

4.1 Requirement Engineering

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

Requirements Engineering Process consists of the following main activities(processes):

Requirements elicitation

This is also known as the gathering of requirements. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc.

Requirements specification

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), 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.
- Reviews, buddy checks (involves two members; one from the development team and one from the testing team), making test cases, etc. are some of the methods used for this.

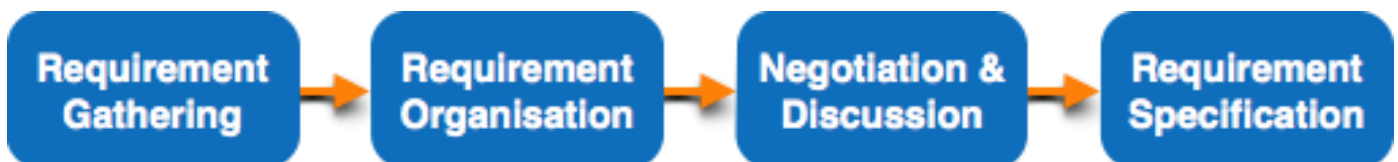
Requirements management

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 in case of end users changing their mind.

4.2 Requirement Elicitation

- Requirement Elicitation is the process of discovering the requirements of a system.
- It is basically collecting requirement from stakeholders, user and customers by conducting meetings, interviews etc.
- It deals with all the activities required in gathering the requirements of a system.
- The developers and engineers work in close coordination with the customers and end users to identify more about the problem to be solved and to bridge the gap between expectation and understanding.

Requirement elicitation process can be depicted using the following diagram:



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

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

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

Some Requirement Elicitation Techniques

- a) Interview
- b) Brainstorming
- c) Use case approach

4.2.1 Interviews

- Interview is conducted basically to understand customer's expectations from software.
- A team conducts interview with different people such as:
 - Managers
 - Stakeholders
 - Users
- It is not possible to conduct interview with each and every person. Hence based on experience and expertise representatives are selected from different groups.
- There are different ways of conducting interviews:
 - Open-ended interviews: There is no pre-set agenda. Context free questions may be asked to understand the problem.
 - Structured interview: Agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.
 - Written interviews
 - Group interviews

4.2.2 Brainstorming series

- Understand customer's expectation with group technique.
- In this different members of groups are invited for the group discussion.
- One by one everyone puts their point like GD in companies.
- It helps to generate variety of new ideas hence providing a platform to share views.
- There are two phases:
 - Generation Phase
 - Consolidation Phase

- Facilitator is there for handling conflicts between people.
- 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.
- Thus at the end a conclusion for a specific problem is gathering a list of ideas.

4.2.3 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 component of the use case design includes 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.
- -----> Stereotype(relationship)
- _____> Generalization

4.3 Requirement analysis


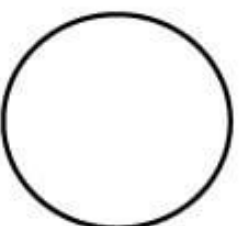

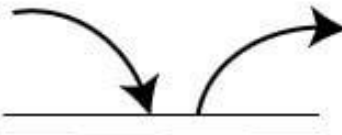
Software requirement analysis means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. Requirement analysis helps organizations to determine the actual needs of stakeholders.

4.3.1 Data flow diagram

- 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.

The following observations about DFDs are essential:

- All names should be unique.
- A DFD does not involve any order of events.
- Suppress logical decisions.
- Do not become bogged down with details.

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Standard symbols for DFDs are:

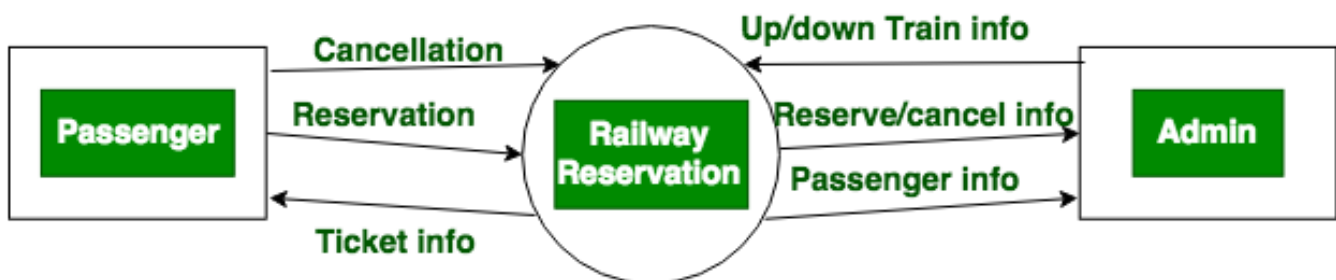
-

- 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 of DFD

0-level DFD:

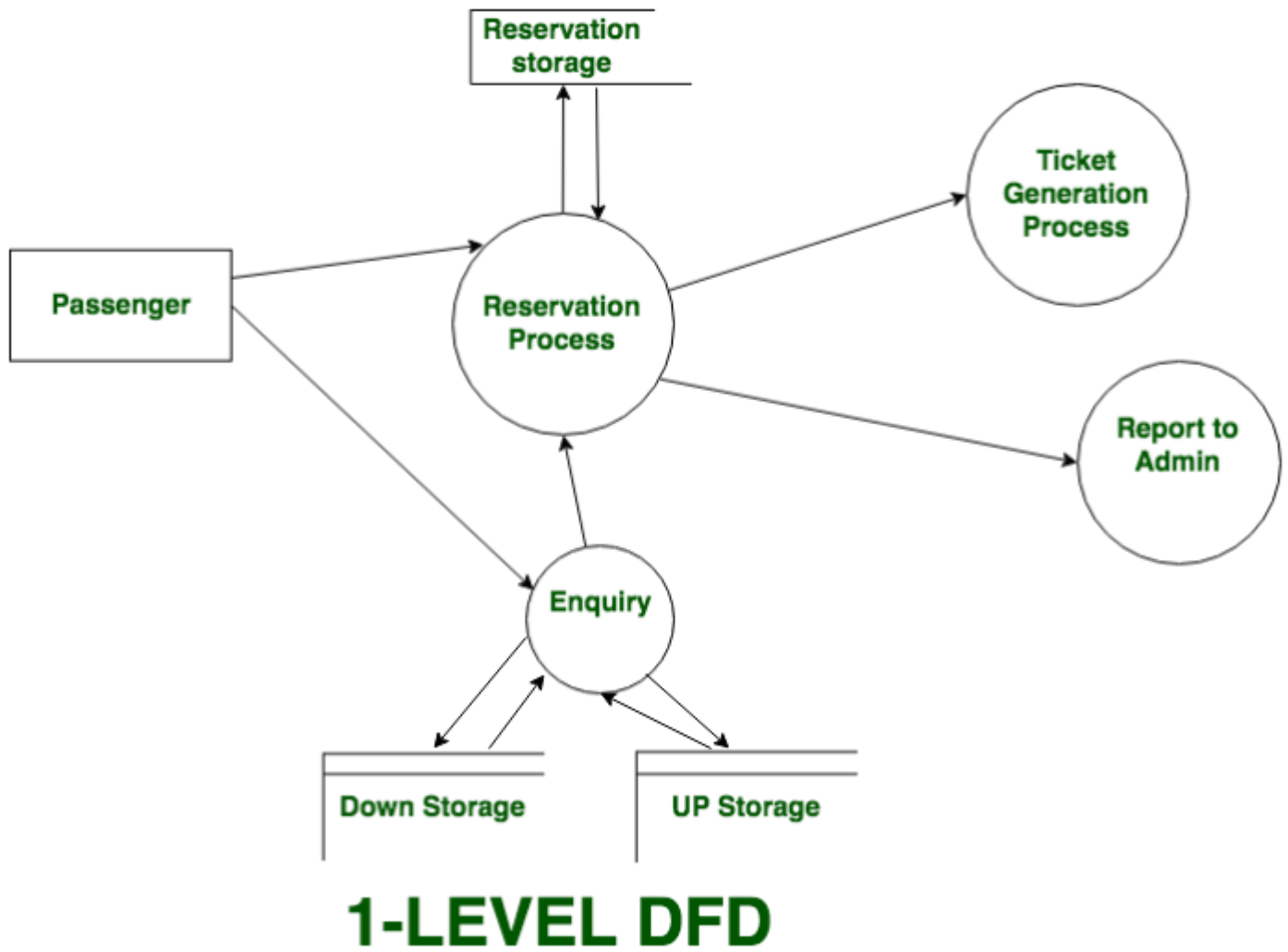
It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.



