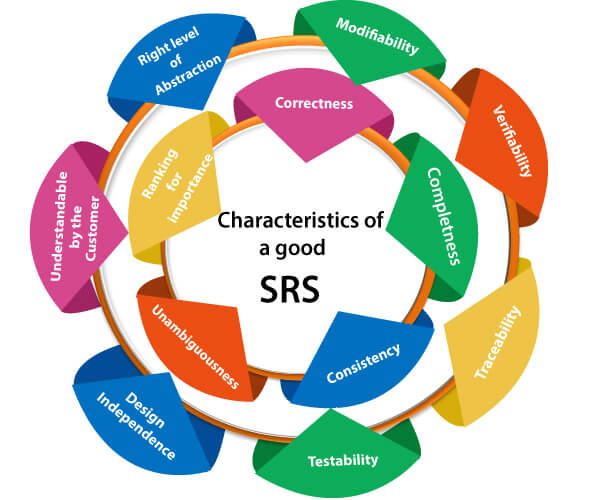
**Unit II: REQUIREMENTS ANALYSIS AND SPECIFICATION**

**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.

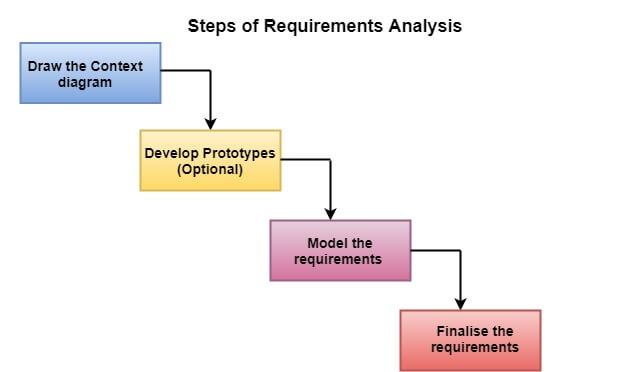
**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

**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.

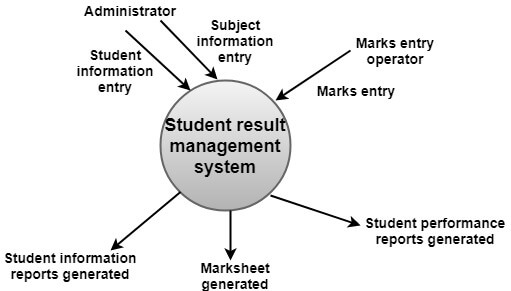
**Requirements Analysis**

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

**The various steps of requirement analysis are shown in fig:**



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



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

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

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

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

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

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

# 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.

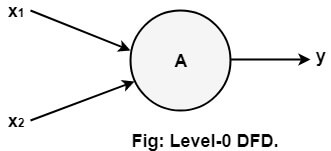
**Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

## Levels in Data Flow Diagrams (DFD)

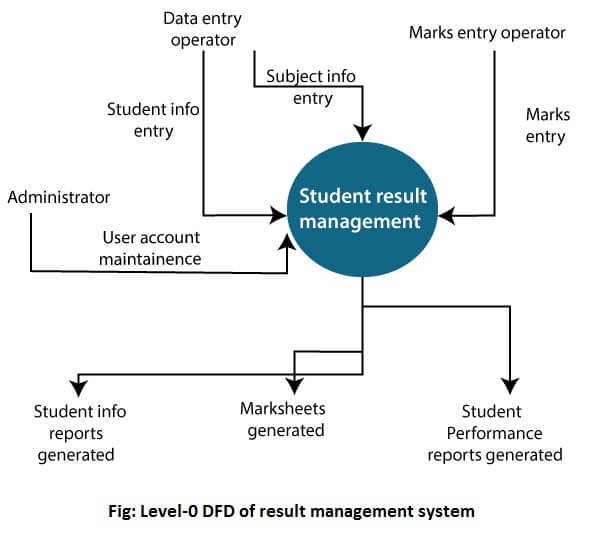
The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram,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

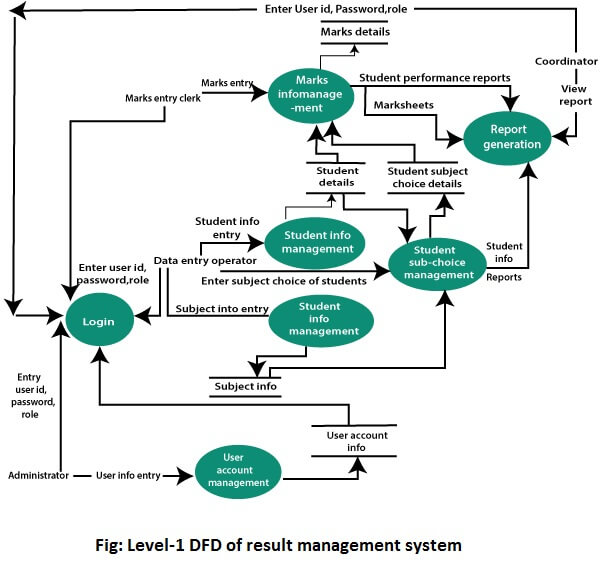


The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.



