# Chapter 4

## ✓ *Requirements Engineering*

Requirements describe

**What not How**

Produces one large document written in natural language contains a description of what the system will do without describing how it will do it.

Crucial process steps

Quality of product → Process that creates it

Without well written document

Developers do not know what to build

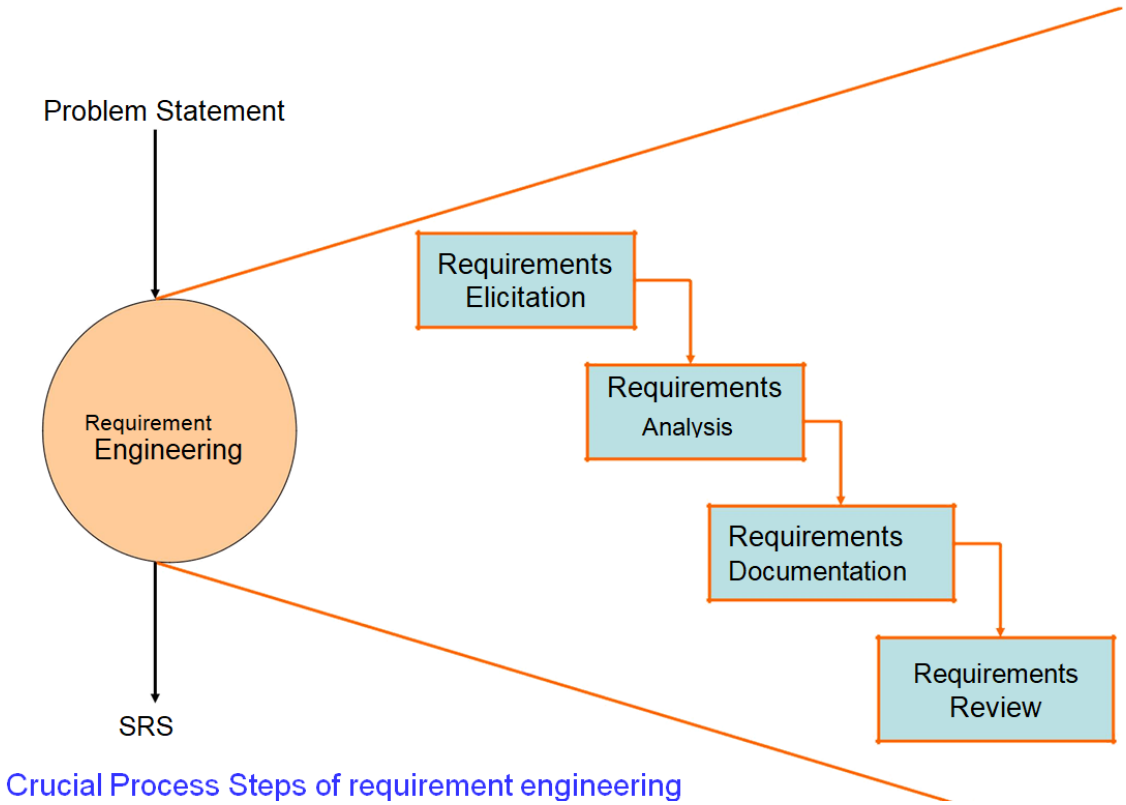
Customers do not know what to expect

What to validate

Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behaviour and its associated constraints.

SRS may act as a contract between developer and customer.

## ➢ Crucial Process Steps




Crucial Process Steps of requirement engineering

**Requirement Elicitation**:- This is also known as gathering of requirements. Here, requirements are identified with the help of customer and existing system processes, if available.

**Requirement Analysis**:- Analysis of requirements starts with requirement elicitation . The requirements are analysed in order to identify inconsistencies, defects, omissions etc. we describe requirements in terms of relationships and also resolve conflicts, if any.

**Requirement Documentation:-** This is the end product of requirement elicitation and analysis. The documentation is very

important as it will be the foundation for the design of the software. The document is known as SRS(Software Requirements Specification).

**Requirements Review**:- The review process is carried out to improve the quality of the SRS. It may also be called as requirements verification.

