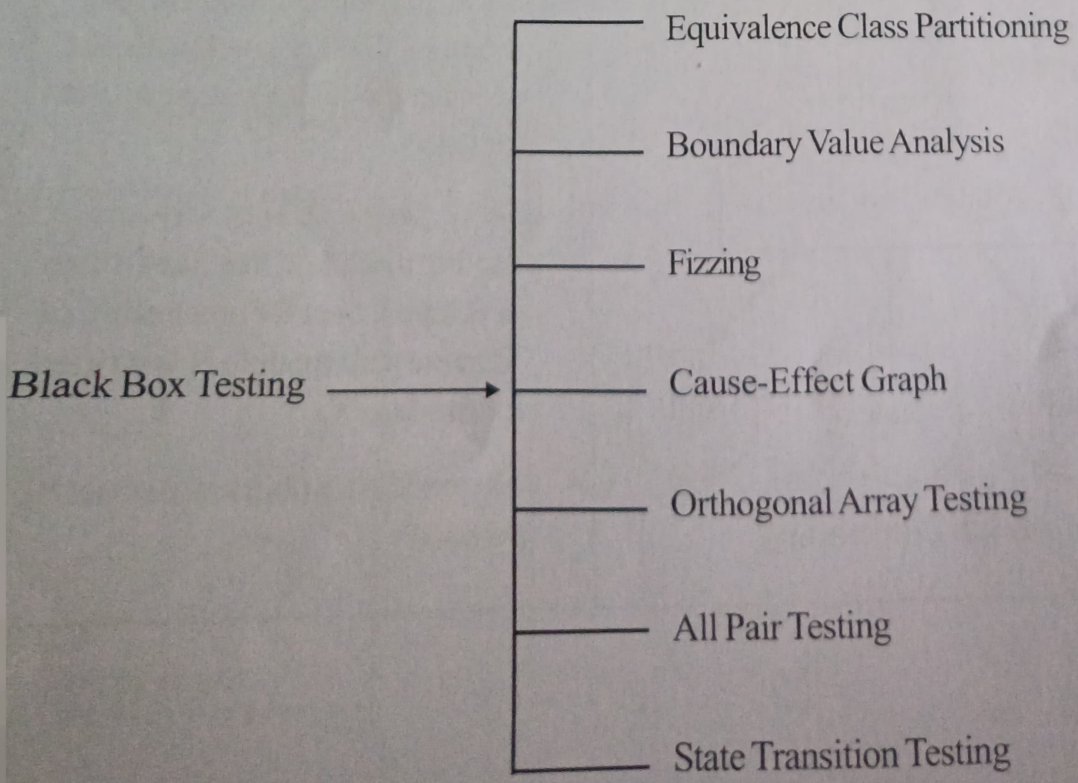


Figure 5.3 Black box testing

- Black box testing treats the software as a "Black Box" - without any knowledge of internal working and it only examines the fundamental aspects of the system. While performing black box test, a tester must know the system architecture and will not have access to the source code.
- The example of black box testing is search engine. You enter text that you want to search in the search bar, and results are returned to you. You don't know the specific process or algorithm of searching.
- There are number of techniques that can be used to design test cases in black box testing method, which are listed below.



(100,200), (101,200), (299,200), (300,200), (301,200).

✢ **Advantages of black box testing**

- It is Efficient for large code segment.
- Tester doesn't need to know the internal structure of the system.
- Tester perception is very simple.
- Programmer and tester are independent of each other.
- Quicker test case development.

✢ **Disadvantages of black box testing**

- It is inefficient testing.
- Without clear specification test cases are difficult to design.
- Only a selected number of test scenarios are actually performed. As a result, there is only limited coverage.

"Black box testing is also known as close box testing and opaque testing."

