

# Unit – II

## Software Requirement Specification

### Software Requirement Specification (SRS) -Problem analysis

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 the system analyst to document the requirements in technical language so that they can be understood and used by the software development team.

SRS should come up with the 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 inaccurately. 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

### 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. Some of them are explained below:

## 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.

## 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.

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

Types of Requirements:

The below diagram depicts the various types of requirements that are captured during SRS.



## 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.


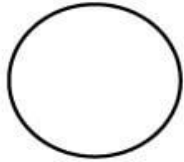

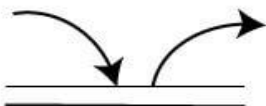
It shows how data enters and leaves the system, what changes the information, and where data is stored.

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 represent the order of events; arrows in DFD represent 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 existing 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; te arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

**Symbols for Data Flow Diagrams**

**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

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

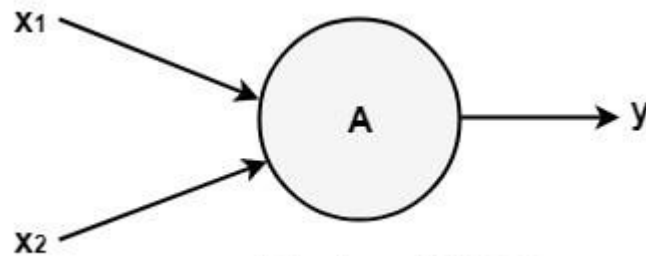
## Levels in Data Flow Diagrams (DFD)

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

### 0-level DFDM

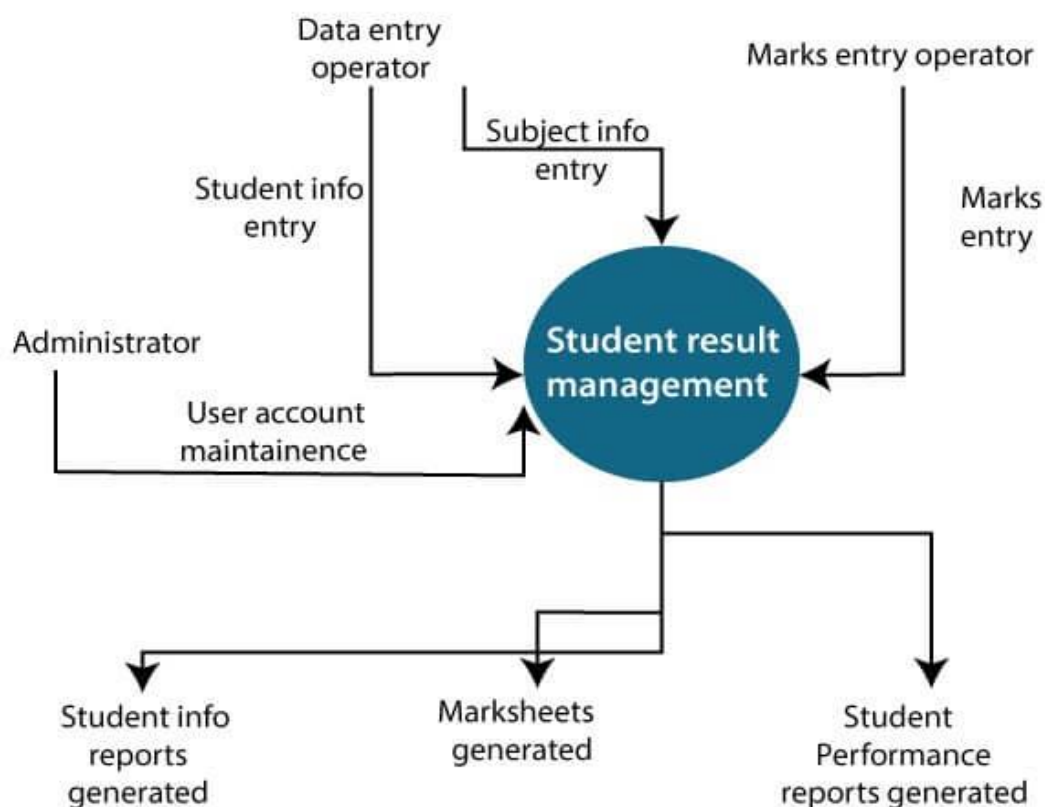
It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a

