

Unit-II

Software Requirements Engineering

Introduction to requirement engineering

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follow:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Elicitation

Elicitation means to find the requirements from anybody.

The requirements are difficult because the **following problems occur in elicitation.**

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time

and it is difficult while developing the project.

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.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.

- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

Eliciting Requirements

Eliciting requirement helps the user for collecting the requirement

Eliciting requirement steps are as follows:

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.

QFD consist of three types of requirement:

Normal requirements

- The objective and goal are stated for the system through the meetings with the customer.
- For the customer satisfaction these requirements should be there.

Expected requirement

- These requirements are implicit.
- These are the basic requirement that not be clearly told by the customer, but also the customer expect that requirement.

Exciting requirements

- These features are beyond the expectation of the customer.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.

For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.

Software Requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

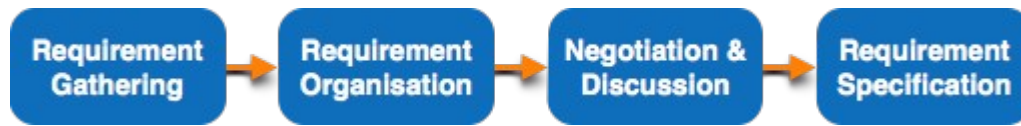
Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the following diagram:



- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.

- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at

client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.

- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negotiation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate

- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategic use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement

- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity
- Finalizing acceptance criteria with client and other stakeholders

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), “You cannot control what you cannot measure.” By his saying, it is very clear how important software measures are.

Let us see some software metrics:

- **Size Metrics** - LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics** - McCabe’s Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
- **Quality Metrics** - Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics** - In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics** - Effort, time and various resources used, represents metrics for resource measurement.

Requirements Elicitation

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

1. Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.

Interviews may be open ended or structured.

1. In open ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views

- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified –

- **Normal requirements** – In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements** – These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements** – It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorized as –
 - It is possible to achieve
 - It should be deferred and the reason for it

- It is impossible to achieve and should be dropped off

5. Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system.

The components of the use case design include three major things – Actor, Use cases, use case diagram.

1. **Actor** – It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
 - Primary actors – It requires assistance from the system to achieve a goal.
 - Secondary actor – It is an actor from which the system needs assistance.
2. **Use cases** – They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. **Use case diagram** – A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
 - A stick figure is used to represent an actor.
 - An oval is used to represent a use case.
 - A line is used to represent a relationship between an actor and a use case.

Requirement Engineering

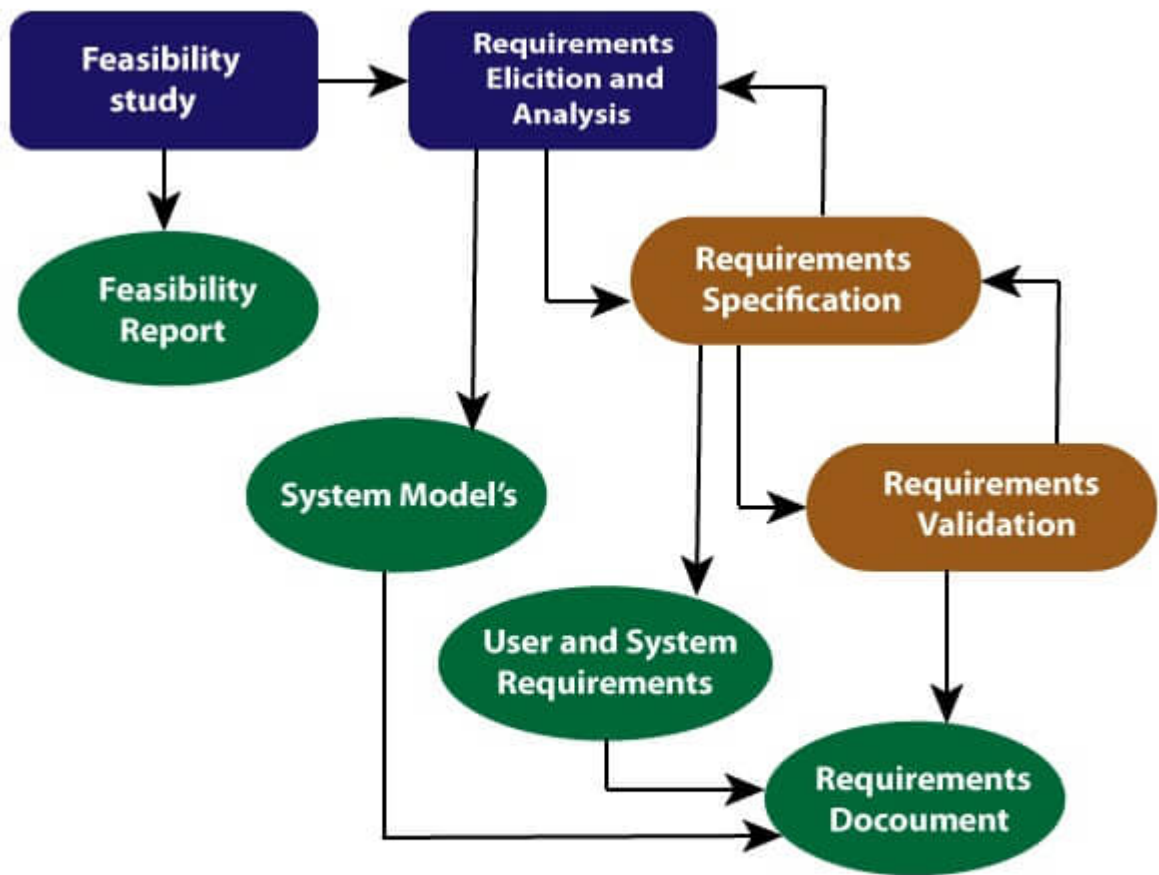
Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating 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 -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



Requirement Engineering Process

1. 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:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

2. 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

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

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 -

- 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.

- o **Test-case generation:** Developing tests for requirements to check testability.
- o **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.