5.3.3 White box testing :

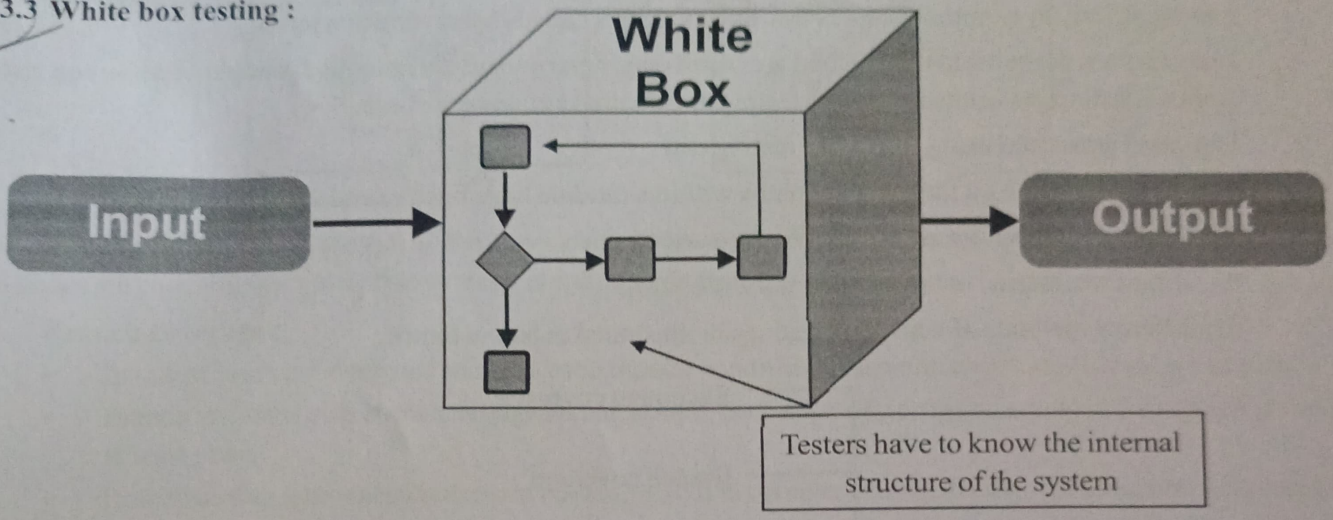