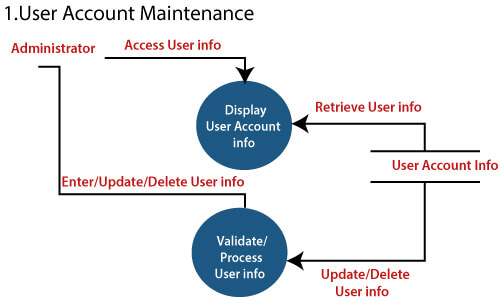
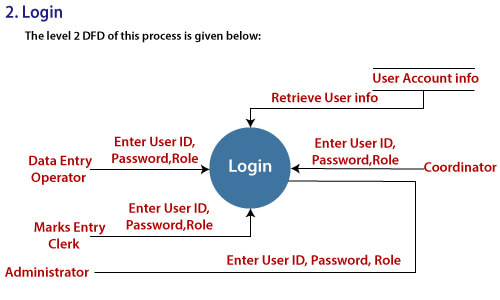
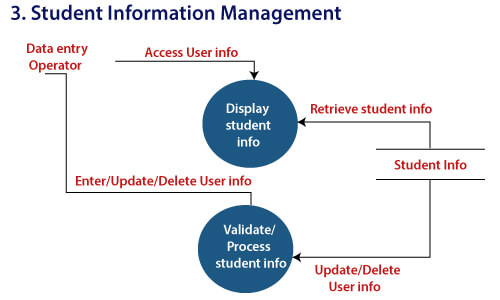
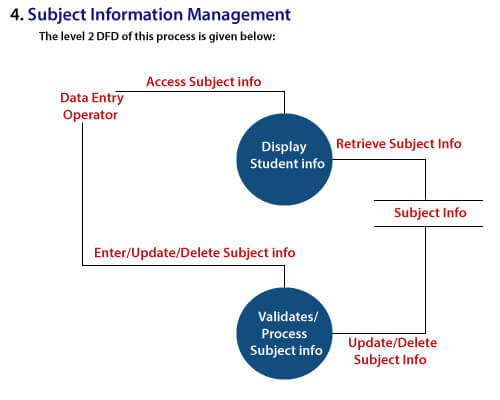
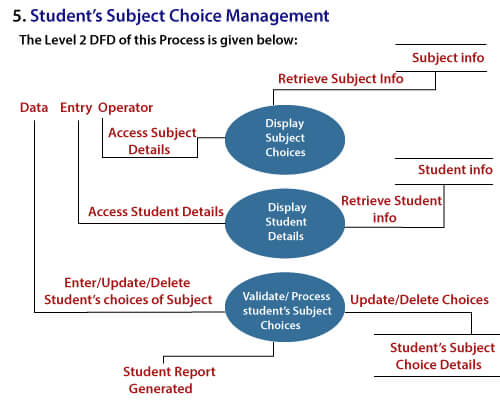
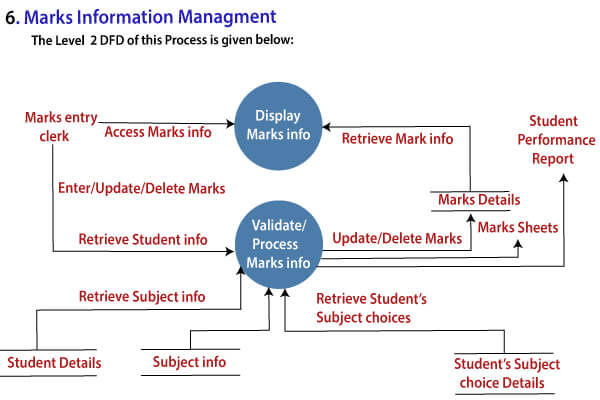
**1-level DFD**

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.

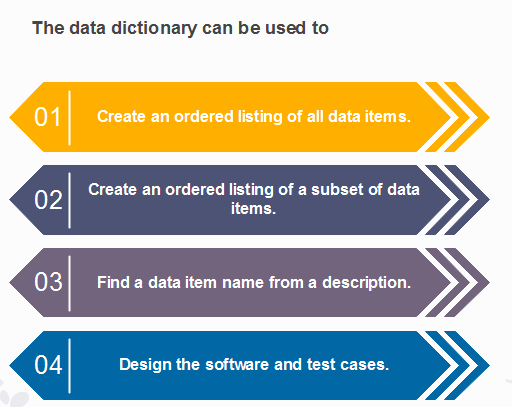
**Related data items** capture relationships between data items e.g., total\_marks must always equal to internal\_marks plus external\_marks.

**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. |



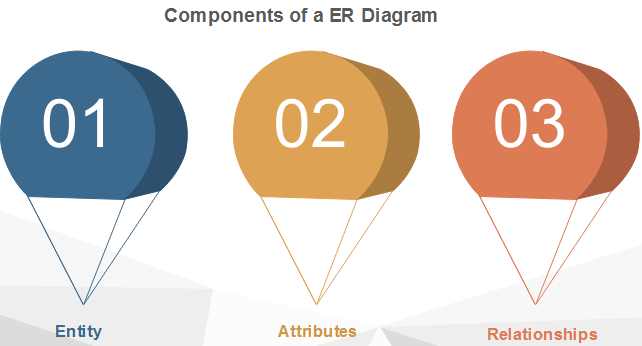
# Entity-Relationship Diagrams

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**

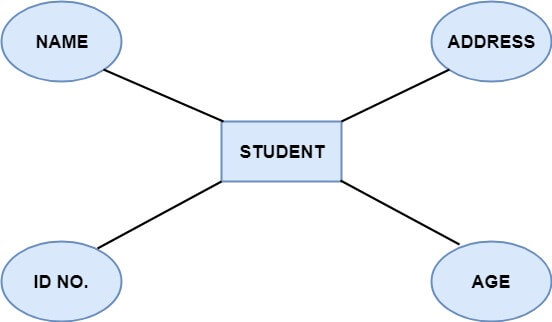
An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need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

