**UNIT-3**

**Requirement Engineering Process**

**Requirements engineering** is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system. The requirements engineering process is an iterative process that involves several steps, including:

* Requirements Elicitation: This is the process of gathering information about the needs and expectations of stakeholders for the software system. This step involves interviews, surveys, focus groups, and other techniques to gather information from stakeholders.
* Requirements Analysis: This step involves analyzing the information gathered in the requirements elicitation step to identify the high-level goals and objectives of the software system. It also involves identifying any constraints or limitations that may affect the development of the software system.
* Requirements Specification: This step involves documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.
* Requirements Validation: This step involves checking that the requirements are complete, consistent, and accurate. It also involves checking that the requirements are testable and that they meet the needs and expectations of stakeholders.
* Requirements Management: This step involves managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant.
* The Requirements Engineering process is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality.

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. it is the disciplined application of proven principle , methods ,tools and notations to describe a proposed system’s intended behavior and its associated constraints.

**Tools involved in requirement engineering:**

* observation report
* Questionnaire ( survey , poll )
* Use cases
* User stories
* Requirement workshop
* Mind mapping
* Role playing
* Prototyping

Requirements Engineering Process consists of the following main activities:

* Requirements elicitation
* Requirements specification
* Requirements verification and validation
* Requirements management

### **Requirements Elicitation:**

 It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project. The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed [here.](https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/) Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve, and the needs and expectations of the stakeholders who will use the system.

There are several techniques that can be used to elicit requirements, including:

* **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
* **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
* **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
* **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
* **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It’s important to document, organize and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent and accurate.

### **Requirements specification:**

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders.

**There are several types of requirements that are commonly specified in this step, including:**

* **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
* **Non-Functional Requirements**: These describe how well the software system should do it. They specify the quality attributes of the system, such as performance , reliability, usability, and security.
* **Constraints:**These describe any limitations or restrictions that must be considered when developing the software system.
* **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

In order to make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent, and accurate.

### **Requirements verification and validation:**

**Verification:** It refers to the set of tasks that ensures that the software correctly implements a specific function.

**Validation:** It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

* The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.
* The requirements should be complete in every sense.
* The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate, and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

* Verification is the process of checking that the requirements are complete, consistent, and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.
* Validation is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.
* V&V is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It’s important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

### **Requirements management:**

Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

There are several key activities that are involved in requirements management, including:

* **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.
* **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
* **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
* **Communication:**This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed in a timely manner.
* **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Requirements management is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality. It also helps to prevent scope creep and to ensure that the requirements are aligned with the project goals.

### Advantages:

* Helps ensure that the software being developed meets the needs and expectations of the stakeholders
* Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant
* Helps ensure that the software is developed in a cost-effective and efficient manner
* Can improve communication and collaboration between the development team and stakeholders

### Disadvantages:

* Can be time-consuming and costly, particularly if the requirements gathering process is not well-managed
* Can be difficult to ensure that all stakeholders’ needs and expectations are taken into account
* Can be challenging to ensure that the requirements are clear, consistent, and complete
* Changes in requirements can lead to delays and increased costs in the development process.
* As a best practice, Requirements engineering should be flexible, adaptable, and should be aligned with the overall project goals.

# Classification of Software Requirements

**A software requirement can be of 3 types:**

* Functional requirements
* Non-functional requirements
* Domain requirements

# Software Requirement

# **Functional Requirements:** These are the requirements that the end user specifically demands as basic facilities that the system should offer. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated. There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax. Functional Requirements in Software Engineering are also called Functional Specification.

**Non-functional requirements:** These are basically the quality constraints that the system must satisfy according to the project contract.Nonfunctional requirements, not related to the system functionality, rather define how the system should perform The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

NFR’s are classified into following types:

* Interface constraints
* Performance constraints: response time, security, storage space, etc.
* Operating constraints
* Life cycle constraints: maintainability, portability, etc.
* Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

They are divided into two main categories: Execution qualities like security and usability, which are observable at run time. Evolution qualities like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

**Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects. Domain requirements can be functional or nonfunctional. Domain requirements engineering is a continuous process of proactively defining the requirements for all foreseeable applications to be developed in the software product line. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

Software requirements can be classified into several categories based on their characteristics and the level of detail. The following are the most common types of software requirements:

**Functional requirements:** These are the requirements that define the functions and features of the software system. They describe what the software should do, and how it should behave when specific user actions or inputs are provided. Functional requirements are often documented as use cases or user stories.