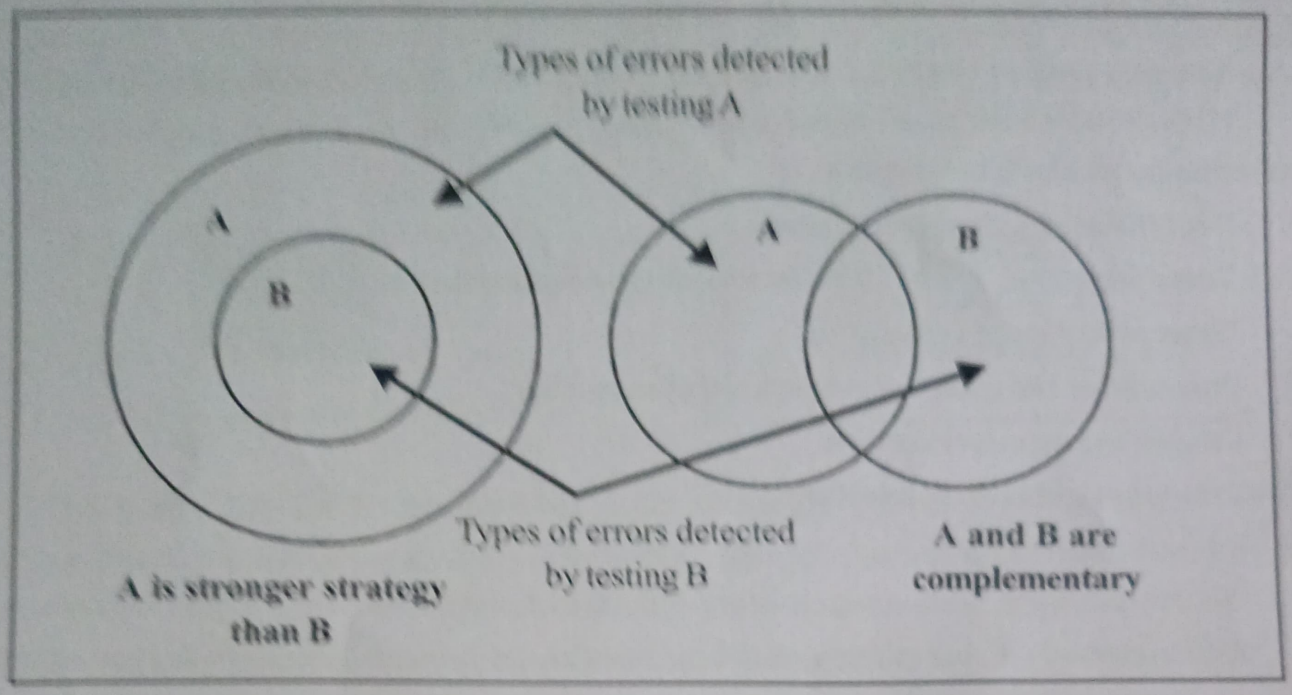
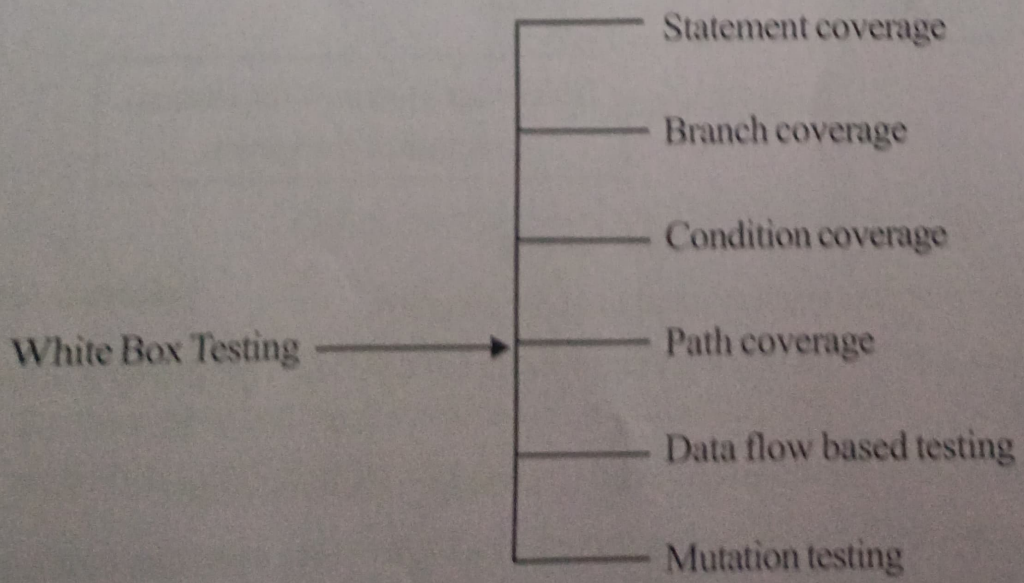