- **State of Practice:-**
  - ***Requirements are difficult to uncover:-*** Today we are automating virtually every kind of task-some that were previously done manually and some. that have never been done before. In either kind of application, it is difficult, if not impossible to identify all the requirements, regardless of the techniques we use. No one can see a brand new system in its entirety. Even if someone could, the description is always incomplete at start. Users and developers must resort to trial and error to identify problems and solutions.
  - ***Requirements change:-*** Because no user can come up with a complete list of requirements at the outset, the requirements get added and changed as the user beg-ins to understand the system and his or her real needs. That is why we always have requirement changes. But, project schedule is seldom adjusted to reflect these modifications. Fluid requirements make it difficult to establish a baseline from which to design and test. Finally, it is hard to justify spending resources to make a requirement specification "perfect", because it will soon change anyway. This is the biggest problem, and there is as yet no technology to overcome it. This problem is often used as an excuse to either eliminate or scale back requirements engineering effort.
  - ***Over reliance on CASE Tools:-*** Computer Aided Software Engineering (CASE) tools are often sold as panaceas. These exaggerated claims, have created a false sense of trust.

which could inflict untold damage on the Software Industry. CASE tools are as important to developers (including requirement writers) as word processors are to authors. However, we must not rely on requirements engineering tools without first understanding and establishing requirements engineering principles, techniques and processes. Furthermore, we must have realistic expectations from the tools.

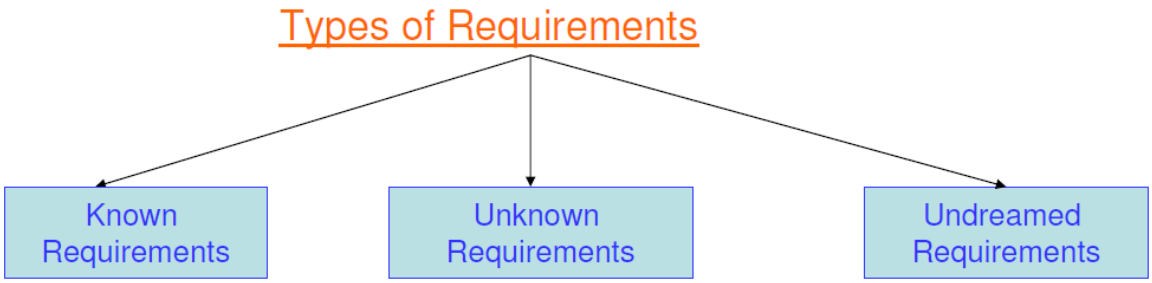
- ***Tight project Schedule:-*** Because of either lack of planning or unreasonable customer demand, many projects start with insufficient time to do a decent job. Sometimes, even the allocated time is reduced while the project is under way. It is also customary to reduce time set apart to analyse requirements, for early start of designing and coding which frequently leads to disaster.
- ***Communication barriers:-*** Requirement engineering is communication intensive activity. Users and developers have different vocabularies, professional backgrounds, and tastes. Developers usually want more precise specifications while users prefer natural language. Selecting either results in misunderstanding and confusion.
- ***Market driven software development***:- Many of the software development is today market driven, developed to satisfy anonymous customers and to keep them coming back to buy upgrades.
- ***Lack of resources:-*** There may not be enough resources to build software that can do everything the customer wants . It is essential to rank requirements so that, in the face of pressure to release the software quickly, the most important can be implemented first.

***Example of Requirement Engineering***

A University wish to develop a software system for the student result management of its M.Tech. Programme. A problem statement is to be prepared for the software development company. The problem statement may

give an overview of the existing system and broad expectations from the new software system.

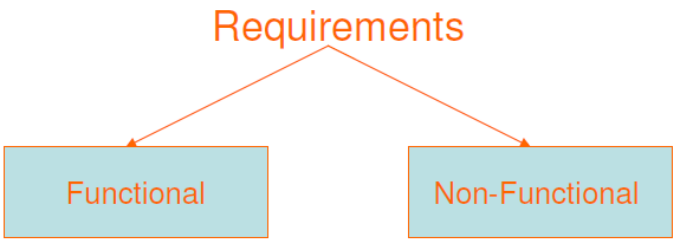
## ✓ *Types of Requirements*




Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User

--- Affected persons




There are different types of requirements such as:

(i) **Known requirements** —Something a stakeholder believes to be implemented,

(ii) **Unknown requirement** – forgotten by the stakeholder because they are not needed right now or needed only by another stakeholder.

