Software Requirement Analysis & Specification: Functional and non-functional, Feasibility studies, Requirements elicitation, Process modelling with physical and logical DFDs, Entity Relationship Diagram, Data Dictionary, Requirement specification: Software requirement Specification (SRS).

Requirements analysis is a very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

Important Topics for Functional vs Non-Functional Requirements

- Functional Requirements
- Non-Functional Requirements
- Extended Requirements
- Difference between Functional Requirements and Non-Functional Requirements:

Functional Requirements

These are the requirements that the end user specifically demands as basic facilities that the system should offer. 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 the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioural requirements. They deal with issues like:

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

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Extended Requirements

These are basically “nice to have” requirements that might be out of the scope of the System.

Example:

- Our system should record metrices and analytics.
- Service heath and performance monitoring.

Difference between Functional Requirements and Non-Functional Requirements:

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example 1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	Example 1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be

for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study

The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility**: In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.
2. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.
3. **Economic Feasibility:** In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.
4. **Legal Feasibility:** In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.
5. **Schedule Feasibility:** In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.
6. **Cultural and Political Feasibility:** This section assesses how the software project will affect the political environment and organizational culture. This analysis takes into account the organization’s culture and how the suggested changes could be received there, as well as any potential political obstacles or internal opposition to the project. It is essential that cultural and political factors be taken into account in order to execute projects successfully.
7. **Market Feasibility:** This refers to evaluating the market’s willingness and ability to accept the suggested software system. Analyzing the target market, understanding consumer wants and assessing possible rivals are all part of this study. It assists in identifying whether the project is in line with market expectations and whether there is a feasible market for the good or service being offered.

8. **Resource Feasibility:** This method evaluates if the resources needed to complete the software project successfully are adequate and readily available. Financial, technological and human resources are all taken into account in this study. It guarantees that sufficient hardware, software, trained labor and funding are available to complete the project successfully.

Aim of Feasibility Study

- The overall objective of the organization are covered and contributed by the system or not.
- The implementation of the system be done using current technology or not.
- Can the system be integrated with the other system which are already exist

Feasibility Study Process

The below steps are carried out during entire feasibility analysis.

1. Information assessment: It assesses the original project concept and establishes the main aims and objectives.
2. Information collection: It collects the necessary information and data required to evaluate the project's many components.
3. Report writing: It produces an in-depth feasibility report that details the analysis and results.
4. General information: It gives a summary of the main points discussed in the report on the feasibility study.

Need of Feasibility Study

Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.

Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and enhance success rate analyzing different parameters associated with proposed project development.

Requirements elicitation is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer's needs and requirements. This article focuses on discussing Requirement Elicitation in detail.

What is Requirement Elicitation?

Requirements elicitation is perhaps the most difficult, most error-prone, and most communication-intensive software development.

1. It can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
2. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
3. It is a critical part of the software development life cycle and is typically performed at the beginning of the project.

4. Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts.
5. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.

Importance of Requirements Elicitation

1. **Compliance with Business Objectives:** The process of elicitation guarantees that the software development endeavours are in harmony with the wider company aims and objectives. Comprehending the business context facilitates the development of a solution that adds value for the company.
2. **User Satisfaction**: It is easier to create software that fulfils end users needs and expectations when they are involved in the requirements elicitation process. Higher user pleasure and acceptance of the finished product are the results of this.
3. **Time and Money Savings:** Having precise and well-defined specifications aids in preventing miscommunication and rework during the development phase. As a result, there will be cost savings and the project will be completed on time.
4. **Compliance and Regulation Requirements**: Requirements elicitation is crucial for projects in regulated industries to guarantee that the software conforms with applicable laws and norms. In industries like healthcare, finance, and aerospace, this is crucial.
5. **Traceability and Documentation:** Throughout the software development process, traceability is based on requirements that are well-documented. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

Requirements Elicitation Activities:

Requirements elicitation includes the subsequent activities. A few of them are listed below:

1. Knowledge of the overall area where the systems are applied.
2. The details of the precise customer problem where the system is going to be applied must be understood.
3. Interaction of system with external requirements.
4. Detailed investigation of user needs.
5. Define the constraints for system development.

Requirements Elicitation Methods:

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

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 maybe 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 a structured interview, an 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

Its objective is to bridge the expectation gap – the 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.

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.