Figure 5.5 White box testing

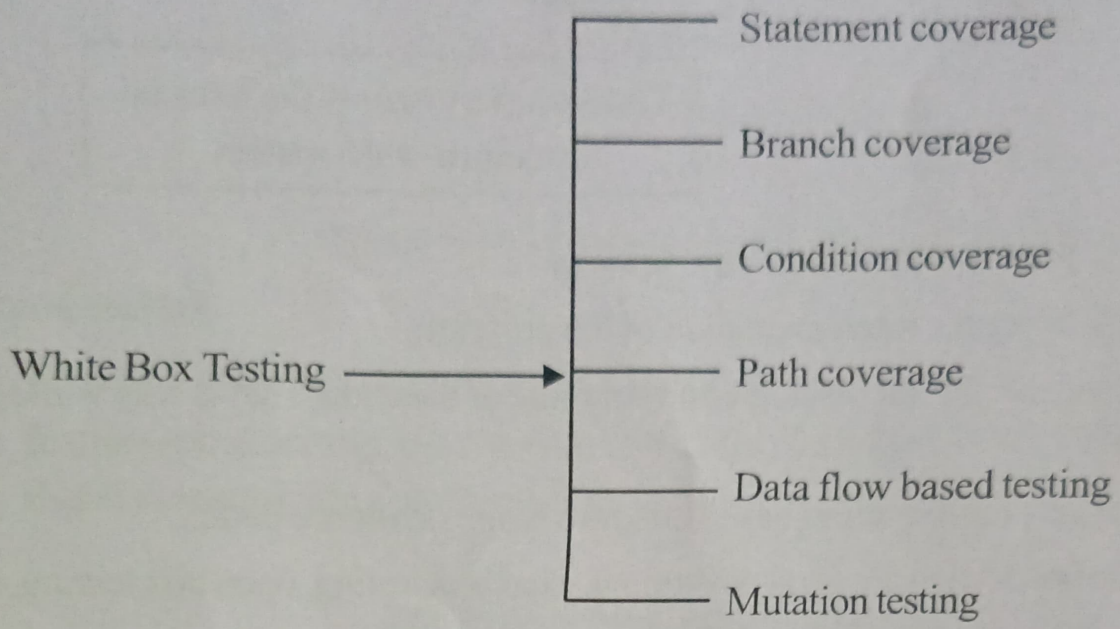
- This method is concerned with testing the implementation of the program.
- The aim of this testing is to investigate the internal logic and structure of the code. That is why white box testing is also called structural testing.
- In white box testing it is necessary for a tester to have full knowledge of source code.
- Some of the synonyms of white box testing are glass box testing, clear box testing, open box testing, transparent box testing, structural testing, logic driven testing and design based testing.
- There are six basic types of testing : unit, integration, function/system, acceptance, regression, and beta. White-box testing is used for three of these six types : Unit, integration and regression testing.
- There are many white box strategies available, in which one is stronger than another. When two testing strategies detect errors that are different, then they are called complementary.
- The concepts of stronger and complementary testing are schematically illustrated in given figure.



- If we test program while it is running, it is called dynamic white box testing, and if we test the program without running it, only by examining and reviewing it then it is called static white box testing.
- Test cases are designed in this method are based on program structure or logic. Example of white box testing is circuit testing. As in electric circuit testing, the internal structure is checked.
- Test cases generated using white box testing can :
  - Guarantee that all independent paths within a module have been exercised at least once.
  - Exercise all decisions whether they are true or false.
  - Exercise internal data structure of the program.
- The different methods of white box testing are illustrated in below figure.



...ods of white box testing are illustrated in below figure.



+ **Statement coverage :**

- The statement coverage is also known as line coverage or segment coverage.
- The principal idea behind the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement.
- It aims to design test cases so that every statement in a program is executed at least once.

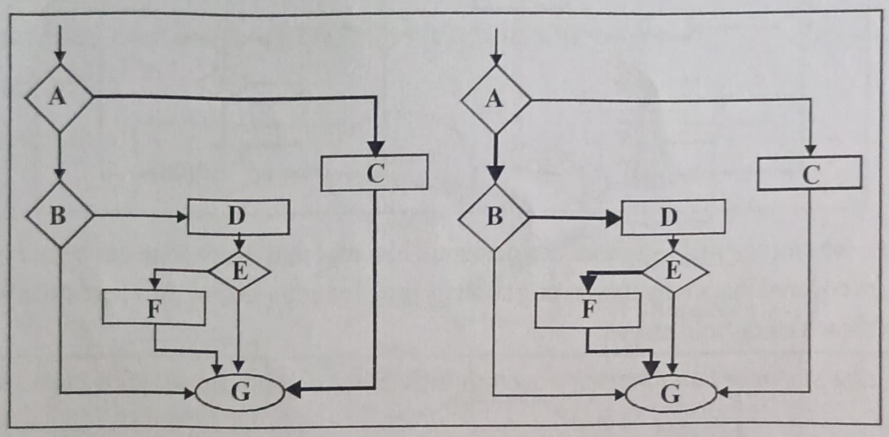
- Statement coverage is also known as node testing.
- However, executing some statement once and observing that it behaves properly for that input value is no guarantee that it will behave correctly for all other input values.