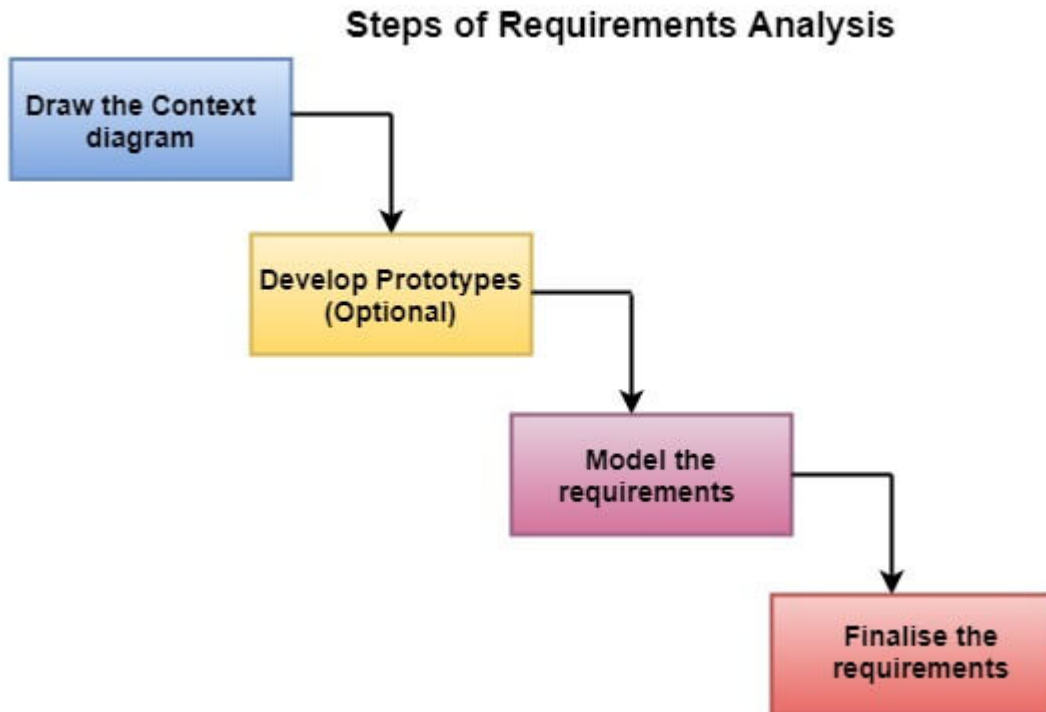
The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

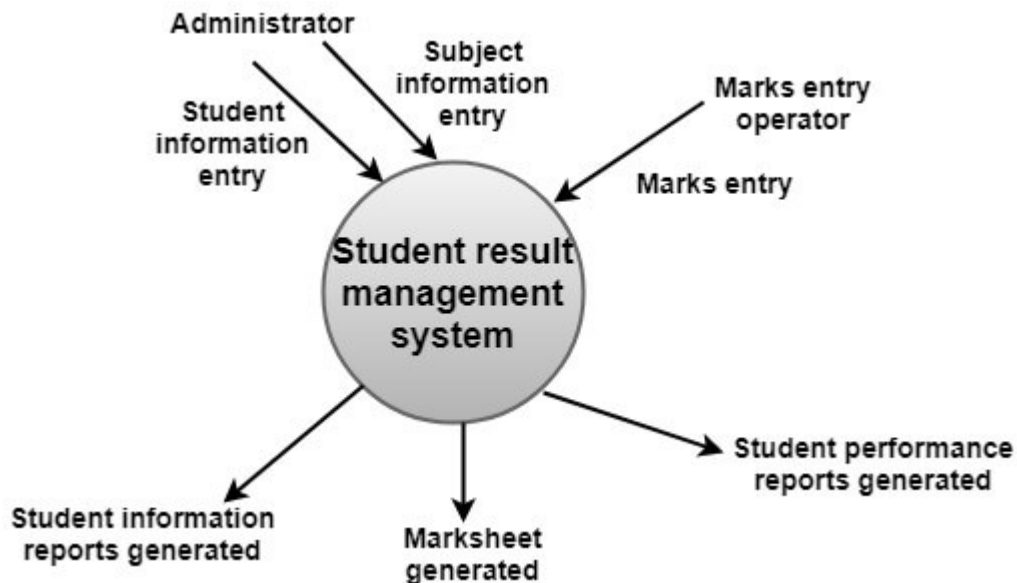
Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

Analysis Modeling in Software Engineering

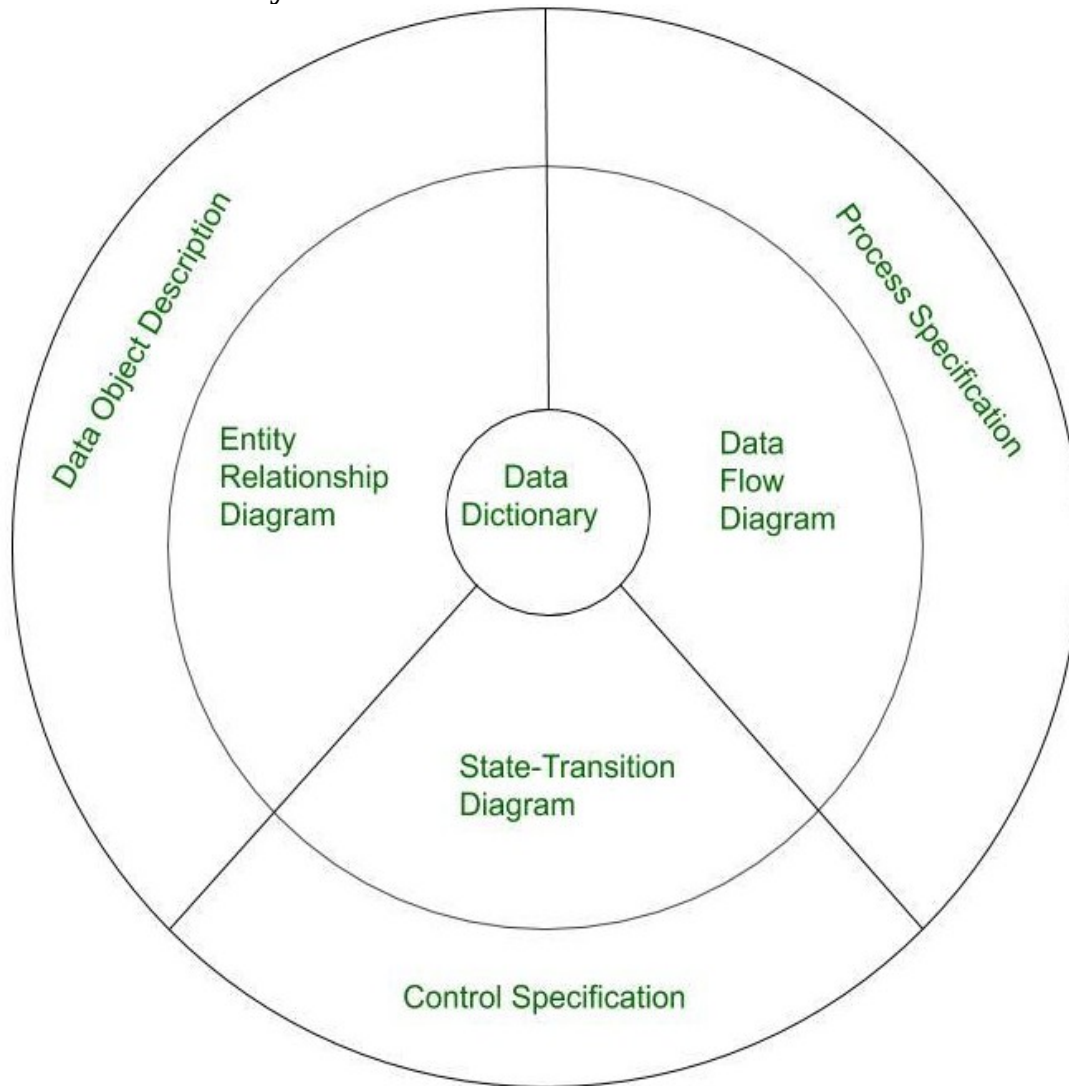
Analysis Model is a technical representation of the system. It acts as a link between system description and design model. In Analysis Modeling, information, behavior and functions of the system is defined and translated into the architecture, component and interface level design in the design modeling.