**Non-functional requirements:** These requirements define the quality attributes of the software system, such as performance, reliability, scalability, usability, security, and compatibility. Non-functional requirements are critical for ensuring that the software meets the user’s needs and expectations.

**Business requirements:** These requirements describe the business needs and objectives that the software system is intended to fulfill. They define the problem or opportunity that the software is addressing, and the benefits that it is expected to provide.

**User requirements:** These requirements describe the needs and expectations of the end-users of the software system. They include information about the users’ tasks, workflows, and preferences, and how the software should support them.

**System requirements:**These requirements describe the hardware and software infrastructure that the software system needs to operate correctly. They include information about the operating system, database management system, network protocols, and other components required to run the software.

**Design requirements:** These requirements describe the technical design of the software system. They include information about the software architecture, data structures, algorithms, and other technical aspects of the software.

**Interface requirements:**These requirements describe the interfaces between the software system and other systems, devices, or services. They include information about the data formats, protocols, and communication channels that the software should use to integrate with other systems.

By categorizing software requirements, software engineers and project managers can ensure that all aspects of the software system are covered, and that the software meets the user’s needs and expectations

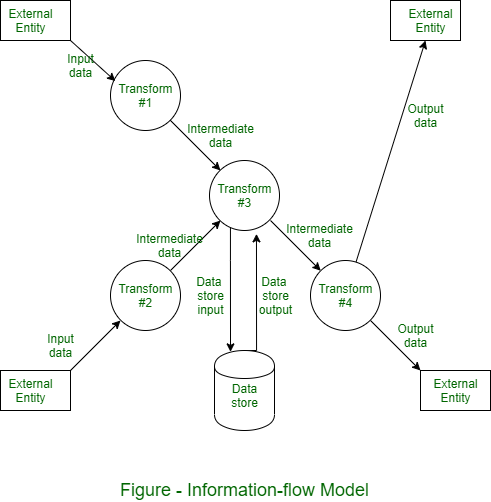
# Functional modelling and Information Flow modelling

In the **Functional Model**, software converts information. and to accomplish this, it must perform at least three common tasks- input, processing and output. When functional models of an application are created, the software engineer emphasizes problem specific tasks. The functional model begins with a single reference level model (i.e., be manufactured). In a series of iterations, more and more functional detail is given, until all system functionality is fully represented.

Information is converted because it flows from a computer-based system. The system takes input in various forms; Hardware, software, and human elements are applied to replace it; And produces in various forms. The transformation (s) or function may be composed of a single logical comparison, a complex numerical method, or a rule- the invention approach of an expert system. The output can light an LED or provide a 200 page report. Instead, we can create a model or flow model for any computer- based system, regardless of size and complexity.

Structural analysis started as an **Information Flow Modelling** technique. A computer-based system can be modeled as an information transform function as shown in figure.

A rectangle represents an external unit. That is, a system element, such as a hardware, a person or another system that provides information for transformation by the software or receives information provided by the software. A circle is used to represent a process or transform or a function that is applied to data and changes it in some way. An arrow is used to represent one or more data items.



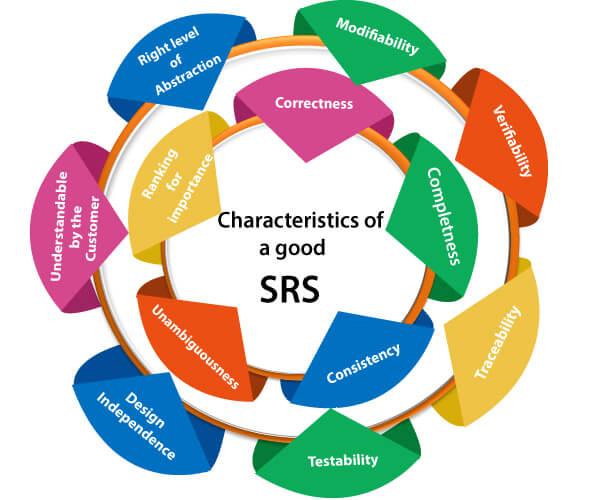
All arrows should be labeled in a [DFD](https://www.geeksforgeeks.org/levels-in-data-flow-diagrams-dfd/). The double line is used to represent data store. There may be implicit procedure or sequence in the diagram but explicit logical details are generally delayed until software design.

Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS



**Following are the features of a good SRS document:**

**1. Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

**2. Completeness:** The SRS is complete if, and only if, it includes the following elements:

**(1).** All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

**(2).** Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

**(3).** Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

**3. Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

**(1).** The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

**(2).** There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

**(3).** Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

**4. Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

**5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

**6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

**7. Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

**8. Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

**There are two types of Traceability:**

**1. Backward Traceability:** This depends upon each requirement explicitly referencing its source in earlier documents.

**2. Forward Traceability:** This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

**9. Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

**10. Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

**11. Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

**12. The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas,for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

## Properties of a good SRS document

**The essential properties of a good SRS document are the following:**

**Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

**Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

**Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

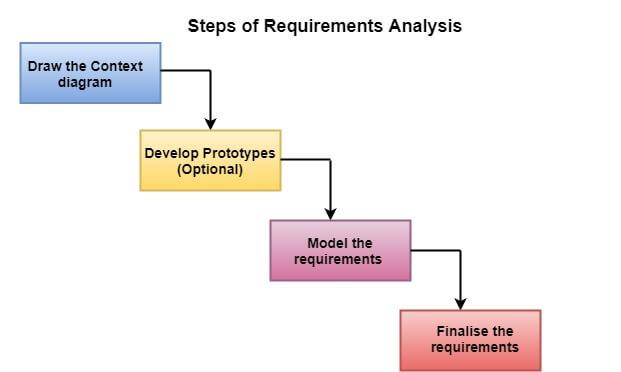
**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

**Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

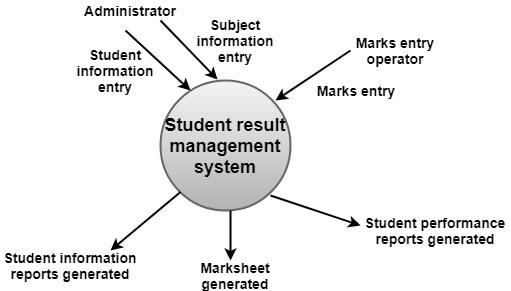
# Requirements Analysis

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:**



1. **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:



1. **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.
2. 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.
3. 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.
4. 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.
5. **(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.
6. **(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.

# 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.

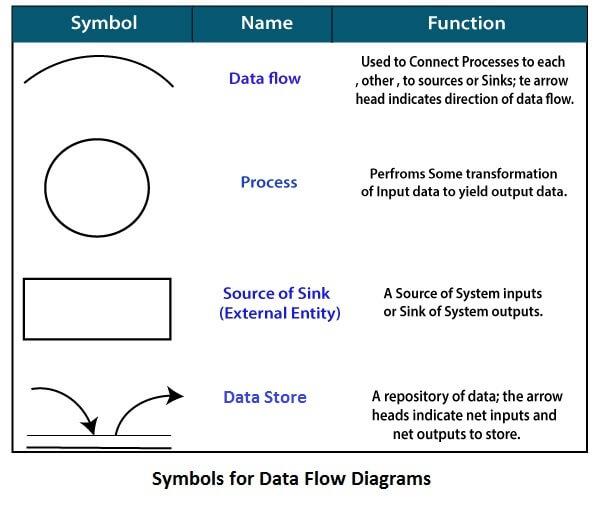
It shows how data enters and leaves the system, what changes the information, and where data is stored.

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 is 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 represents 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:



**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.

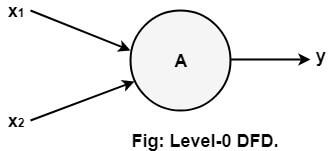
**Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

## Levels in Data Flow Diagrams (DFD)

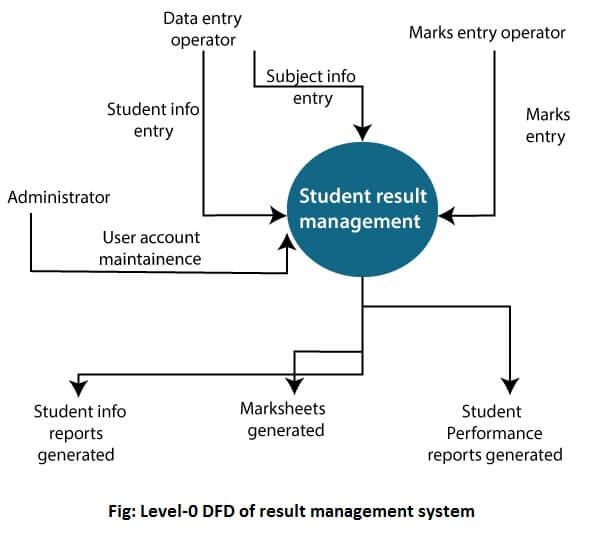
The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram,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