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  $x_1$  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.**

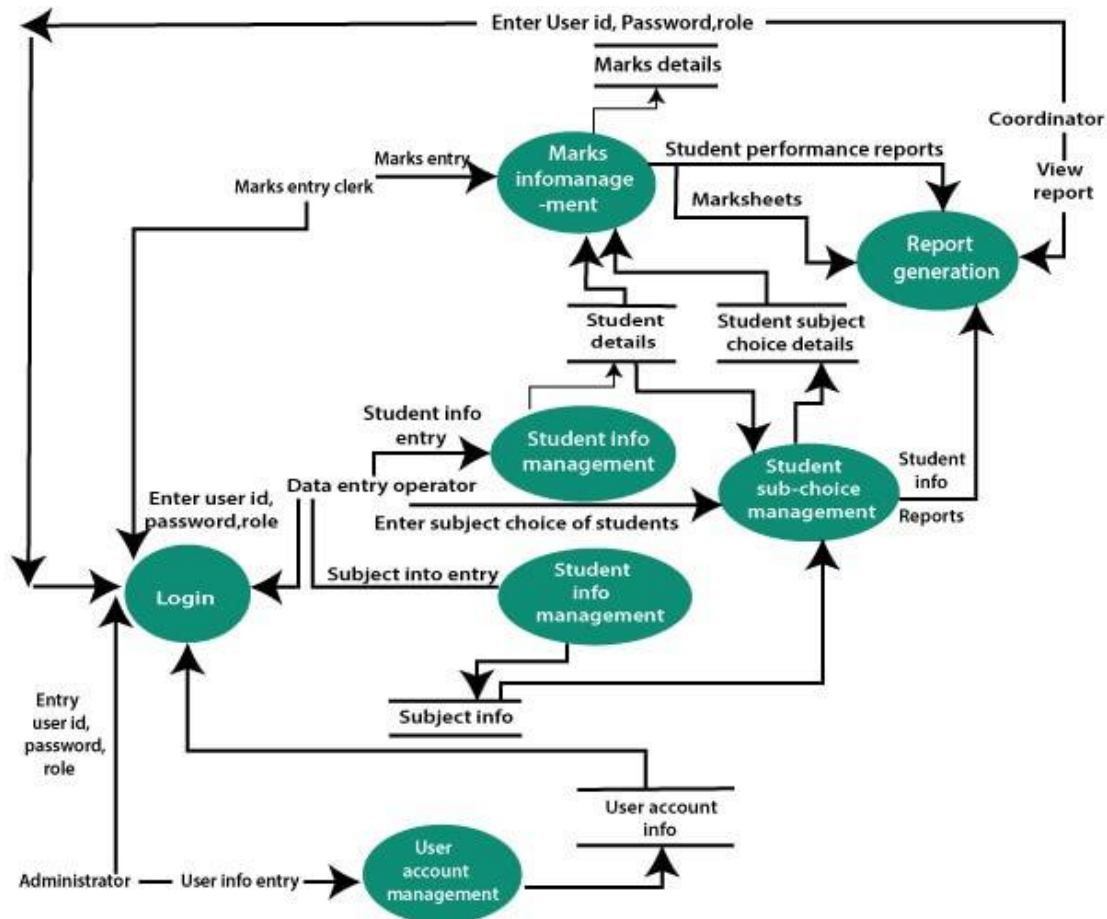
The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.