Objectives of Analysis Modeling:

1. It must establish a way of creation of software design.

2. It must describe requirements of customer.
3. It must define set of requirements which can be validated, once the software is built.

Elements of Analysis Model:



1. Data Dictionary:

It is a repository that consists of description of all data objects used or produced by software. It stores the collection of data present in the software. It is a very crucial element of the analysis model. It acts as a centralized repository and also helps in modeling of data objects defined during software requirements.

2. **Entity Relationship Diagram (ERD):**

It depicts relationship between data objects and used in conducting of data modeling activity. The attributes of each object in the Entity Relationship Diagram can be described using Data object description. It provides the basis for activity related to data design.

3. **Data Flow Diagram (DFD):**

It depicts the functions that transform data flow and it also shows how data is transformed when moving from input to output. It provides the additional information which is used during the analysis of information domain and serves as a basis for the modeling of function. It also enables the engineer to develop models of functional and information domain at the same time.

4. **State Transition Diagram:**

It shows various modes of behavior (states) of the system and also shows the transitions from one state to other state in the system. It also provides the details of how system behaves due to the consequences of external events. It represents the behavior of a system by presenting its states and the events that cause the system to change state. It also describes what actions are taken due to the occurrence of a particular event.

5. **Process Specification:**

It stores the description of each functions present in the data flow diagram. It describes the input to a function, the algorithm that is applied for transformation of input, and the output that is produced. It also shows regulations and barriers imposed on the performance characteristics that are applicable to the process, and layout constraints that could influence the way in which the process will be implemented.

6. **Control Specification:**

It stores the additional information about the control aspects of the software. It is used to indicate how the software behaves when an event occurs and which processes are invoked due to the occurrence of the event. It also provides the details of the processes which are executed to manage events.

7. **Data Object Description:**

It stores and provides the complete knowledge about a data object present and

used in the software. It also gives us the details of attributes of the data object present in Entity Relationship Diagram. Hence, it incorporates all the data objects and its attributes.

Data Modeling: Conceptual, Logical, Physical Data Model Types

What is Data Modeling?

Data modeling (data modeling) is the process of creating a data model for the data to be stored in a database. This data model is a conceptual representation of Data objects, the associations between different data objects, and the rules. Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

Data Model

The **Data Model** is defined as an abstract model that organizes data description, data semantics, and consistency constraints of data. The data model emphasizes on what data is needed and how it should be organized instead of what operations will be performed on data. Data Model is like an architect's building plan, which helps to build conceptual models and set a relationship between data items.

The two types of Data Modeling Techniques are

1. Entity Relationship (E-R) Model
2. UML (Unified Modeling Language)

We will discuss them in detail later.

This Data Modeling Tutorial is best suited for fresher's, beginners as well as experienced professionals. In this data model tutorial, data modeling concepts in detail-

- **Why use Data Model?**
- **Types of Data Models**
 - **Conceptual Data Model**
 - **Logical Data Model**
 - **Physical Data Model**
- **Advantages and Disadvantages of Data Model**

Why use Data Model?

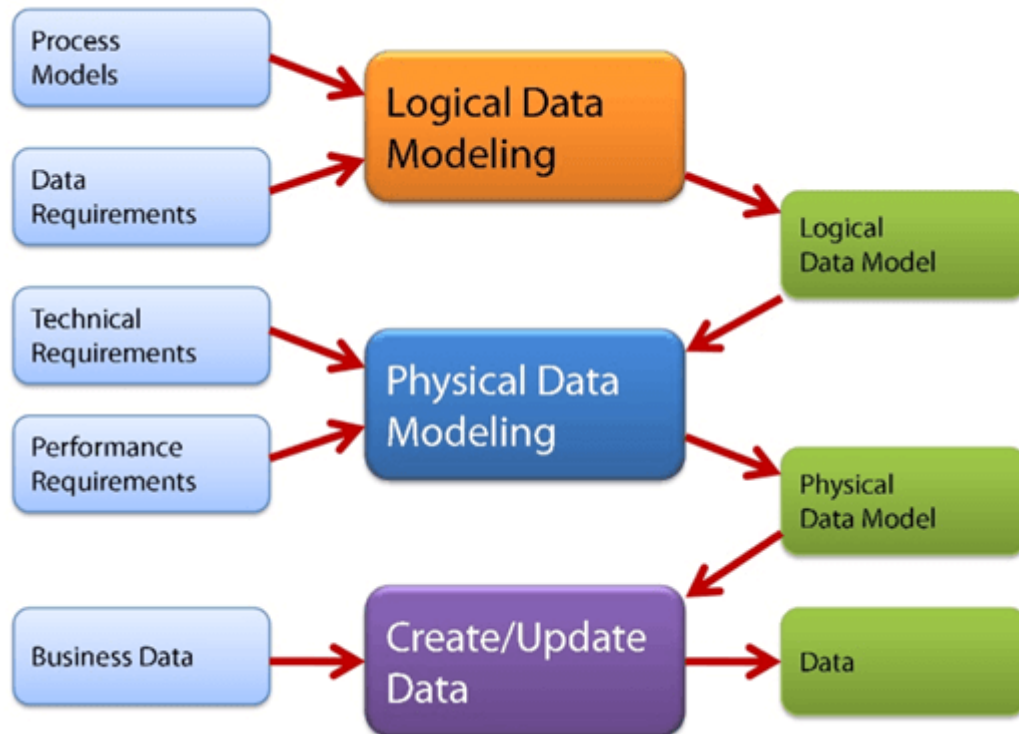
The primary goal of using data model is:

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.
- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

Types of Data Models

Types of Data Models: There are mainly three different types of data models: conceptual data models, logical data models, and physical data models, and each one has a specific purpose. The data models are used to represent the data and how it is stored in the database and to set the relationship between data items.

1. **Conceptual Data Model:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data



es Architects. The purpose is to organize scope and define business concepts and rules.

