



DNYANSAGAR ARTS AND COMMERCE COLLEGE, BALEWADI, PUNE – 45

**Software Engineering (CA303)
SYBBA(CA)-3rd Sem**

**By
Prof Gayatri A Amate**



Unit : I

Introduction to system concepts



Introduction

we are surrounded by systems. There are many systems such as transportation system, education , manufacturing and human economic activity systems.

System: “A system is an orderly grouping of independent components linked together according to a plan to achieve a specific objective or goal”



System

- A system is a set or group of components that interact to accomplish some purpose.
- System interacts with their environment.
- E.g brain , spinal cord, nerves etc.

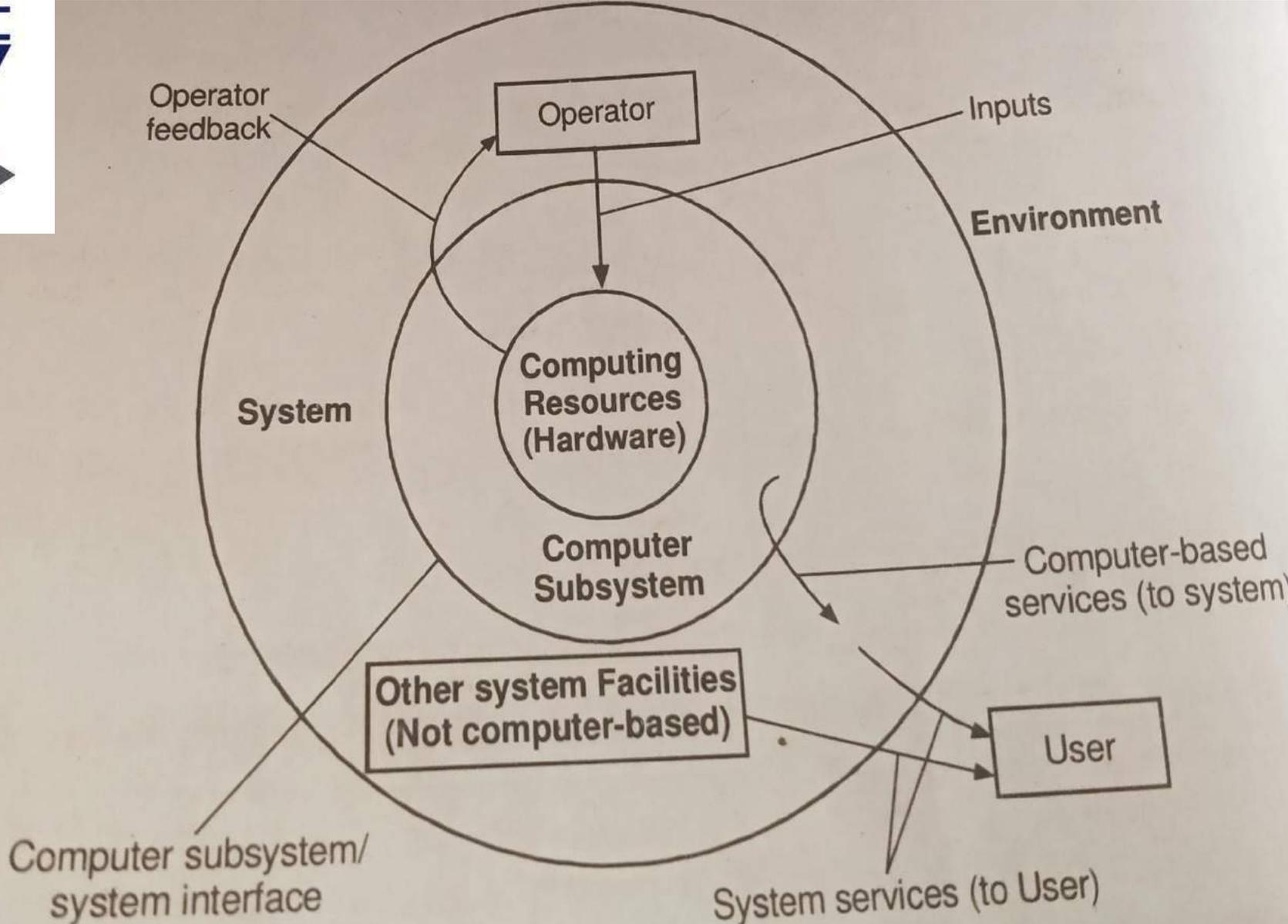


Fig. 1.1: System relationships



System consist of

- Standards: which acceptable performance.
- Measurement: Measuring actual performance.
- Compare: Comparing actual performance.
- Feedback: A method for feedback.



Elements of system

- Input and Output
- Processors
- Control
- Feedback
- Environment
- Boundaries and Interface



I. Input and Output

- A major objective of a system is to produce the output as per the user's requirement.
- The output could be in the form of goods information as services. It must be done with expectations of user.
- Inputs are elements that enter the system for processing.
- Output is outcome of processing.



Processors:

- The processor is the element of a system that involves the actual transformation of Input into output.
- Processor should be designed of such type that it can accept the input in the given form and can give output in desired format.



Control:

- The control elements guide the system. The control element controls the working of the system at all stages.
- It is necessary to control input, process and output, continuously, in order to get desired results.
- Management support is required for screening control supporting the objective proposed change



Feedback

- Feedback measures output against a standard in some form of cybernetic procedure includes communication and control.
- Feedback may be positive or negative. Positive feedback reinforces the performance of system.



5. Environment

- All the things which are outside system are called environment of the system.
- The environment does affect working or progress of the system.



Boundaries and Interface:

- The boundary indicates the extent or limit of the system.
- The boundary divides the things into the system and its environment.
- The things which are inside boundary are part of system otherwise which are outside boundary are its environment.



Interface:

- It means interaction of different system parts with each other as interaction of the system with system outside its boundaries.
- **Boundaries**
- are the limits that identify its components, process and interrelationships when it interfaces with another system.



Characteristics of a system

Following are the some important characteristics of a system:

1. Organization
2. Interaction,
3. Interdependence,
4. Integration
5. Central objective

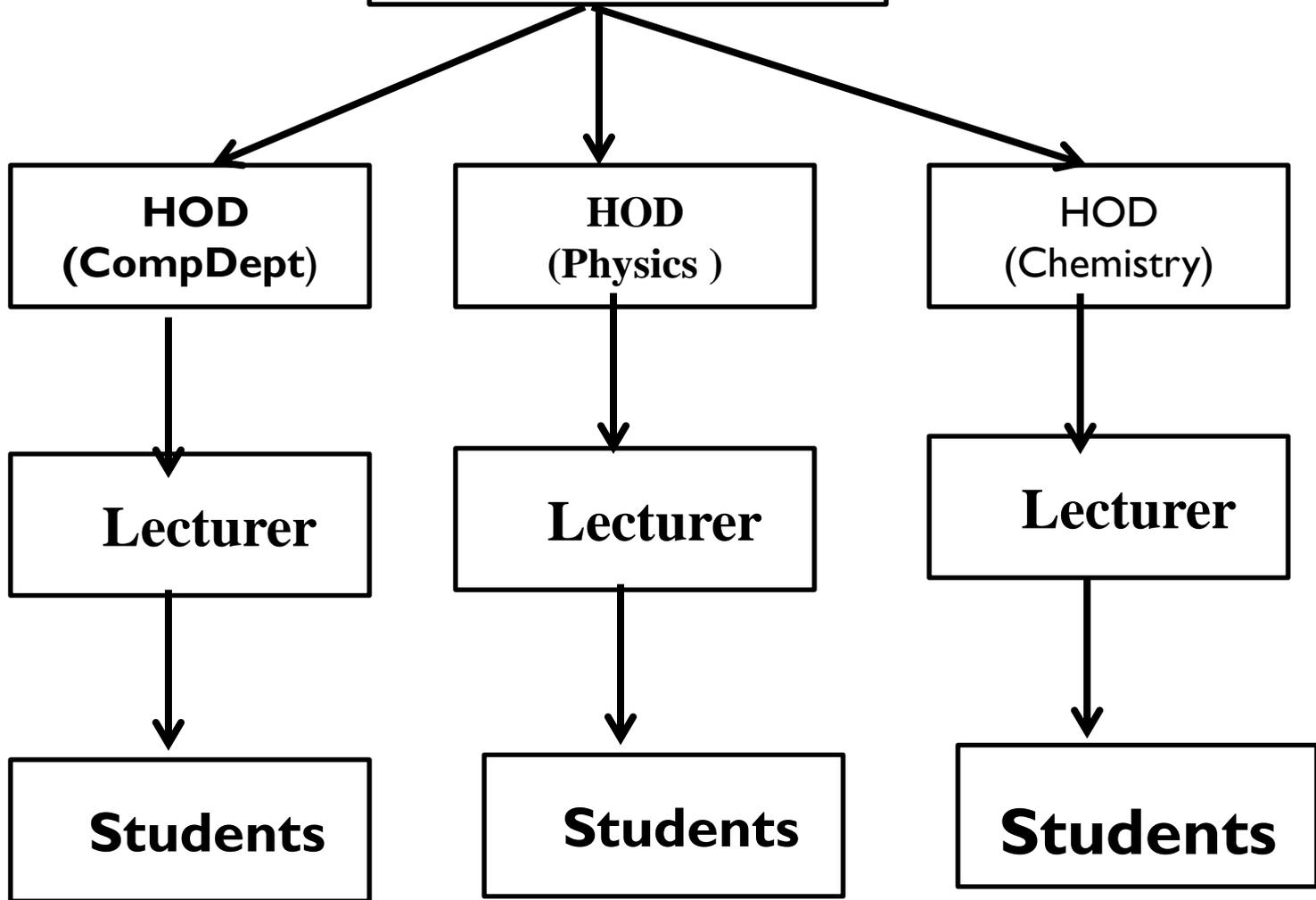


Organization

- Basically, Organization means the arrangement of components that helps to achieve objectives.
- Example in college system , the hierarchical relationship starting with the chairman on top and leading downward to peon, represent the organization structure.



Principal





Interaction:

- Interaction refers to the manner in which each component interacts with other components of the system.
- E.g. In a computer system all component are interact with other



Interdependence:

- Interdependence means parts of organization depend on one another.
- They are coordinated and linked together according to plan.
- In computer system, the three units Input, system unit, Output is interdependent for proper functioning.



Integration:

- Integration is concerned with how a system is tied together in order to achieve common goal, thus forming integration.
- It means that parts of system work together within system even though each part performs unique function.



Central Objective:

- Objectives may be stated or real.
- The stated objective and real objective of the system could differ based on the policy of the company.
- The user should develop a central objective by taking into consideration real objective and stated objective .
- The important point is that the users must know the central objective of computer application in analysis for successful design and conversion .



Types of systems:

Physical System

- The physical system could be static or dynamic in nature.
- Static means which do not change as far as working or life of the system is concerned.
on other hand dynamic system may change due to processing of system.



Example

- In computer system the hardware parts are static but the data which changes due to processing is dynamic. These both together form physical system.



2. Abstract system:

- The system which are represented conceptually(Non Existing)
- It is prepared by studying the physical system.
- The computer is a physical system and its block diagram is called as abstract system
A model representation of real or planned system.



System Model

- The use of model makes it easier for analyst to visualize relationship in system study.

System Model

The analyst begins by creating a model of reality i.e fact, relationship, procedure...etc with which the system is concerned. Every computer system deal with real world



Various system model are used to show benefits of abstracting systems to model form. The major of this type models are:

Schematic models: It shows a two dimensional system elements and their linkages.

Flow system models: It shows the flow of material , energy and information that hold system together. A widely used example is program evaluation and review technique(PERT)



Static system models: This type of model exhibits a pair of relationship such as activity time or cost quantity. E.g. Gantt Chart

Dynamic system models: Business organizations are dynamic systems. It depicts constantly an ongoing constantly changing the system. It consists of:

1. Inputs that enter the system.
2. Processor through which transformation takes place.
3. The programs required for processing.
4. Output that results from processing.



3. Open and closed system:

a. Open system:

- Another classification of systems is based on their degree of independence.
- An open system is a one which does not provide for its own control or modification. It does not supervise itself so it needs to be supervised by people.
- For example: If the high speed printer used with computer systems did not have a switch to sense whether paper is In the printer, then a person would have to notice when the paper runs out and



Signal the system to stop printing.

Characteristics of Open system

Inputs from outside: Open systems are self-adjusting and self-regulating. When functioning properly, an open system reaches a steady state.

Entropy: All dynamic systems tend to run down over time, resulting in entropy or loss of energy. Open systems resist entropy by seeking new inputs or modifying the processes to return to a steady state.



Characteristics of Open system

3. Process, Output and cycle: Open systems produce useful output and operate in cycle.

B. Closed system:

A closed system is one which automatically controls or modifies its own operation by responding to data generated by the system itself

For example: High speed printers used with computer systems usually have a switch that senses whether there is paper in the printer. If the paper runs out, the switch signals to stop printing.



4. Manmade system:

- With the help of information system, we can define some standard for the working of the system. This we can try to make the system to work according to the standards defined.
- We can define information system as a set of devices, procedures, rules but most of the work performs manually.
- It provides instructions, commands and feedback. It determines nature of relationship among decision makers.



5. Formal Information system:

- A formal information system is represented by organization chart. It gives a representation of the different parts of system and flow of information among them.
- **Categories of information**
 - **1. Strategic information:** It relates to long range planning in system.
 - **2. Management**



- Strategic Information
 - Managerial Information
 - Operational Information
-
- The first level relates to long range planning policies that are of direct interest to upper management
 - The second level is of direct use of middle management and department heads for implementation and control.
 - The Third level is short term, daily information is used to operate department and enforce day-to-day rules and regulations of information



6. Informal Information system:

- The informal information system is employee based system design to meet personnel and vocational needs and to help in the solution of work-related it is considered to be a useful system because it works within the framework of the business and its stated policies.
- It is related with what is happening practically rather than what is shown on paper, it is an employee based system designed to meet personal and vocational needs and help work related problem.



7. Computer Based Information System:

- Computer based information is more accurate, more neat and attractive it is possible to perform different operations easily. Security of data is possible in this system.



8. Management Information System (MIS)

A management information system (MIS) is a computer system consisting of hardware and software that serves as the backbone of an organization's operations. An MIS gathers data from multiple online systems, analyzes the information, and reports data to aid in management decision-making.



Advantages:

- Processing time and number of programs written are reduced.
- All applications share centralized files.
- Storage space duplication is eliminated.
- Data are stored once in database and are easily accessible when needed



9. Decision Support system (DSS)

- A decision support system (DSS) is a computerized system that gathers and analyzes data, synthesizing it to produce comprehensive information reports.
- A decision support system differs from an ordinary operations application, whose function is just to collect data.
- Decision support systems allow for more informed decision-making, timely problem-solving, and improved efficiency in dealing with issues or operations, planning, and even management



The primary purpose of using a DSS is to present information to the customer in an easy-to-understand way. A DSS system is beneficial because it can be programmed to generate many types of reports, all based on user specifications. For example, the DSS can generate information and output its information graphically, as in a bar chart that represents projected revenue or as a written report.



10. Expert System(ES)

Expert Systems

- Artificial Intelligence is a piece of software that simulates the behavior and judgment of a human or an organization that has experts in a particular domain is known as an expert system. It does by acquiring relevant knowledge from its knowledge base and interpreting it according to the user's problem. The data in the knowledge base is added by humans that are expert in a particular domain and this software is used by a non-expert user to acquire some information. It is widely used in many areas such as medical diagnosis, accounting, coding, games etc.