**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**

## Entity-Relationship Diagrams

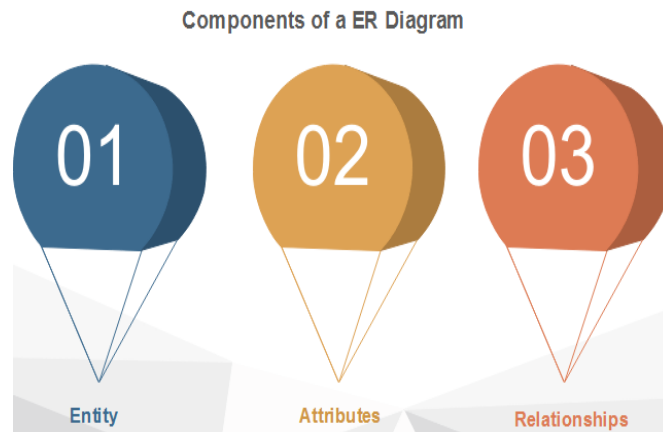
ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

### 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 is a real-world object, which is distinguishable (identifiable) from rest of the objects. 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

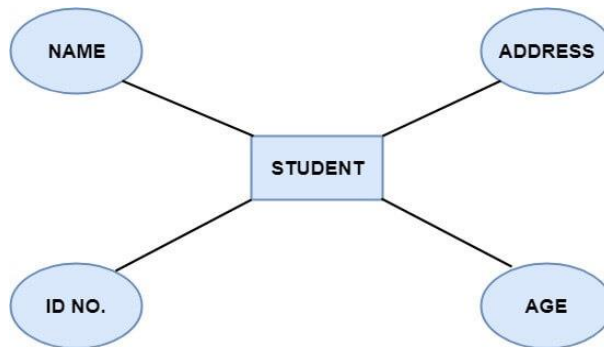
An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need 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.

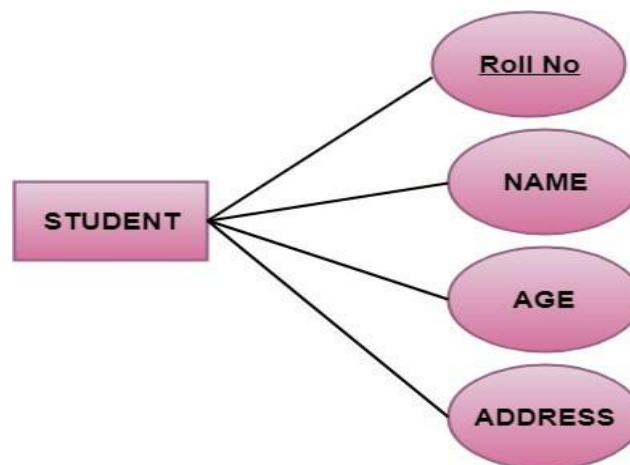
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



**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.

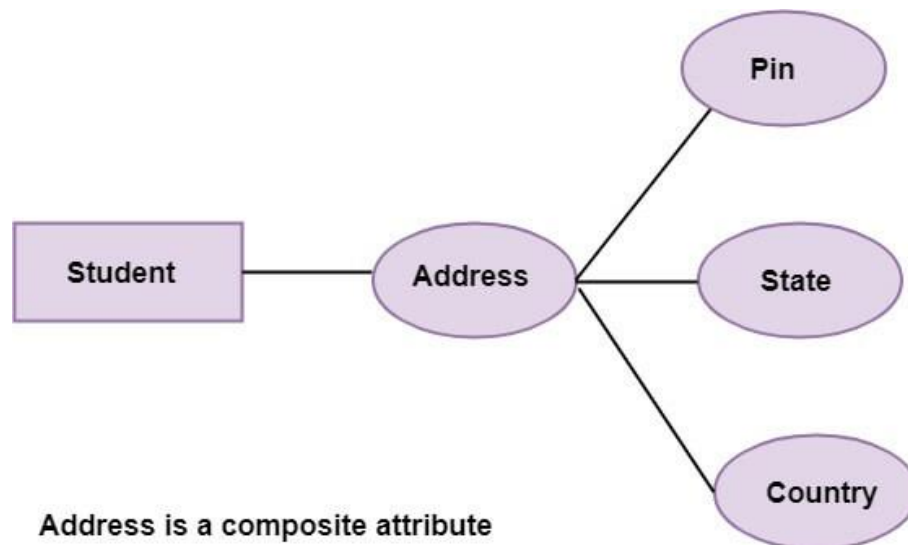


**There are mainly three types of keys:**

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.