2. **Logical Data Model:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical Data Model:** This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.
4. of Data Model

Conceptual Data Model

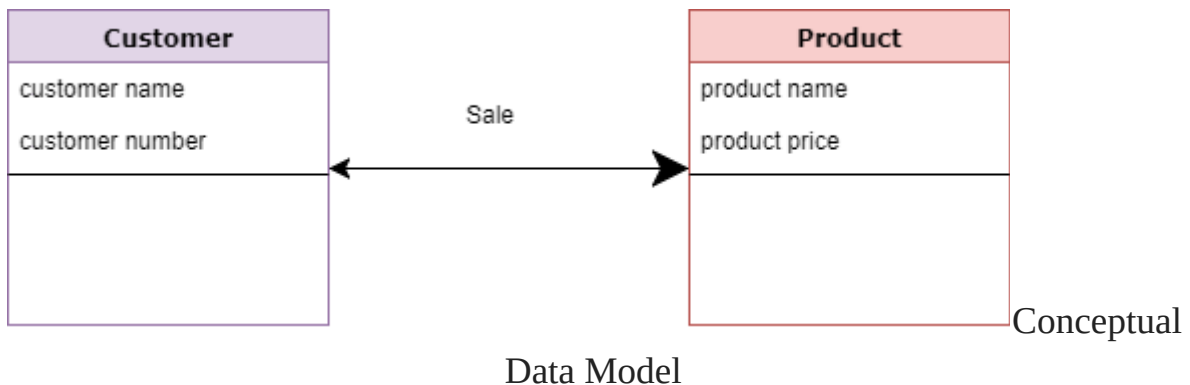
A **Conceptual Data Model** is an organized view of database concepts and their relationships. The purpose of creating a conceptual data model is to establish entities, their attributes, and relationships. In this data modeling level, there is hardly any detail available on the actual database structure. Business stakeholders and data architects typically create a conceptual data model.

The 3 basic tenants of Conceptual Data Model are

- **Entity:** A real-world thing
- **Attribute:** Characteristics or properties of an entity
- **Relationship:** Dependency or association between two entities

Data model example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity
- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product



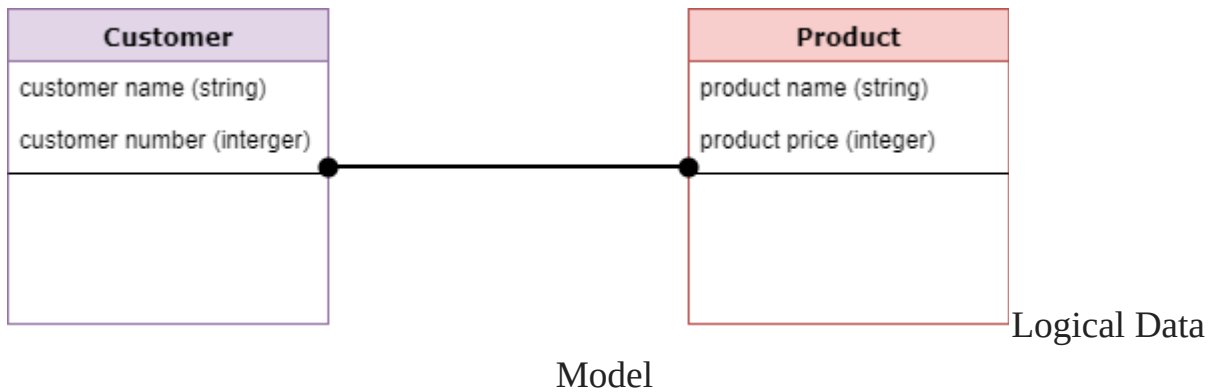
Characteristics of a conceptual data model

- Offers Organization-wide coverage of the business concepts.
- This type of Data Models are designed and developed for a business audience.
- The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

Logical Data Model

The **Logical Data Model** is used to define the structure of data elements and to set relationships between them. The logical data model adds further information to the conceptual data model elements. The advantage of using a Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.



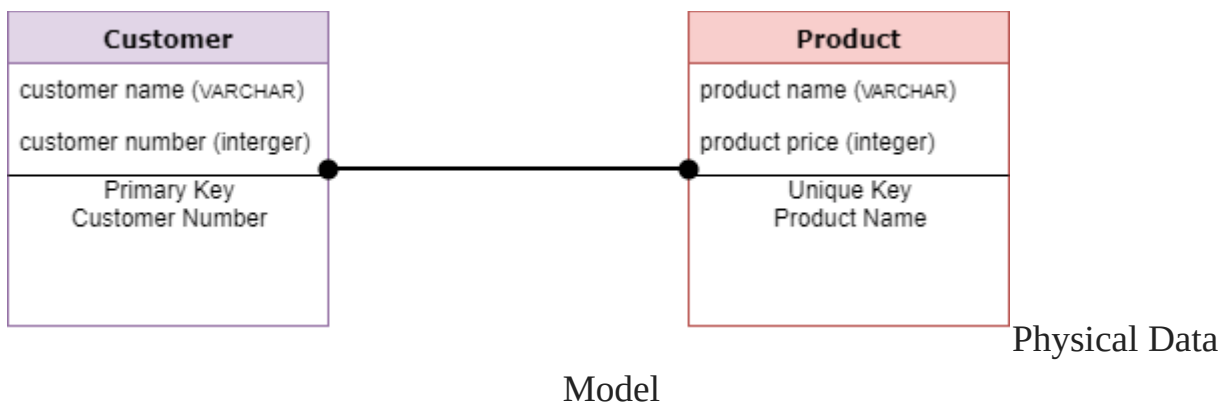
At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

Characteristics of a Logical data model

- Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
- Designed and developed independently from the DBMS.
- Data attributes will have datatypes with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

Physical Data Model

A **Physical Data Model** describes a database-specific implementation of the data model. It offers database abstraction and helps generate the schema. This is because of the richness of meta-data offered by a Physical Data Model. The physical data model also helps in visualizing database structure by replicating database column keys, constraints, indexes, triggers, and other RDBMS features.



Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it may be integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and null ability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact data types, lengths assigned and default values.
- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

Advantages and Disadvantages of Data Model:

Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.
- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

Disadvantages of Data model:

- To develop Data model one should know physical data stored characteristics.
- This is a navigational system produces complex application development, management. Thus, it requires knowledge of the biographical truth.
- Even smaller change made in structure requires modification in the entire application.
- There is no set data manipulation language in DBMS.

Conclusion

- Data modeling is the process of developing data model for the data to be stored in a Database.
- Data Models ensure consistency in naming conventions, default values, semantics, and security while ensuring quality of the data.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- There are three types of conceptual, logical, and physical.
- The main aim of conceptual model is to establish the entities, their attributes, and their relationships.
- Logical data model defines the structure of the data elements and set the relationships between them.
- A Physical Data Model describes the database specific implementation of the data model.
- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The biggest drawback is that even smaller change made in structure requires modification in the entire application.
- Reading this Data Modeling tutorial, you will learn from the basic concepts such as What is Data Model? Introduction to different types of Data Model, advantages, disadvantages, and data model example.

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. 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:

- o Name of the data item
- o Aliases
- o Description/purpose

- o Related data items
- o Range of values
- o Data structure definition/Forms

The **name of the data item** is self-explanatory.

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.