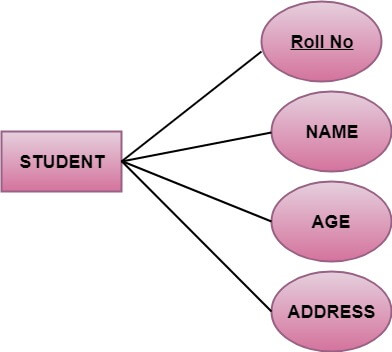
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



**There are four types of Attributes:**

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

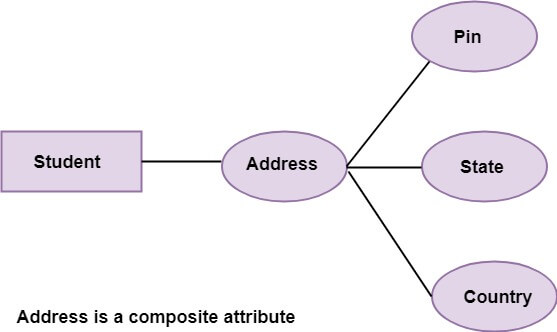
**1. Key attribute:** Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll\_number of a student makes him identifiable among students.



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

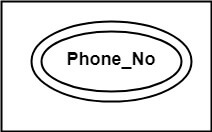
1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.
2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
3. **Primary key:** A primary key is one of the candidate keys chosen by the database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

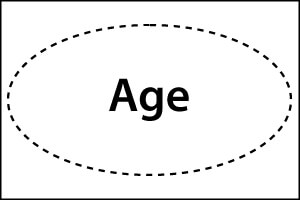
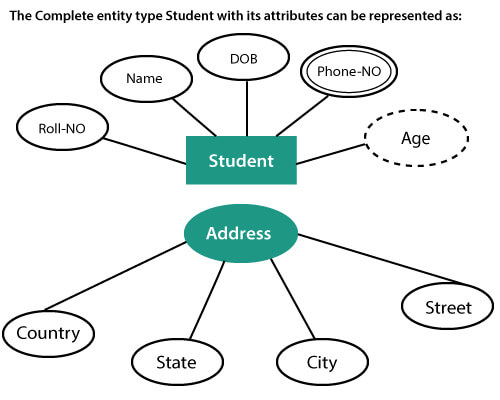


**3. Single-valued attribute:** Single-valued attribute contain a single value. For example, Social\_Security\_Number.

**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.

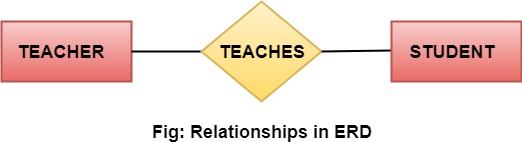


**5. Derived attribute:** Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date\_of\_birth. In the ER diagram, Derived attributes are depicted by the dashed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

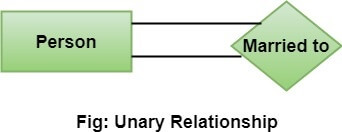
A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

## Degree of a relationship set

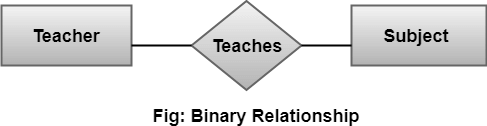
The number of participating entities in a relationship describes the degree of the relationship. The three most common relationships in E-R models are:

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.

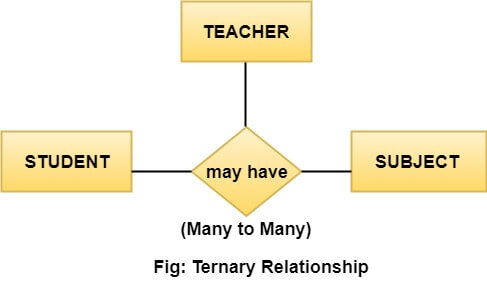


**2. Binary relationship:** It is a relationship between the instances of two entity types. For example, the Teacher teaches the subject.



**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

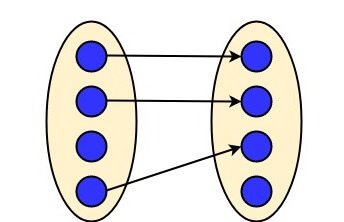
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

### Types of Cardinalities

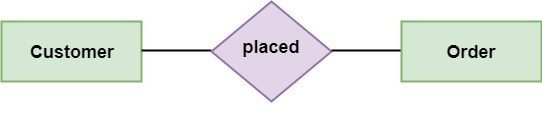
**1. One to One:** One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.