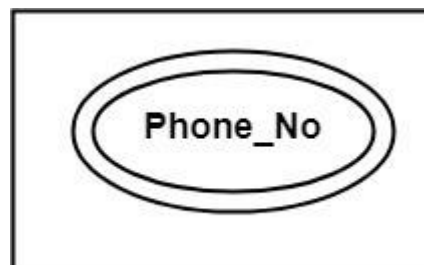


**2. Composite attribute:** An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

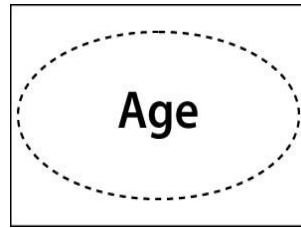


**3. Single-valued attribute:** Single-valued attribute contain a single value. For example, Social\_Security\_Number.

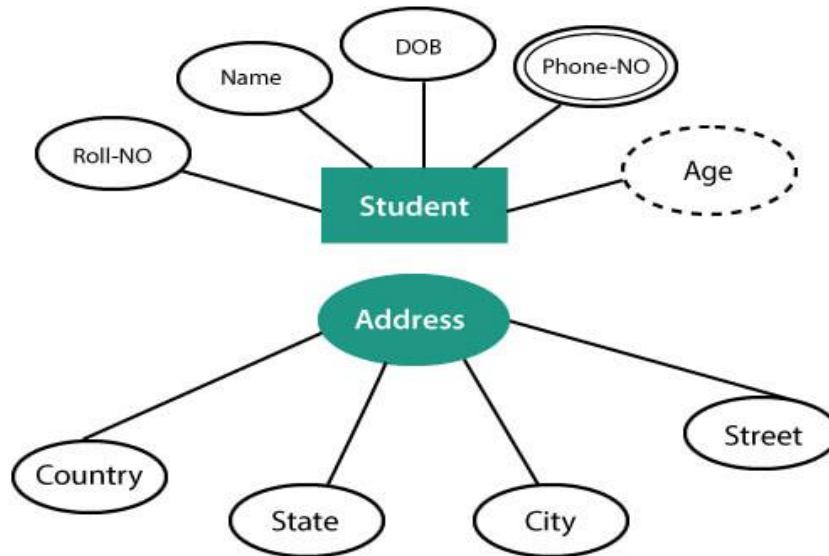
**4. Multi-valued Attribute:** If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



**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.



The Complete entity type Student with its attributes can be represented as:



### 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.



Fig: Relationships in ERD

#### Relationship set

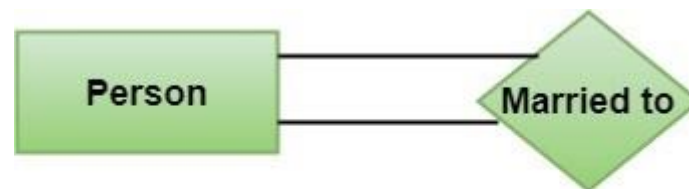
A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

#### 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.



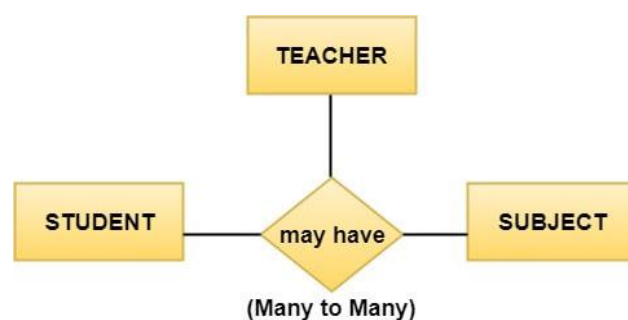
**Fig: Unary Relationship**

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



**Fig: Binary Relationship**

**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.



**Fig: Ternary Relationship**

## Cardinality

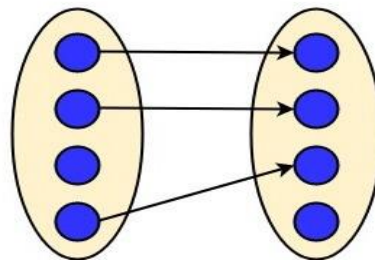
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

## 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.



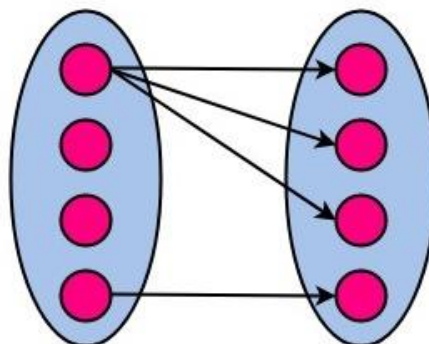
Using Sets, it can be represented as:



**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:

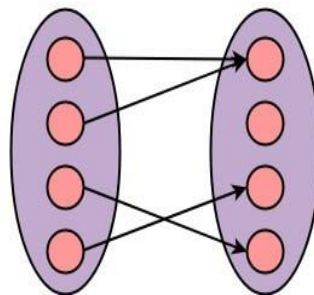


**3. Many to One:** More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated

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.



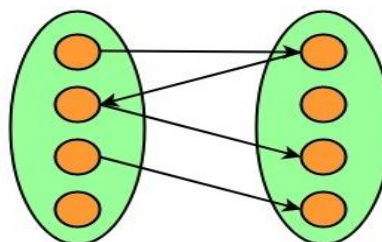
Using Sets, it can be represented as:



**4. 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:



Analysts use various tools to understand and describe the information system. One of the ways is using structured analysis.

## Structured Analysis

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

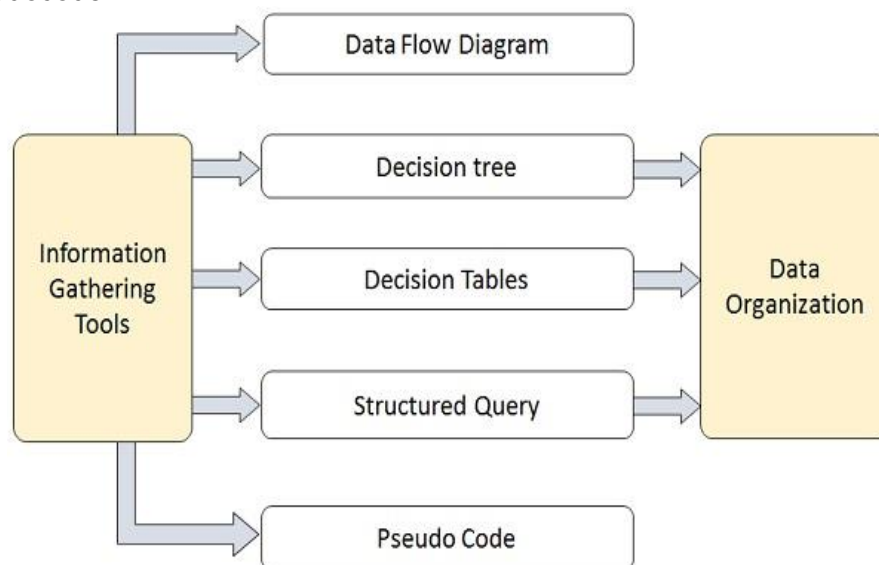
It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

## Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are –

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



## Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

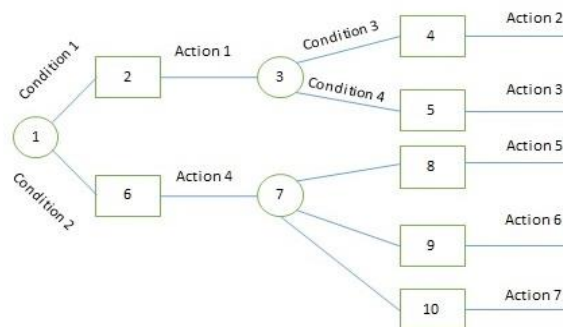
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table –

Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15

## Decision Trees

Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree –



## Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

- It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
- It is a matrix containing row or columns for defining a problem and the actions.

### Components of a Decision Table

- **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** – It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table –

CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N



Regular Customer	-	Y	N	-
<b>ACTIONS</b>				
Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

## Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

- It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
- It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions –

```

if customer pays advance
then
    Give 5% Discount
else
    if purchase amount >=10,000
    then
        if the customer is a regular customer
        then Give 5% Discount
        else No Discount
        end if
    else No Discount
    end if
end if

```

## Software Requirements Specification (SRS) 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:

- **Correctness:**

User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

- **Completeness:**

Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

- **Unambiguousness:**

A SRS is said to be unambiguous if all the requirements stated have only one interpretation. Some of the ways to prevent unambiguousness include the use of modeling techniques like ER diagrams, proper reviews and buddy checks, etc.

- **Ranking for importance:**

There should be a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

- **Modifiability:**

SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

- **Verifiability:**

A SRS is verifiable if there exist a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

- **Traceability:**

One should be able to trace a requirement to design components and then to code segments in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

- **Testability:**

A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

Understandable by the customer:

An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should

be avoided to as much extent as possible. The language should be kept easy and clear.

## **COMPONENTS OF THE SRS**

Here we describe some of the system properties that an SRS should specify. The basic issues, an SRS must address are:

- 1. Functional requirements**
- 2. Performance requirements**
- 3. Design constraints**
- 4. External interface requirements**

Conceptually, any SRS should have these components. Now we will discuss them one by one.

### **1. Functional Requirements**

Functional requirements specify what output should be produced from the given inputs. So they basically describe the connectivity between the input and output of the system. For each functional requirement:

1. A detailed description of all the data inputs and their sources, the units of measure, and the range of valid inputs be specified:
2. All the operations to be performed on the input data obtain the output should be specified, and
3. It must clearly state what the system should do if system behaves abnormally when any invalid input is given or due to some error during computation. Specifically, it should specify the behaviour of the system for invalid inputs and invalid outputs.

### **2. Performance Requirements (Speed Requirements)**

This part of an SRS specifies the performance constraints on the software system. All the requirements related to the performance characteristics of the system must be clearly specified. Performance requirements are typically expressed as processed transactions per second or response time from the system for a user event or screen refresh time or a combination of these. It is a good idea to pin down performance requirements for the most used or critical transactions, user events and screens.

### **3. Design Constraints**

The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

**Standard Compliance:** It specifies the requirements for the standard the system must follow. The standards may include the report format and according procedures.

**Hardware Limitations:** The software needs some existing or predetermined hardware to operate, thus imposing restrictions on the design. Hardware limitations can include the types of machines to be used operating system availability memory space etc.

**Fault Tolerance:** Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive, so they should be minimized.

**Security:** Currently security requirements have become essential and major for all types of systems. Security requirements place restrictions on the use of certain commands control access to database, provide different kinds of access, requirements for different people, require the use of passwords and cryptography techniques, and maintain a log of activities in the system.

#### **4. External Interface Requirements**

For each external interface requirements:

1. All the possible interactions of the software with people, hardware and other software should be clearly specified,
2. The characteristics of each user interface of the software product should be specified and
3. The SRS should specify the logical characteristics of each interface between the software product and the hardware components for hardware interfacing.