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

Steps Of Requirements Elicitation:

1. Identify all the stakeholders, e.g., 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.

Features of Requirements Elicitation:

1. **Stakeholder engagement:** Requirements elicitation involves engaging with stakeholders such as customers, end-users, project sponsors, and subject-matter experts to understand their needs and requirements.
2. **Gathering information:** Requirements elicitation involves gathering information about the system to be developed, the business processes it will support, and the end-users who will be using it.
3. **Requirement prioritization:** Requirements elicitation involves prioritizing requirements based on their importance to the project's success.
4. **Requirements documentation:** Requirements elicitation involves documenting the requirements in a clear and concise manner so that they can be easily understood and communicated to the development team.
5. **Validation and verification:** Requirements elicitation involves validating and verifying the requirements with the stakeholders to ensure that they accurately represent their needs and requirements.
6. **Iterative process:** Requirements elicitation is an iterative process that involves continuously refining and updating the requirements based on feedback from stakeholders.
7. C**ommunication and collaboration:** Requirements elicitation involves effective communication and collaboration with stakeholders, project team members, and other relevant parties to ensure that the requirements are clearly understood and implemented.
8. **Flexibility:** Requirements elicitation requires flexibility to adapt to changing requirements, stakeholder needs, and project constraints.

Advantages of Requirements Elicitation:

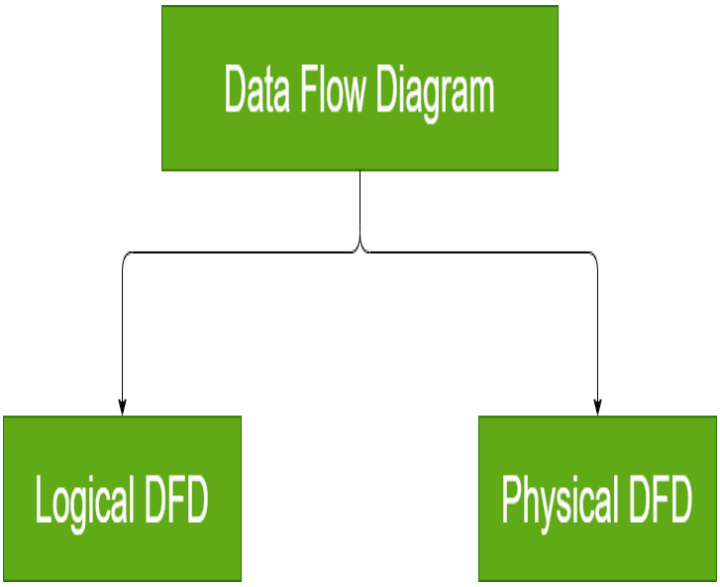
1. **Clear requirements:** Helps to clarify and refine customer requirements.
2. **Improves communication:** Improves communication and collaboration between stakeholders.

3. **Results in good quality software:** Increases the chances of developing a software system that meets customer needs.
4. **Avoids misunderstandings:** Avoids misunderstandings and helps to manage expectations.
5. **Supports the identification of potential risks:** Supports the identification of potential risks and problems early in the development cycle.
6. **Facilitates development of accurate plan:** Facilitates the development of a comprehensive and accurate project plan.
7. **Increases user confidence:** Increases user and stakeholder confidence in the software development process.
8. **Supports identification of new business opportunities:** Supports the identification of new business opportunities and revenue streams.

Disadvantages of Requirements Elicitation:

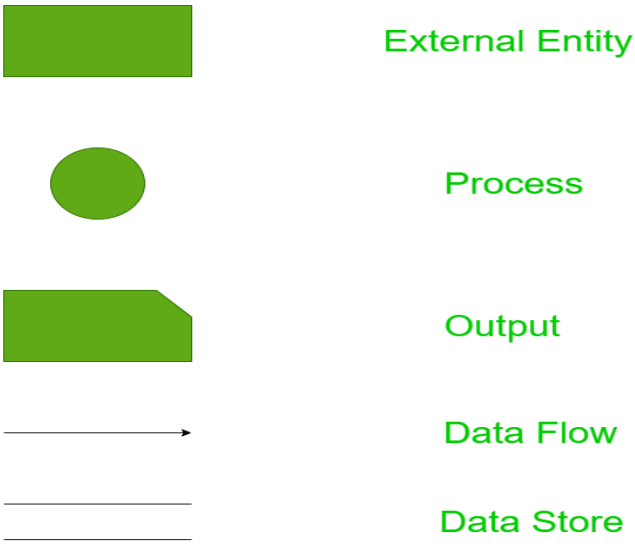
1. **Time consuming:** Can be time-consuming and expensive.
2. **Skills required:** Requires specialized skills and expertise.
3. **Impacted by changing requirements:** May be impacted by changing business needs and requirements.
4. **Impacted by other factors:** Can be impacted by political and organizational factors.
5. **Lack of commitment from stakeholders:** Can result in a lack of buy-in and commitment from stakeholders.
6. **Impacted by conflicting priorities:** Can be impacted by conflicting priorities and competing interests.
7. **Sometimes inaccurate requirements:** May result in incomplete or inaccurate requirements if not properly managed.
8. **Increased development cost:** Can lead to increased development costs and decreased efficiency if requirements are not well-defined.