(iii) **Undreamt requirements** —Stakeholder may not be able to think of new requirements due to limited domain knowledge.

**Functional and Non-functional Requirements**:-Software requirements are broadly classified as functional and non-functional requirements.

(i) **Functional requirements**: These are related to the expectations from the intended software. They describe what the software has to do. They are also called product features. Sometimes, functional requirements may also specify what the software should not do.

(ii) **Non-Functional requirements**: Non-functional requirements are mostly quality requirements that stipulate how well the software does what it has to do. Non-functional requirements that are especially important to users include specifications of desired performance, availability, reliability, usability and flexibility. Non-functional requirements for developers are maintainability, portability and testability.

**User and System Requirements** :-User requirements are written for the users and include functional and non-functional requirements. User may not be the experts of the software field; hence simple language should be used. The software terminologies, notations etc. should be avoided. User requirements should specify the external behaviour of the system with some constraints and quality parameters. However, design issues should be avoided. Hence, we should only highlight the overview of the system without design characteristics. System requirements are derived from user requirements. They are expanded form of user requirements. They may be used as input to the designers for the preparation of software design document. The user and system requirements are the parts of software requirements and specification (SRS) document.

**Interface Specification**:- Interface issues are very important, for the customers. A good software may not be appreciated if interfaces are not as per expectations of the customers. The types of interlaces are
1. **Procedural interfaces**: Some services are offered by calling

interface procedures. These interfaces are called Application Programming Interfaces (API).

2. **Data structures**: Data structures are used to transfer information from one module to another module. This can be represented using graphical data models.

3. **Representations of data**: The issues related to Ordering of bits are established here. Some are common in embedded systems and microprocessor applications. One of the methods to describe these is to use a diagram of the structure with annotations explaining the function of each group.

## ✓ *Feasibility Study*

***Purpose of feasibility study****:-*"evaluation or analysis of the potential impact of a proposed project or program."

***Focus of feasibility studies***

- Is the product concept viable?
- Will it be possible to develop a product that matches the project's vision statement?
- What are the current estimated cost and schedule for the project?

***Focus of feasibility studies***

- How big is the gap between the original cost & schedule targets & current estimates?
- Is the business model for software justified when the current cost & schedule estimate are considered?
- Have the major risks to the project been identified & can they be surmounted?
- Is the specifications complete & stable enough to support remaining development work?

✓ ***Requirement Elicitation:-*** Requirements elicitation is perhaps the most difficult, most critical, most error-prone, and most communication intensive aspect of software development. Elicitation can succeed only through an effective customer-developer partnership.

The real requirements actually reside in user's mind. Hence the most important goal of requirement engineering is to find out what users really need. Users need can be identified only if we understand the expectations of the users from the desired software.

1. **Interviews:-** After receiving the problem statement from the customer, the first step is to arrange a meeting with the customer. During the meeting or interview, both the parties would like to understand each other. Normally specialised developers, often called 'requirement engineers' interact with the customer. The objective of conducting an interview is to understand the customer's expectations from the software. Both parties have different feelings, goals, opinions, vocabularies, understandings, but one thing is common, both want the project to be a success. With this in mind, requirement engineers normally arrange interviews. Requirement engineers must be open minded and should not approach the interview with pre-conceived notions about what is required.

   Interview may be **open-ended or structured**. In open-ended interview, there is no pre-set agenda. Context free questions may be asked to understand the problem and to have an overview of the situation. For example, for a "result management system", requirement engineer may ask:

   • Who is the controller of examination ?

- Who has requested for such a software ?
- How many officers are placed in the examination division ?
- Who will use the software ?
- Who will explain the manual system ?
- Is there any opposition for this project ?
- How many stakeholders are computer friendly ?

such questions help to identify all stakeholders who will have interest in the software to be developed.