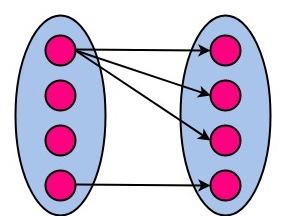
**Using Sets, it can be represented as:**



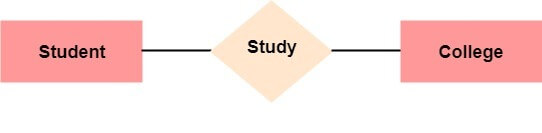
**2. One to many:** When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.



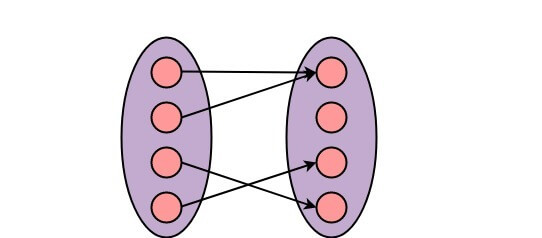
**Using Sets, it can be represented as:**



**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.



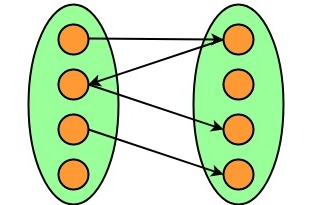
**Using Sets, it can be represented as:**



**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:**



**Functional vs Non Functional Requirements**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 29 Apr, 2020

Requirements analysis is 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*.

**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 basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

**Non-functional requirements:** These are basically 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 other. They are also called non-behavioral requirements.  
They basically deal with issues like:

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

Following are the differences between Functional 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 |

**oftware Requirement Specification (SRS) Format**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 13 Dec, 2021

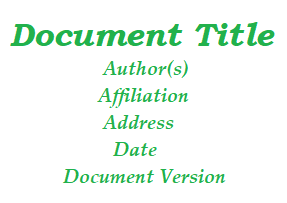
In order to form a [good SRS](https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-good-srs/), here you will see some points which can be used and should be considered to form a structure of good SRS. These are as follows :

1. Introduction

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

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

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



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

1. **Introduction :**
   * **(i) Purpose of this Document –**  
     At first, main aim of why this document is necessary and what’s purpose of document is explained and described.
   * **(ii) Scope of this document –**  
     In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
   * **(iii) Overview –**  
     In this, description of product is explained. It’s simply summary or overall review of product.
2. **General description :**  
   In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
3. **Functional Requirements :**  
   In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
4. **Interface Requirements :**  
   In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.
5. **Performance Requirements :**  
   In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.
6. **Design Constraints :**  
   In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc.
7. **Non-Functional Attributes :**  
   In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
8. **Preliminary Schedule and Budget :**  
   In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
9. **Appendices :**  
   In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

**Software Engineering | Requirements Engineering Process**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 27 Feb, 2020

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. 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 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 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 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.

**ypes of Feasibility Study in Software Project Development**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 21 Sep, 2021

**Feasibility Study** in [Software Engineering](https://www.geeksforgeeks.org/software-engineering-introduction-to-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](https://www.geeksforgeeks.org/software-engineering-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.

1. **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.

1. **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.

1. **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.

1. **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.

**Feasibility Study Process :**   
The below steps are carried out during entire feasibility analysis. 

1. Information assessment
2. Information collection
3. Report writing
4. General information

**Need of Feasibility Study :**   
Feasibility study is so important stage of [Software Project Management Process](https://www.geeksforgeeks.org/software-engineering-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.

**What is Feasibility Study? Types of Feasibility. Explain Feasibility Study Process**

By Dinesh Thakur

**Feasibility** is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. [Information](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

• To analyze whether the software will meet organizational requirements.

• To determine whether the software can be implemented using the current technology and within the specified budget and schedule.

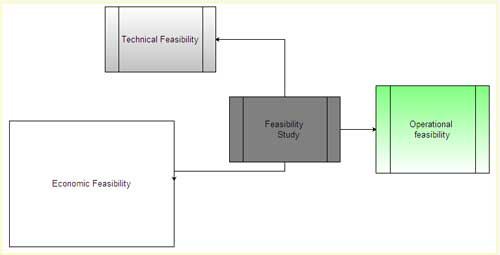
• To determine whether the software can be integrated with other existing software.

We’ll be covering the following topics in this tutorial:

* [Types of Feasibility](https://ecomputernotes.com/software-engineering/feasibilitystudy#Types_of_Feasibility)
* [Feasibility Study Process](https://ecomputernotes.com/software-engineering/feasibilitystudy#Feasibility_Study_Process)

**Types of Feasibility**

Various types of feasibility that are commonly considered include technical feasibility, operational feasibility, and economic feasibility.



Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements. Technical feasibility also performs the following tasks.

• Analyzes the technical skills and capabilities of the software development team members.

• Determines whether the relevant technology is stable and established.

• Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

Operational feasibility assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed. Operational feasibility also performs the following tasks.

• Determines whether the problems anticipated in user requirements are of high priority.

• Determines whether the solution suggested by the software development team is acceptable.

• Analyzes whether users will adapt to a new software.

• Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

Economic feasibility determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software. Software is said to be economically feasible if it focuses on the issues listed below.

• Cost incurred on software development to produce long-term gains for an organization.

• Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis).

• Cost of hardware, software, development team, and training.

**Feasibility Study Process**

Feasibility study comprises the following steps.