**0-level DFDM**

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 x1 and x2 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:

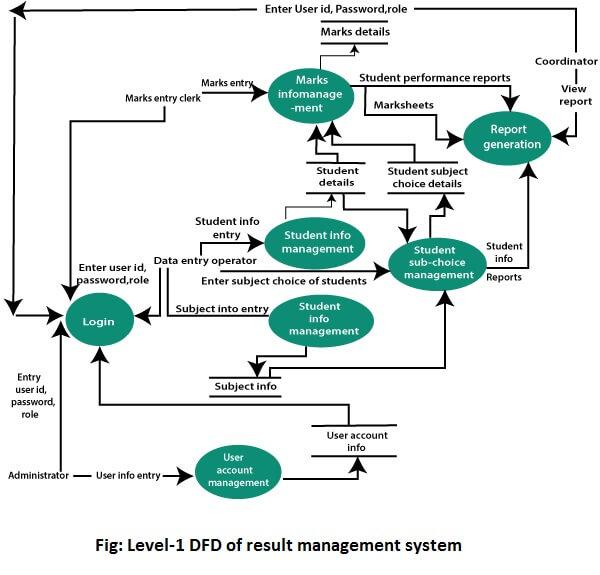


The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.



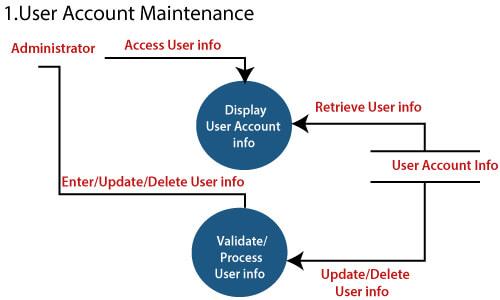
**1-level DFD**

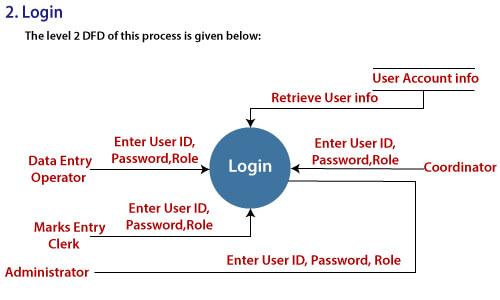
In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

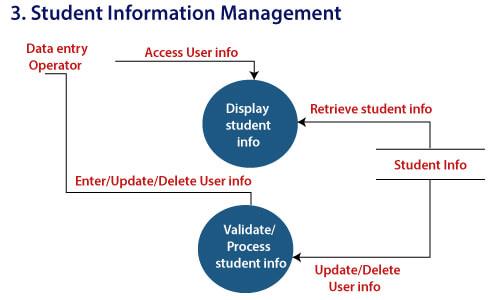


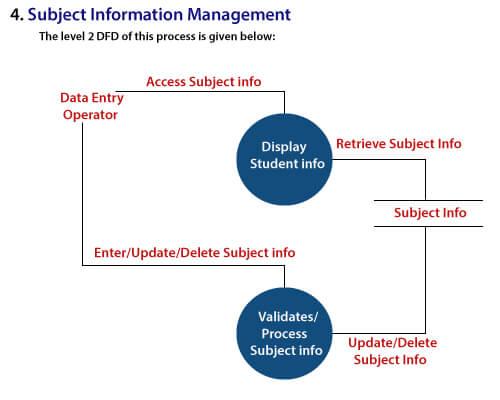
**2-Level DFD**

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.









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:

* Name of the data item
* Aliases
* Description/purpose
* Related data items
* Range of values
* 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.