Example :

```
int x, y;
if x > y
        printf("x is greater");
else
        printf("y is greater");
```

For this code test case should be :

{(5,4), (4,5)}



In above figure, we can see that all the nodes (A to G) have been tested by designing test cases in two different ways. All nodes are covered, that's why this strategy is called node testing.

✢ **Branch coverage :**

- In it, test cases are designed to make each branch condition to assume true and false values in turn.
- Branch testing is also known as edge testing as in it, each edge of a program's control flow graph is traversed at least once.
- Branch testing guarantees statement coverage, so it is stronger testing strategy than statement coverage.
- We can take above example for branch testing as well.

Example :

```
int compute_gcd(x, y)
int x, y;
{
1  while (x! = y){
2        if (x>y) then
3              x= x - y;
4        else y= y - x;
5  }
6  return x;
   }
```

For this code test case should be:

{(x=3,y=3), (x=4,y=3), (x=3,y=4)}

all mutants.

- Major disadvantage of mutation testing is → it is very expensive to compute as large number of generated.
- It is not suitable for manual testing.

Advantages of white box testing methods

- Reveal (બતાવવું, ઉજાગર કરવું) hidden errors in the code.
- White box testing is very thorough as the entire code and structures are tested.

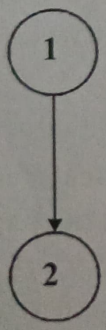
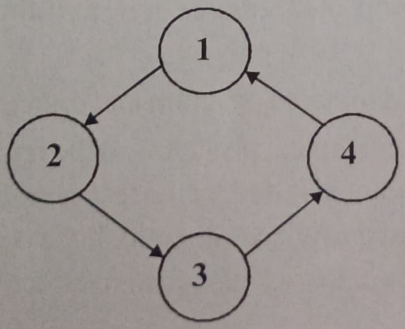
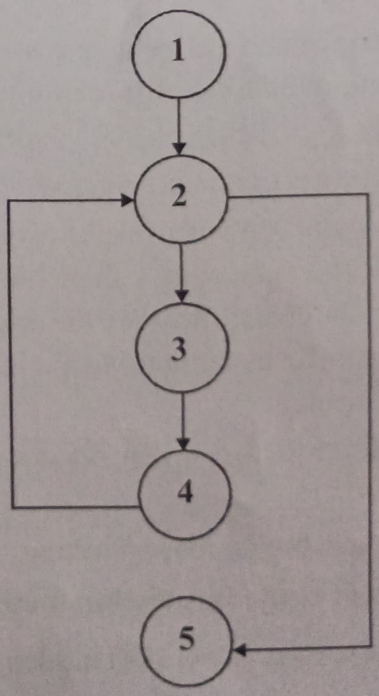
- Testing can start early in SDLC even if GUI is not available.
- Out product will be qualitative if white box testing is successful.

Disadvantages of white box testing methods

- White box testing can be quite complex and expensive.
- It is time consuming as it takes more time to test fully.
- It requires professional resources.
- In-depth knowledge about the programming language is necessary to perform white box testing.

5.3.4 Control Flow Graph (CFG) :

- A control flow graph describes the sequence in which the different instructions of a program get executed. In other words, a control flow graph describes how the control flows through the program.
- In order to draw the control flow graph of a program, all the statements of a program must be numbered first. The different numbered statements serve as nodes of the control flow graph.
- An edge from one node to another node exists if the execution of the statement representing the first node can result in the transfer of control to the other node.
- The CFG for any program can be easily drawn by knowing how to represent the sequence, selection, and iteration type of statements in the CFG.
- Below figure summarizes how the CFG for these three types of statements can be drawn.