- An expert system is an AI software that uses knowledge stored in a knowledge base to solve problems that would usually require a human expert thus preserving a human expert's knowledge in its knowledge base. They can advise users as well as provide explanations to them about how they reached a particular conclusion or advice.
- One expert system may contain knowledge from more than one human experts thus making the solutions more efficient.



Limitations:

- Don't have human-like decision making power.
- Can't possess human capabilities.
- Can't produce correct result from less amount of knowledge.
- Requires excessive training.

Advantages:

- Low accessibility cost.
- Fast response.
- Not affected by emotions unlike humans.
- Low error rate.
- Capable of explaining how they reached a solution.



Executive information system (EIS)

An executive information system (EIS) is a decision support system (DSS) used to assist senior executives in the decision-making process. It does this by providing easy access to important data needed to achieve strategic goals in an organization. An EIS normally features graphical displays on an easy-to-use interface.

Executive information systems can be used in many different types of organizations to monitor enterprise performance as well as to identify opportunities and problems.



Advantages of EIS

1. EIS is easy for upper level executives.
2. EIS provides timely delivery of company summary information.
3. EIS filters data for management.



Integrated system:

- Integrated systems, or systems integration, is the process of bringing together component sub-systems into one functional system. It provides a system with coherence by making the parts or components work together, or ‘building or creating a whole from parts
- It consists of individual computers may be workstations or multiple systems.



- Each of them runs a set of standard software and deals initially with it's own application.
- It is a different approach, sometimes called the integrated model and provides truly distributed system by designing it from scratch. In this integrated model and provides truly.



Sub System:

- A Larger system divided into subparts that subparts is known as subsystem.
- For example:
- Microprocessor of a motherboard. Where Motherboard is system and microprocessor is subsystem.



Transaction processing system(TPS)

- A Transaction processing system is a type of information system. TPS collect, store, modify and retrieve the transactions of an organization. A transaction is an event that generates or modifies data that is eventually stored in an information system.
- It works on transaction in database system. This system is useful in online shopping, online booking.



Unit 2 Introduction to Software Engineering

Definition of Software

- **Software** is defined as collection of computer programs, procedures, rules and data. Software Characteristics are classified into six major components:

Definition of Software Engineering

- Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.
- Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Definition of Software Engineering

- Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.



SOFTWARE CHARACTERISTICS

- **Software is developed:** It is not manufactured. It is not something that will automatically roll out of an assembly line. It ultimately depend on individual skill and creative ability
- **Software does not Wear Out :** Software is not susceptible to the environmental melodies and it does not suffer from any effects with time
-
- **Most Software is Created and Not Assembled from Existing Components**

- Several models of software quality factors and their categorization have been suggested over the years. The classic model of software quality factors, suggested by McCall, consists of 11 factors (McCall et al., 1977). Similarly, models consisting of 12 to 15 factors, were suggested by Deutsch and Willis (1988) and by Evans and Marciniak (1987).
- All these models do not differ substantially from McCall's model. The McCall factor model provides a practical, up-to-date method for classifying software requirements (Pressman, 2000).



McCall's Factor Model

- This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.
- Product operation factors – Correctness, Reliability, Efficiency, Integrity, Usability.
- Product revision factors – Maintainability, Flexibility, Testability.
- Product transition factors – Portability, Reusability, Interoperability.



Product Operation Software Quality Factors

- According to McCall's model, product operation category includes five software quality factors, which deal with the requirements that directly affect the daily operation of the software. They are as follows –



Correctness

- These requirements deal with the correctness of the output of the software system. They include –
- Output mission
- The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be affected by incomplete data.
- The up-to-dateness of the information defined as the time between the event and the response by the software system.
- The availability of the information.



Reliability

- Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.



Efficiency

- It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).
- It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.



Integrity

- This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.



Usability

- Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

Product Revision Quality Factors

- According to McCall's model, three software quality factors are included in the product revision category. These factors are as follows –
 - Maintainability
 - This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.



Flexibility

- This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.



Testability

- Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.



Product Transition Software Quality Factor

- According to McCall's model, three software quality factors are included in the product transition category that deals with the adaptation of software to other environments and its interaction with other software systems. These factors are as follows –



Portability

- Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. The software should be possible to continue using the same basic software in diverse situations.



Reusability

- This factor deals with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.



Interoperability

- Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware. For example, the firmware of the production machinery and testing equipment interfaces with the production control software.



Software Processes

- The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.
- A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:
 1. The software must evolve to meet changing client needs.

Software Processes

- 1. Software specifications:** The functionality of the software and constraints on its operation must be defined.
- 2. Software development:** The software to meet the requirement must be produced.
- 3. Software validation:** The software must be validated to ensure that it does what the customer wants.
- 4. Software evolution**



The Software Process Model

- A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described. :

The Software Process Model

- Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering. Some examples of the types of software process models that may be produced are

- I. A workflow model: This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.

- 2. A dataflow or activity model: This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers

- **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible



Software Engineering | SDLC V-Model

- V-Model
- V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

- There are the various phases of Verification Phase of V-model:
- Business requirement analysis: This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
- System Design: In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.

- **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.

- **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as **Low-Level Design**

- **Coding Phase:**After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

- **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/units



Integration Testing

- Integration Testing: Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.

System Testing:

- System Testing: System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.

Acceptance Testing:

- **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.



When to use V-Model?

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

Advantage (Pros) of V-Model:

- Easy to Understand.
- Testing Methods like planning, test designing happens well before coding.
- This saves a lot of time. Hence a higher chance of success over the waterfall model.
- Avoids the downward flow of the defects.
- Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

- Very rigid and least flexible.
- Not a good for a complex project.
- Software is developed during the implementation stage, so no early prototypes of the software are produced.
- If any changes happen in the midway, then the test documents along with the required documents, has to be updated



**UNIT 3: SOFTWARE
DEVELOPMENT LIFE
CYCLE**

I Introduction

- **Software Development Life Cycle (SDLC)** is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

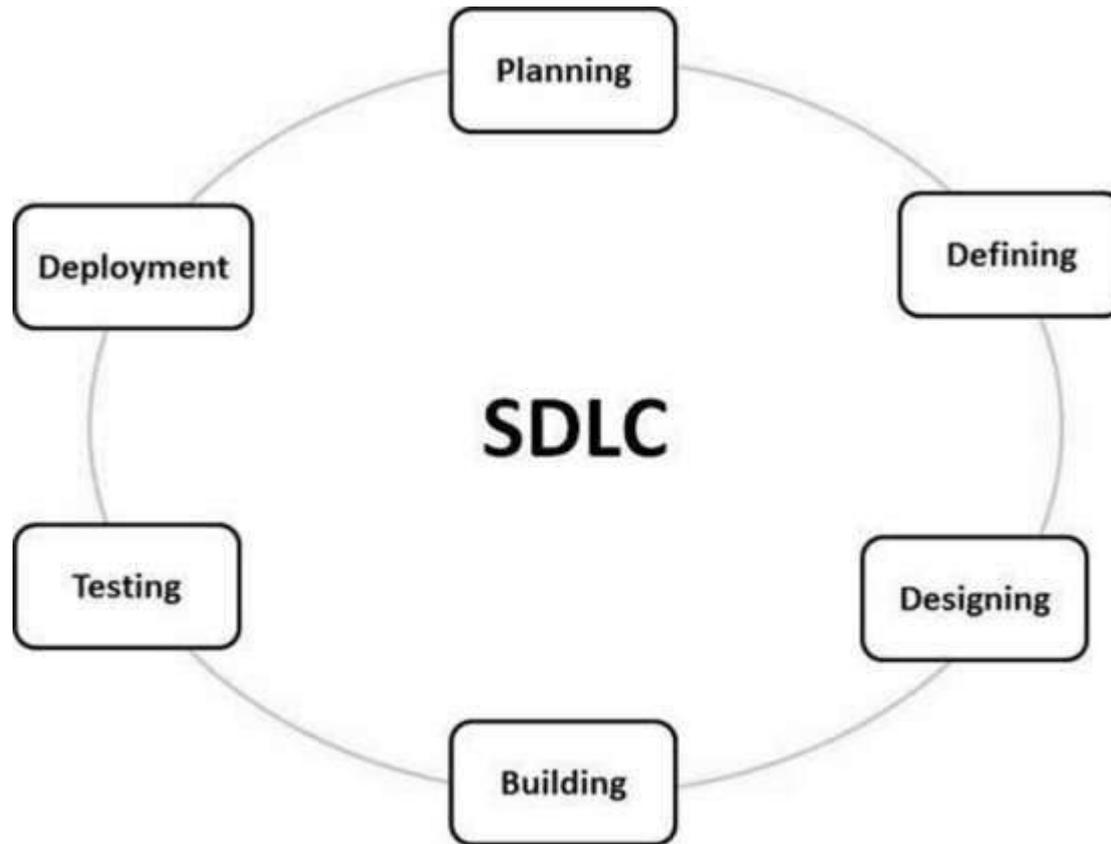
- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.

- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software



What is SDLC?

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





Stage I: Planning and Requirement Analysis

- Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Stage I: Planning and Requirement Analysis

- Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

- Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

- SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

Stage 4: Building or Developing the Product

- In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

- .

Stage 4: Building or Developing the Product

- Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed



Stage 5: Testing the Product

- This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

- Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).



Stage 6: Deployment in the Market and Maintenance

- Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

- There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

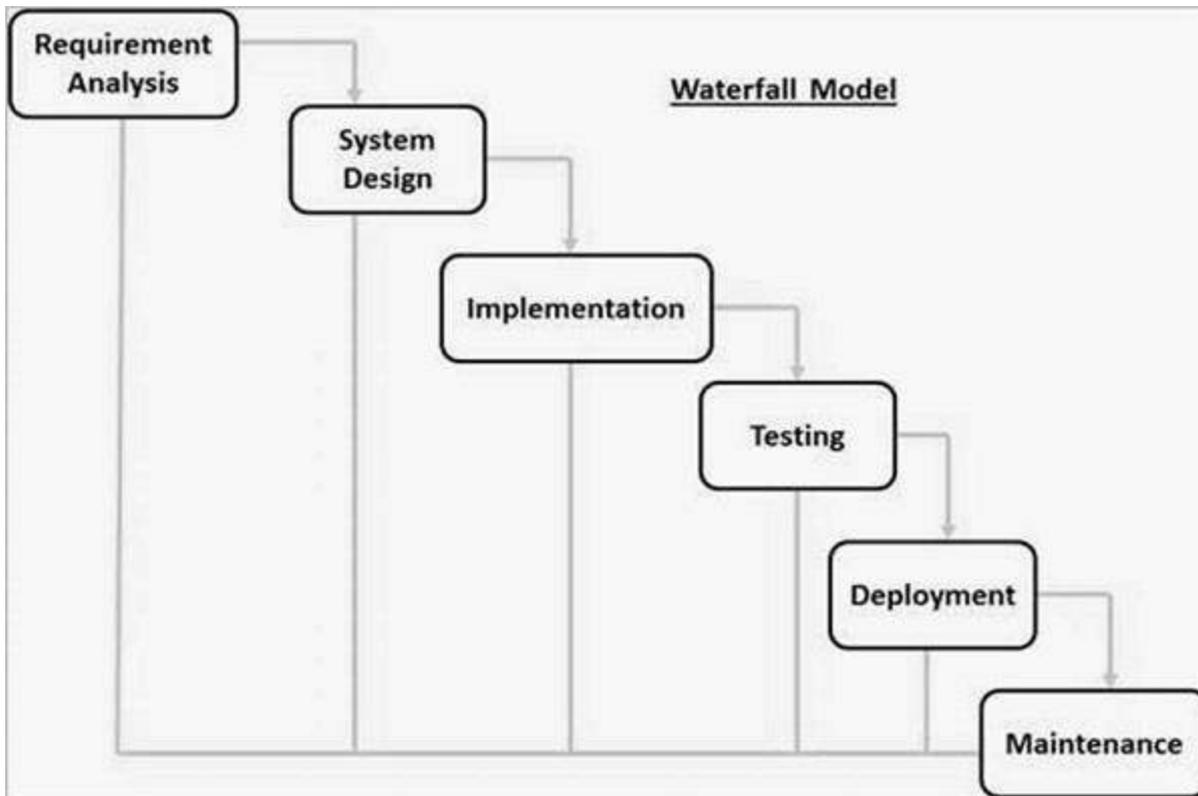
- Following are the most important and popular SDLC models followed in the industry –
- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model
- Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.



Waterfall Model - Design

- Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

- The following illustration is a representation of the different phases of the Waterfall Model.
- SDLC Waterfall Model



- The sequential phases in Waterfall model are –
- Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design

- **System Design** – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation

- **Implementation** – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Integration and Testing

- **Integration and Testing** – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system

- **Deployment of system** – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance

- **Maintenance** – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

- All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.



Waterfall Model - Application

- Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are –
- Requirements are very well documented, clear and fixed.
- Product definition is stable.

- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.



Waterfall Model - Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

Waterfall Model - Advantages

- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.

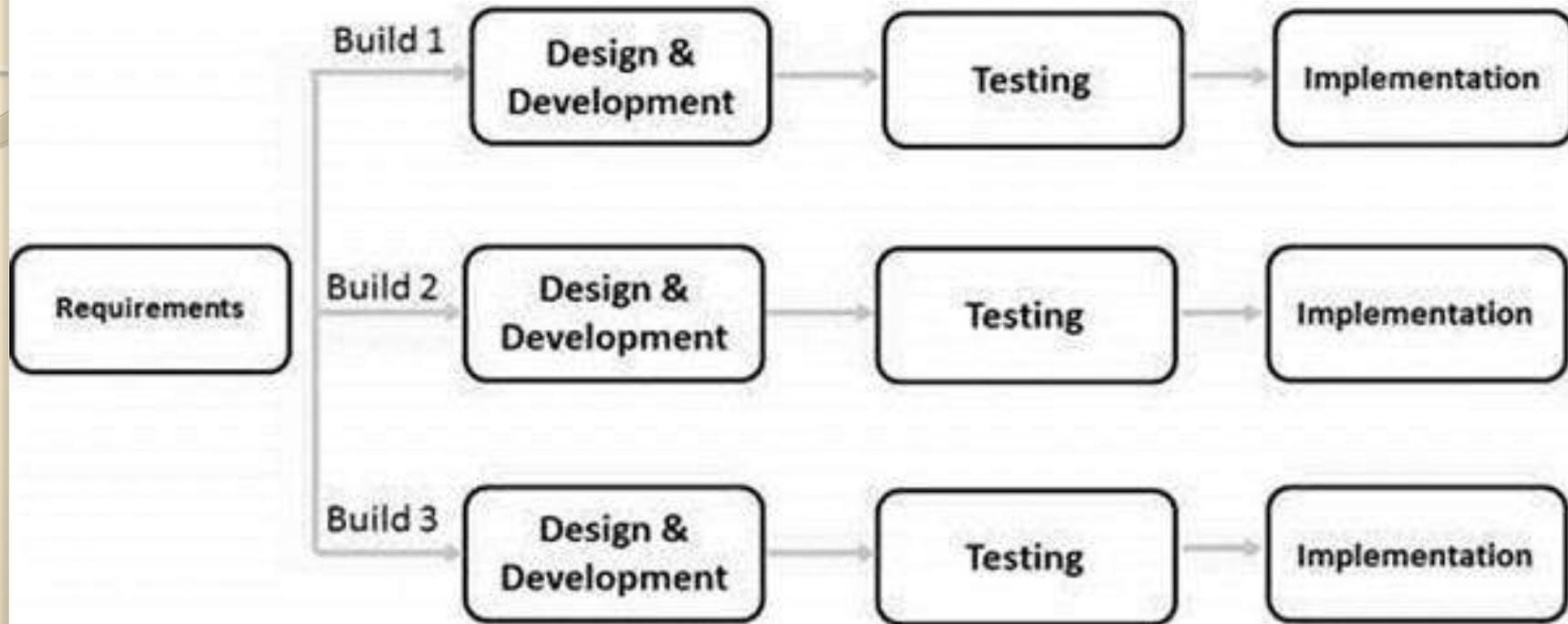


Waterfall Model - Disadvantages

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang" at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Model - Design

- Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).



- Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

- In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

- The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.

Iterative Model - Application

- Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios –
- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.

Iterative Model - Application

- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

Iterative Model - Pros

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.

disadvantages of the Iterative and Incremental SDLC Model

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.

Spiral Model - Design

- The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.



Spiral Model - Design

- Spiral Model - Design
- The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.



Identification

- This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

Identification

- This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.



Design

- The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.



Construct or Build

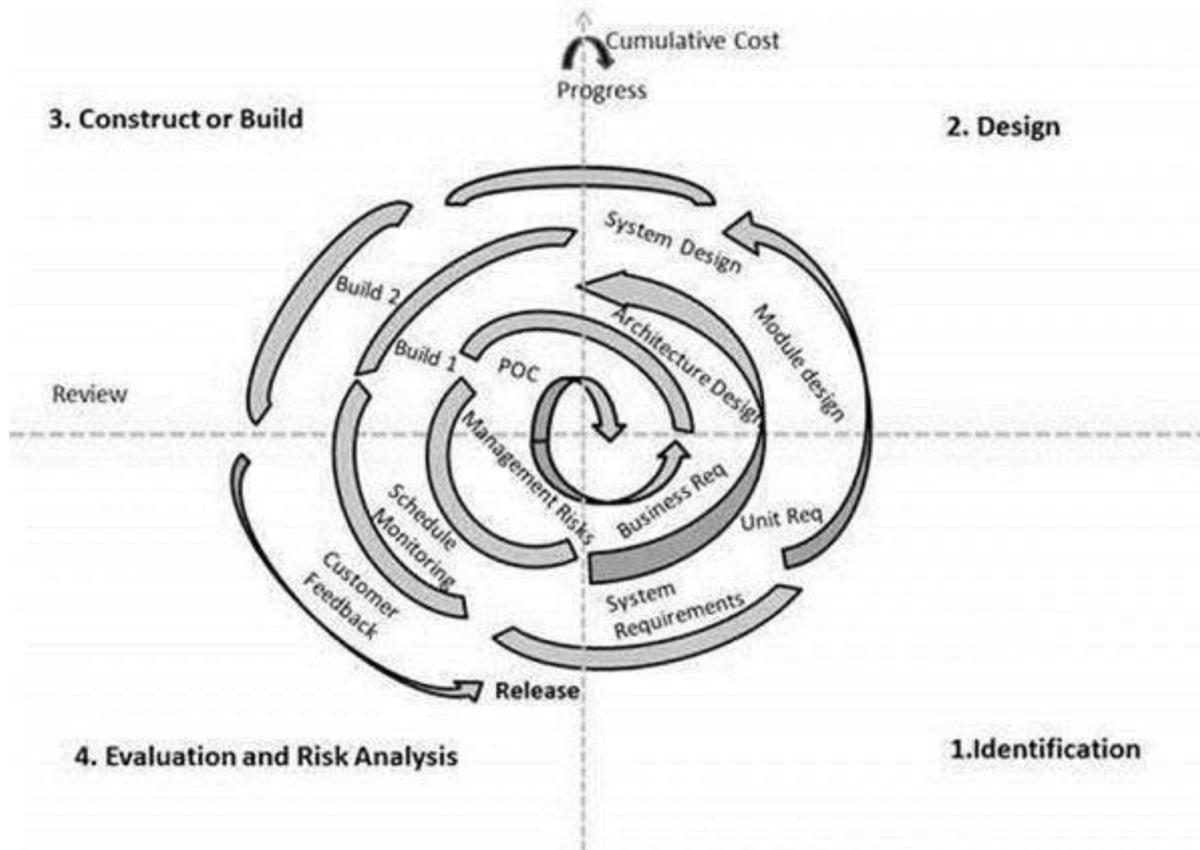
- The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Construct or Build

- Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback

Evaluation and Risk Analysis

- Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.
- The following illustration is a representation of the Spiral Model, listing the activities in each phase.



- Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.



Spiral Model Application

- The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

uses of a Spiral Model –

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

- The advantages of the Spiral SDLC Model are as follows –
- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows –

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

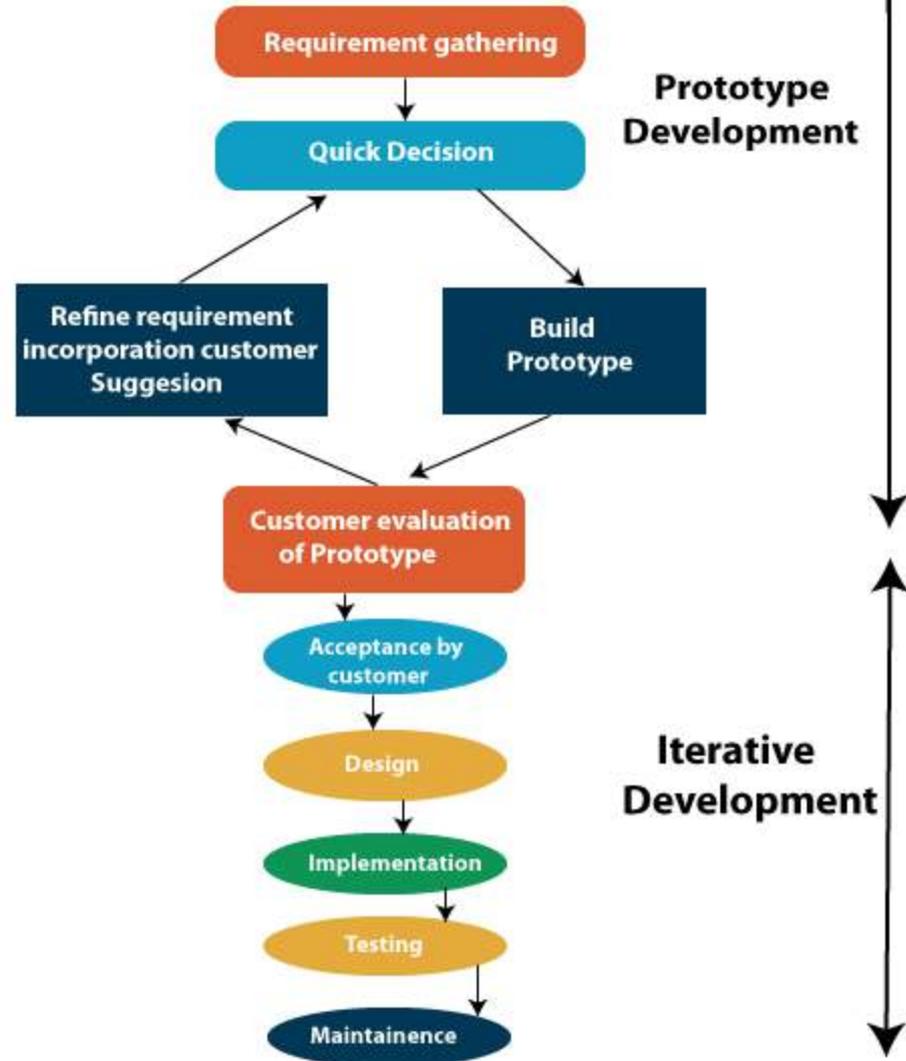
Prototype Model

- The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possibly exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software.

Prototype Model

- In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Fig: Prototype Model





Steps of Prototype Model

- Requirement Gathering and Analyst
- Quick Decision
- Build a Prototype
- Assessment or User Evaluation
- Prototype Refinement
- Engineer Product

Advantage of Prototype Model

- Reduce the risk of incorrect user requirement
- Good where requirement are changing/uncommitted
- Regular visible process aids management
- Support early product marketing
- Reduce Maintenance cost.
- Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

- An unstable/badly implemented prototype often becomes the final product.
- Require extensive customer collaboration
- Costs customer money
- Needs committed customer
- Difficult to finish if customer withdraw

Disadvantage of Prototype Model

- May be too customer specific, no broad market
- Difficult to know how long the project will last.
- Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
- Prototyping tools are expensive.
- Special tools & techniques are required to build a prototype.
- It is a time-consuming process.



UNIT 4 : REQUIREMENT ENGINEERING



Introduction

- Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process..

- Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

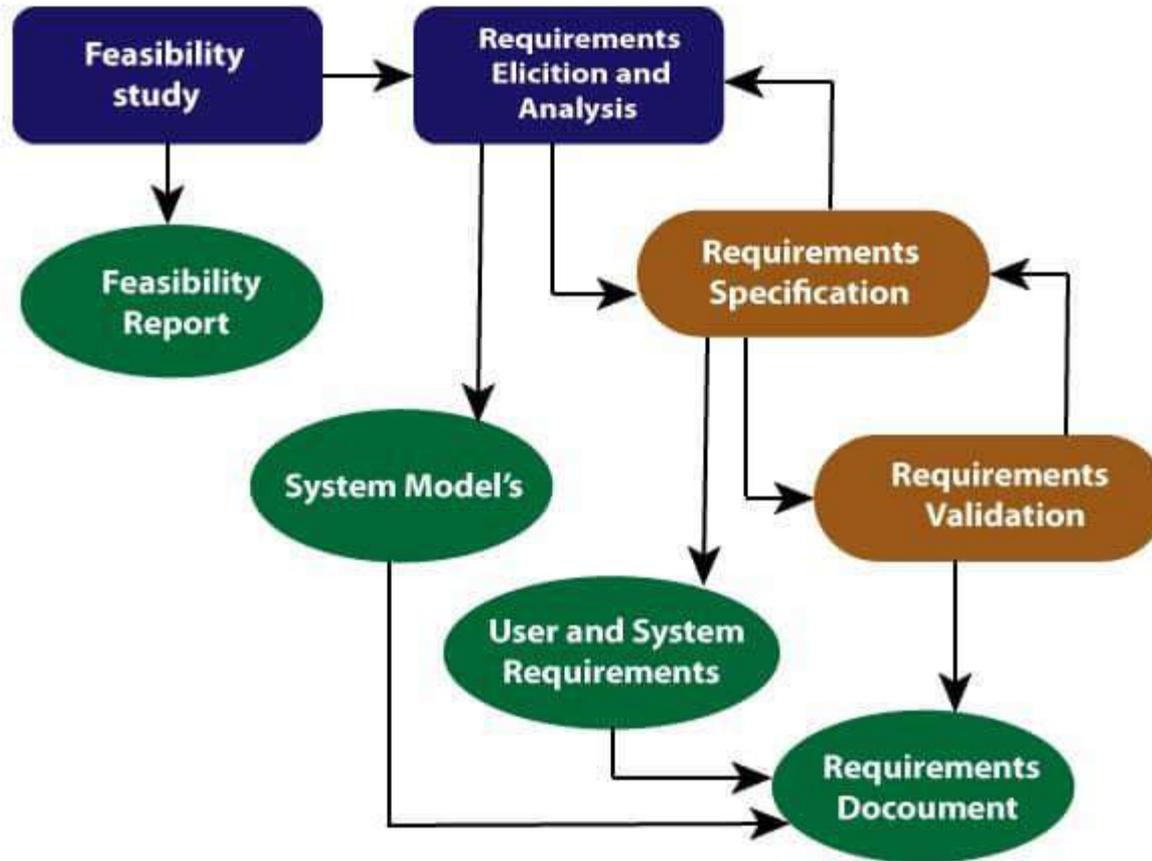
- Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints



Requirement Engineering Process

- It is a four-step process, which includes -
- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management

Requirement Engineering Process



Requirement Engineering Process



Feasibility Study:

- The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.



Types of Feasibility:

- **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
- **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
- **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.



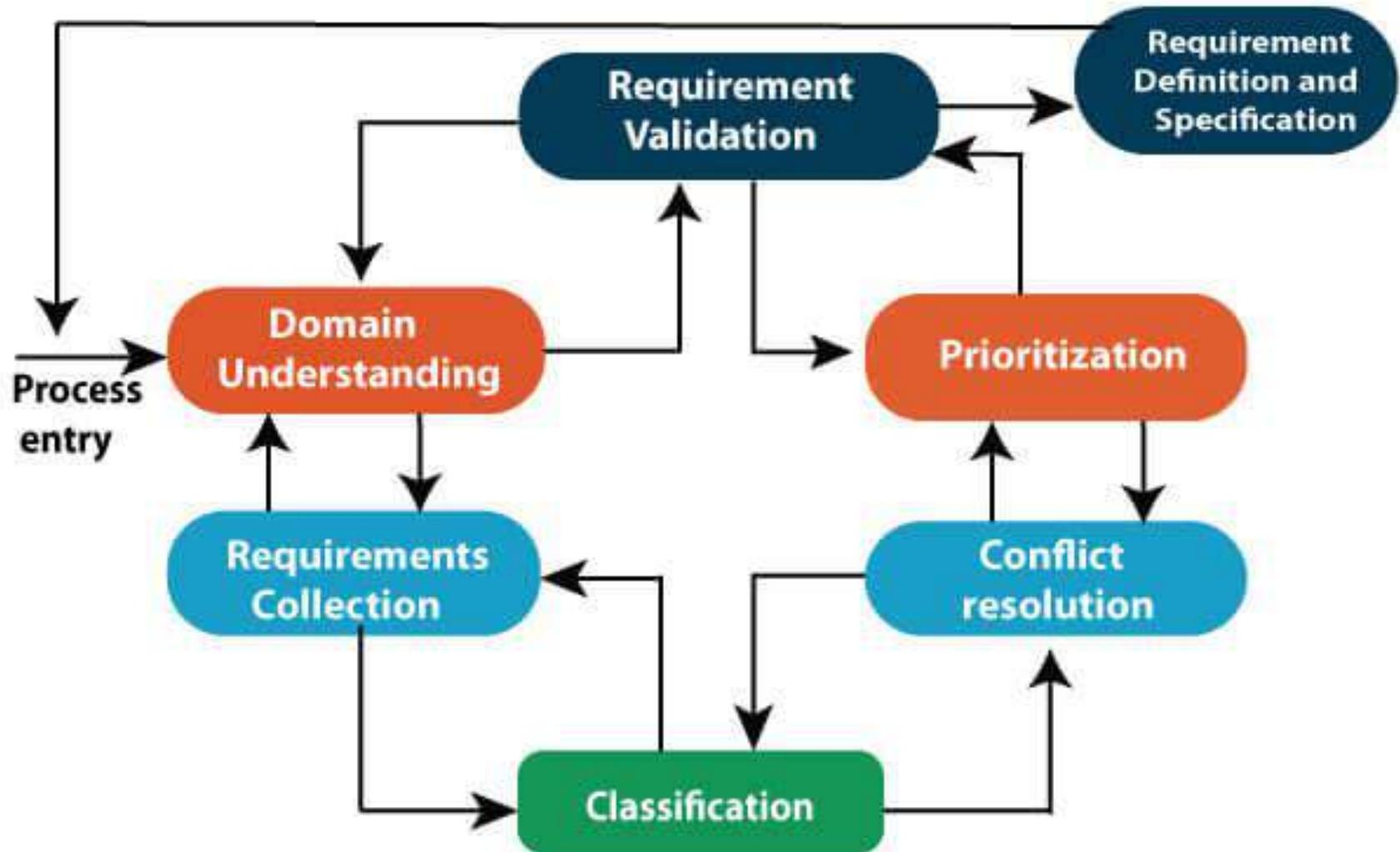
Requirement Elicitation and Analysis:

- This is also known as the gathering of requirements. Here, requirements are identified with the help of customers and existing systems processes, if available.
- Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements

Elicitation and Analysis Process



3. Software Requirement Specification:

- Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

- The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.
- Data Flow Diagrams: Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "E-R diagram." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

- After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

Software Requirement Validation

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- Requirements reviews/inspections: systematic manual analysis of the requirements.
- Prototyping: Using an executable model of the system to check requirements.
- Test-case generation: Developing tests for requirements to check testability.
- Automated consistency analysis: checking for the consistency of structured requirements descriptions.