In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview. Interview may be started with simple question to set people at ease. After making atmosphere comfortable and calm, specific questions may be asked to understand the requirements. The customer may be allowed to voice his or her own perceptions about a possible solution.

**Selection of stakeholder:** It will be impossible to interview every stakeholder. representatives from groups must be selected based on their technical expertise, domain knowledge, credibility, and accessibility. There are several groups to be considered for conducting interviews:

(i) **Entry level personnel**: They may not have sufficient domain knowledge and experience, but may be very useful for fresh ideas and different views.

(ii) **Mid-level stakeholders**: They have better domain knowledge and experience of the project. They know the sensitive, complex and critical areas of the project. Hence, requirement engineers may be able to extract meaningful and useful information. Project leader should always be interviewed.

(iii) **Managers or other Stakeholders:** Higher level management officers like Vice_ Presidents, General Managers, Managing Directors should also be interviewed. Their expectations may provide different but rich information for the software development.

(iv) **Users of the software**: This group is perhaps the most important because they will spend more time interacting with the software than anyone else. Their information may be eye opener and may be original at times. Only caution required is that they may be biased towards existing systems.

**Types of questions:** Questions should be simple and short. Two or three questions rolled into one can lead to compound requirements statements that are difficult to interpret and test. It is important to prepare questions, but reading from the questionnaire or only sticking to it is not desirable. We

should be open for any type of discussion and any direction of the interview. For the "result management system" we may ask:

• Are there any problems with the existing system ?

• Have you faced calculation errors in past ?

• What are the possible reasons of malfunctioning ?

• How many students are enrolled presently ?

• What are the possible benefits of computerising this system ?

• Are you satisfied with current processes and policies ?

• How are you maintaining the records of previous students 9

• What data, required by you, exists in other systems `?

• What problems do you want this system to solve ?

- Do you need additional functionality for improving the performance of the system?
- What should be the most important goal of the proposed development

2.**Brainstorming Sessions** :- Brainstorming is a group technique that may be used during requirements elicitation to understand the requirements. The group discussions may

lead to promote creative thinking. Group discussions may lead to new ideas quickly and help to promote creative thinking.

Brainstorming has become very popular and is being used by most of the companies. It promotes creative thinking, generates new ideas and provides platform to share views, apprehensions expectations and difficulties of implementation. All participants are encouraged to say whatever ideas come to mind, whether they seem relevant or not. No one will be criticized for any idea, no matter how goofy it seems, as the responsibility of the participant is to generate views and not to vet them.

This group technique may be carried out with specialised groups like actual users, middle level managers etc., or with total stakeholders. Sometimes unnatural groups are created that may not be appreciated and are uncomfortable for participants. At times, only superficial responses may be gathered to technical questions. In order to handle such situations, a highly trained facilitator may be required. The facilitator may handle group bias and group conflicts carefully. The facilitator should also be cautious about individual egos, dominance and will be responsible for smooth conduct of brainstorming sessions. He or she will encourage the participants, ensure proper individual and group behaviour and help to capture the ideas. The facilitator will follow a published agenda and restart the creative

process if it falters. Every idea will be documented in such a way that everyone can see it. White boards, overhead transparencies or a computer projection system can be used to make it visible to every participant. After the session, a detailed report will be prepared and facilitator will review the report. Every idea will be written in simple english so that it conveys same meaning to every stakeholder. Incomplete ideas may be listed separately and should be discussed at length to make them complete ideas, if possible. Finally, a document will be prepared which will have list of requirements and their priority, if possible.

3.**Facilitated Application Specification Technique** :- This approach is similar to brainstorming sessions and the objective is to bridge the expectation think they are going to get. In order to reduce expectation gap, a team oriented approach is developed for requirements gathering and is called Facilitated Application Specification Technique(FAST).

**FAST session preparations** Each FAST attendee is asked to make a list of objects that are:

(I)part of the environment that surrounds the system

(ii) produced by the system

(iii)used by the system.