Sequence	Selection	Iteration
1. a=5 2. b=a*5	1. if (a>b) 2. print(a is greater) 3. else print(b is greater) 4. print(a+b)	1. a=5 2. while(a>0) { 3. print(a) 4. a=a-1 } 5. b=a;
		

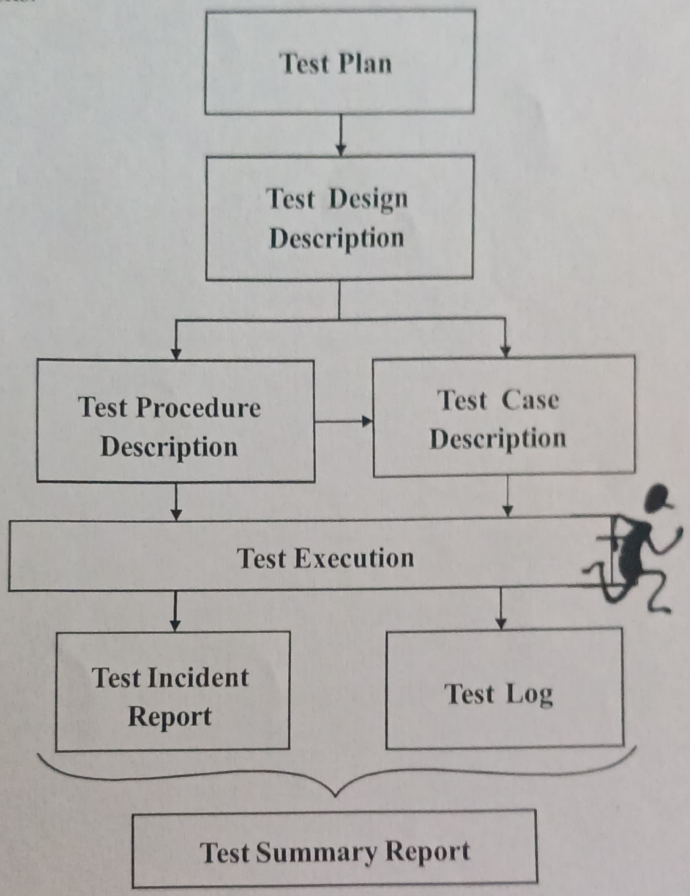
and if statement). So the cyclomatic complexity is : 2 + 1 = 3.

✢ **Difference between Black box testing and White box testing :**

Black box testing	White box testing
• Synonyms of black box testing are functional testing, close box testing, data driven testing and opaque testing.	• Synonyms of white box testing are structural testing, glass box testing, clear box testing, open box testing, logic driven testing.
• No need to know internal structure of the system.	• Internal structure of the system must be known.
• It is concerned with results.	• It is concerned with details and internal workings of the system.
• Performed by end users and also by testers and developers.	• Normally done by testers and developers.
• Granularity is low.	• Granularity is low.
• It is least exhaustive and time consuming.	• Potentially most exhaustive and time consuming.
• Not suited for algorithm testing.	• It is suited for algorithm testing.
• Example : search engine.	• Example : electrical circuit testing.

5.4 TEST DOCUMENTATION

- The documentation which is generated towards the end of testing or during the testing, it is the test summary reports or test documentation.
- It provides summary of test suits which has been applied to the system.
- It specify how many test suits are successful, how many are unsuccessful and what is the degree of successful and unsuccessful.
- Test documentation should be followed the IEEE standards 829.
- Test documentation should includes : scope, approach, resources and schedule of the testing activity to identify the items being tested, the features to be tested, the testing tasks to be performed, responsible personnel and the associated risks.



- A test design specification/description → to identify the features to be tested and associated tests.
- A test case specification/description → to define test cases identified by test design specifications.
- A test procedure specification → to specify the steps for executing a set of test cases.
- A test item transmittal report → to identify the test items being transmitted for testing.
- A test log' to document the generated results.
- A test incident report → to document occurred events during testing process.
- A test summary report → to summarize the results of the testing activities and to provide evaluation of these results.