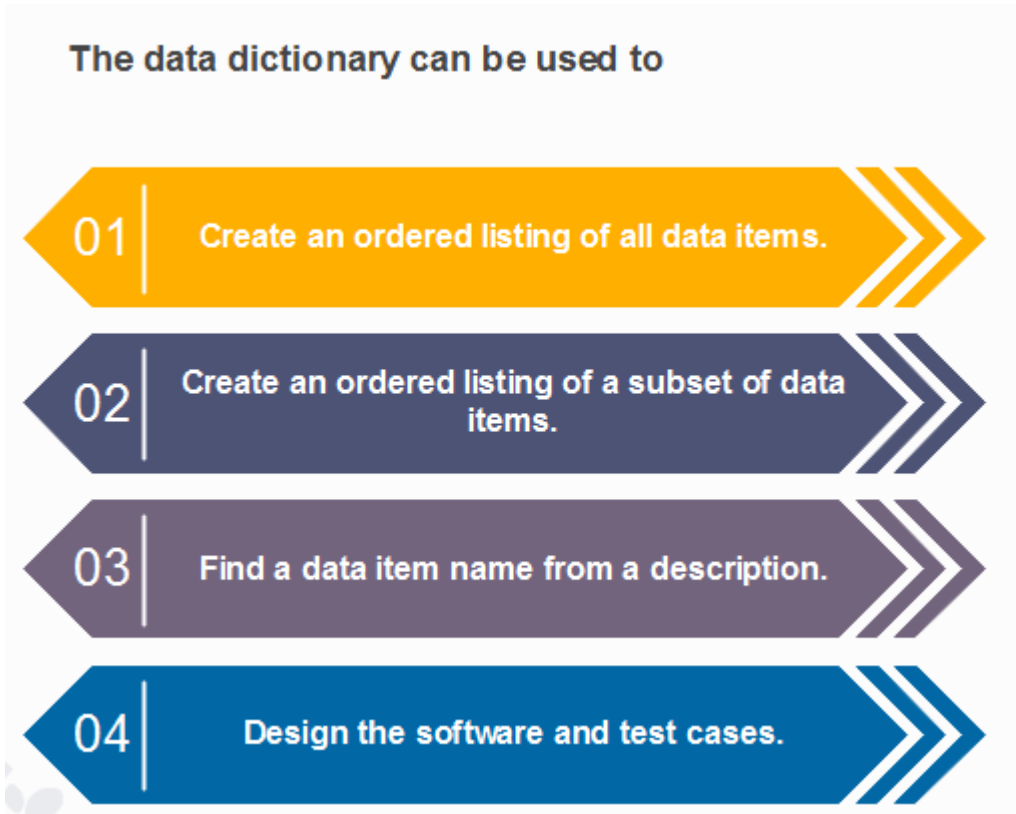
Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.

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.

The mathematical operators used within the data dictionary are defined in the table:

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.
$x=a \ x$	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



Introduction flow oriented modeling in software engineering:

Although stream oriented model is perceived as the obsolete method by some software engineers, it remains to remain one of the most widely applied requirements analysis notations being used today. Flow oriented model focuses on integrated analysis and design, is the place to down method and uses a graphic technique describing data flows and the transformations that are used as information moves from input to output.

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.


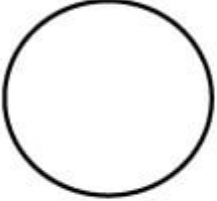

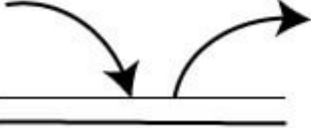
It shows how data enters and leaves the system, what changes the information, and where data is stored.

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 chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows in a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. 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 exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

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 or 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 fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y , then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

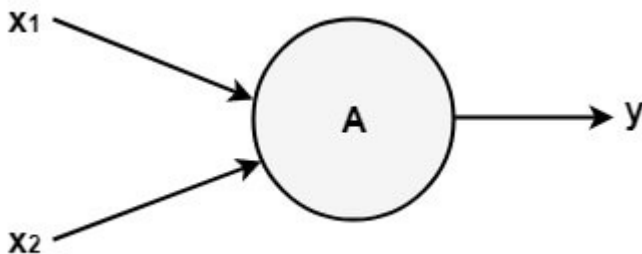


Fig: Level-0 DFD.

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.

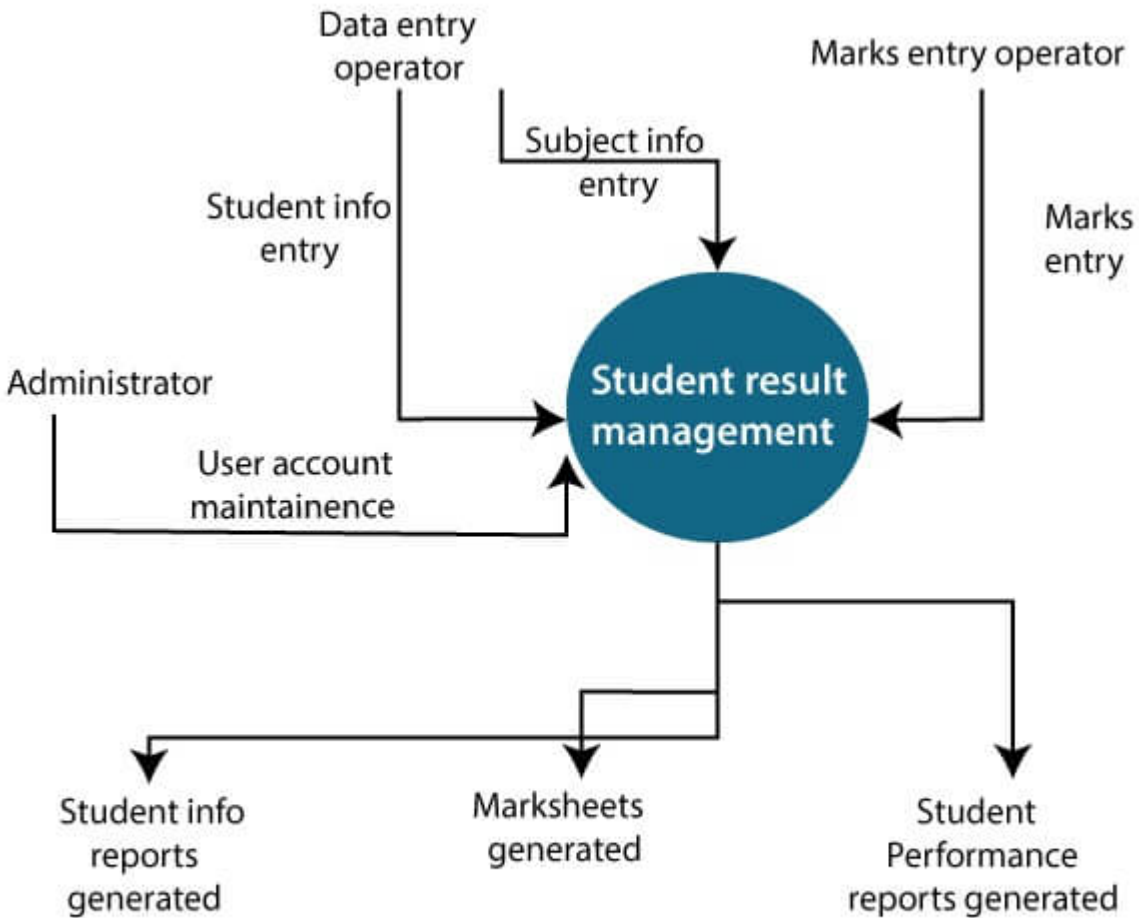


Fig: Level-0 DFD of result management system

1-level DFD

In 1-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 sub processes.