Data Flow Diagram (DFD) is a graphical representation of data flow in any system. It is capable of illustrating incoming data flow, outgoing data flow and store data. Data flow diagram describes anything about how data flows through the system. Sometimes people get confused between data flow diagram and flowchart. There is a major difference between data flow diagram and flowchart. The flowchart illustrates flow of control in program modules. Data flow diagrams illustrate flow of data in the system at various levels. Data flow diagram does not have any control or branch elements. **Types of DFD :** DFD is of two types:



1. **Logical DFD:** Logical data flow diagram mainly focuses on the system process. It illustrates how data flows in the system. Logical DFD is used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another.
2. **Physical DFD:** Physical data flow diagram shows how the data flow is actually implemented in the system. Physical DFD is more specific and close to implementation.

Components of Data Flow Diagram: Following are the components of the data flow diagram that are used to represent source, destination, storage and flow of data.



- **Entities:** Entities include source and destination of the data. Entities are represented by rectangle with their corresponding names.
- **Process:** The tasks performed on the data is known as process. Process is represented by circle. Somewhere round edge rectangles are also used to represent process.
- **Data Storage:** Data storage includes the database of the system. It is represented by rectangle with both smaller sides missing or in other words within two parallel lines.
- **Data Flow:** The movement of data in the system is known as data flow. It is represented with the help of arrow. The tail of the arrow is source and the head of the arrow is destination.

Importance of Data Flow Diagram: Data flow diagram is a simple formalism to represent the flow of data in the system. It allows a simple set of intuitive concepts and rules. It is an elegant technique that is useful to represent the results of structured analysis of software problem as well as to represent the flow of documents in an organization.

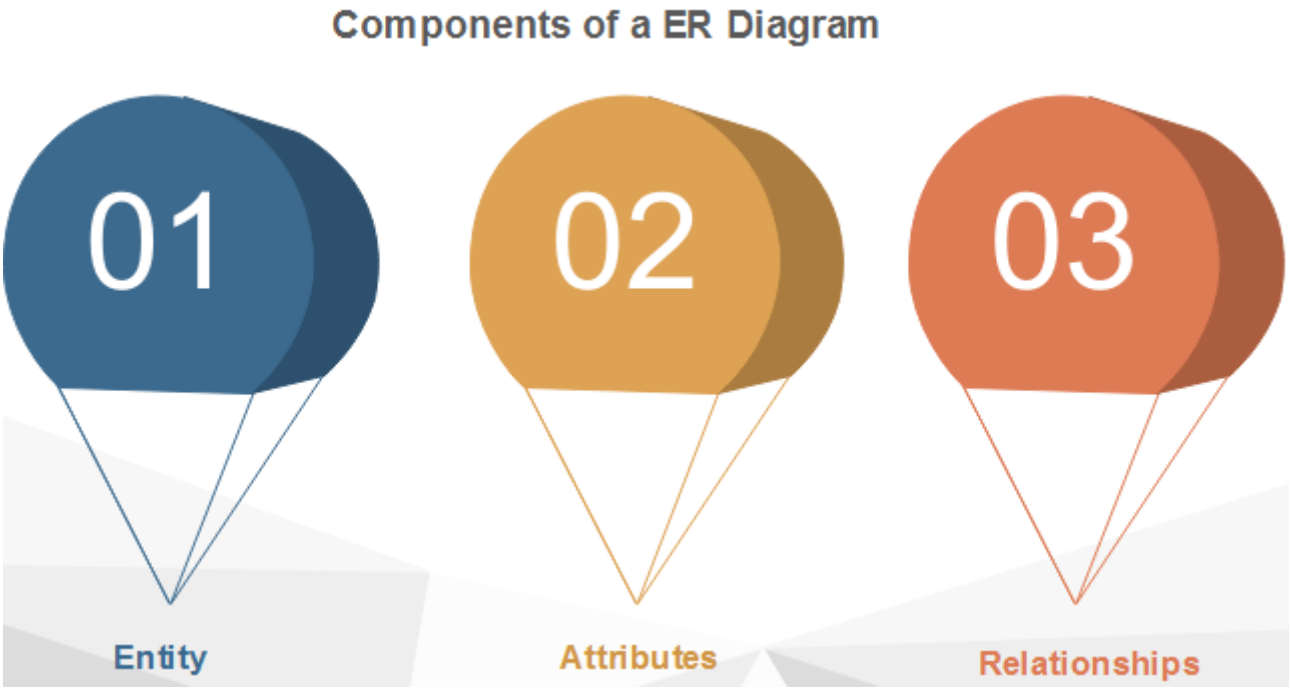
Entity-Relationship Diagrams

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.