In addition, each attendee is asked to make another list of services (processes or functions) that manipulate or interact with the objects. Finally, lists of constraints (e.g.,

cost, size) and performance criteria (e.g., speed, accuracy) are also developed. The attendees are informed that the lists are not expected to be exhaustive but are expected to reflect each person's perception of the system

**Activities of FAST session**

The activities during FAST session may have the following steps:

• Each participant presents his or her lists of objects, services, constraints, and performance for discussion. Lists may be displayed in the meeting by using board, large sheet of paper or any other mechanism, so that they are visible to all the participants.

• The combined lists for each topic are prepared by eliminating redundant entries and adding new ideas.

• The combined lists are again discussed and consensus lists are finalised by the facilitator.

• Once the consensus lists have been completed, the team is divided into smaller subteams, each works to develop mini-specifications for one or more entries of the lists.

• Each subteam then presents mini-specifications to all FAST attendees. After discussion, additions or deletions are made to the lists. We may get new objects, services, constraints, or performance requirements to be added to original lists.

• During all discussions, the team may raise an issue that cannot be resolved during the meeting. An issues list is prepared so that these ideas will be considered later.

• Each attendee prepares a list of validation criteria for-product/system and presents the list to the team. A consensus list of validation criteria is then created.

- A subteam may be asked to write the complete draft specifications using all inputs from the fast meeting.

4. **Quality Function Deployment**:- It is a quality management technique that helps to incorporate the voice of the customer. The voice is then translated into technical requirements. These technical requirements are documented and result is the SRS document. These requirements are further translated into design document. Here, customer satisfaction is of prime QFD emphasizes an then deploys these values throughout the S.E process. Three types of requirements are identified:-

(i) **Normal requirements:** The objectives and goals of the proposed software are discussed with the customer. If this category of requirements (normal) are present, the customer is satisfied. Examples related to result management system might be: entry of marks, calculation of results, merit list report, failed students report, etc. (iii) (ii)**Expected requirements:** These requirements are implicit to the software product and may be so obvious that customer does not explicitly state them. If such

requirements are not present, customer will be dissatisfied with the software. Examples of expected requirements may be: protection from unauthorised access, some warning system for wrong entry of data, the feasibility for modification of any record only by a fool proof system for the identification of person along-with date and time of modification, etc.

(iii) **Exciting requirements:** Some features go beyond the customer's expectations and prove to be very satisfying when present. Examples of exciting requirements for result management system may be: if an unauthorised access is noticed by the software, it should immediately shutdown all the processes and an E-mail is generated to the system administrator, an additional copy of important files is maintained and may be accessed by system administrator only, sophisticated virus protection system etc.

The QFD method has the following steps

(i) Identify all the stakeholders e.g., customers, users, and developers. Also identify any initial constraints identified by the customer that affect requirements development.

(ii) List out requirements from customer ; inputs, considering different viewpoints. Requirements are expression of what the system will do, which is both perceptible and of value to customers. Some customer's expectations may be unrealistic or

ambiguous and may be translated into realistic or unambiguous requirements if possible.

(iii) A value indicating a degree of importance , is assigned to each requirement. Thus, customer determines the importance of each requirement on a scale of 1 to 5 as given below:

5 points: Very important

4 points: important

3 points: not important, but nice to have

2 points: not important

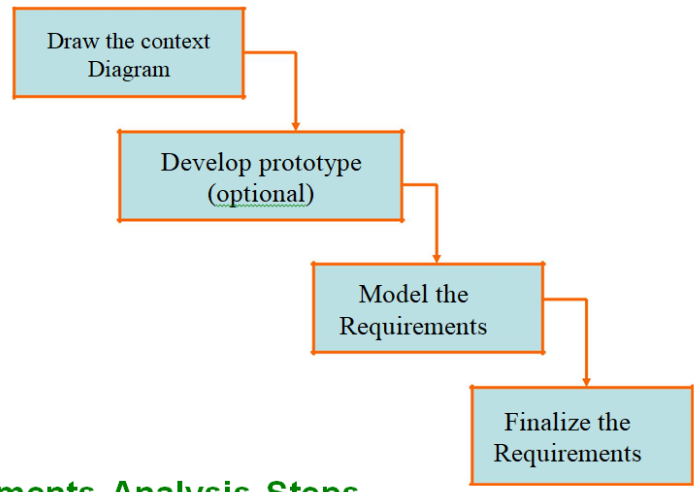
1 point: unrealistic, requires further exploration