0-LEVEL DFD

1-level DFD:

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



2-level DFD:

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

Range of values – It will represent all possible answers.

Data Dictionary Notations tables :

<u>Notations</u>	<u>Meaning</u>
$X = a+b$	X consists data elements a and b.
$X = [a/b]$	X consists of either elements a or b.
$X = a X$	X consists of optimal(best) data elements a.
$X = y[a]$	X consists of y or more events of data element a
$X = [a] z$	X consists of z or less events of data element a
$X = y [a] z$	X consists of some events of data elements between y and z.

Features of Data Dictionary:

- It helps in designing test cases and designing the software.
- It is very important for creating an order list from a subset of the items list.
- It is very important for creating an order list from a complete items list.
- The data dictionary is also important to find the specific data item object from the list.

4.3.3 Entity-Relationship Diagram

- Entity relationship diagrams are used in software engineering during the planning stages of the software project.
- They help to identify different system elements and their relationships with each other.
- It is often used as the basis for data flow diagrams or DFD's as they are commonly known.

For example, an inventory software used in a retail shop will have a database that monitors elements such as purchases, item, item type, item source and item price. Rendering this information through an ER diagram would be something like this:

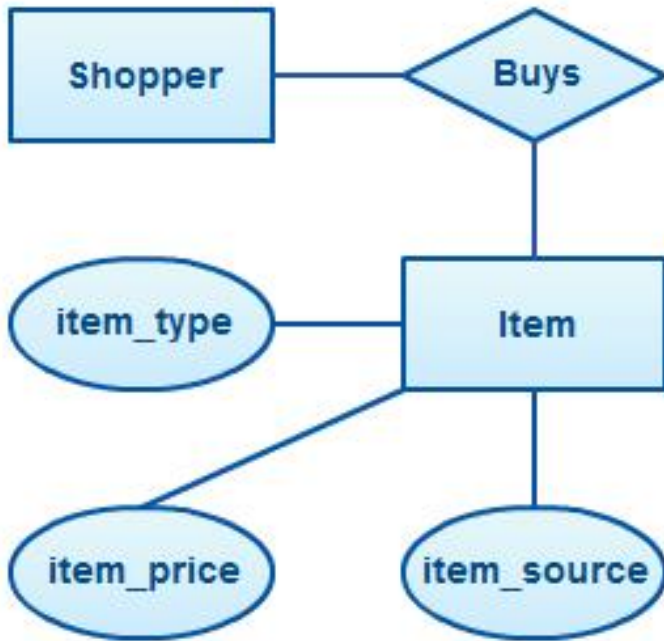


Fig: ER diagram example with entity having attributes

Entity Relationship Diagram (ERD) Symbols and Notations

Entity - An entity can be a person, place, event, or object that is relevant to a given system