Software Requirement Management:

- Requirement management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

- The priority of requirements from different viewpoints changes during development process.
- The business and technical environment of the system changes during the development.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

Requirement Gathering Techniques

- Techniques describe how tasks are performed under specific circumstances. A task may have none or one or more related techniques. A technique should be related to at least one task.
- The following are some of the well-known requirements gathering techniques –



Brainstorming

- Brainstorming is used in requirement gathering to get as many ideas as possible from group of people. Generally used to identify possible solutions to problems, and clarify details of opportunities.



Document Analysis

- Reviewing the documentation of an existing system can help when creating AS–IS process document, as well as driving gap analysis for scoping of migration projects. In an ideal world, we would even be reviewing the requirements that drove creation of the existing system – a starting point for documenting current requirements. Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.



Focus Group

- A focus group is a gathering of people who are representative of the users or customers of a product to get feedback. The feedback can be gathered about needs/opportunities/problems to identify requirements, or can be gathered to validate and refine already elicited requirements. This form of market research is distinct from brainstorming in that it is a managed process with specific participants.



Interview

- Interviews of stakeholders and users are critical to creating the great software. Without understanding the goals and expectations of the users and stakeholders, we are very unlikely to satisfy them. We also have to recognize the perspective of each interviewee, so that, we can properly weigh and address their inputs. Listening is the skill that helps a great analyst to get more value from an interview than an average analyst.

Observation

- By observing users, an analyst can identify a process flow, steps, pain points and opportunities for improvement. Observations can be passive or active (asking questions while observing). Passive observation is better for getting feedback on a prototype (to refine requirements), where active observation is more effective at getting an understanding of an existing business process. Either approach can be used.

Definition of Fact-finding Techniques

- Fact finding is process of collection of data and information based on techniques which contain sampling of existing documents, research, observation, questionnaires, interviews, prototyping and joint requirements planning. System analyst uses suitable fact-finding techniques to develop and implement the current existing system.

- Collecting required facts are very important to apply tools in System Development Life Cycle because tools cannot be used efficiently and effectively without proper extracting from facts. Fact-finding techniques are used in the early stage of System Development Life Cycle including system analysis phase, design and post implementation review.

- Facts included in any information system can be tested based on three steps: data-facts used to create useful information, process- functions to perform the objectives and interface- designs to interact with users.



Fact-finding techniques

- There are seven common fact-finding techniques
- Sampling of existing documentation, forms and databases
- Research and Site visits
- Observation of the work environment
- Questionnaires
- Interviews
- Prototyping



Software Requirement Specification (SRS) Format

- In order to form a good SRS, here you will see some points which can be used and should be considered to form a structure of good SRS. These are as follows

- Introduction
- **(i)** Purpose of this document
- **(ii)** Scope of this document
- **(iii)** Overview

- . General description
- 3. Functional Requirements
- 4. Interface Requirements
- 5. Performance Requirements
- 6. Design Constraints
- 7. Non-Functional Attributes
- 8. Preliminary Schedule and Budget
- 9. Appendices



Software Requirement Specification (SRS)

- Format as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-requirements depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers.

- Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.



Introduction :

- (i) Purpose of this Document –
- At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- (ii) Scope of this document –
- In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
- (iii) Overview –
- In this, description of product is explained. It's simply summary or overall review of product.

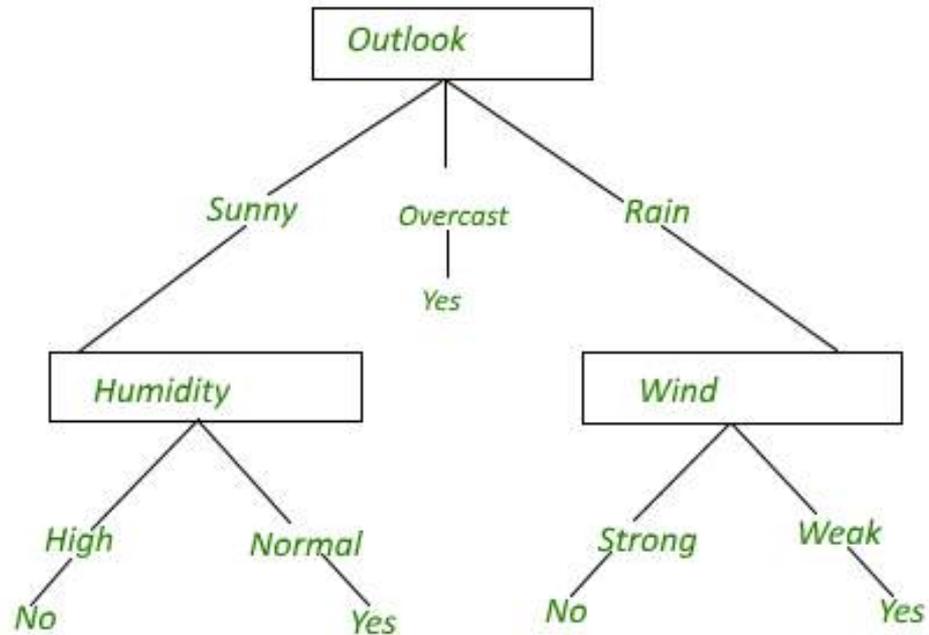


UNIT 5: ANALYSIS AND DESIGN TOOLS

Decision Tree

- Decision Tree : Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label

Decision Tree for *PlayTennis*



Construction of Decision Tree :

- A tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning classification.

- .The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery.

- Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on

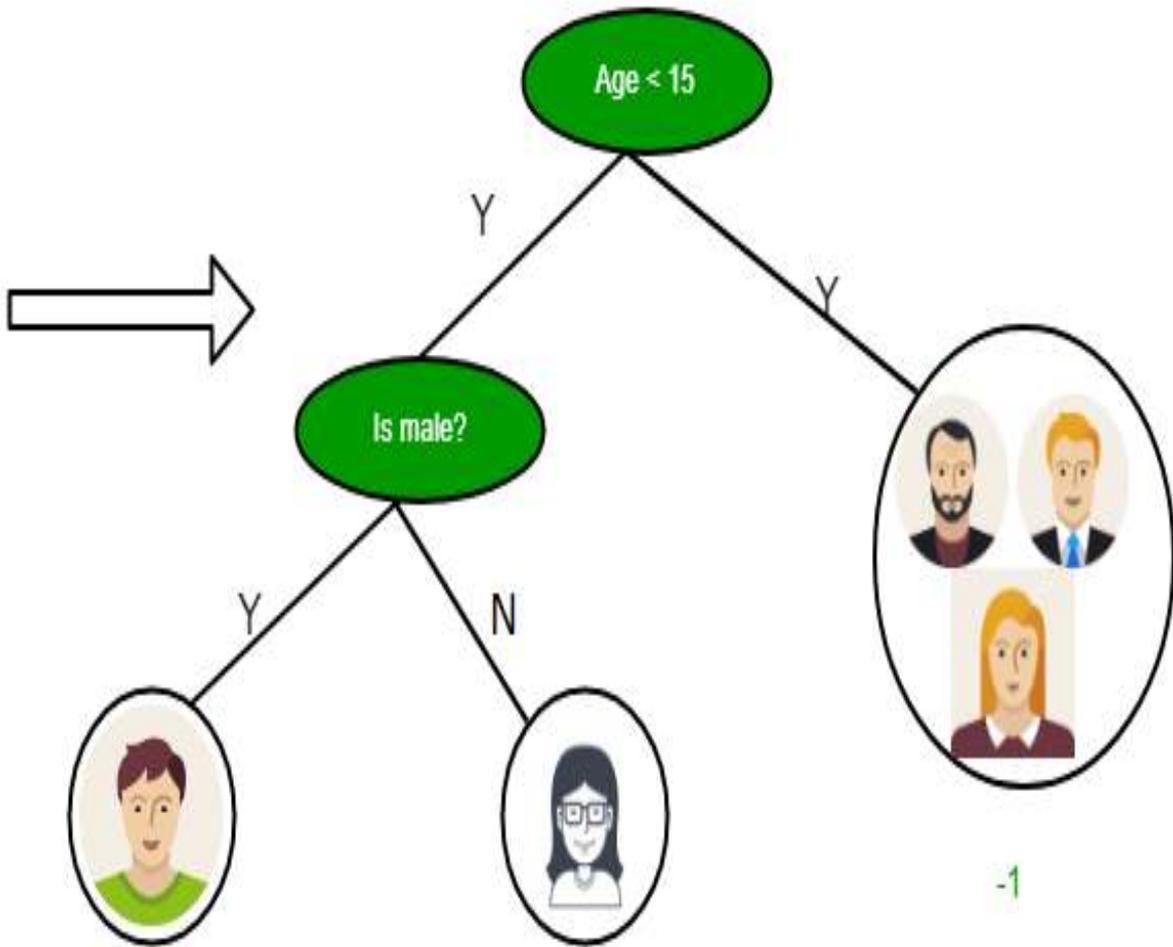
Decision Tree Introduction with example

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.

Input: Age, Gender, Occupation, ...



Does the person likes computer games

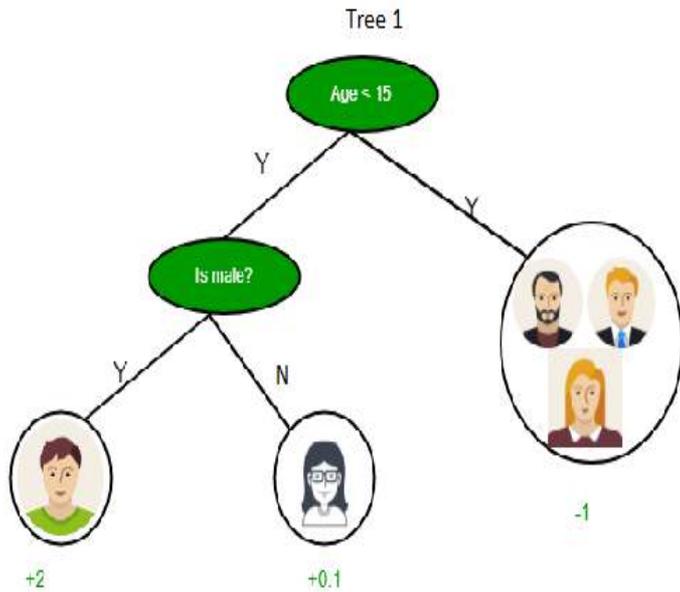


Prediction score in each leaf → +2

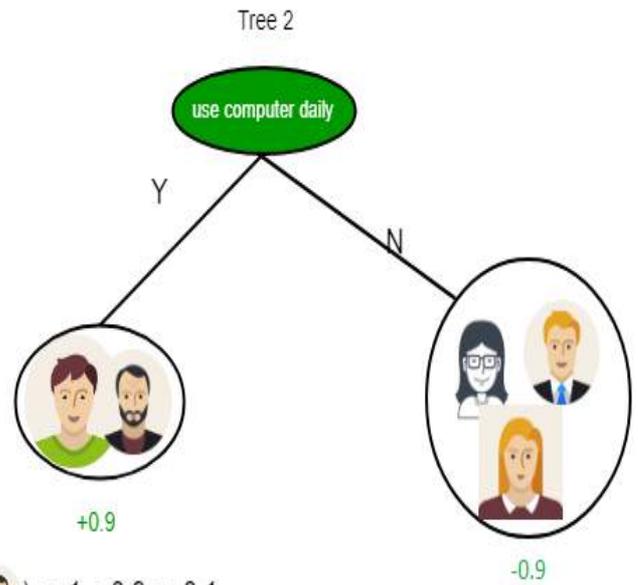
+0.1

-1

- Below are some assumptions that we made while using decision tree:
- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.



If () = 2 + 0.9 = 2.9



If () = -1 + 0.9 = -0.1

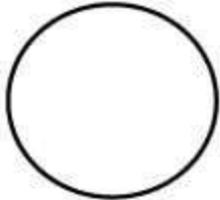
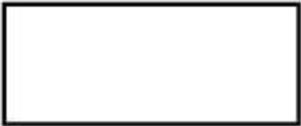
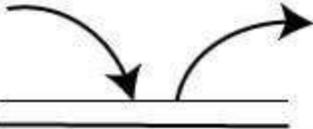
Data Flow Diagrams (DFD)

- Levels in Data Flow Diagrams (DFD)
- In Software engineering DFD(data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher-level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

- Data Flow Diagrams
- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble char

- The following observations about DFDs are essential:
- All names should be unique. This makes it easier to refer to elements in the DFD.
- Remember that DFD is not a flow chart. Arrows in a flow chart represent the order of events; arrows in DFD represent flowing data. A DFD does not involve any order of events.
- .

- Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represent decision points with multiple existing paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
- Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

- Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.
- Data Flow: A curved line shows the flow of data into or out of a process or data store.
- .

- **Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.
- **Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs



Levels in Data Flow Diagrams (DFD)

- The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD:

-
- It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

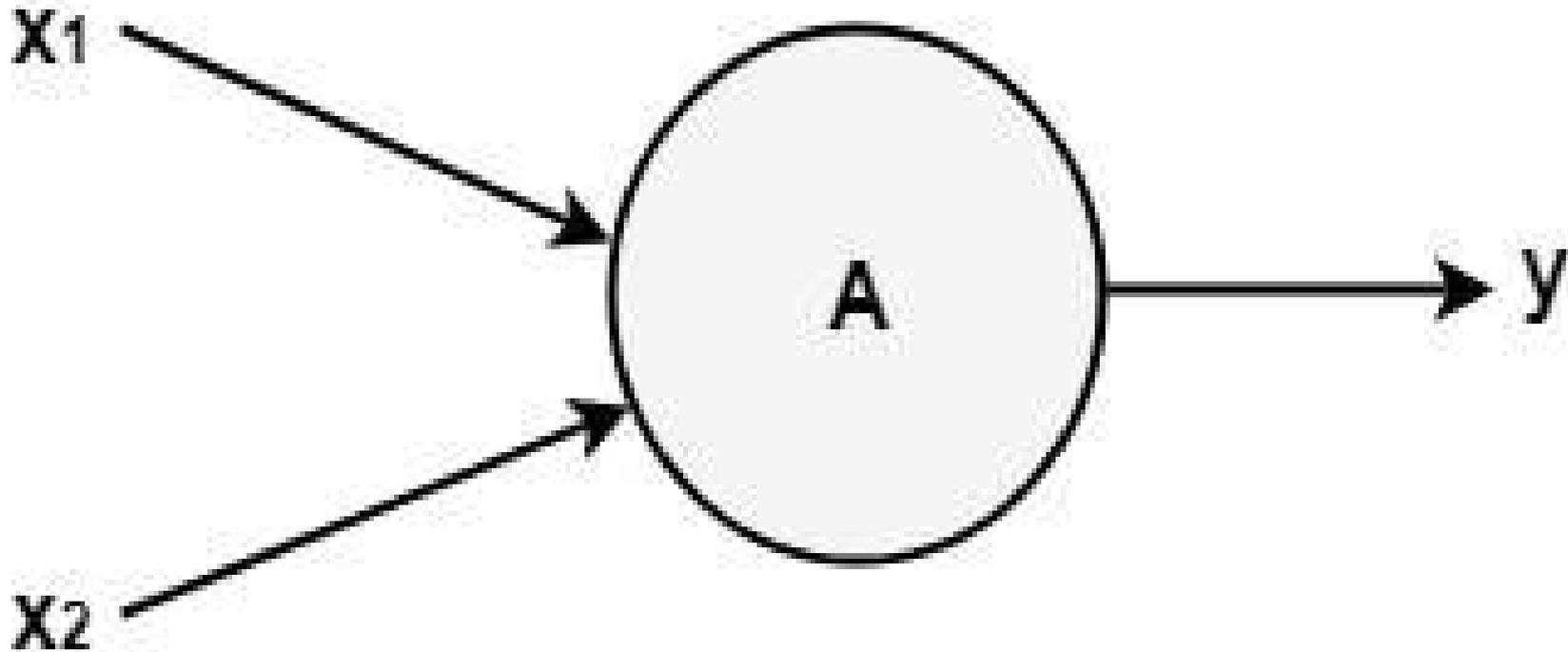


Fig: Level-0 DFD.

- The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed

Example

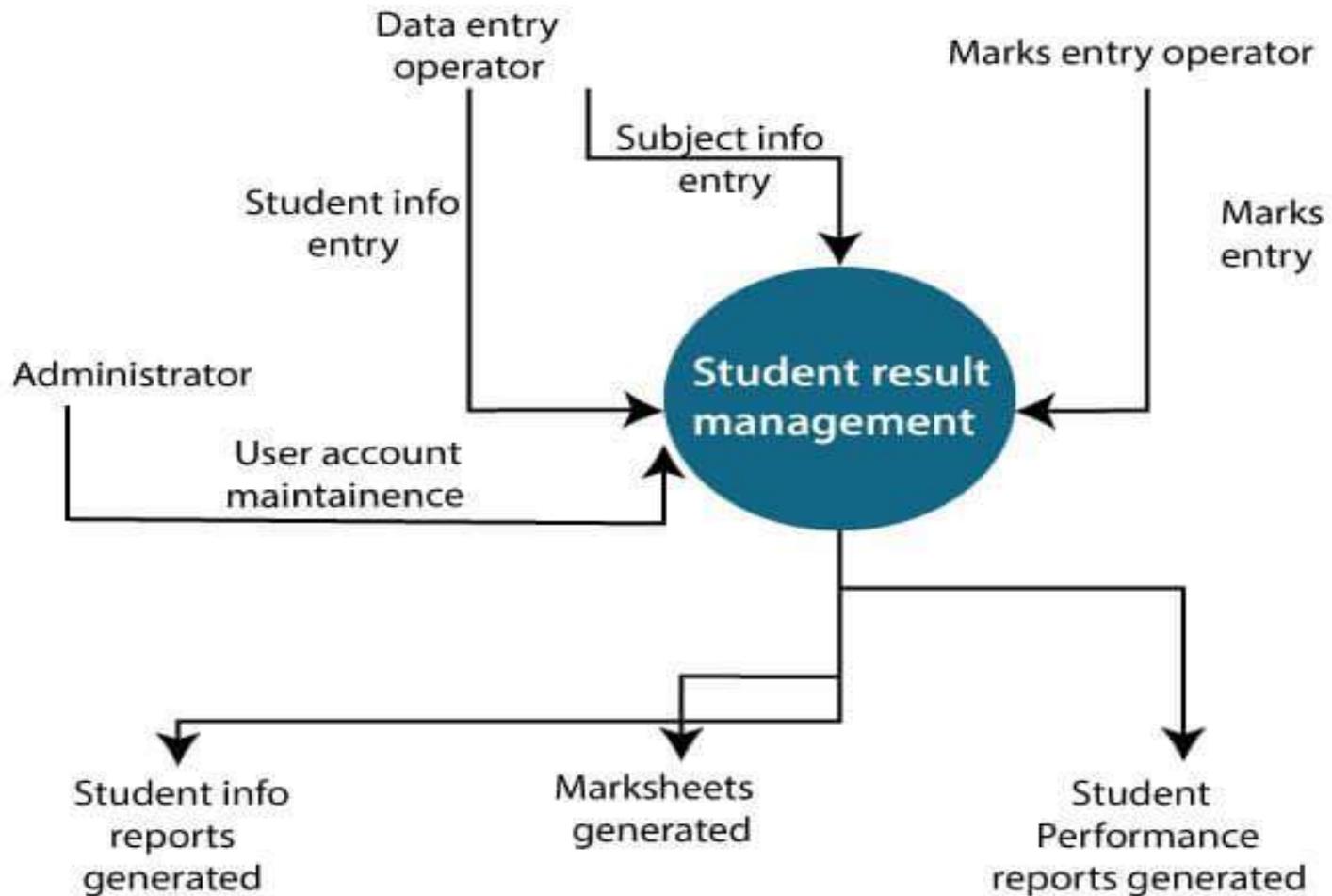


Fig: Level-0 DFD of result management system

I-level DFD

- In I-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

I-level DFD:

- In I-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

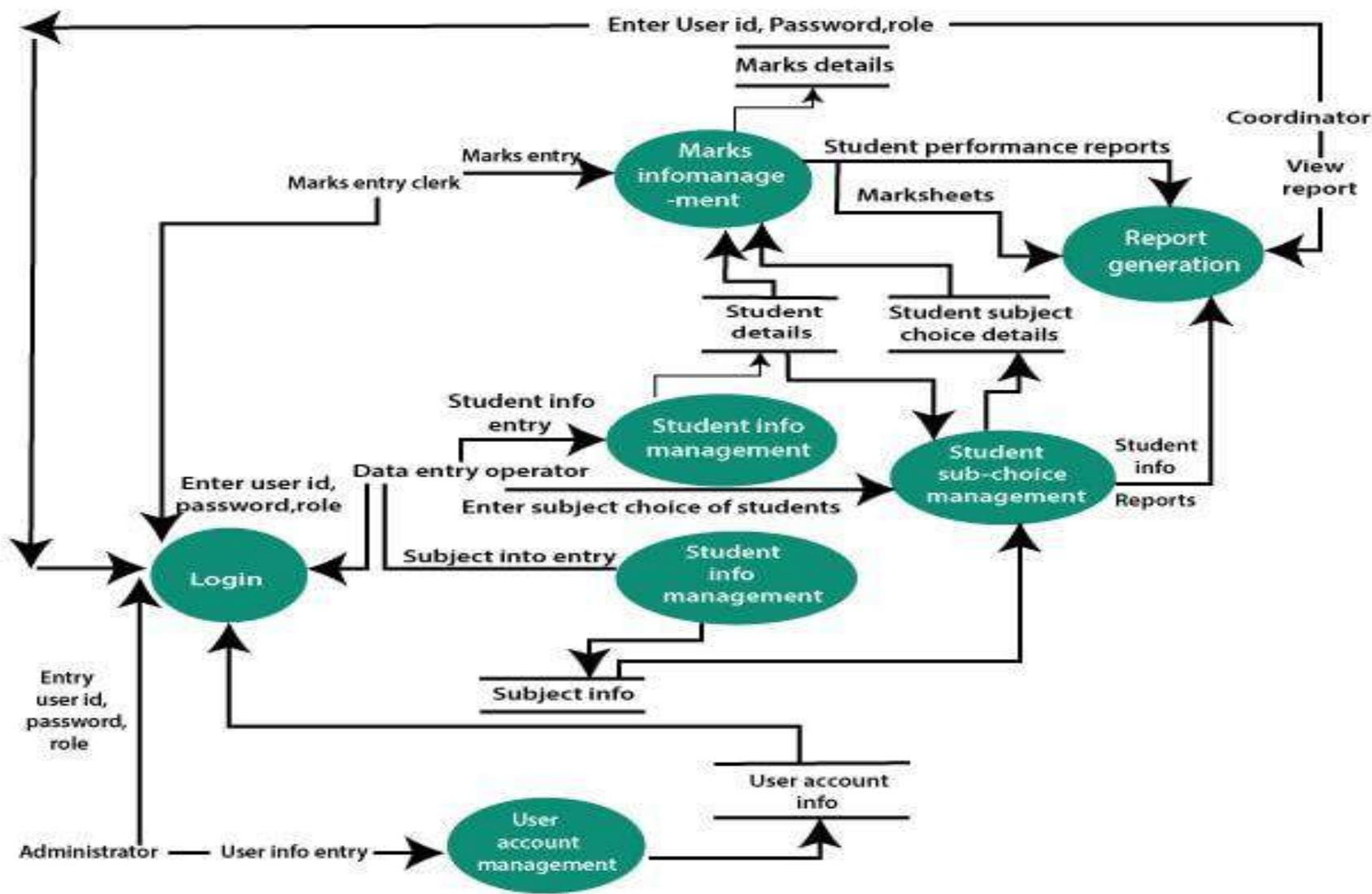
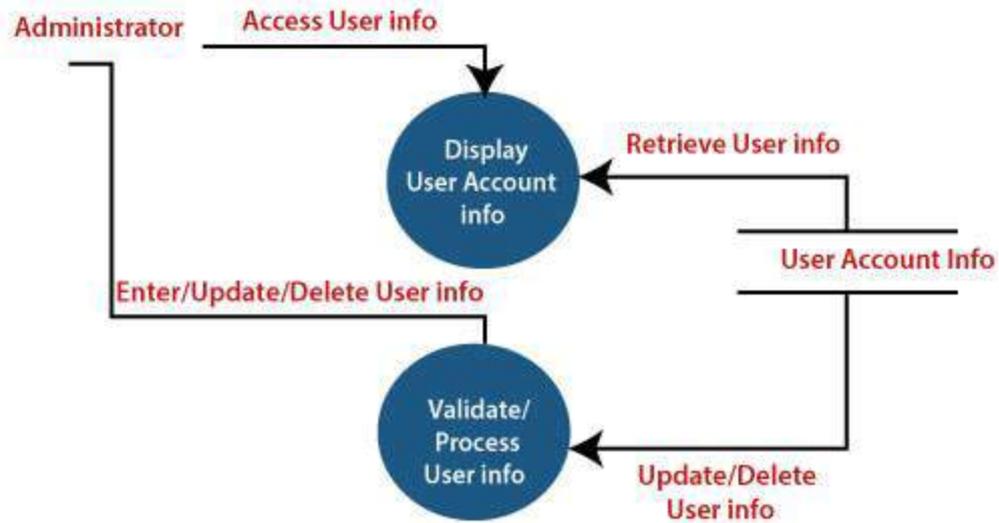


Fig: Level-1 DFD of result management system

2-Level DFD

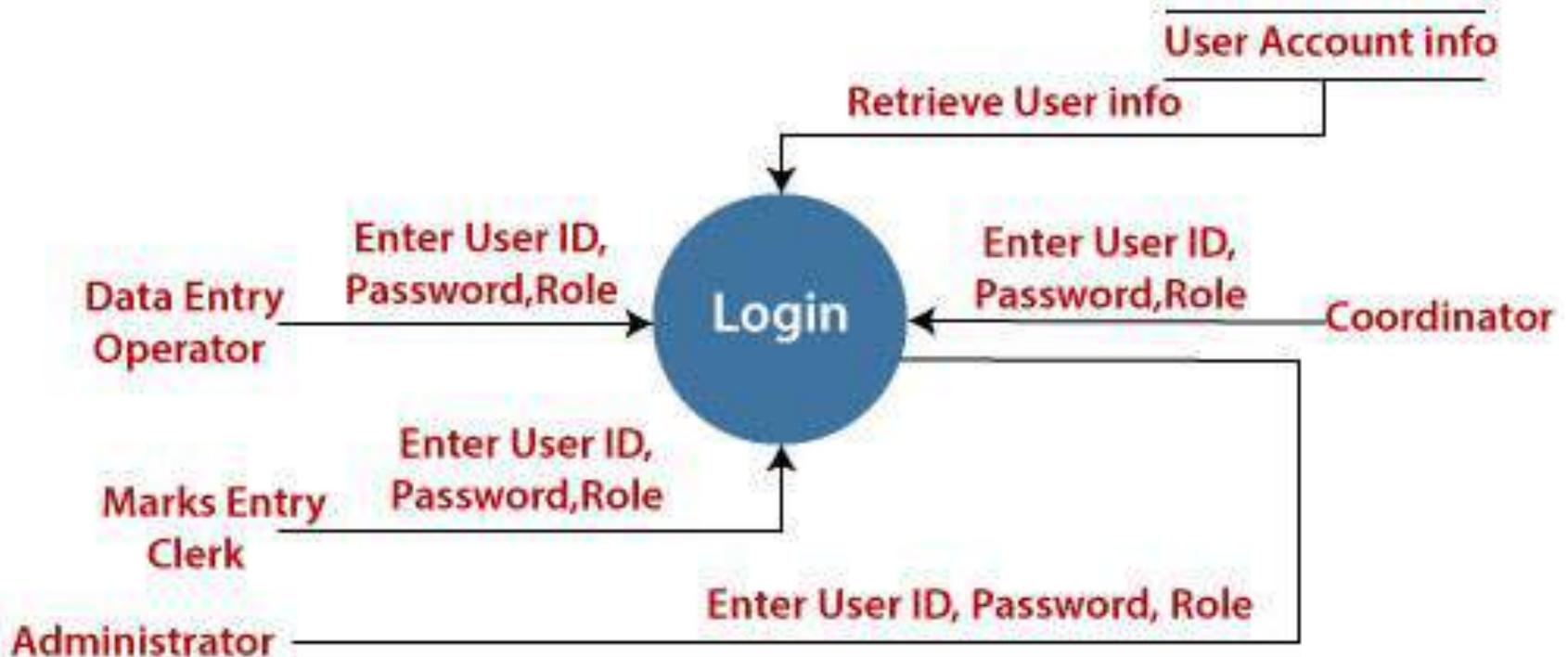
- 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

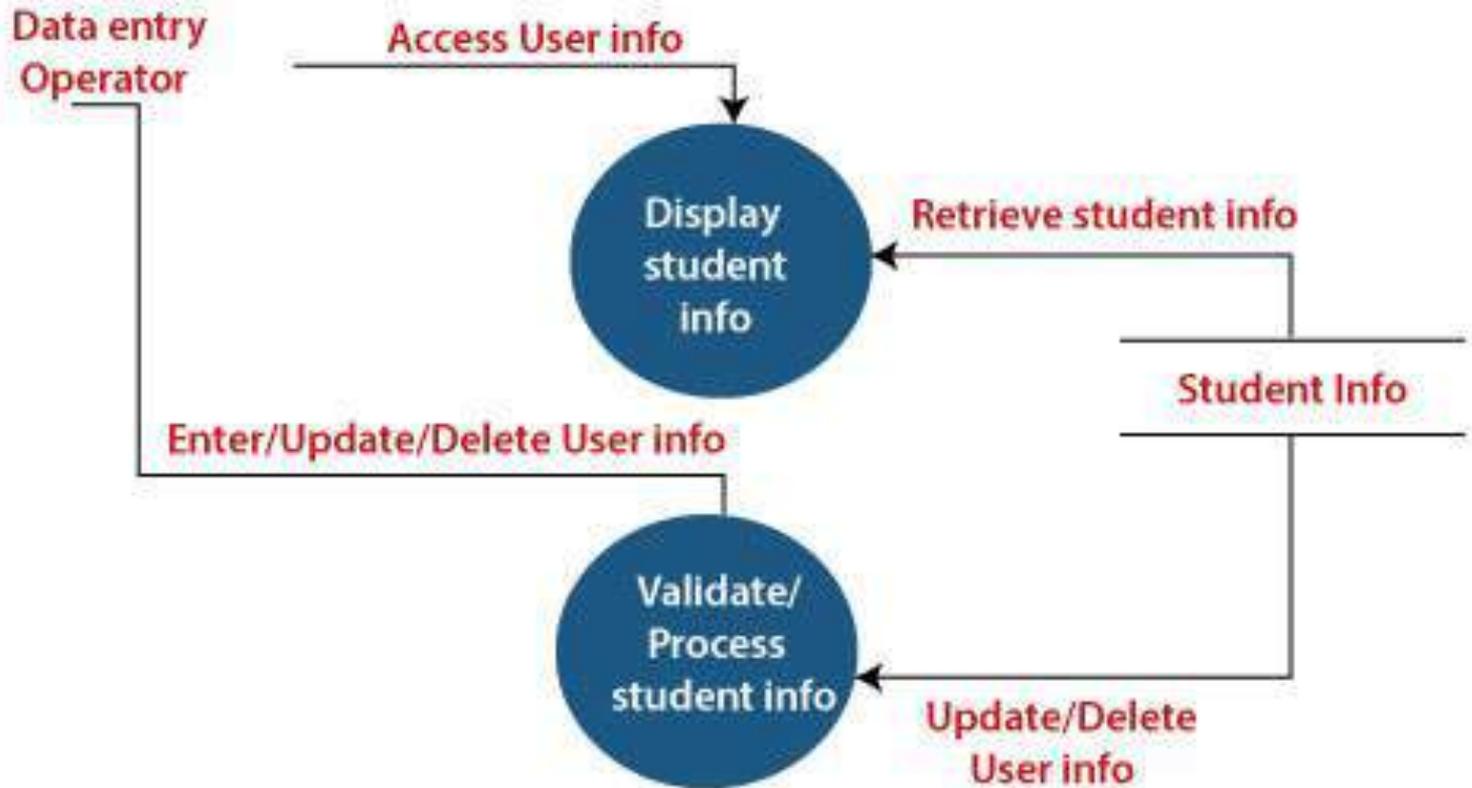


2. Login

The level 2 DFD of this process is given below:

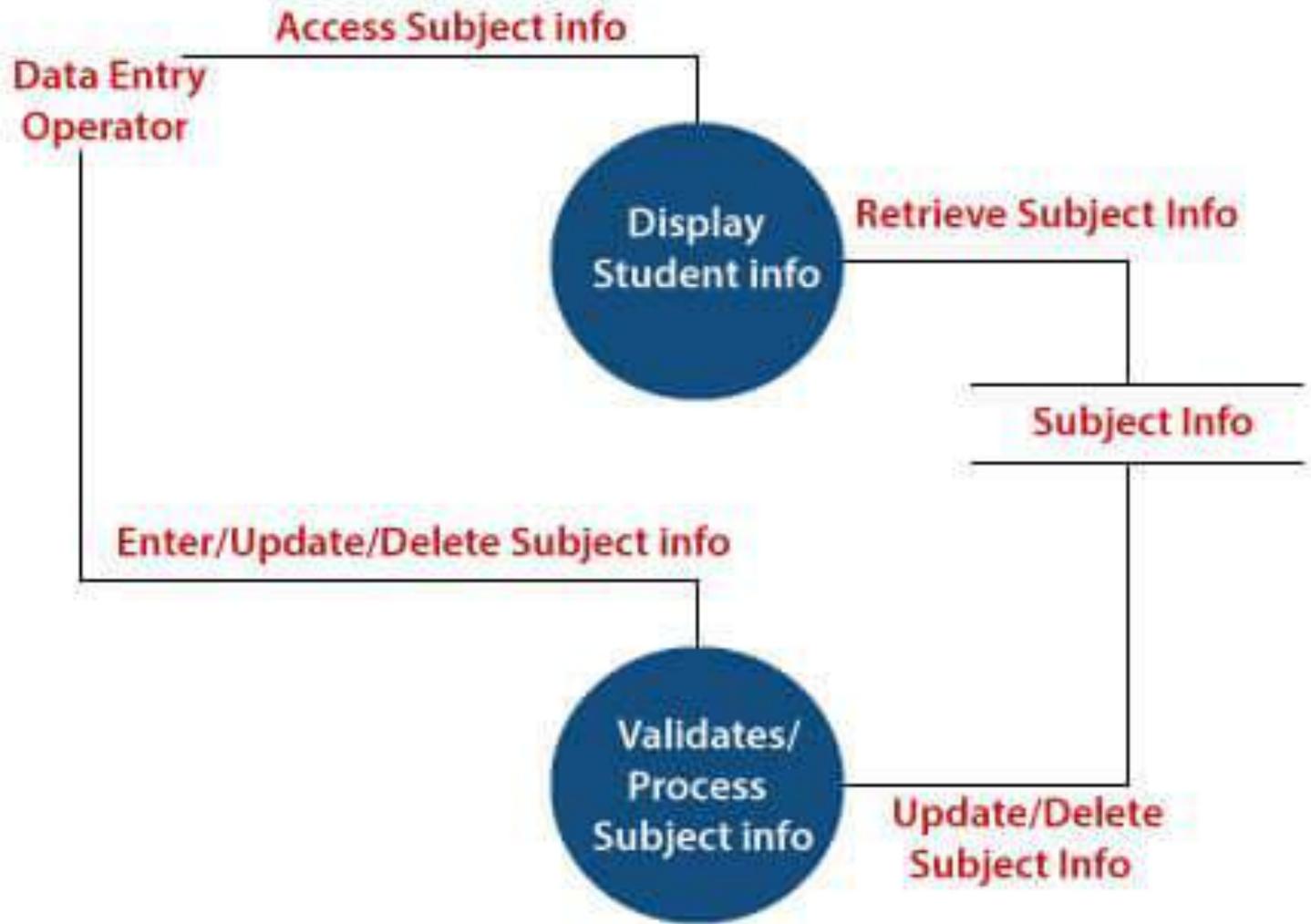


3. Student Information Management



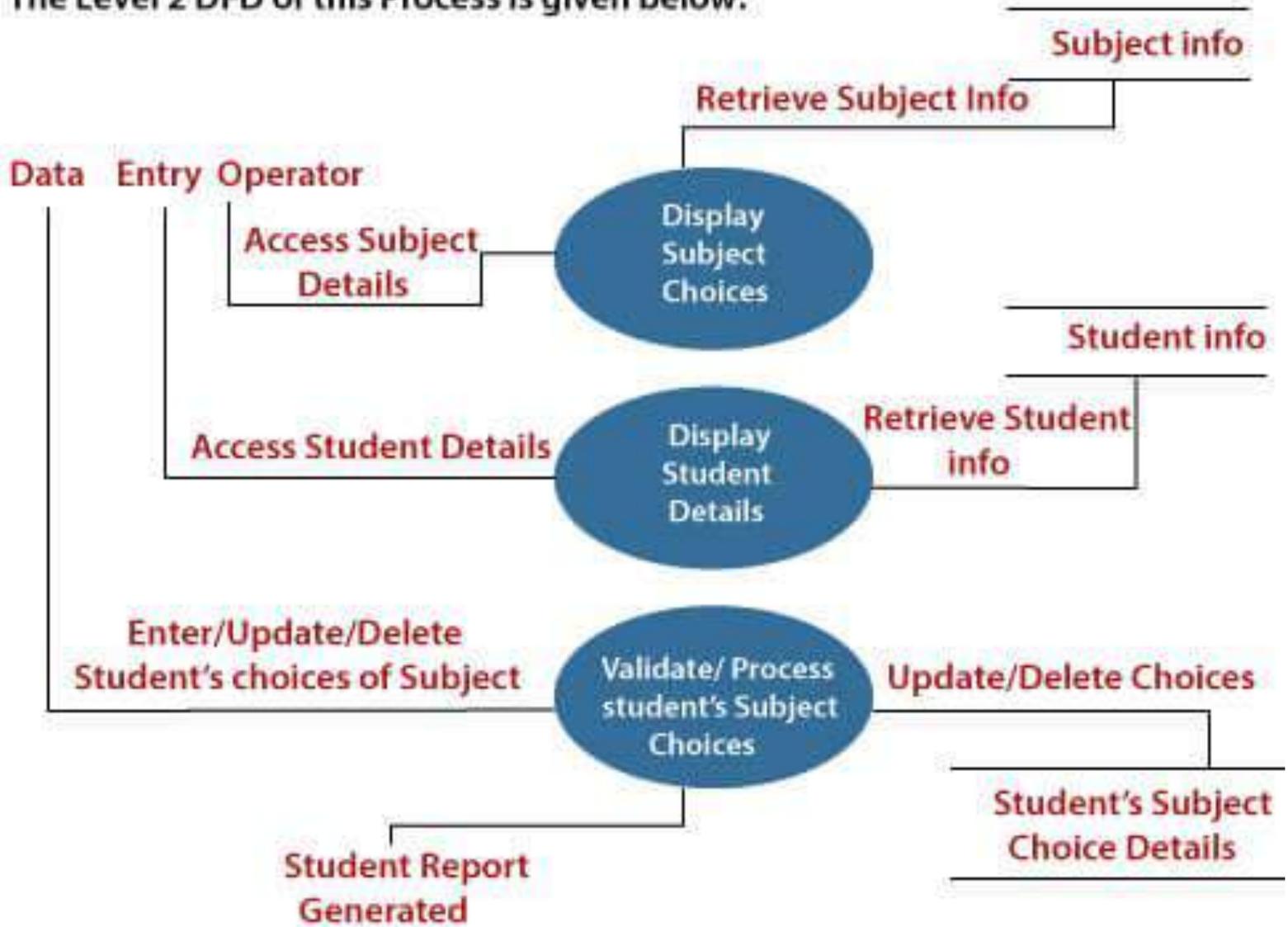
4. Subject Information Management

The level 2 DFD of this process is given below:



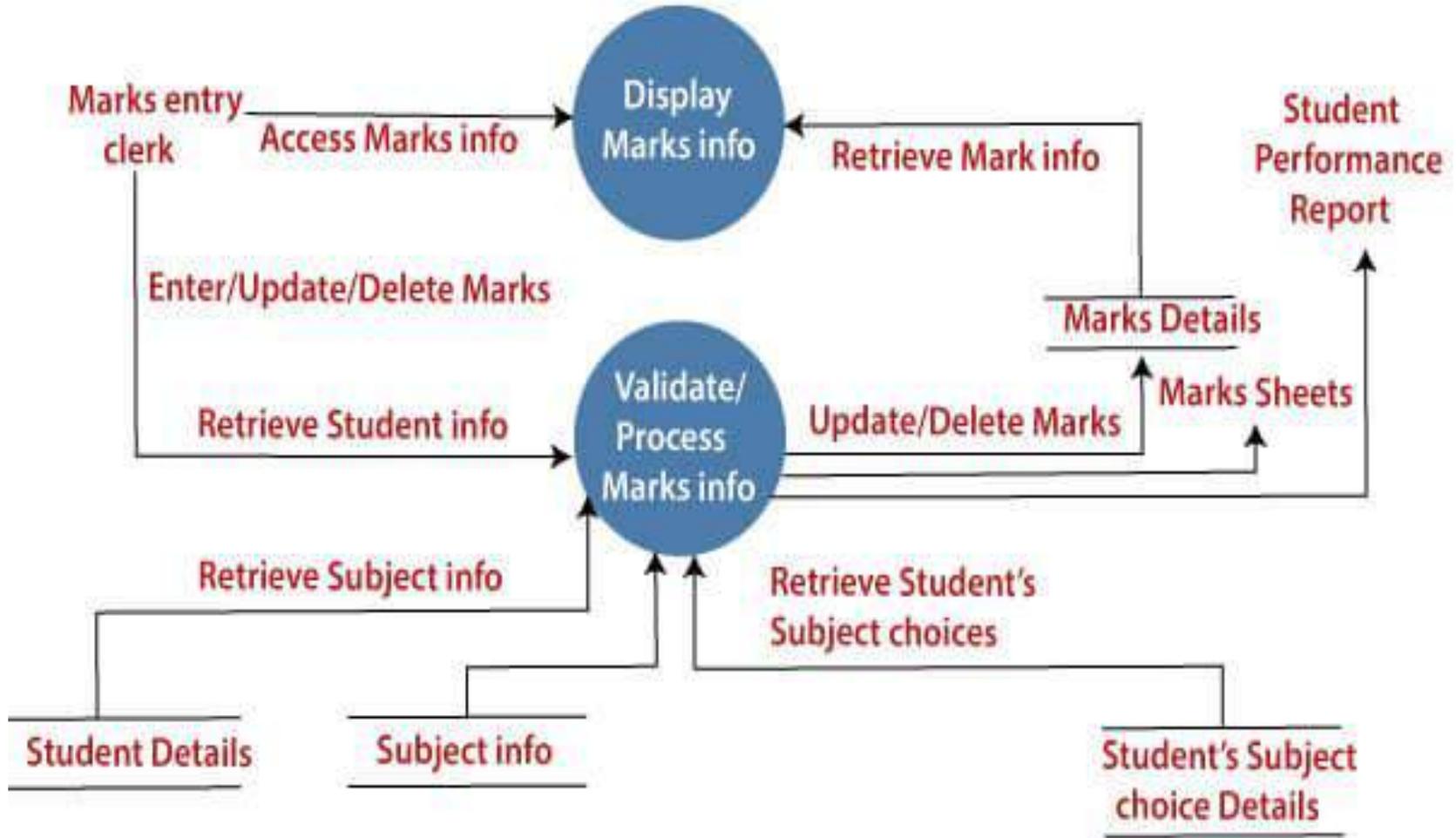
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



Data Dictionaries

- A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data.



Data Dictionaries

- The data dictionary is an essential component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

- The data dictionary, in general, includes information about the following:
 - Name of the data item
 - Aliases
 - Description/purpose
 - Related data items
 - Range of values
 - Data structure definition/Forms

- Aliases include other names by which this data item is called DEO for Data Entry Operator and DR for Deputy Registrar.
- Description/purpose is a textual description of what the data item is used for or why it exists.
- Related data items capture relationships between data items e.g., total_marks must always equal to internal_marks plus external_marks.

- Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.
- Data structure Forms: Data flows capture the name of processes that generate or receive the data items. If the data item is primitive, then data structure form captures the physical structures of the data item. If the data is itself a data aggregate, then data structure form capture the composition of the data items in terms of other data items.

Notations	Meaning
$x = a + b$	x includes of data elements a and b.
$x = [a/b]$	x includes of either data elements a or b.
$x = a^*$	includes of optimal data elements a.
$x = y[a]$	x includes of y or more occurrences of data element a
$x = [a]z$	x includes of z or fewer occurrences of data element a
$x = y[a]z$	x includes of some occurrences of data element a which are between y and z.



The advantages of using a data dictionary are:

- It is a valuable reference in any organization because it provides documentation.
- It improves the communication between system analyst and user by establishing consistent definitions of various items terms and procedures.
- It is a good tool for manage operators and other members of the development team to understand requirements and design.

The advantages of using a data dictionary are:

- It helps the analyst to simplify the structure for meeting the data requirements of the system.
- It is just like a store of all data elements information that can link all phases of software development life cycle.
- It is used to remove the redundancy in data definition.
- It is an important step building a database. Most data base management system has a data dictionary as a standard feature.
- During implementation, it serves as a base against which developers compare their data description.



Disadvantage of Data Dictionary

- There are some disadvantages given below:
- It does not provide functional details.
- It is not acceptable to many nontechnical users.

Input Design

- In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.
- Therefore, the quality of system input determines the quality of system output. Well designed input forms and screens have following properties –

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- .

- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding –
- What are the inputs needed for the system?
- How end users respond to different elements of forms and screens

Objectives for Input Design

- The objectives of input design are –
- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.



Output Design

- The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.



Objectives of Output Design

- The objectives of input design are –
- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end users requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

Structured Design

- Structured Design is a systematic methodology to determine design specification of software. The basic principles, tools and techniques of structured methodology are discussed in this chapter. It covers the four components of software design, namely, architectural design, detail design, data design and interface design. This chapter describes the following concepts, tools and techniques of structured design:

- Coupling and cohesion
- Structure chart
- Transaction analysis and transform analysis
- Program flowchart
- Structured flowchart
- HIPO documentation

Structure Chart

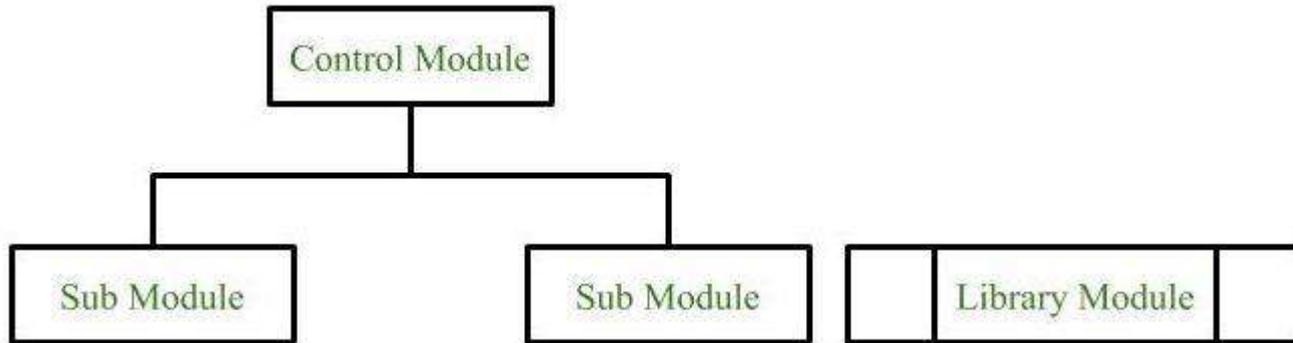
- Structure Chart represent hierarchical structure of modules. It breaks down the entire system into lowest functional modules, describe functions and sub-functions of each module of a system to a greater detail. Structure Chart partitions the system into black boxes (functionality of the system is known to the users but inner details are unknown). Inputs are given to the black boxes and appropriate outputs are generated.
- .

- Modules at top level called modules at low level. Components are read from top to bottom and left to right. When a module calls another, it views the called module as black box, passing required parameters and receiving results



Symbols used in construction of structured chart

- Module
- It represents the process or task of the system. It is of three types.
- Control Module
- A control module branches to more than one sub module.
- Sub Module
- Sub Module is a module which is the part (Child) of another module.
- Library Module
- Library Module are reusable and invocable from any module.





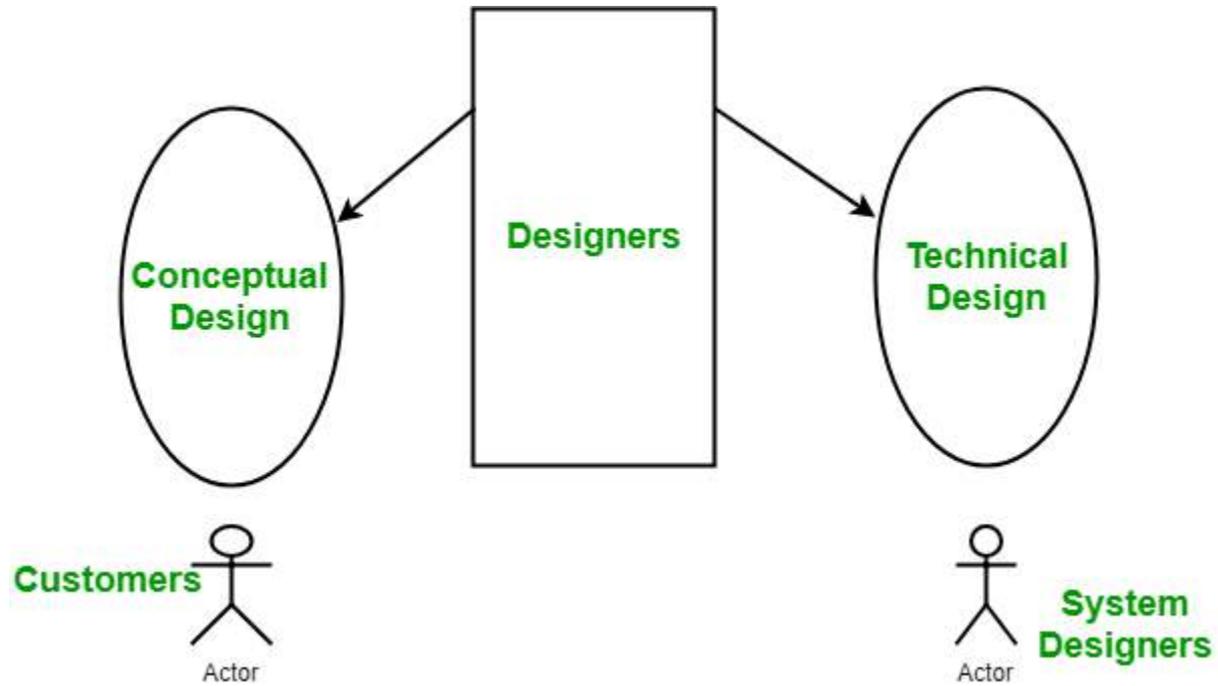
Software Engineering | Coupling and Cohesion

- Introduction: The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem given in the SRS (Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD)



Software Engineering | Coupling and Cohesion

- Basically, design is a two-part iterative process. First part is Conceptual Design that tells the customer what the system will do. Second is Technical Design that allows the system builders to understand the actual hardware and software needed to solve customer's problem.





Conceptual design of system:

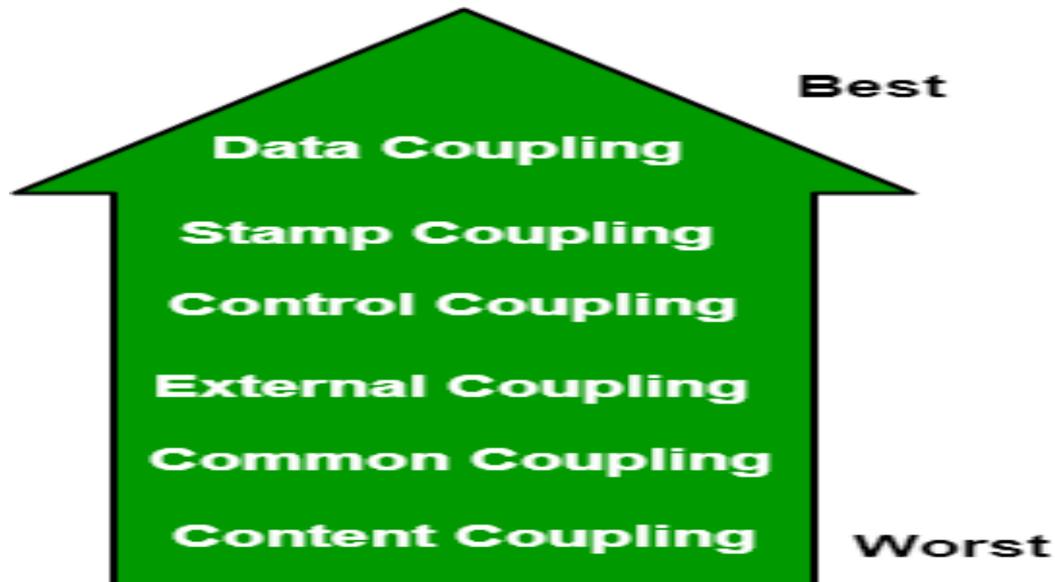
- Written in simple language i.e. customer understandable language.
- Detail explanation about system characteristics.
- Describes the functionality of the system.
- It is independent of implementation.
- Linked with requirement document.

Modularization

- **Modularization:** Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:
- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

Coupling

- **Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.





Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.

Stamp Coupling

- In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.

Control Coupling

- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.

External Coupling

- In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

Common Coupling

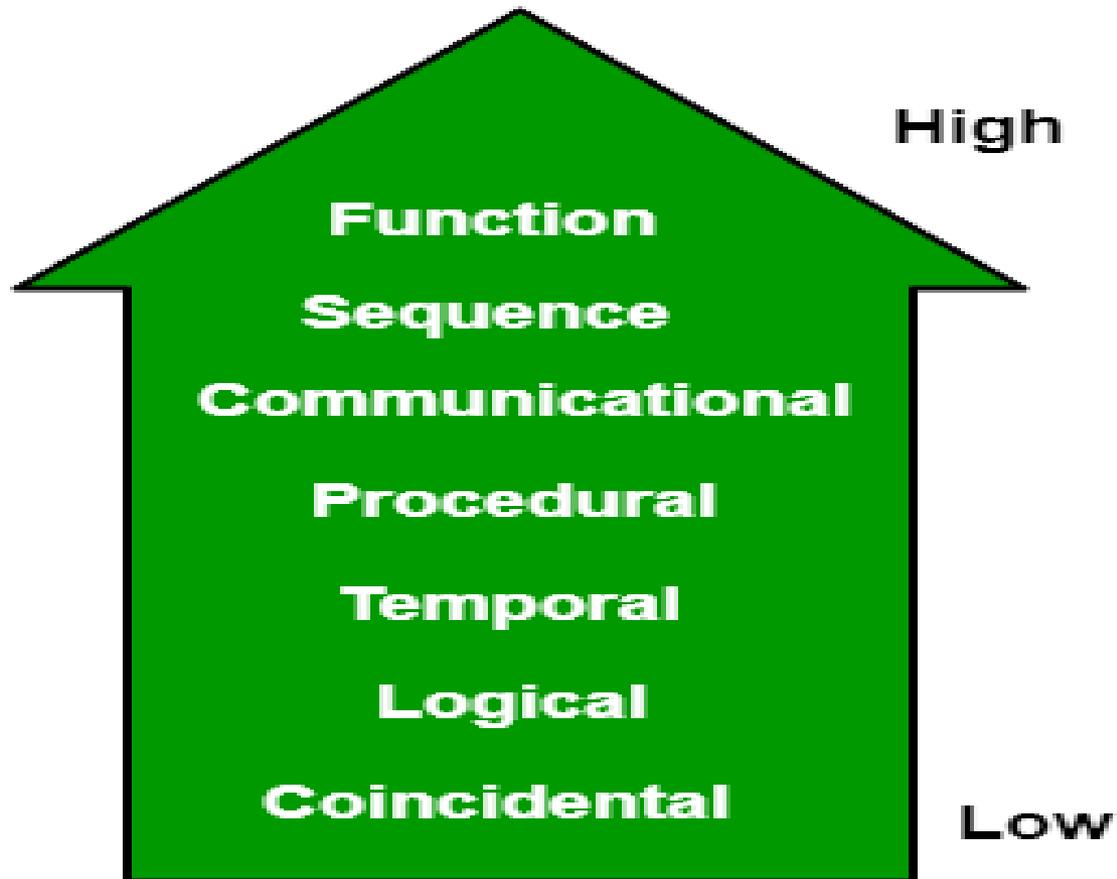
- The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.

Content Coupling

- In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion

- Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Cohesion

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record,

- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time.

- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.

- **Coincidental Cohesion:** The elements are not related (unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.



Unit 6 : Software Testing



What is Software Testing

- Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce a quality product.



Definition of Software Testing

- A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.



Testing Methods:

- Static Testing
- Dynamic Testing



Static Testing:

- It is also known as Verification in Software Testing.
- Verification is a static method of checking documents and files. Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.
- Activities involved here are Inspections, Reviews, Walkthroughs

Dynamic Testing:

- It is also known as Validation in Software Testing.
- Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.
- Activities involved in this is Testing the software application (Desktop application, Web application, Mobile Apps)

Type of Software testing

- We have various types of testing available in the market, which are used to test the application or the software.
- With the help of below image, we can easily understand the type of software testing:



Manual Testing

- Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.
- Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Automation testing

- Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

- There are different methods that can be used for software testing. This chapter briefly describes the methods available.



Black-Box Testing

- The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	T

White-Box Testing

- White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code.
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.
- The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.
Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	



Grey-Box Testing

- Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

- Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan

Advantages	Disadvantages
Offers combined benefits of black-box and white-box testing wherever possible.	Since the access to source code is not available, the ability to go over the code and test coverage is limited.
Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.	The tests can be redundant if the software designer has already run a test case.
Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.
The test is done from the point of view of the user and not the designer.	



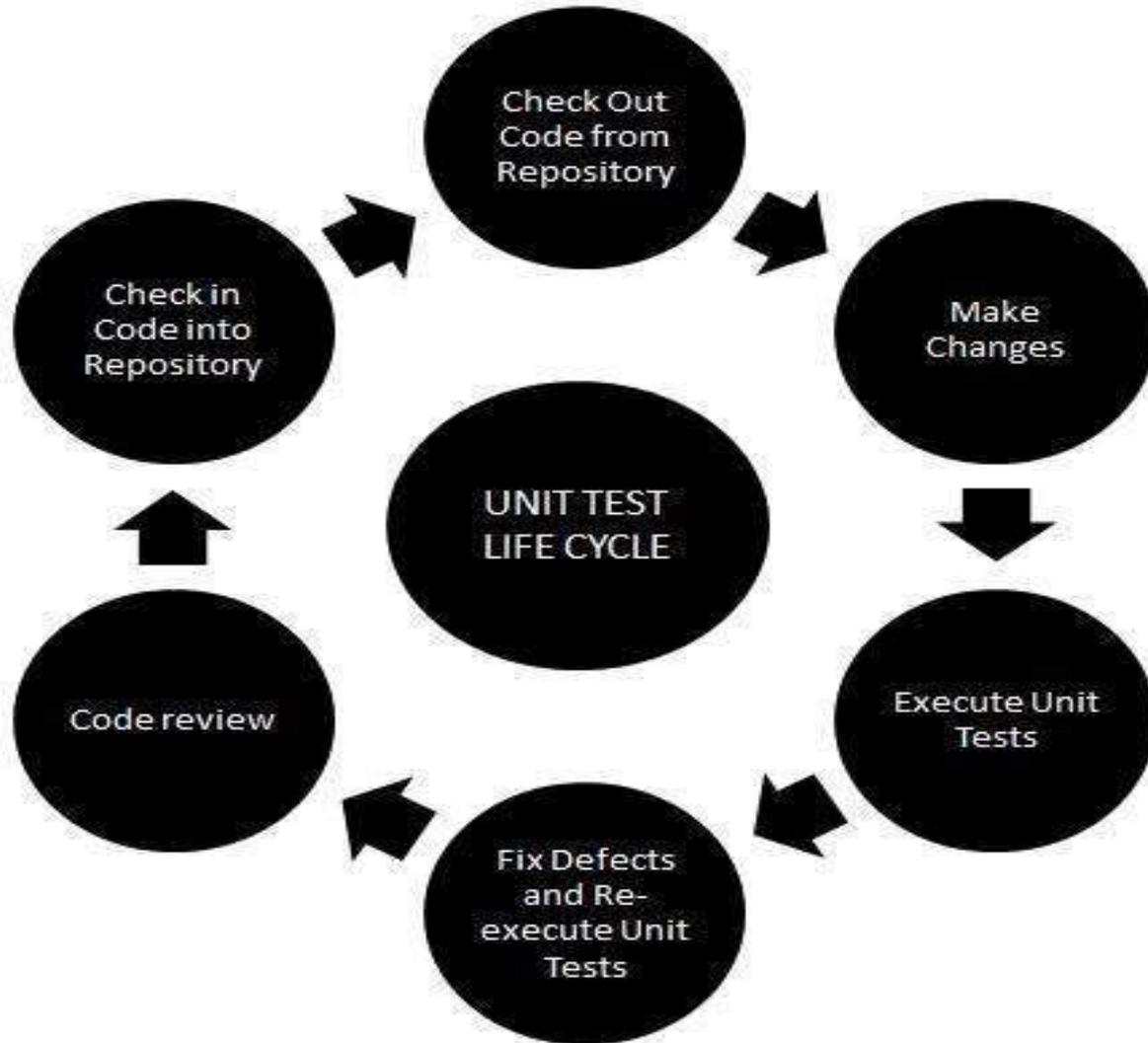
What is Unit Testing?

- Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.
- The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing - Advantages

- Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing LifeCycle:



Unit Testing Techniques:

- **Black Box Testing** - Using which the user interface, input and output are tested.
- **White Box Testing** - used to test each one of those functions behaviour is tested.
- **Gray Box Testing** - Used to execute tests, risks and assessment methods.

Integration testing

- Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration test approaches –

- I. Big-Bang Integration Testing –
- It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems.

- If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.



Advantages:

- It is convenient for small systems.
- Disadvantages:
 - There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
 - High risk critical modules are not isolated and tested on priority since all modules are tested at once.



Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up Integration Testing –

- In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.



Advantages:

- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.



Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

Mixed Integration Testing –

- A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested.

- In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. A mixed integration testing is also called sandwiched integration testing.

- Advantages:
- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

- Disadvantages:
- For mixed integration testing, require very high cost because one part has Top-down approach while another part has bottom-up approach.
- This integration testing cannot be used for smaller system with huge interdependence between different modules.



System Testing

- is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
- In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

- System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing
- System Testing is performed after the integration testing and before the acceptance testing.

Acceptance Testing



System Testing



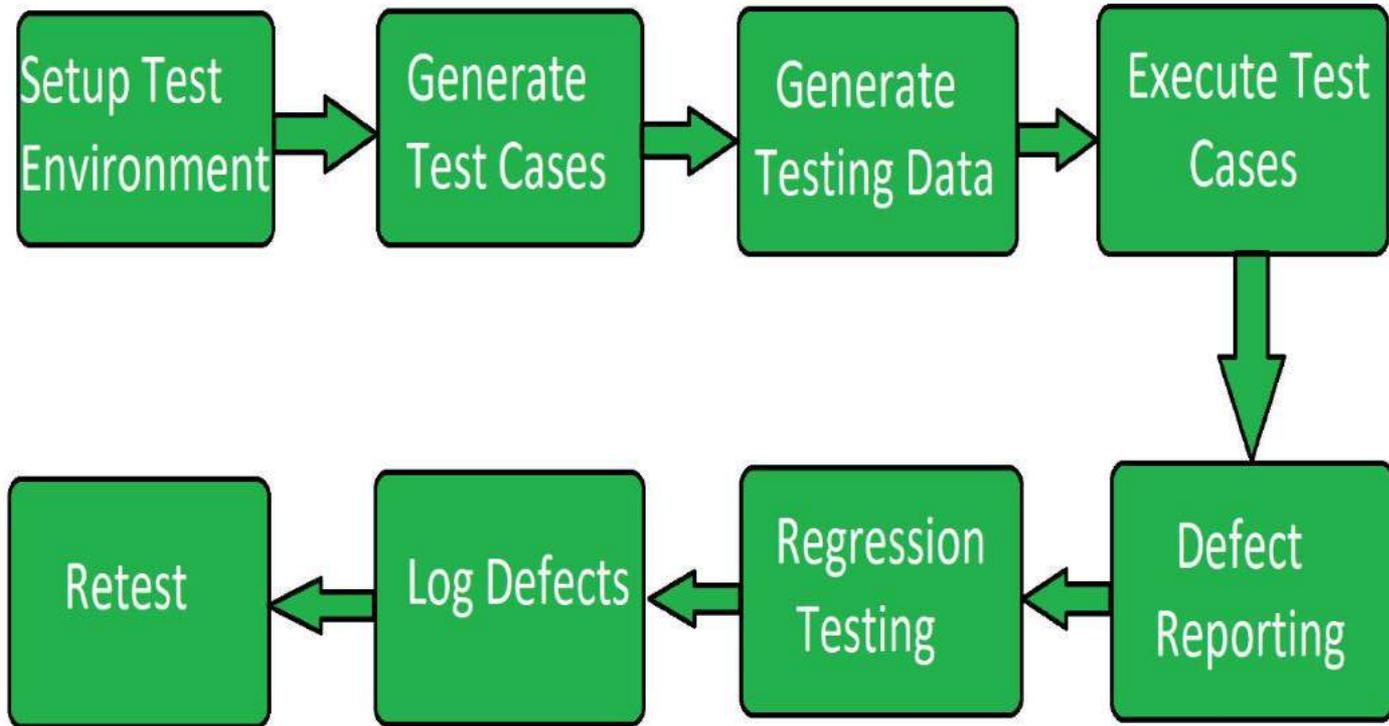
Integration Testing



Unit Testing

- System Testing Process:
- System Testing is performed in the following steps:
 - Test Environment Setup:
 - Create testing environment for the better quality testing.
 - Create Test Case:
 - Generate test case for the testing process.
 - Create Test Data:

- Generate the data that is to be tested.
- Execute Test Case:
- After the generation of the test case and the test data, test cases are executed.
- Defect Reporting:
- Defects in the system are detected.
- Regression Testing:
- It is carried out to test the side effects of the testing process.
- Log Defects:
- Defects are fixed in this step.
- Retest:
- If the test is not successful then again test is performed.



- **Types of System Testing:**
- **Performance Testing:**
- Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:**

- Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- Stress Testing:
 - Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- Scalability Testing:
 - Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

- System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS).
- .



**UNIT 7: SOFTWARE
MAINTENANCE AND
SOFTWARE RE-
ENGINEERING**

Software Maintenance

- Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.



Need for Maintenance

- Software Maintenance must be performed in order to:
 - Correct faults.
 - Improve the design.
 - Implement enhancements.
 - Interface with other systems.
 - Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
 - Migrate legacy software.
 - Retire software.

Categories of Software

Maintenance –

- Corrective maintenance:
- Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.
- Adaptive maintenance:
- This includes modifications and updates when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

- Perfective maintenance:
- A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.
- Preventive maintenance:
- This type of maintenance includes modifications and updations to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.



Reverse Engineering –

- **Reverse Engineering –**
Reverse Engineering is processes of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. It is also called back Engineering.

Software Reverse Engineering

- **Software Reverse Engineering –**
Software Reverse Engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code. Reverse Engineering is becoming important, since several existing software products, lack proper documentation, are highly unstructured, or their structure has degraded through a series of maintenance efforts.



Why Reverse Engineering?

- **Why Reverse Engineering?**
- Providing proper system documentatiuon.
- Recovery of lost information.
- Assisting with maintenance.
- Facility of software reuse.
- Discovering unexpected flaws or faults.

- Used of Software Reverse Engineering –
- Software Reverse Engineering is used in software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code.
- Reverse engineering is also useful in software testing, it helps the testers to study the virus and other malware code .

Software Re-engineering

- Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

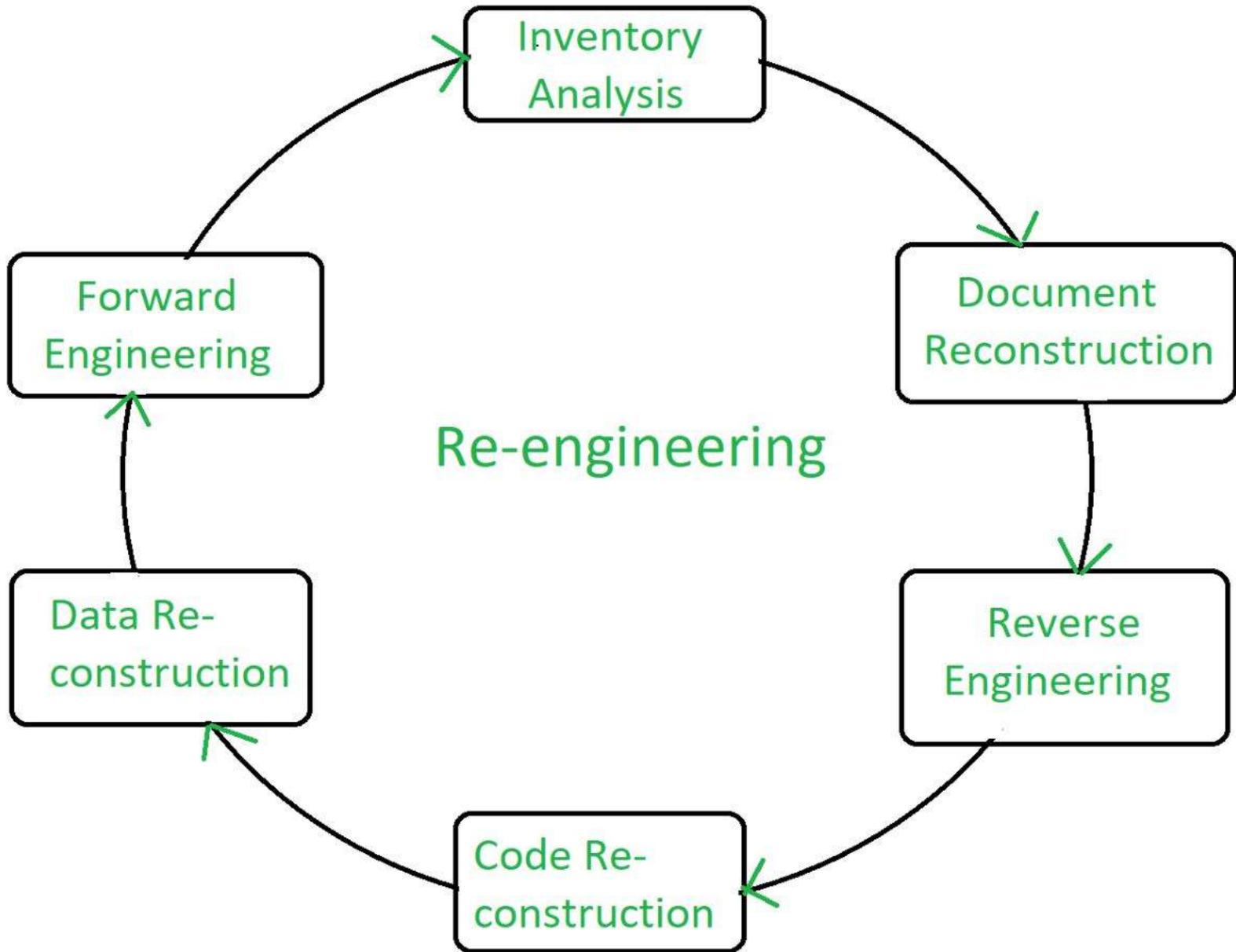
Objectives of Re-engineering:

- Objectives of Re-engineering:
- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.



Steps involved in Re-engineering:

- **Steps involved in Re-engineering:**
- Inventory Analysis
- Document Reconstruction
- Reverse Engineering
- Code Reconstruction
- Data Reconstruction
- Forward Engineering



- Re-engineering Cost Factors:
- The quality of the software to be re-engineered
- The tool support available for re-engineering
- The extent of the required data conversion
- The availability of expert staff for re-engineering

Advantages of Re-engineering

- Reduced Risk:
- As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems which may arise in new software development.
- Reduced Cost:
- The cost of re-engineering is less than the costs of developing new software.



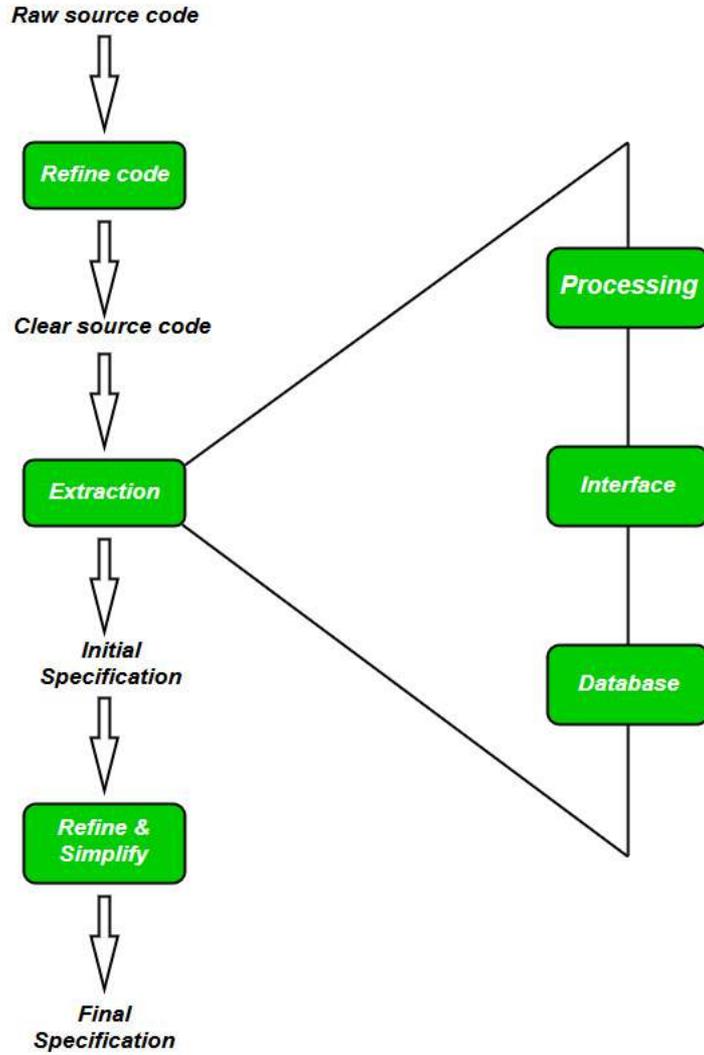
Reverse Engineering

- Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.



Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



- **Collection Information:**
- This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.
- **Examining the information:**
- The information collected in step-1 as studied so as to get familiar with the system.

- **Extracting the structure:**
- This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.
- **Recording the functionality:**
- During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

- **Recording data flow:**
- From the information extracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.
- **Recording control flow:**
- High level control structure of the software is recorded.

- **Review extracted design:**
- Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.
- **Generate documentation:**
- Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use



Forward Engineering:

- Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering is required high proficiency skill. It takes more time to construct or develop an application.

**System
Specification**

**Design and
Implementation**

New System

Forward Engineering

- In forward engineering, the application are developed with the given requirements.
- Forward Engineering is high proficiency skill.
- Forward Engineering takes more time to develop an application.
- The nature of forward engineering is Prescriptive.

- In forward engineering, production is started with given requirements.
- The example of forward engineering are construction of electronic kit, construction of DC MOTOR etc.



- Thank you