## 5. Use Case Approach( Done in Lab)

### ✓ Requirement Analysis

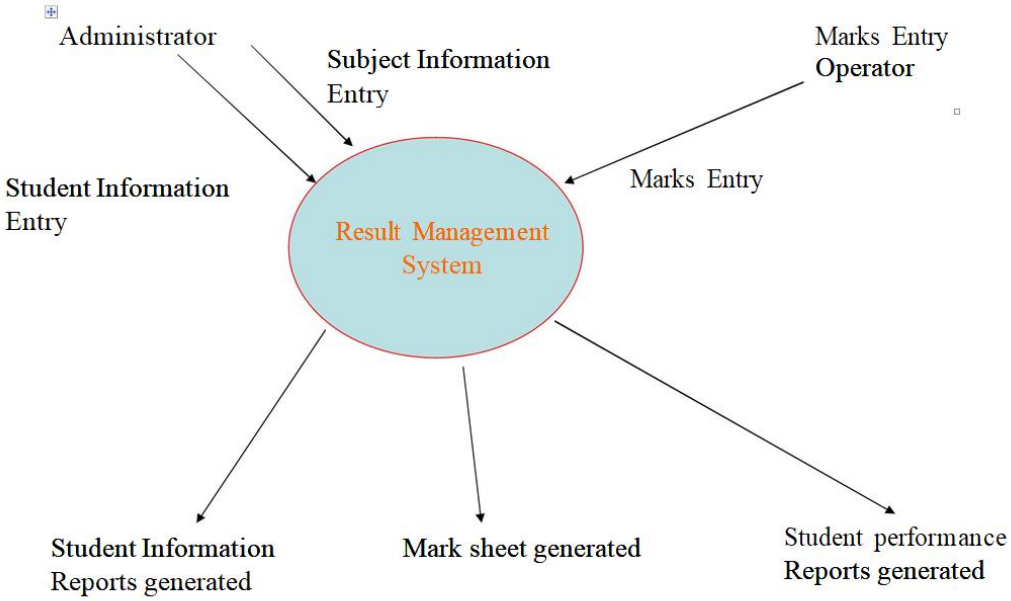
We analyze, refine and scrutinize requirements to make consistent & unambiguous requirements.

Steps




Requirements Analysis Steps

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




(ii) **Development of a prototype (optional):** One effective way to find out what the customer really wants is to construct a prototype, something that looks and preferably acts like a part of the system they say they want. We can use their feedback to continuously modify the prototype until the customer is satisfied. Hence prototype helps the client to visualise the proposed system and increase the understanding of requirements.

**(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 requirement,- Such models include data flow diagrams, entity relationship diagrams, data dictionaries, state-transition diagrams etc.

**(iv)Finalise the requirements:** After modelling the requirements, we will have better understanding of system behaviour. The inconsistencies and ambiguities

have been identified and corrected. Flow of data amongst various modules has been analysed. Elicitation and analysis activities have provided better insight to the system. Now we finalise the analysed requirement and next step is to document these requirements in a prescribed format.

# 1.Data Flow Diagram

## Data Flow Diagrams

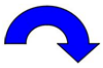
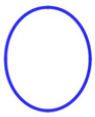
DFD show the flow of data through the system.