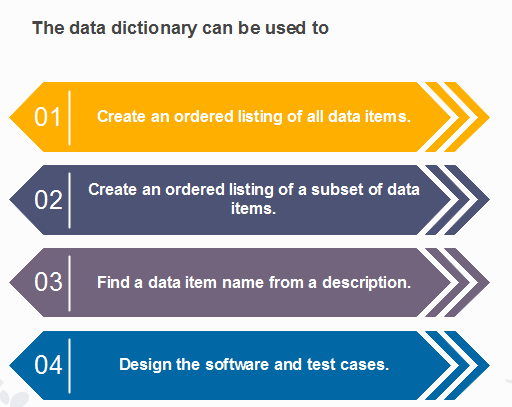
**Related data items** capture relationships between data items e.g., total\_marks must always equal to internal\_marks plus external\_marks.

**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. |



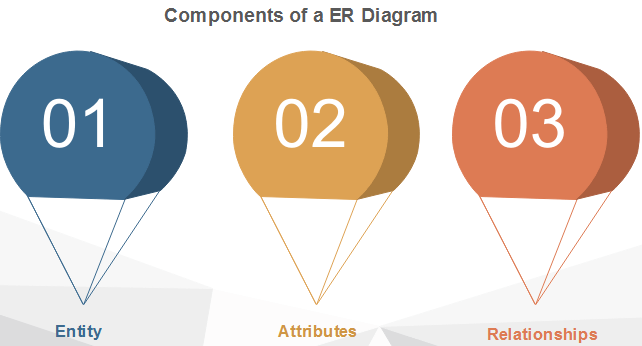
Entity-Relationship Diagrams

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD

* The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
* The ERD serves as a documentation tool.
* Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of an ER Diagrams



### 1. Entity

An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

**Entity Set**

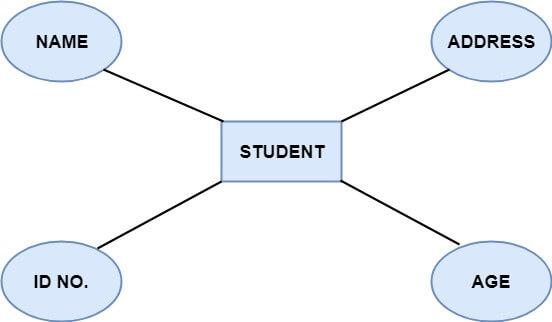
An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.



### 2. Attributes

Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

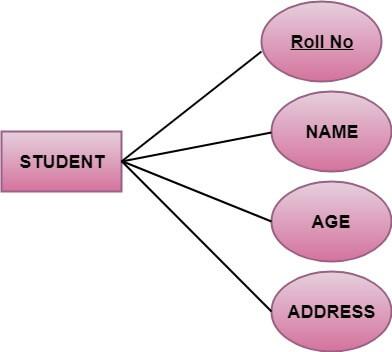
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



**There are four types of Attributes:**

1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

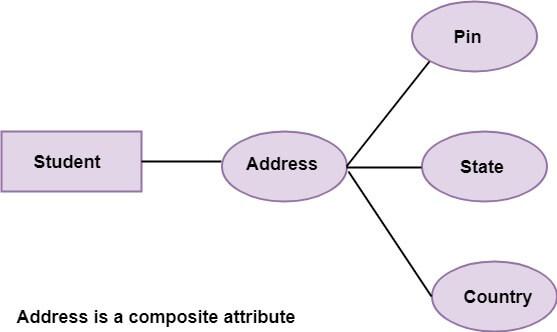
**1. Key attribute:** Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll\_number of a student makes him identifiable among students.



**There are mainly three types of keys:**

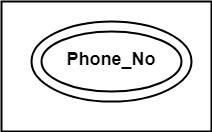
1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.
2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
3. **Primary key:** A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

**2. Composite attribute:** An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

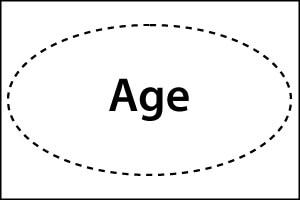


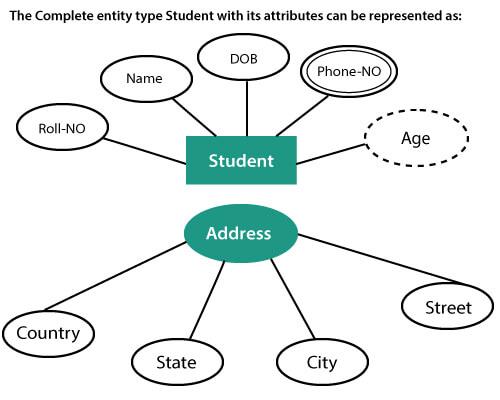
**3. Single-valued attribute:** Single-valued attribute contain a single value. For example, Social\_Security\_Number.

**4. Multi-valued Attribute:** If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



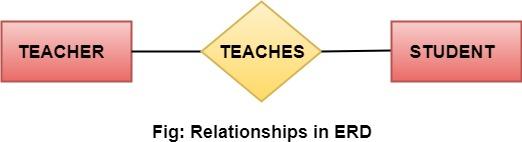
**5.Derived attribute:** Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date\_of\_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.





## 3. Relationships

The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works\_at a department, a student enrolls in a course. Here, Works\_at and Enrolls are called relationships.



Relationship set

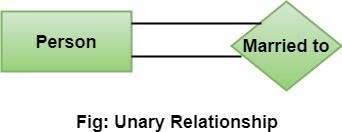
A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of a relationship set

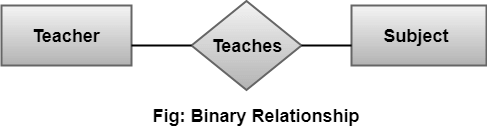
The number of participating entities in a relationship describes the degree of the relationship. The three most common relationships in E-R models are:

1. Unary (degree1)
2. Binary (degree2)
3. Ternary (degree3)

**1. Unary relationship:** This is also called recursive relationships. It is a relationship between the instances of one entity type. For example, one person is married to only one person.

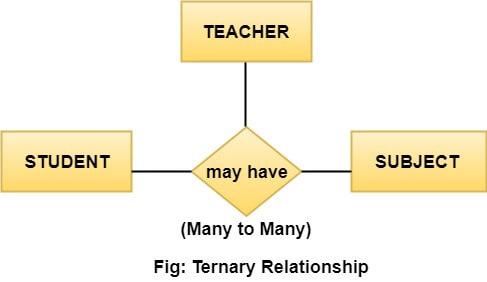


**2. Binary relationship:** It is a relationship between the instances of two entity types. For example, the Teacher teaches the subject.



**3. Ternary relationship:** It is a relationship amongst instances of three entity types. In fig, the relationships "**may have**" provide the association of three entities, i.e., TEACHER, STUDENT, and SUBJECT. All three entities are many-to-many participants. There may be one or many participants in a ternary relationship.

In general, "**n**" entities can be related by the same relationship and is known as **n-ary** relationship



## Cardinality

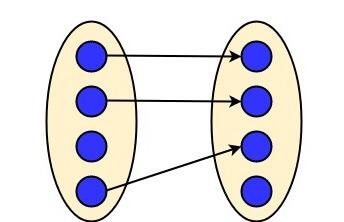
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

### Types of Cardinalities

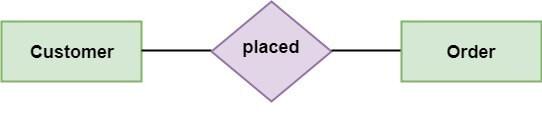
**1. One to One:** One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.



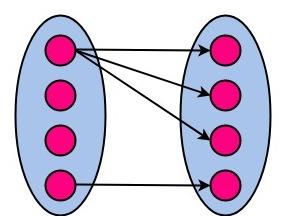
**Using Sets, it can be represented as:**



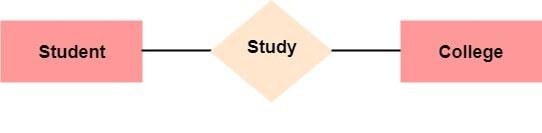
**2. One to many:** When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.



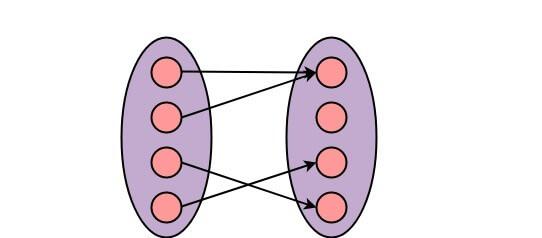
**Using Sets, it can be represented as:**



1. **Many to One:** More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. For example - many students can study in a single college, but a student cannot study in many colleges at the same time.



**Using Sets, it can be represented as:**



1. **Many to Many:** One entity from A can be associated with more than one entity from B and vice-versa. For example, the student can be assigned to many projects, and a project can be assigned to many students.



**Using Sets, it can be represented as:**