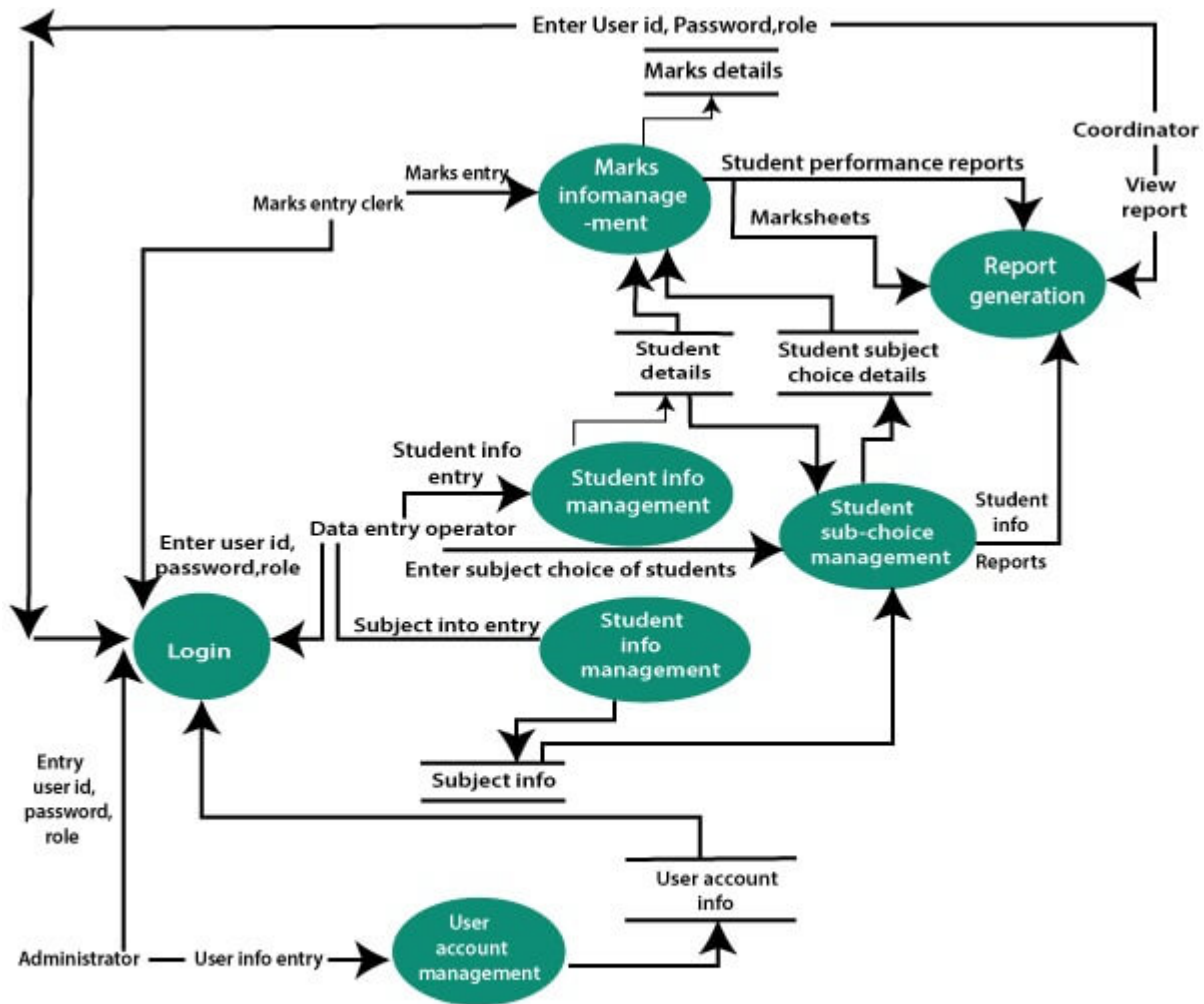
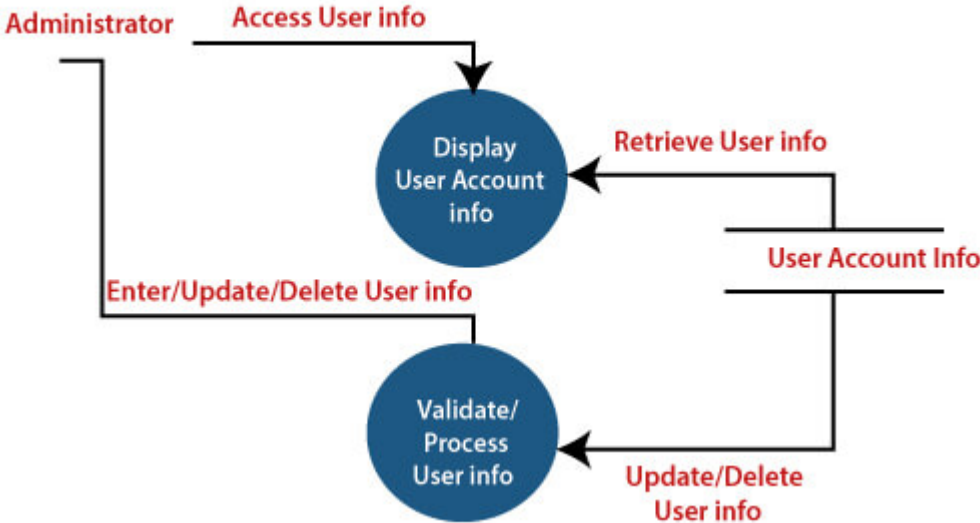


Fig: Level-1 DFD of result management system

2-Level DFD

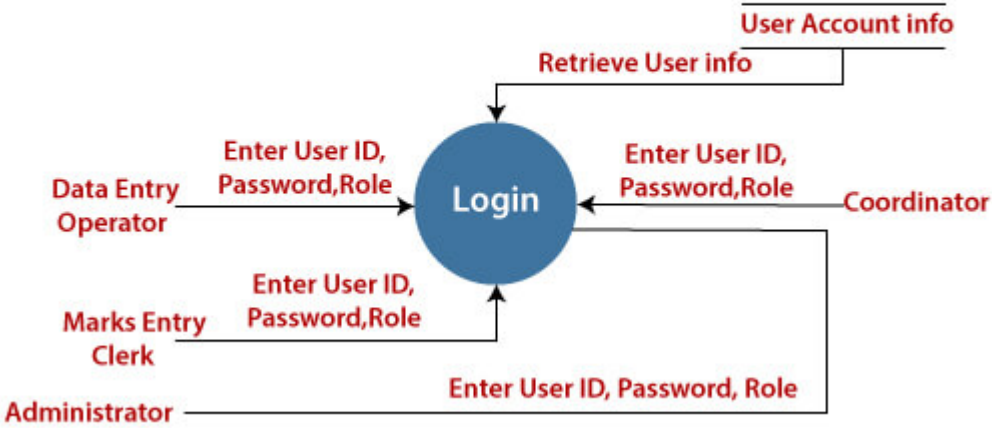
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