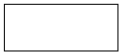
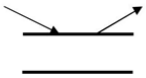
--All names should be unique

It is not a flow chart

Suppress logical decisions

Defer error conditions & handling until the end of the analysis

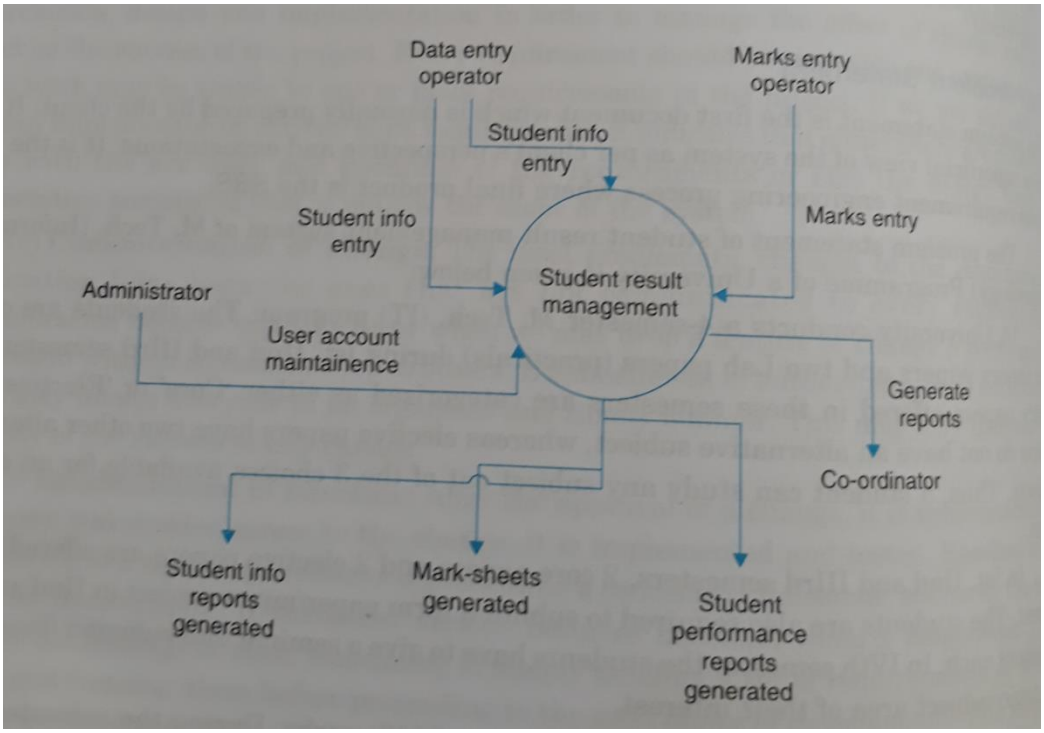
| Symbol | Name | Function |
|---|---|---|
|  | Data Flow | Connect process |
|  | Process | Perform some transformation of its input data to yield output data. |

| Symbol | Name | Function |
|---|---|---|
|  | Source or sink | A source of system inputs or sink of system outputs |
|  | Data Store | A repository of data, the arrowhead indicate net input and net outputs to store |