**•** [**Information**](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) **assessment:** Identifies information about whether the system helps in achieving the objectives of the organization. It also verifies that the system can be implemented using new technology and within the budget and whether the system can be integrated with the existing system.

**• Information collection:** Specifies the sources from where information about software can be obtained. Generally, these sources include users (who will operate the software), organization (where the software will be used), and the software development team (which understands user requirements and knows how to fulfill them in software).

**• Report writing:** Uses a feasibility report, which is the conclusion of the feasibility study by the software development team. It includes the recommendations whether the software development should continue. This report may also include information about changes in the software scope, budget, and schedule and suggestions of any requirements in the system.

**• General information:** Describes the purpose and scope of feasibility study. It also describes system overview, project references, acronyms and abbreviations, and points of contactto be used. **System overview** provides description about the name of the organization responsible for the software development, system name or title, system category, operational status, and so on. **Project references** provide a list of the references used to prepare this document such as documents relating to the project or previously developed documents that are related to the project. **Acronyms and abbreviations** provide a list of the terms that are used in this document along with their meanings. **Points of contact** provide a list of points of organizational contact with users for information and coordination. For example, users require assistance to solve problems (such as troubleshooting) and collect information such as contact number, e-mail address, and so on.

**Management summary:** Provides the following information.

**• Environment:** Identifies the individuals responsible for software development. It provides information about input and output requirements, processing requirements of the software and the interaction of the software with other software. It also identifies system security requirements and the system’s processing requirements

**• Current functional procedures:** Describes the current functional procedures of the existing system, whether automated or manual. It also includes the data-flow of the current system and the number of team members required to operate and maintain the software.

**• Functional objective:** Provides information about functions of the system such as new services, increased capacity, and so on.

**• Performance objective:** Provides information about performance objectives such as reduced staff and equipment costs, increased processing speeds of software, and improved controls.

**• Assumptions and constraints:** Provides information about assumptions and constraints such as operational life of the proposed software, financial constraints, changing hardware, software and operating environment, and availability of information and sources.

**• Methodology:** Describes the methods that are applied to evaluate the proposed software in order to reach a feasible alternative. These methods include survey, modeling, benchmarking, etc.

**• Evaluation criteria:** Identifies criteria such as cost, priority, development time, and ease of system use, which are applicable for the development process to determine the most suitable system option.

**• Recommendation:** Describes a recommendation for the proposed system. This includes the delays and acceptable risks.

**• Proposed software:** Describes the overall concept of the system as well as the procedure to be used to meet user requirements. **In** addition, it provides information about improvements, time and resource costs, and impacts. Improvements are performed to enhance the functionality and performance of the existing software. Time and resource costs include the costs associated with software development from its requirements to its maintenance and staff training. Impacts describe the possibility of future happenings and include various types of impacts as listed below.

**• Equipment impacts:** Determine new equipment requirements and changes to be made in the currently available equipment requirements.

**• Software impacts:** Specify any additions or modifications required in the existing software and supporting software to adapt to the proposed software.

**• Organizational impacts:** Describe any changes in organization, staff and skills requirement.

**• Operational impacts:** Describe effects on operations such as user-operating procedures, data processing, data entry procedures, and so on.

**• Developmental impacts:** Specify developmental impacts such as resources required to develop databases, resources required to develop and test the software, and specific activities to be performed by users during software development.

**• Security impacts:** Describe security factors that may influence the development, design, and continued operation of the proposed software.

**• Alternative systems:** Provide description of alternative systems, which are considered in a feasibility study. This also describes the reasons for choosing a particular alternative system to develop the proposed software and the reason for rejecting alternative systems.

**Requirements Elicitation & Analysis**

It’s a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constraints.

Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.

It may also involve different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

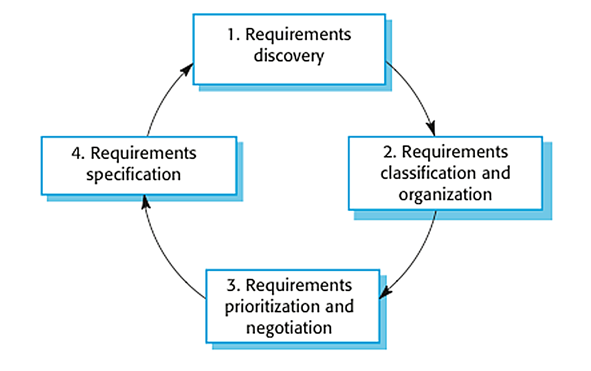
A stakeholder is anyone who has direct or indirect influence on the requirements.

**The requirements elicitation and analysis has 4 main process**

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub-components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may arise from some conflicts.

Here are the 4 main processes of requirements elicitation and analysis.



The process of requirements elicitation and analysis — From [Software Engineering, 9th edition, Chapter 4](https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html), by [Ian Sommerville](https://sites.google.com/site/iansommerville/home)

It shows that it’s an iterative process with feedback from one activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

**1. Requirements Discovery**

It’s the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exists).

It can be done using some techniques, like interviews, scenarios, prototypes, etc, which help the stockholders to understand what the system will be like.

**Gathering and understanding the requirements is a difficult process**

That’s because stakeholders may not know what exactly they want the software to do, or they may give unrealistic requirements.

They may give different requirements, which will result in conflict between the requirements, so we have to discover and resolve these conflicts.