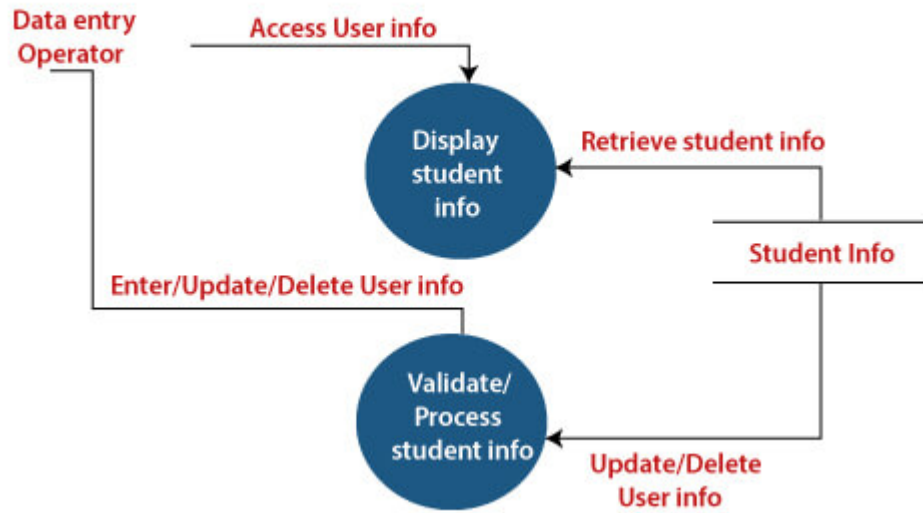


2. Login

The level 2 DFD of this process is given below:

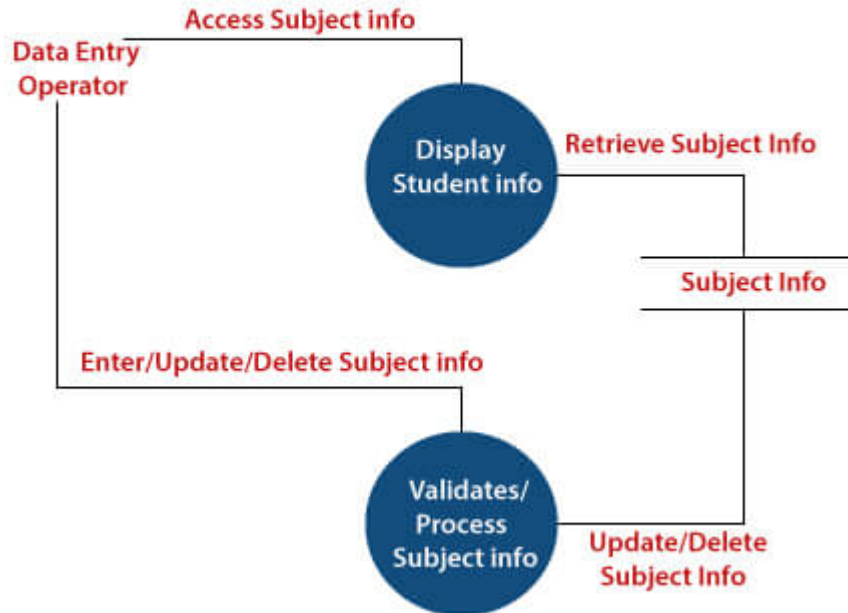


3. Student Information Management



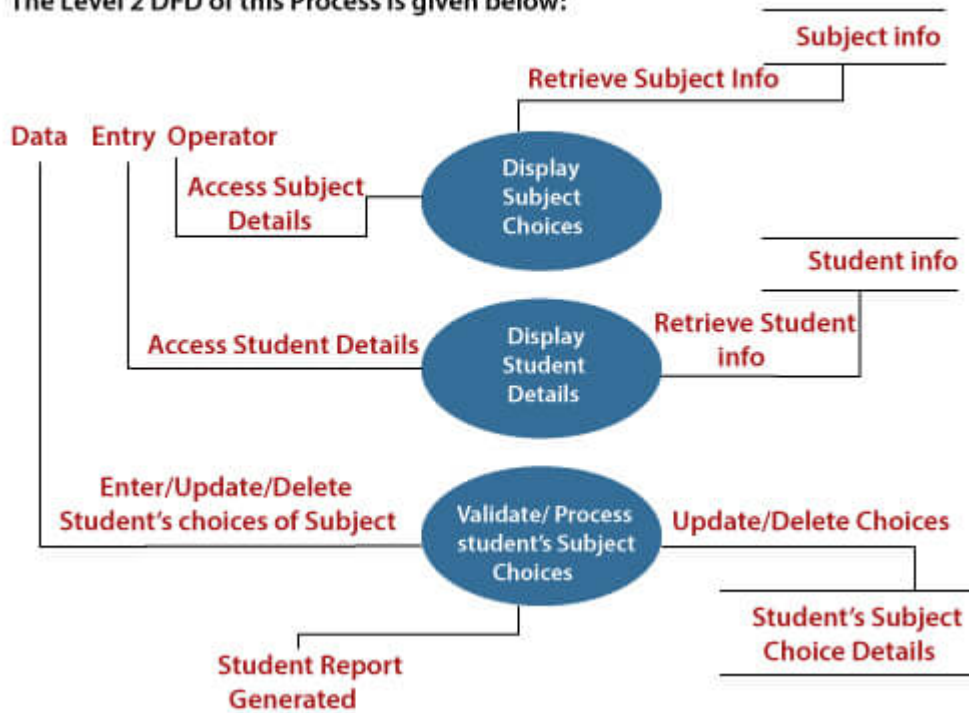
4. Subject Information Management

The level 2 DFD of this process is given below:



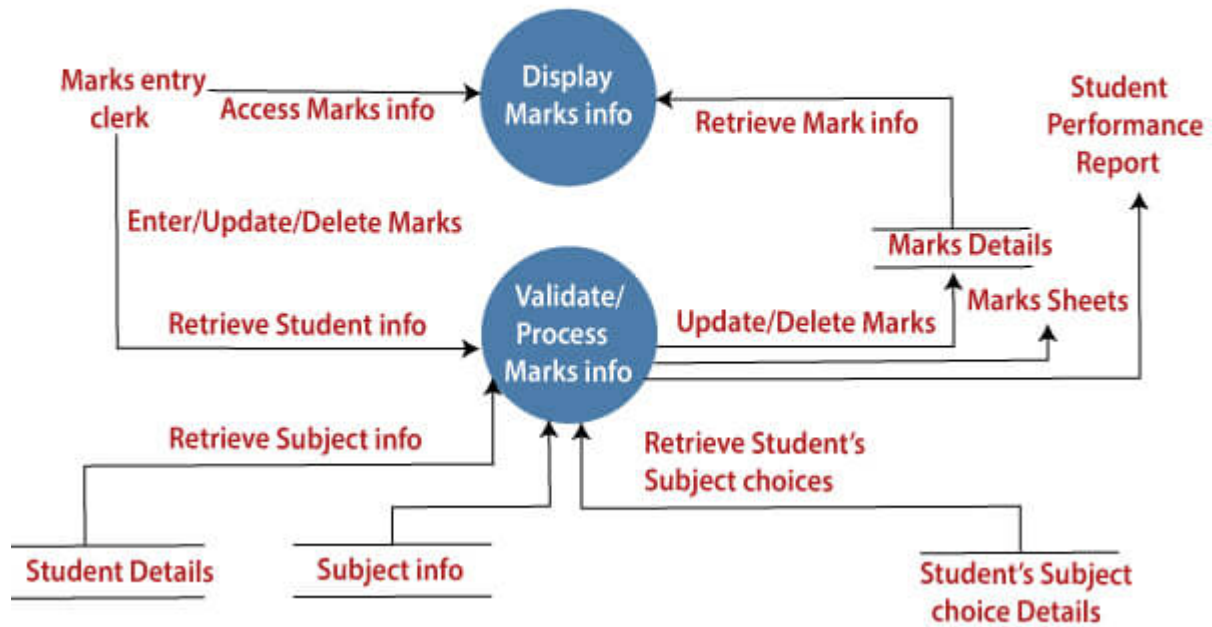
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



Behavioral Model

Behavioral Model is specially designed to make us understand behavior and factors that influence behavior of a System.

Behavior of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system.

So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram.

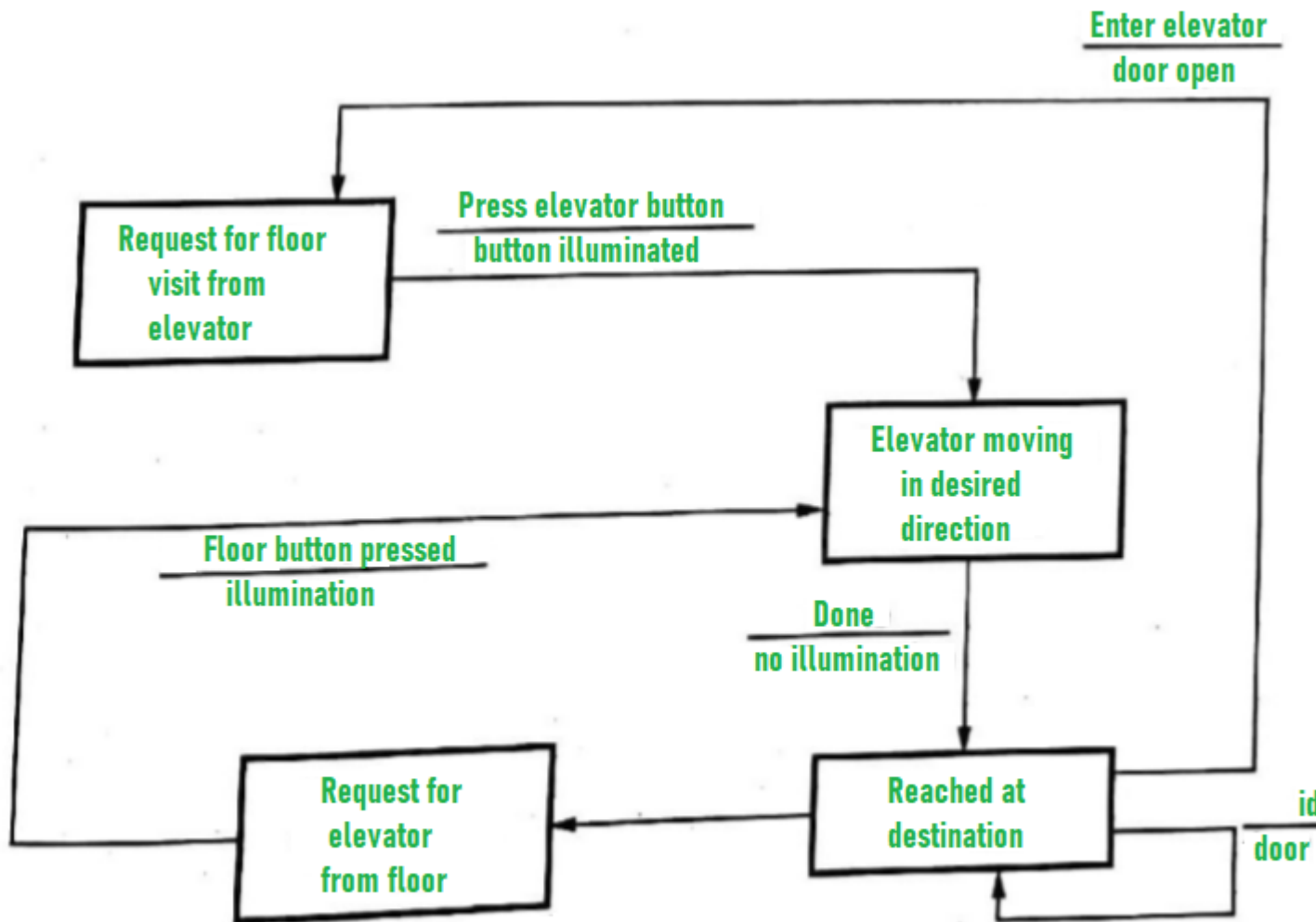
Example :

Consider an Elevator. This elevator is for n number of floors and has n number of buttons one for each floor.

Elevator's working can be explained as follows :

1. **Elevator buttons** are type of set of buttons which is there on elevator. For reaching a particular floor you want to visit, "elevator buttons" for that particular floor is pressed. Pressing, will cause illumination and elevator will start moving towards that particular floor for which you pressed "elevator buttons". As soon as elevator reaches that particular floor, illumination gets canceled.
2. **Floor buttons** are another type of set of buttons on elevator. If a person is on a particular floor and he wants to go on another floor, then elevator button for that floor is pressed. Then, process will be same as given above. Pressing, will cause illumination and elevator to start moving, and when it reaches on desired floor, illumination gets canceled.
3. When there is no request for elevator, it remains closed on current floor.

State Transition Diagram for an elevator system is shown below -



STATE TRANSITION DIAGRAM

Advantages :

- Behavior and working of a system can easily be understood without any effort.
- Results are more accurate by using this model.
- This model requires less cost for development as cost of resources can be minimal.
- It focuses on behavior of a system rather than theories.

Disadvantages :

- This model does not have any theory, so trainee is not able to fully understand basic principle and major concept of modeling.
- This modeling cannot be fully automated.

- Sometimes, it's not easy to understand overall result.
- Does not achieve maximum productivity due to some technical issues or any errors.