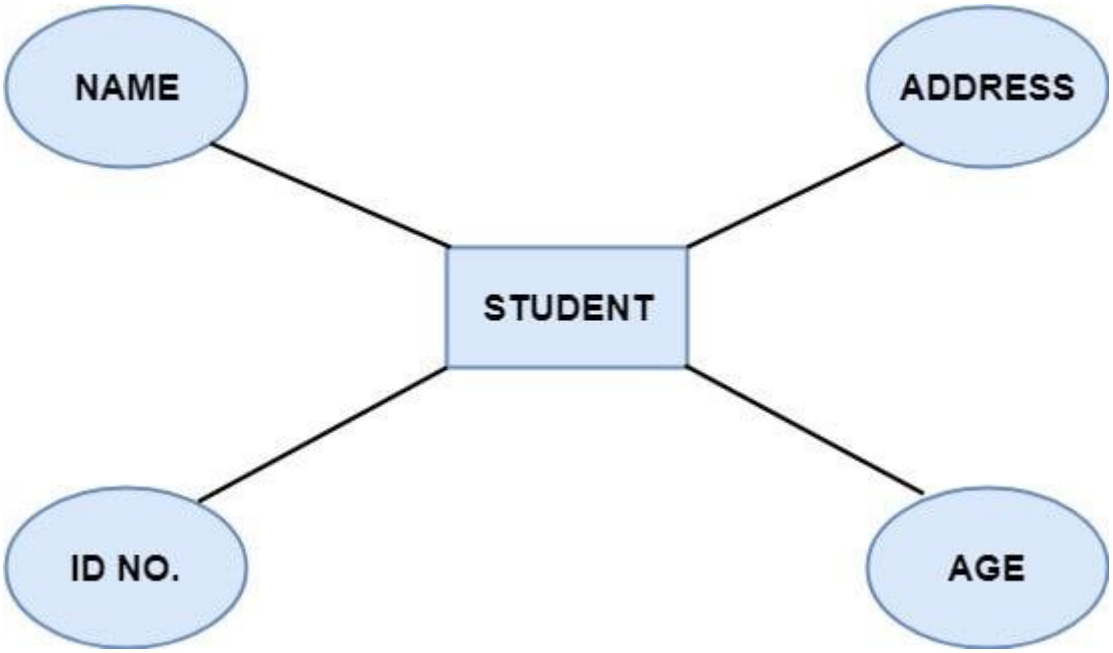
Entity Set

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.

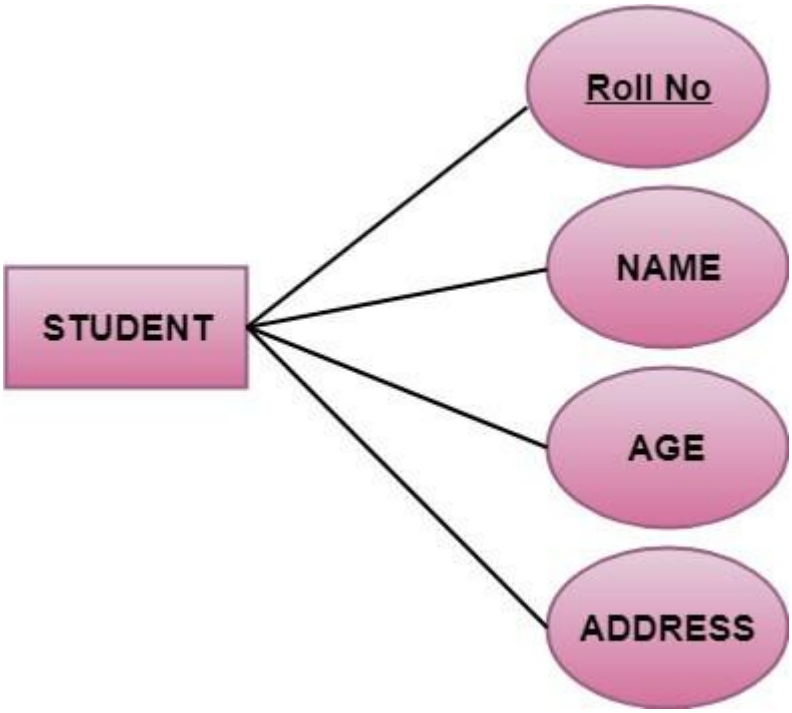
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