Also, there might be some factors that influence the stakeholder’s decision, like, for example, managers at a company or professors at the university want to take full control over the management system.

**Interviews**

In Interviews, requirements engineering teams put the questions to the stakeholder about the system that’s currently used, and the system to be developed, and hence they can gather the requirements from the answers.

The questions fall under two categories:

1. *Closed-Ended* Questions: A pre-defined set of questions.
2. *Open-Ended* questions: There is no pre-defined expected answer, they are more generic questions. It’s used to explore issues that are not clear in a less structured way.

In practice, interviews usually use a mixture of both. Usually, start with the open-ended questions, and then use the closed-ended questions to be more specific about some requirements that aren’t clear yet.

Interviews are good to get an overall understanding of what stakeholders need, how they might interact with the new system and the difficulties they face with the current system.

However, interviews aren’t so helpful in understanding the domain requirements. This is for two reasons:

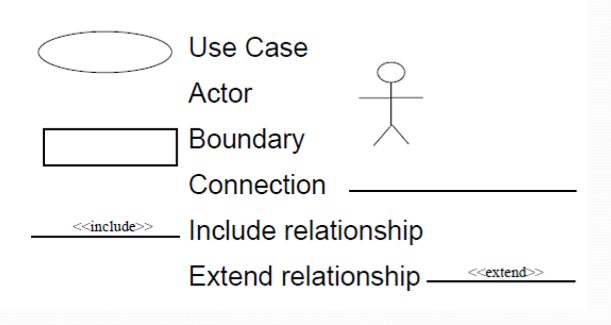
1. Domain requirements may be expressed using special domain terminologies, and software engineers often find it difficult to understand and it’s easy for them to misunderstand.
2. Sometimes stakeholders won’t tell you some requirements because they assume it’s so fundamental and it doesn’t worth mentioning, or they find it difficult to explain, which won’t be taken into consideration in the requirements.

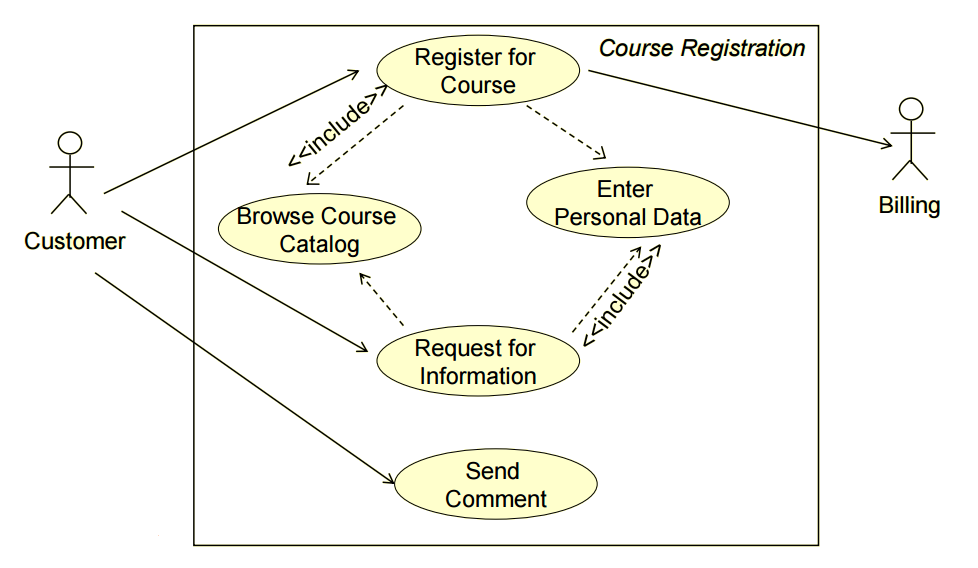
**Use Cases & Scenarios**

The use cases and scenarios are two different techniques, but, usually, they are used together.

Use cases identify interactions between the system and its users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.

Use case involves some symbols to describe the system:





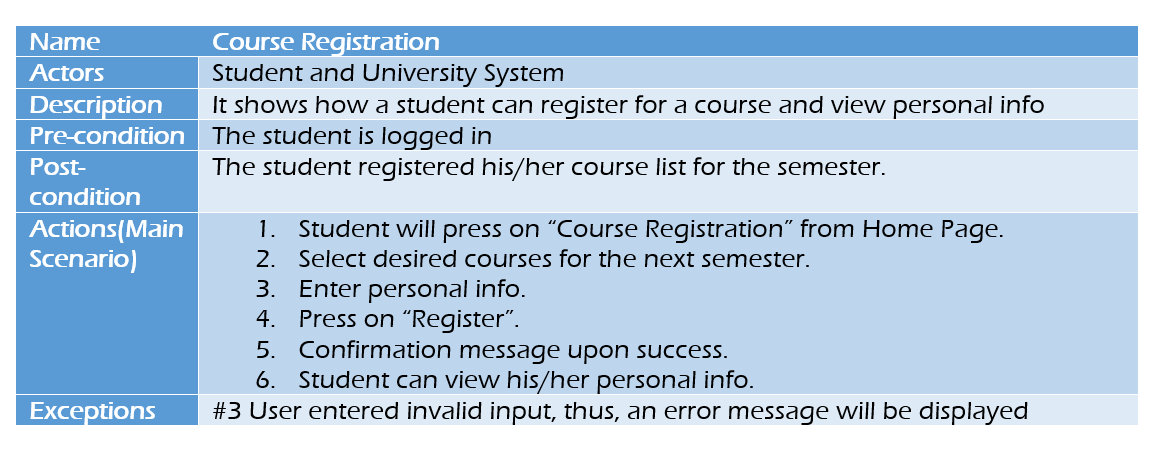
Use case diagram symbols and an example

1. **Actors**: Are those who interact with the system; human or other systems
2. **Interaction (Use Case)**: It denotes the name of the interaction (verb). It’s represented as a named ellipse.
3. **Connection**: Lines that link between the actors and the interactions.
4. **Include** **Relationship**: It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude** **Relationship**: It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on its own without the extending interaction.

Now, we are going to use scenarios to describe the interactions in each use case textually. They should have a format and include the following:

1. A description of the initial situation.
2. A description of the flow of the events or interactions with the system
3. A description of the exceptions, or in other words, what can go wrong, and how it can be handled.
4. Any concurrent activities should be mentioned
5. A description of the final state.

Here is the example for a scenario for the use case example above.



Scenario

Use cases and scenarios are effective techniques for eliciting the requirements. But, because they focus on the interactions with the system, they aren’t effective for eliciting high-level business, non-functional, or domain requirements.

The next two phases are about analyzing requirements: determining whether the stated requirements are clear, complete, consistent, and unambiguous, group-related requirements and organizing them into related components, and resolving any apparent conflicts.

**2. Requirements Classification & Organization**

It’s very important to organize the overall structure of the system.

Putting related requirements together, and decomposing the system into sub-components of related requirements. Then, we define the relationship between these components.

What we do here will help us in the decision of identifying the most suitable architectural design patterns.

**3. Requirements Prioritization & Negotiation**

We previously explained why eliciting and understanding the requirements is not an easy process.

One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. *Why?* because it’s hard to satisfy all parties if it’s not impossible.

This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.

We shouldn’t reach a situation where a stakeholder is not satisfied because his requirements are not taken into consideration.

Prioritizing your requirements will help you later to focus on the essentials and core features of the system, so you can meet the user expectations. It can be achieved by giving every piece of function a priority level. So, functions with higher priorities need higher attention and focus*.*

**4. Requirements Specification**

The requirements are then documented. We’ll discuss requirements specifications in more detail in “[Requirements Engineering — Requirements Specification](https://medium.com/omarelgabrys-blog/requirements-engineering-elicitation-analysis-part-5-2dd9cffafae8)”.

**Software Engineering | Requirements Validation Techniques**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 25 Apr, 2019

**Requirements validation** is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.

In the requirements validation process, we perform a different type of test to check the requirements mentioned in the [Software Requirements Specification (SRS)](https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-good-srs/), these checks include:

* Completeness checks
* Consistency checks
* Validity checks
* Realism checks
* Ambiguity checks
* Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**  
   Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.
2. **Prototyping:**  
   In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.
3. **Requirements Reviews:**  
   In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.
4. **Automated Consistency Analysis:**  
   This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

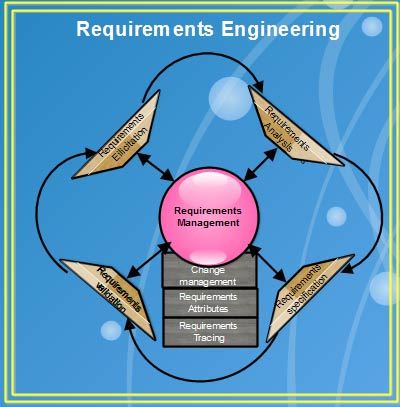
1. **Walk-through:**  
   A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
   * Checking early whether the idea is feasible or not.
   * Obtaining the opinions and suggestion of other people.
   * Checking the approval of others and reaching an agreement.

**Requirements Management Process in Software Engineering**

By Dinesh Thakur

Once a system has been deployed, new requirements inevitably emerge. It is difficult for the users to anticipate the effect of these new requirements (if a new system is developed for these requirements) on the organization. Thus, to understand and control changes to system requirements, requirements management is performed.

**Requirements management** can be defined as a process of eliciting, documenting, organizing, and controlling changes to the requirements. Generally, the process of requirements management begins as soon as the requirements document is available, but ‘planning’ for managing the changing requirements should start during the requirements elicitation process.



The essential activities performed in requirements management are listed below.

1. Recognizing the need for change in the requirements
2. Establishing a relationship amongst stakeholders and involving them in the requirements engineering process
3. Identifying and tracking requirements attributes.

Requirements management enables the development team to identify, control, and track requirements and changes that occur as the software development process progresses. Other advantages associated with the requirements management are listed below.

1. **Better control of complex projects:** This provides the development team with a clear understanding of what, when, and why the software is to be delivered. The resources are allocated according to user-driven priorities and relative implementation effort.
2. **Improved software quality:** This ensures that the software performs according to the requirements to enhance software quality. This can be achieved when the developers and testers have a precise understanding of what to develop and test.
3. **Reduced project costs and delays:** This minimizes errors early in the development cycle as it is expensive to ‘fix’ errors at the later stages of the development cycle. As a result, the project costs also reduce.
4. **Improved team communication:** This facilitates early involvement of users to ensure that their needs are achieved.
5. **Easing compliance with standards and regulations:** This ensures that standards involved with software compliance and process improvement have a thorough understanding of requirements management. For example, CMM addresses requirements management as one of the first steps to improve software quality.
6. All the user requirements are specified in the software requirements specification. The project manager as part of requirements management tracks the requirements for the current project and those which are planned for the next release.

We’ll be covering the following topics in this tutorial:

* [Requirements Management Process](https://ecomputernotes.com/software-engineering/requirementsmanagementprocess#Requirements_Management_Process)
* [Requirements Change Management](https://ecomputernotes.com/software-engineering/requirementsmanagementprocess#Requirements_Change_Management)

**Requirements Management Process**

Requirements management starts with planning, which establishes the level of requirements management needed. After planning, each requirement is assigned a unique ‘identifier’ so that it can be crosschecked by other requirements. Once requirements are identified, requirements tracing is performed.

**Requirements tracing** is a medium to trace requirements from the start of development process till the software is delivered to the user. The objective of requirements tracing is to ensure that all the requirements are well understood and included in test plans and test cases. Various advantages of requirements tracing are listed below.

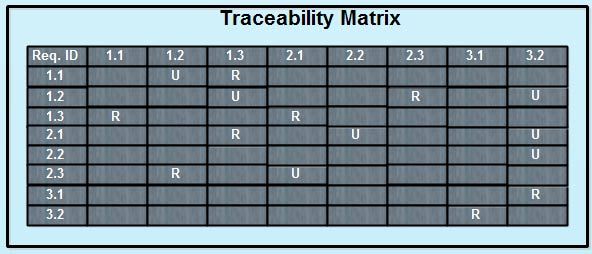
1. It verifies whether user requirements are implemented and adequately tested.
2. It enables user understanding of impact of changing requirements.

Trace ability techniques facilitate the impact of analysis on changes of the project, which is under development. Traceability [information](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) is stored in a traceability matrix, which relates requirements to stakeholders or design module. The traceability matrix refers to a table that correlates high-level requirements with the detailed requirements of the product. Mainly, five types of traceability tables are maintained. These are listed in Table.

In a traceability matrix, each requirement is entered in a row and column of the matrix. The dependencies between different requirements are represented in the cell at a row and column intersection. ‘U’ in the row and column intersection indicates the dependencies of the requirements in the row on the column and ‘R’ in the row and column intersection indicates the existence of some other weaker relationship between the requirements.

**Table Types of Traceability Tables**

|  |  |
| --- | --- |
| **Traceability Table** | **Description** |
| Features traceability | Indicates how requirements relate to important features specified by the user. |
| Source traceability | Identifies the source of each requirement by linking the requirements to the stakeholders who proposed them. When a change is proposed, [information](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) from this table can be used to find and consult the stakeholders. |
| Requirements traceability | Indicates how dependent requirements in the SRS are related to one another. Information from this table can be used to evaluate the number of requirements that will be affected due to the proposed change(s). |
| Design traceability | Links the requirements to the design modules where these requirements are implemented. Information from this table can be used to evaluate the impact of proposed requirements changes on the software design and implementation. |
| Interface traceability | Indicates how requirements are related to internal interface and external interface of a system. |



Note that a traceability matrix is useful when less number of requirements are to be managed. However, traceability matrices are expensive to maintain when a large system with large requirements is to be developed. This is because large requirements are not easy to manage. Due to this, the traceability information of large system is stored in the ‘requirements [database](https://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms)’ where each requirement is explicitly linked to related requirements. This helps to assess how a change in one requirement affects the different aspects of the system to be developed.

**Requirements Change Management**

Requirements change management is used when there is a request or proposal for a change in the requirements. The advantage of this process is that the changes to the proposals are managed consistently and in a controlled manner. Note that many activities of requirements management are similar to software configuration management activities.

An efficient requirements change management process undergoes a number of stages for changes to the requirements. These stages are listed below.

1. **Problem analysis and change specification:** The entire process begins with identification of problems to the requirements. The problem or proposal is analyzed to verify whether the change is valid. The outcome of the analysis is provided to the ‘change requester’ and a more specific requirements change proposal is then made.
2. **Change analysis and costing:** The effect of a change requested on the requirement is assessed according to traceability information. The cost for this can be estimated on the basis of modification made to the design and implementation. After the analysis is over, a decision is made whether changes are to be made.
3. **Change implementation:** Finally, the changes are made to the requirements document, system design and implementation. The requirements document is organized in such a manner so that changes to it can be made without extensive rewriting. Minimizing the external references and making document sections modular achieves changeability in the document. By doing this, individual sections can be changed and replaced without affecting other parts of the document.



**Petri Nets-Data Dictionary**

Petri nets — Formal technique for describing concurrent interrelated activities

Consists of four parts

(1)             A set of places

(2)             A set of transitions

(3)             An input function

(4)             An output function

Originally of interest to automata theorists Found wide applicability in computer science

Performance evaluation Operating systems

Marking of a Petri net Assignment of tokens Tokens enable transitions

Petri nets are non-deterministic

If more than one transition is able to fire, then any one can be fir

Use of inhibitor arcs an important extension: Small circle instead of arrow

A transition is enabled if

At least one token is in each of its (normal) input arcs and No tokens on any of its inhibitor input arcs