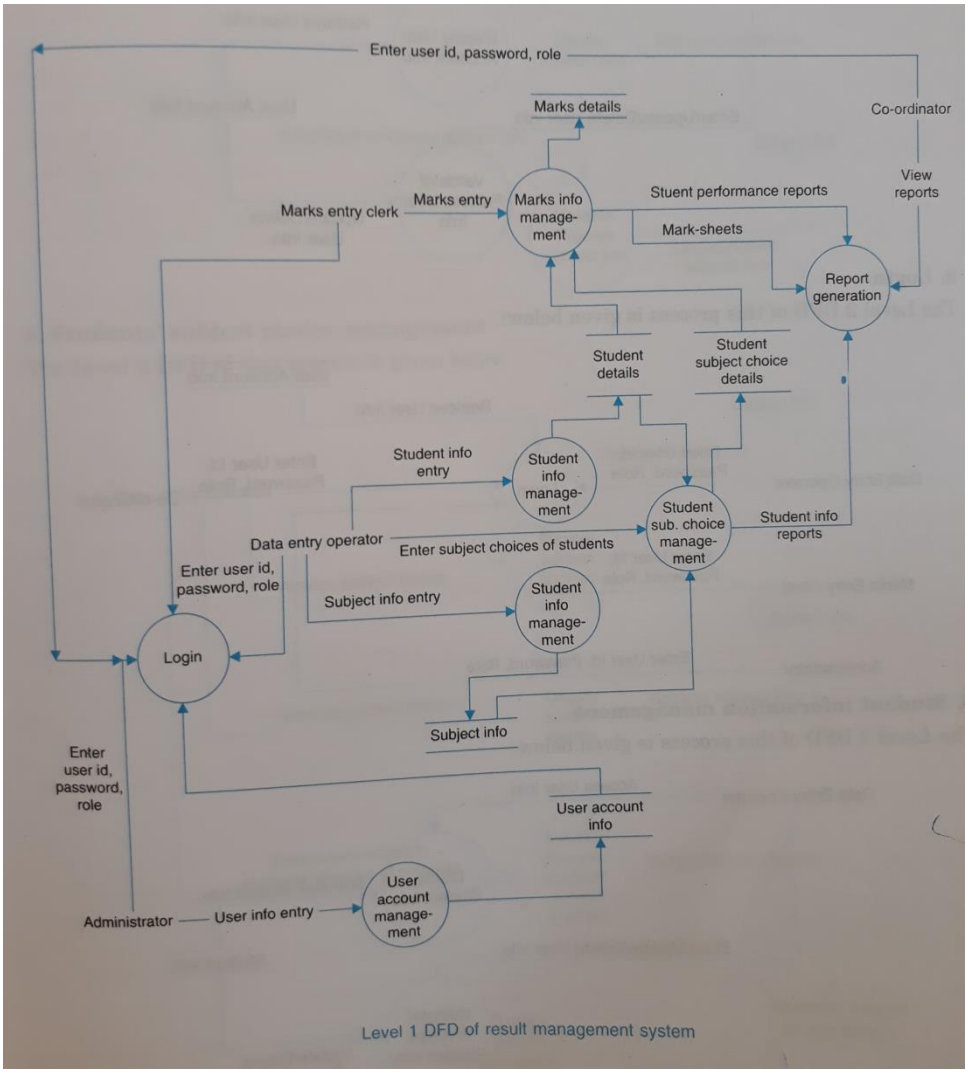
## Leveling

DFD represent a system or software at any level of abstraction.

A level 0 DFD is called fundamental system model or context model represents entire software element as a single bubble with input and output data indicating by incoming & outgoing arrows.




Level 0 DFD(Result Management System)




Level 1DFD

# 2.Data Dictionaries:

DFD → DD

Data Dictionaries are simply repositories to store information about all data items defined in DFD.

Includes :

Name of data item

Aliases (other names for items)

Description/Purpose

Related data items
Range of values
Data flows
Data structure definition

| Notation | Meaning |
|---|---|
| x= a+b | x consists of data element a & b |
| x={a/b} | x consists of either a or b |
| x=(a) | x consists of an optional data element a |
| x= y{a} | x consists of y or more occurrences |

## 3.ER Diagrams(Do Yourself)

**4.Software Prototyping:-** Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design**.**

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely −

**Throwaway/Rapid Prototyping**

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are

understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

## Evolutionary Prototyping

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. By using evolutionary prototyping, the well-understood requirements are included in the prototype and the requirements are added as and when they are understood.

- ***Requirement Documentation*:-**

## *SRS*

The SRS is a specification for a particular software product, program or set of programs that performs certain functions in a specific environment.

SRS Should

- ✓ Correctly define all requirements
- ✓ not describe any design details
- ✓ not impose any additional constraints

### Characteristics of SRS

**Correct:-**An SRS is correct if and only if every requirement stated therein is one that the software shall meet.

**Unambiguous:-**An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation.

**Complete:-**An SRS is complete if and only if, it includes the following elements**.**All significant requirements, whether related to functionality, performance, design constraints, attributes or external interfaces.Responses to both valid & invalid inputs.Full Label and

references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure.

**Consistent:-**An SRS is consistent if and only if, no subset of individual requirements described in it conflict.

**Ranked for importance and/or Stability:-**If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement.

**Verifiable:-**An SRS is verifiable, if and only if, every requirement stated therein is verifiable.

**Modifiable:-**An SRS is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining structure and style.

**Traceable:-**An SRS is traceable, if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.