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

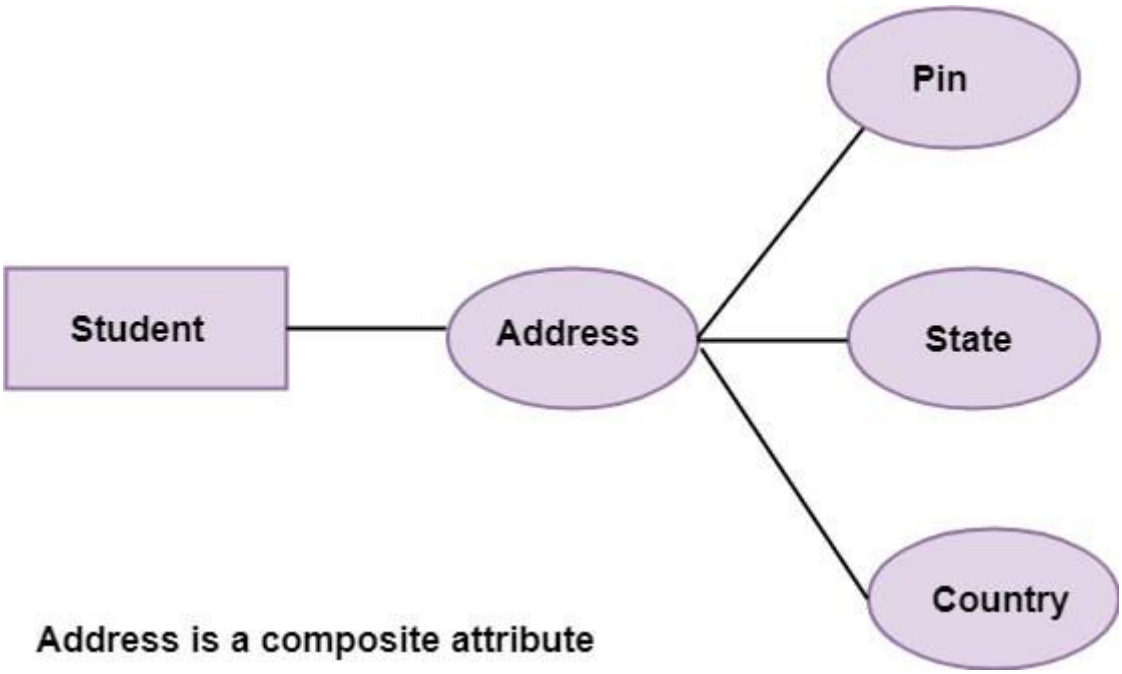
1. Key attribute: Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll_number of a student makes him identifiable among students.



There are mainly three types of keys:

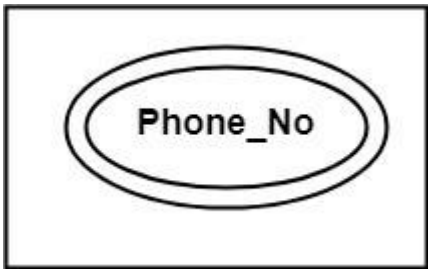
1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.
2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
3. **Primary key:** A primary key is one of the candidate keys chosen by the database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

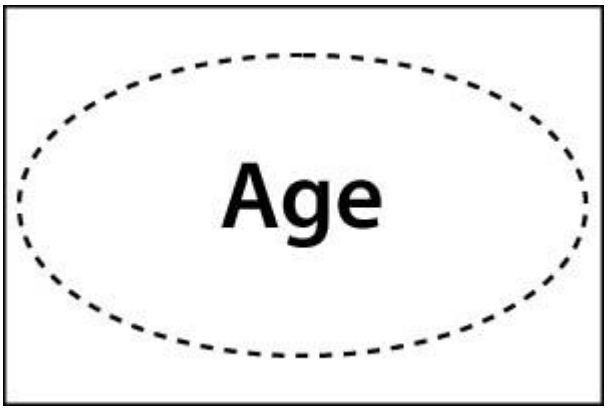


3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

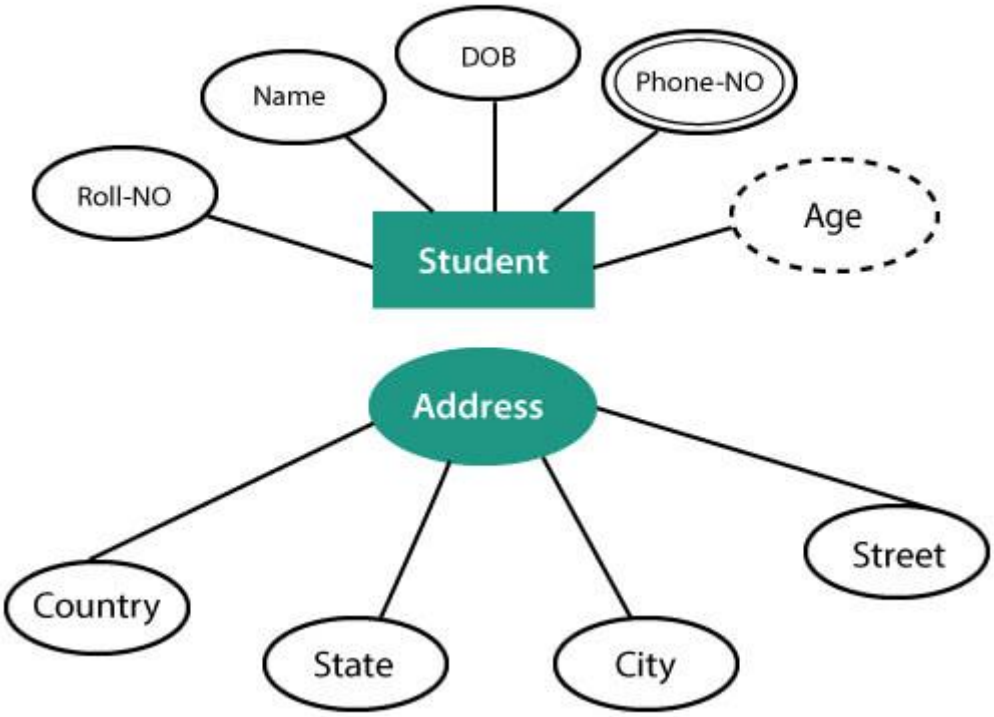
4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



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.



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.

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

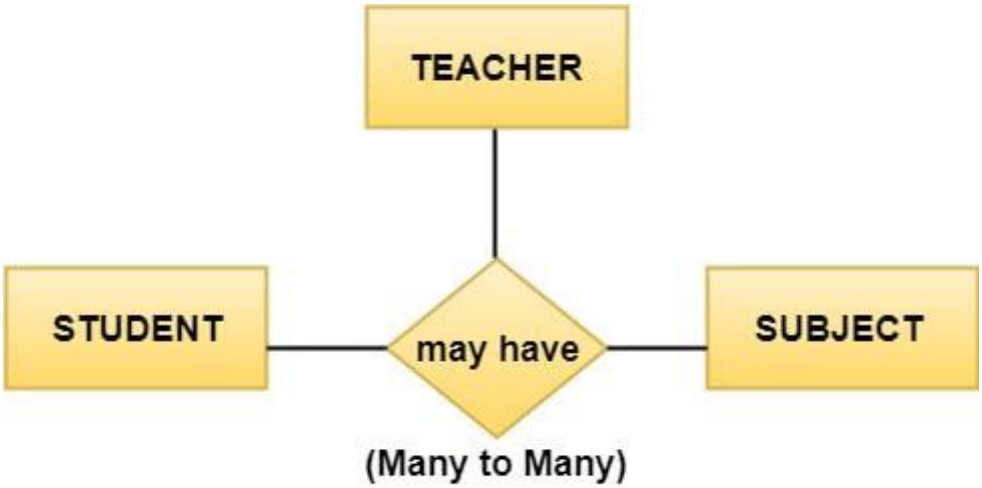


Fig: Ternary Relationship

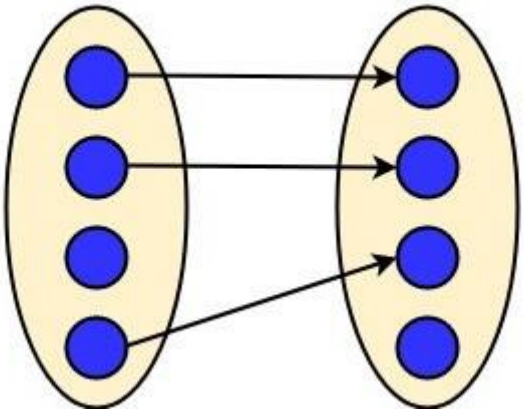
Cardinality

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

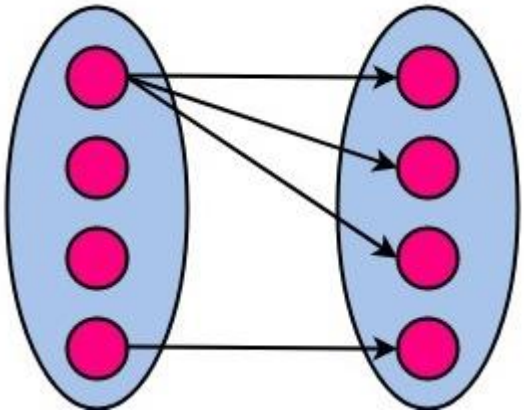
1. One to One: One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.

Using Sets, it can be represented as:



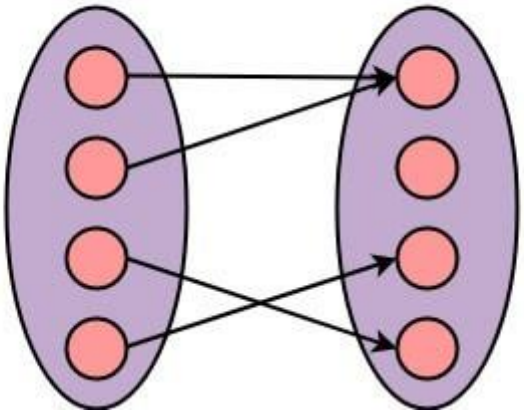
2. One to many: When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.

Using Sets, it can be represented as:



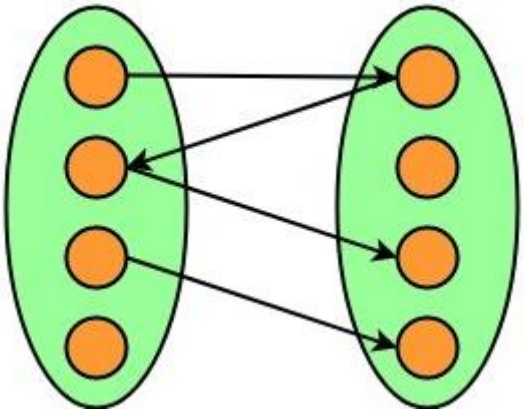
3. Many to One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. For example - many students can study in a single college, but a student cannot study in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



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.

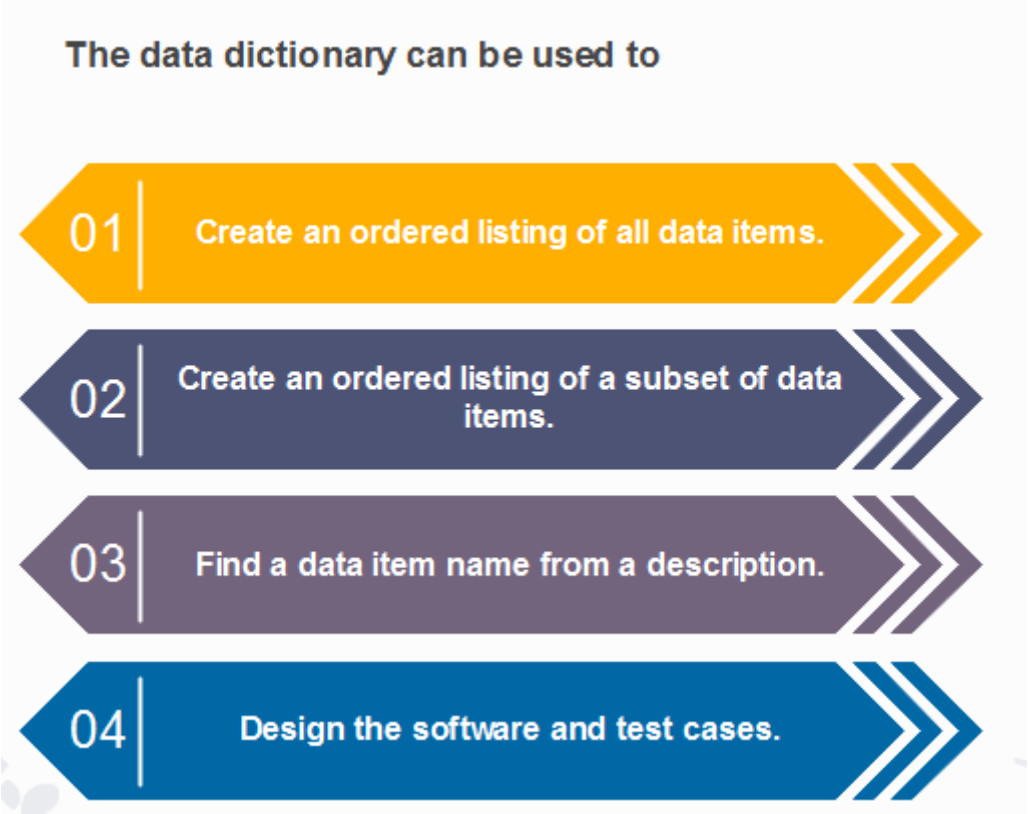
Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.

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.

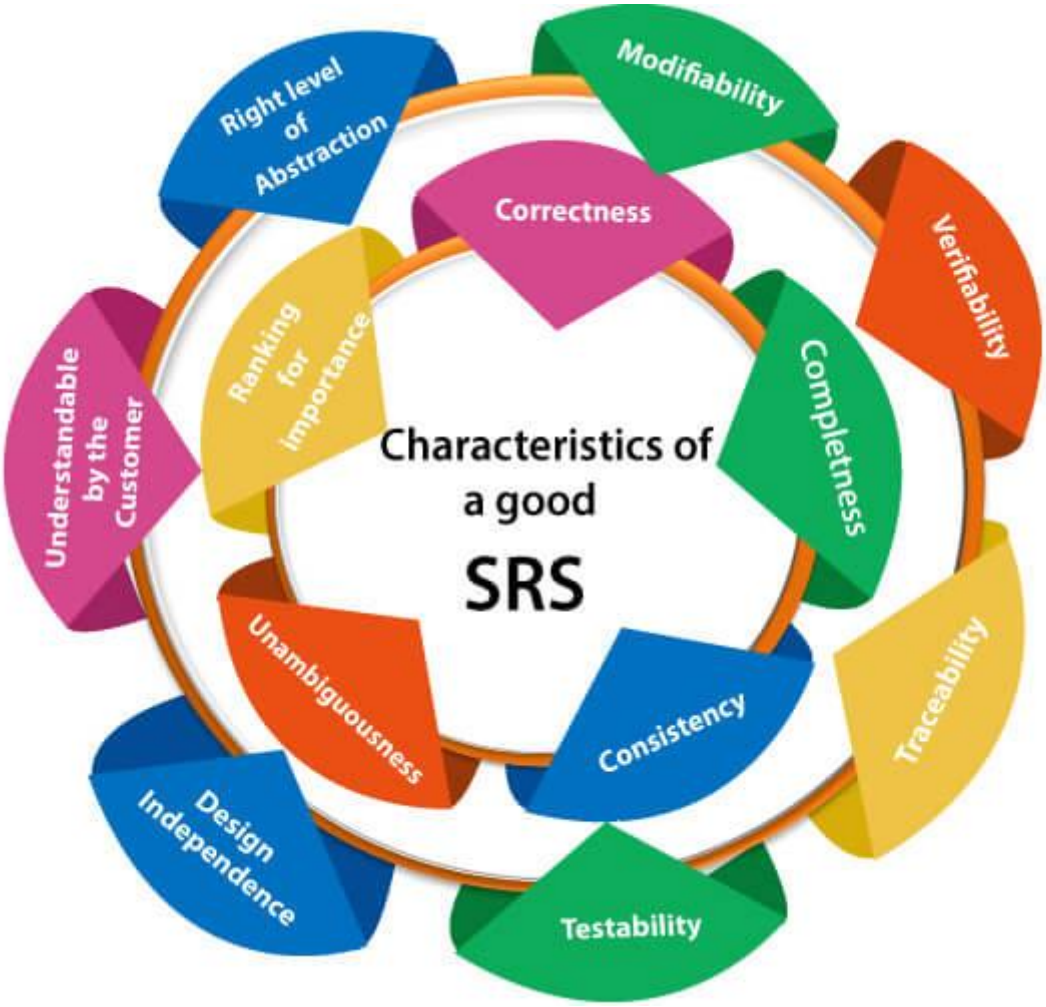


Software Requirement Specifications

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.

Note: It is essential to specify the responses to both valid and invalid values.

(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.