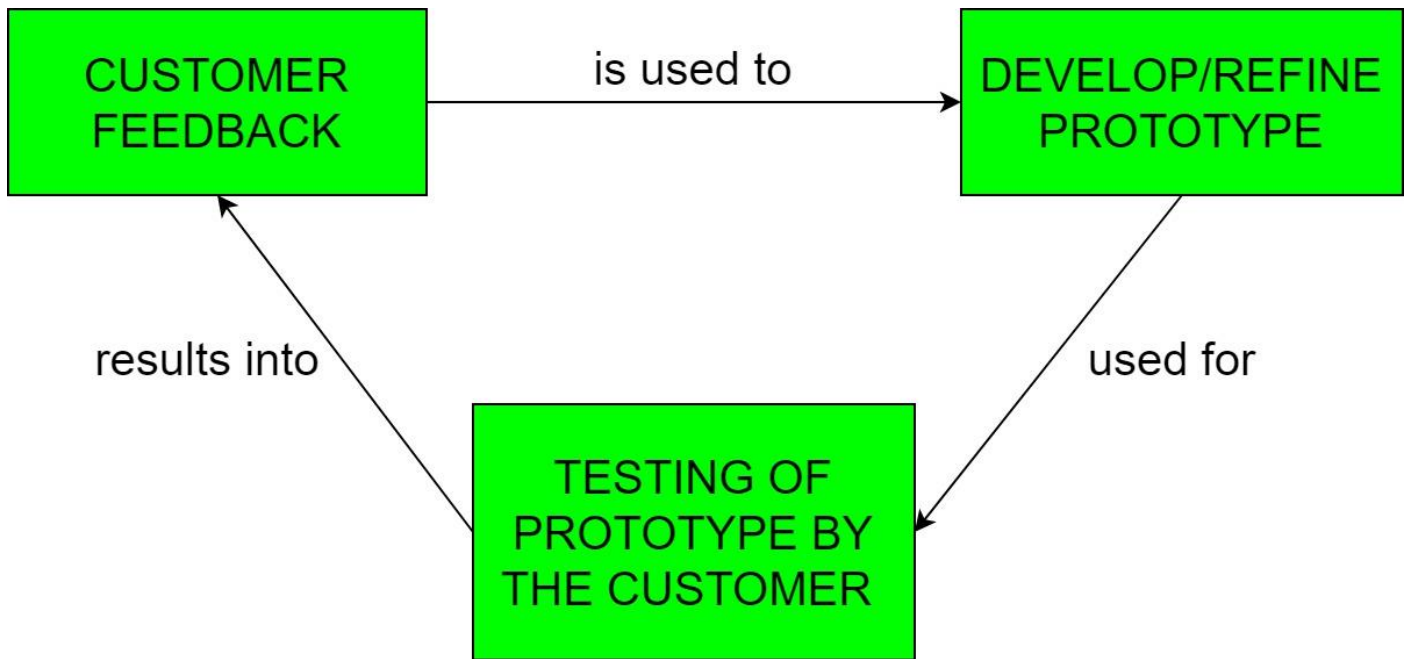
Weak Entity - A weak entity is an entity that depends on the existence of another entity.

Attributes - An attribute is a property, trait, or characteristic of an entity, relationship, or another attribute.

- **Multivalued Attribute** - If an attribute can have more than one value it is called a multi-valued attribute.
- **Derived Attribute** - An attribute based on another attribute. This is found rarely in ER diagrams. For example, for a circle, the area can be derived from the radius.

Relationship - A relationship describes how entities interact.

4.3.4 Software prototyping



In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle.

There **are four types of model** available:

A) Rapid Throwaway Prototyping – This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype.

B) Evolutionary Prototyping – In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

C) Incremental Prototyping – In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order.

Advantages –

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier

Disadvantages –

- Costly w.r.t time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.

4.4 Requirement documentation

Software documentation is a written piece of text that is often accompanied with a software program. This makes the life of all the members associated with the project more easy.

For example before development of any software product requirements are documented which is called as Software Requirement Specification (SRS).

Types Of Software Documentation :

Requirement Documentation : It is the description of how the software shall perform and which environment setup would be appropriate to have the best out of it.

Architectural Documentation : Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles and working.

Technical Documentation : These contain the technical aspects of the software like API, algorithms etc. It is prepared mostly for the software devs.

End-user Documentation : As the name suggests these are made for the end user. It contains support resources for the end user.

4.4.1 Nature of SRS

Nature of Software Requirement Specification (SRS):

The basic issues that SRS writer shall address are the following:

1. **Functionality:** What the software is supposed to do?

2. **External Interfaces:** How does the software interact with people, system's hardware, other hardware and other software?
3. **Performance:** What is the speed, availability, response time, recovery time etc.
4. **Attributes:** What are the considerations for portability, correctness, maintainability, security, reliability etc.
5. **Design Constraints Imposed on an Implementation:** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment etc.

4.4.2 Characteristics of SRS (In Unit 2 Notes)

4.4.3 Organization of a SRS

It depends to a large extent on the system analyst, S/he is often guided by policies & standards followed by company.

Also, the organization of document & issues to a large extent depend on type of product being developed.

However, irrespective of company principles & product type, three basic issue that any SRS document should discuss are:-

1. Functional Requirement

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.

For example, in a hospital management system, a doctor should be able to retrieve the information of his patients.

2. Non- functional requirement

These are basically the quality constraints that the system must satisfy according to the project contract. They basically deal with issues like:

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

3. Domain requirements:

Domain requirements are the requirements which are characteristic of a particular category or domain of projects. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement.