

## UNIT - 2

---

# **SOFTWARE REQUIREMENTS ANALYSIS AND SPECIFICATIONS**

# Requirement 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.

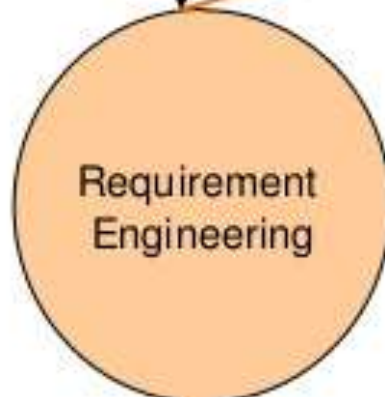
Crucial process steps

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

Problem Statement



SRS

Requirements  
Elicitation

Requirements  
Analysis

Requirements  
Documentation

Requirements  
Review

Crucial Process Steps of requirement engineering

# Requirement Engineering

---

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

SRS may act as a contract between developer and customer.

## -- State of practice

- Requirements are difficult to uncover
- Requirements change
- Tight project Schedule
- Communication barriers
- Market driven software development
- Lack of resources

# Requirement Engineering

---

## EXAMPLE:

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.

# Requirement Engineering

---

## EXAMPLE:

A university wishes to develop a software system for library management activities. Design the problem statement for the software company.

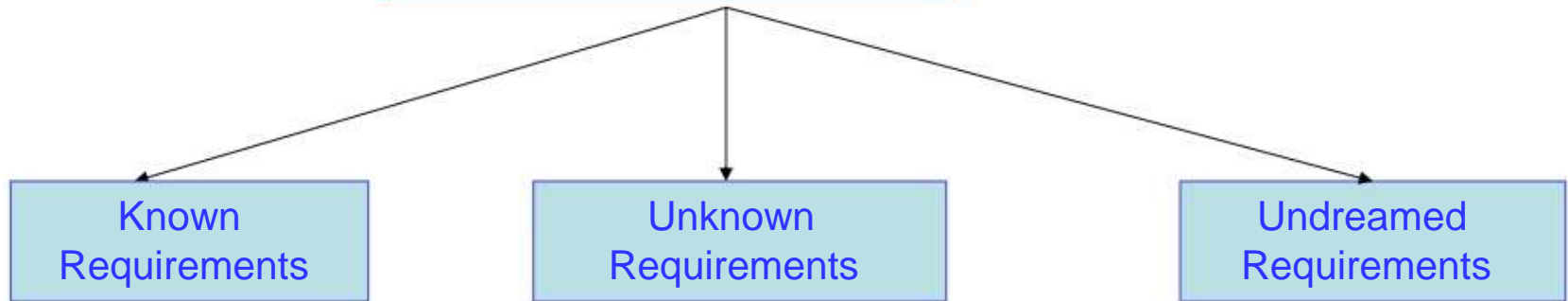
## SOLUTION:

1. Issue of books
2. Return of books
3. Query processing

# Types of Requirements

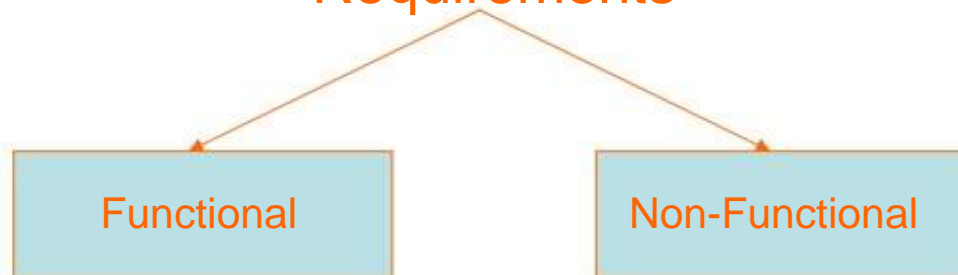
---

## Types of Requirements



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

## Requirements

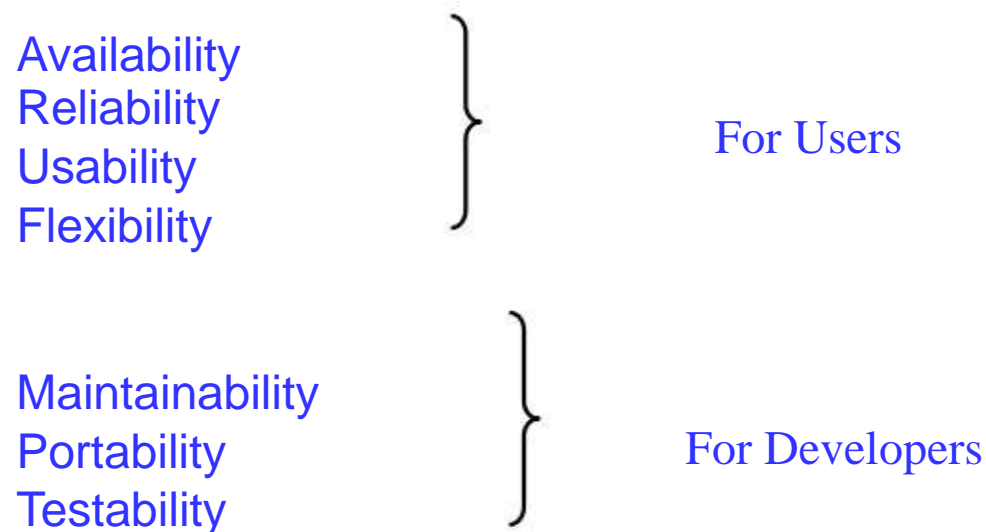


# Types of Requirements

---

Functional requirements describe **what the software has to do**. They are often called product features.

Non Functional requirements are mostly **quality requirements**. That stipulate how well the software does, what it has to do.





# Types of Requirements

---

## User and system requirements

- User requirement are written for the users and include functional and non functional requirement.
- System requirement are derived from user requirement.
- The user and system requirements are the parts of software requirement and specification (SRS) document.

# Types of Requirements

---

## Interface Specification

- Important for the customers.

### TYPES OF INTERFACES

- Procedural interfaces (also called Application Programming Interfaces (APIs)).
- Data structures
- Representation of data.

# Feasibility Study

---

A **Feasibility Study** is a study made before committing to a project. It leads to a decision:

- go ahead
- do not go ahead
- think again

The **main purpose** of the feasibility study is to answer one main question:

**“Should we proceed with this project idea?”**

# Feasibility Study

---

The **purpose of feasibility study** is not to solve the problem, but to determine whether the problem is worth solving.

It concentrates on the following area :-

1. Operation Feasibility
2. Technical Feasibility
3. Economic Feasibility

# Elements of a good Feasibility Study

---

There are total six parts of any feasibility study :

- The Project Scope
- The Current Analysis
- Requirements
- Approach
- Evaluation
- Review

# Requirements Elicitation

---

Perhaps

- Most difficult
- Most critical
- Most error prone
- Most communication intensive

Succeed

└→ effective customer developer partnership

Selection of any method

1. It is the only method that we know
2. It is our favorite method for all situations
3. We understand intuitively that the method is effective in the present circumstances.

Normally we rely on first two reasons.

# Requirements Elicitation

---

There are number of requirement elicitation methods:-

1.Interviews

2.Brainstorming Sessions

3.FAST

4.Quality function deployment

5.Use Case Approach

# 1. Interviews

---

Both parties have a common goal 

Success of the  
project

Selection of stakeholder: Groups to be considered for interviews:-

1. Entry level personnel
2. Middle level stakeholder
3. Managers
4. Users of the software (Most important)



# 1. Interviews

---

Types of questions.

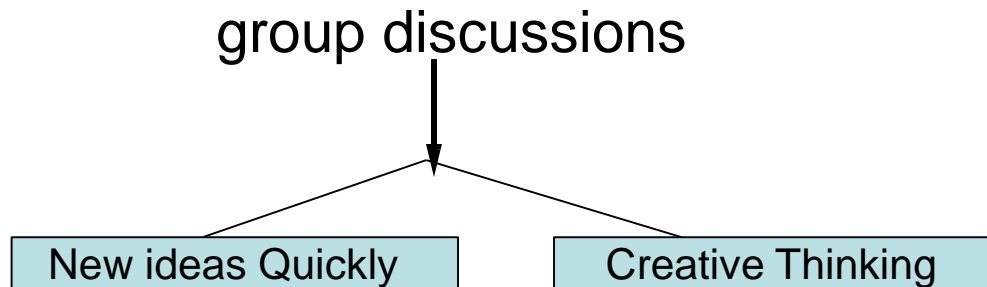
1. Any problems with existing system
2. Any Calculation errors
3. Possible reasons for malfunctioning
4. No. of Student Enrolled
5. Possible benefits
6. Satisfied with current policies
7. How are you maintaining the records of previous students?
8. Any requirement of data from other system
9. Any specific problems
10. Any additional functionality
11. Most important goal of the proposed development

At the end, we may have wide variety of expectations from the proposed software

## 2. Brainstorming Sessions

---

It is a group technique



### Groups

1. Users 2. Middle Level managers 3. Total Stakeholders

## 2. Brainstorming Sessions

---

**A Facilitator may handle group bias, conflicts carefully.**

- Facilitator may follow a published agenda
- Every idea will be documented in a way that everyone can see it.
- A detailed report is prepared.

# 3. Facilitated Application specification Techniques (FAST)

---

**Objective** is to bridge the expectation gap.

- Similar to brainstorming sessions.
- Team oriented approach
- Creation of joint team of customers and developers.

# 3. Facilitated Application specification Techniques (FAST)

---

## Guidelines

1. Arrange a meeting at a neutral site.
2. Establish rules for participation.
3. Informal agenda to encourage free flow of ideas.
4. Appoint a facilitator.
5. Prepare definition mechanism board, worksheets, wall stickier.
6. Participants should not criticize or debate.

# 3. Facilitated Application specification Techniques (FAST)

---

## FAST session Preparations

Each attendee is asked to make a list of objects that are:

1. Part of environment that surrounds the system.
2. Produced by the system.
3. Used by the system.
  - A. List of constraints
  - B. Functions
  - C. Performance criteria

# 3. Facilitated Application specification Techniques (FAST)

---

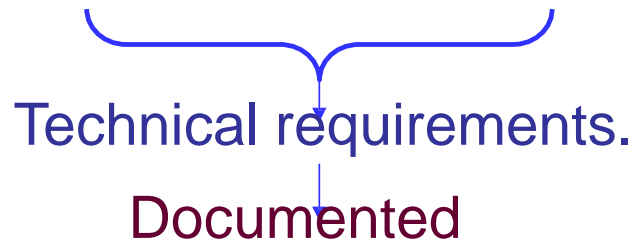
## Activities of FAST session

1. Every participant presents his/her list
2. Combine list for each topic
3. Discussion
4. Consensus list
5. Sub teams for mini specifications
6. Presentations of mini-specifications
7. Validation criteria
8. A sub team to draft specifications

# 4. Quality Function Deployment

---

- Incorporate voice of the customer



Technical requirements.  
Documented

Prime concern is customer satisfaction



What is important for customer?

- Normal requirements
- Expected requirements
- Exciting requirements



# Requirements Elicitation

---

## Steps

1. Identify stakeholders
2. List out requirements
3. Degree of importance to each requirement.

# Requirements Elicitation

---

- 5 Points : V. Important
- 4 Points : Important
- 3 Points : Not Important but nice to have
- 2 Points : Not important
- 1 Points : Unrealistic, required further exploration

Requirement Engineer may categorize like:

- (i) It is possible to achieve
- (ii) It should be deferred & Why
- (iii) It is impossible and should be dropped from consideration

First Category requirements will be implemented as per priority assigned with every requirement.

# 5. The Use Case Approach

---

- Ivar Jacobson & others introduced Use Case approach for elicitation & modeling.

- Use Case – give functional view

- The terms

Use Case

Use Case Scenario

Use Case Diagram

} Often Interchanged

} But they are different

Use Cases are structured outline or template for the description of user requirements modeled in a structured language like English.

**Use case    Scenarios are unstructured descriptions of user requirements.**

---

Use case diagrams are graphical representations that may be decomposed into further levels of abstraction.

## Components of Use Case approach

Actor:

An actor or external agent, lies outside the system model, but interacts with it in some way.

Actor → Person, machine, information  
System

- 
- Cockburn distinguishes between Primary and secondary actors.
  - A Primary actor is one having a goal requiring the assistance of the system.
  - A Secondary actor is one from which System needs assistance.

## Use Cases

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied.

- 
- \* It describes the sequence of interactions between actors and the system necessary to deliver the services that satisfies the goal.
  - \* Alternate sequence
  - \* System is treated as black box.

Thus

Use Case captures who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals.

---

\*defines all behavior required of the system,  
bounding the scope of the system.

Jacobson & others proposed a template for writing  
Use cases as shown below:

**1. Introduction**

Describe a quick background of the use case.

**2. Actors**

List the actors that interact and participate in the  
use cases.

**3. Pre Conditions**

Pre conditions that need to be satisfied for the use  
case to perform.

**4. Post Conditions**

Define the different states in which we expect the system  
to be in, after the use case executes.

---

## 5. Flow of events

### 1. Basic Flow

List the primary events that will occur when this use case is executed.

### 2. Alternative Flows

Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow.

A use case can have many alternative flows as required.

## 6.Special Requirements

Business rules should be listed for basic & information flows as special requirements in the use case narration .These rules will also be used for writing test cases. Both success and failures scenarios should be described.

## 7.Use Case relationships

For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability

Use Case Template.



---

## Use Case Guidelines

1. Identify all users
2. Create a user profile for each category of users including all roles of the users play that are relevant to the system.
3. Create a use case for each goal, following the use case template maintain the same level of abstraction throughout the use case. Steps in higher level use cases may be treated as goals for lower level (i.e. more detailed), sub-use cases.

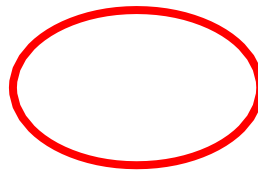
---

## Use case Diagrams

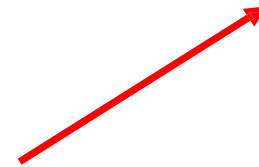
- represents what happens when actor interacts with a system.
- captures functional aspect of the system.



Actor



Use Case



Relationship between actors and use case and/or between the use cases.

- Actors appear outside the rectangle.
- Use cases within rectangle providing functionality.
- Relationship association is a solid line between actor & use cases.

---

\*Use cases should not be used to capture all the details of the system.

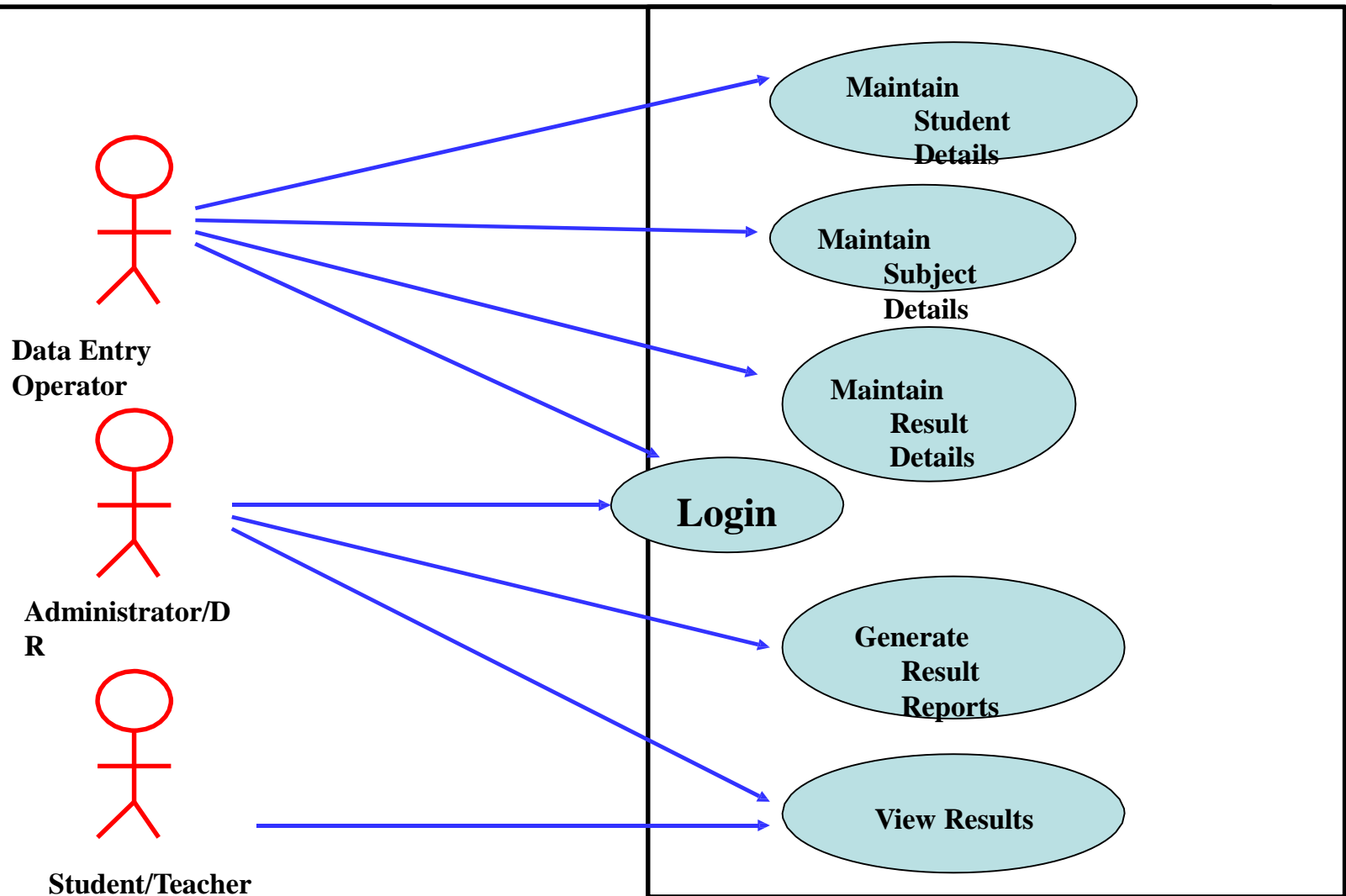
\*Only significant aspects of the required functionality

\*No design issues

\*Use Cases are for “what” the system is , not “how” the system will be designed

\* Free of design characteristics

# Use case diagram for Result Management System



---

## 1. Maintain student Details

Add/Modify/update students details like name, address.

## 2. Maintain subject Details

Add/Modify/Update Subject information semester wise

## 3. Maintain Result Details

Include entry of marks and assignment of credit points for each paper.

## 4. Login

Use to Provide way to enter through user id & password.

## 5. Generate Result Report

Use to print various reports

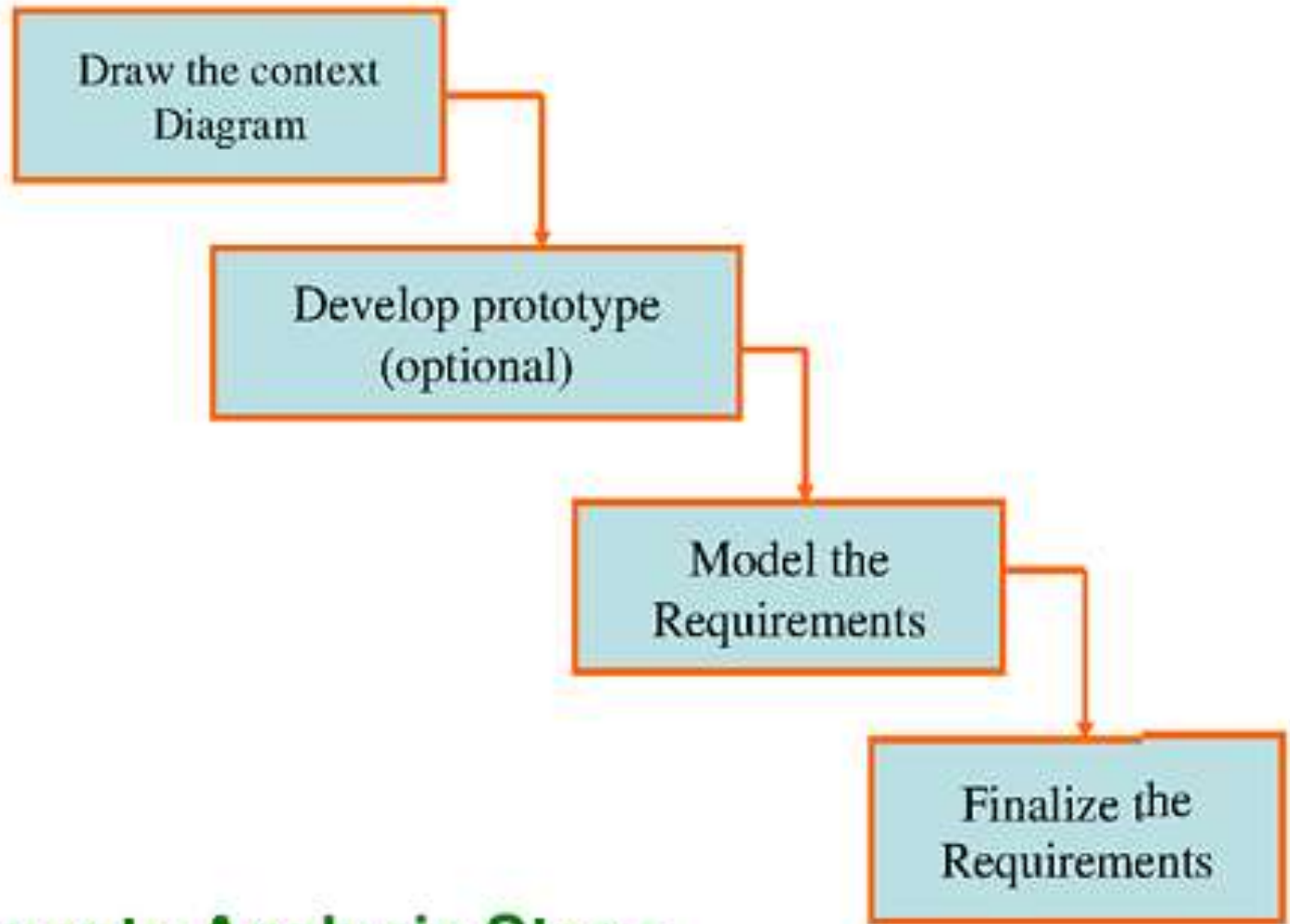
## 6. View Result

- (i) According to course code
- (ii) According to Enrollment number/roll number

# REQUIREMENT ANALYSIS

---

## Steps

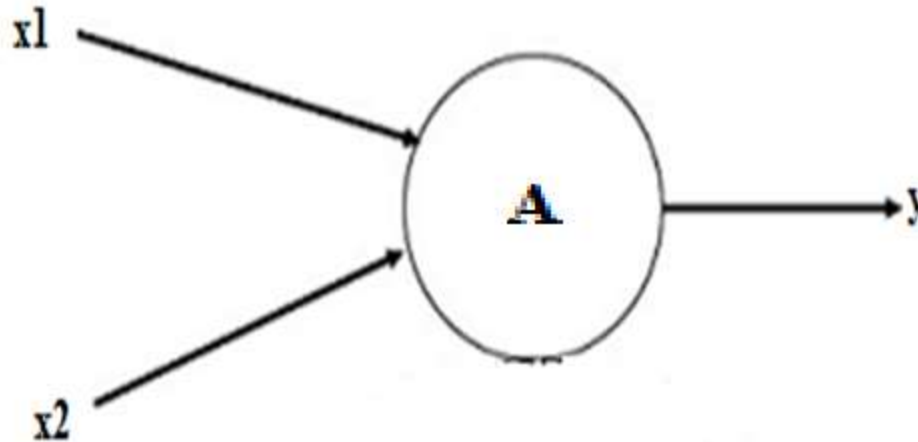


## Requirements Analysis Steps

# Draw the Context Diagram

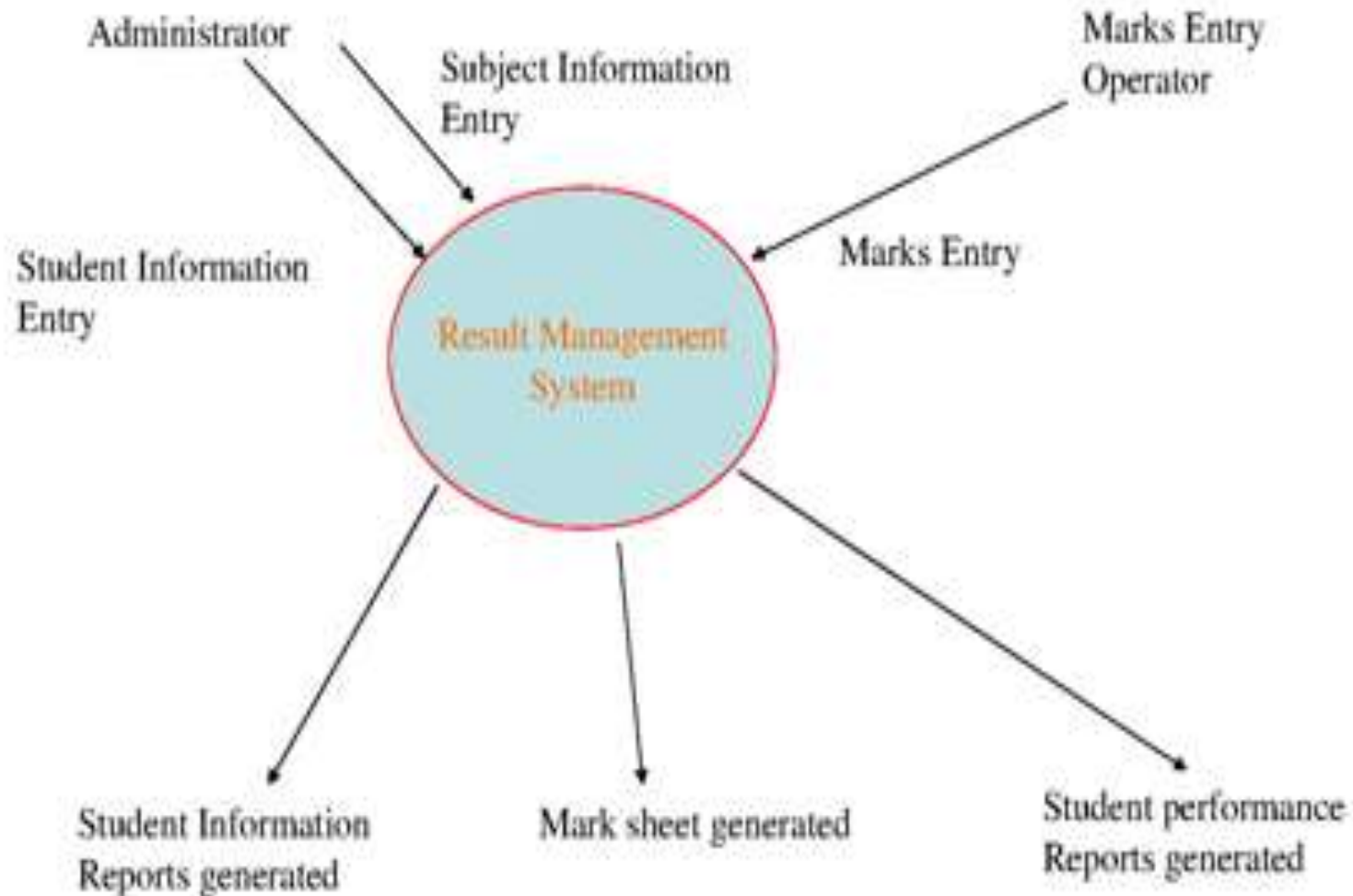
---

- This is also called as **Fundamental System Model** or **Level-0 DFD**.
- It represents the entire system element as a **Single Bubble** with input and output data indicated by incoming and outgoing arrows.
- For example, if bubble A has two inputs x1 and x2, and one output y that represents A should have exactly two external inputs and one external output.



# Context Diagram of Result Mgmt System

---





# Model the Requirements

---

This process usually consists of various graphical representations of the functions, data entities, external entities and the relationships between them.

This graphical view **may help to find incorrect, inconsistent, and missing requirements.**

Such model include –

- Data flow Diagrams
- Data Dictionaries
- ER Diagrams

# 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 & error handling until the end of the analysis

# Standard Symbols for DFD's

---

**Symbol**

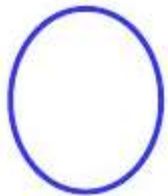
**Name**

**Function**



**Data flow**

**Used to connect processes to each other**



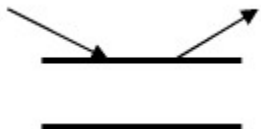
**Process**

**Performs some transformation of input data to yield output data**



**Source or sink**

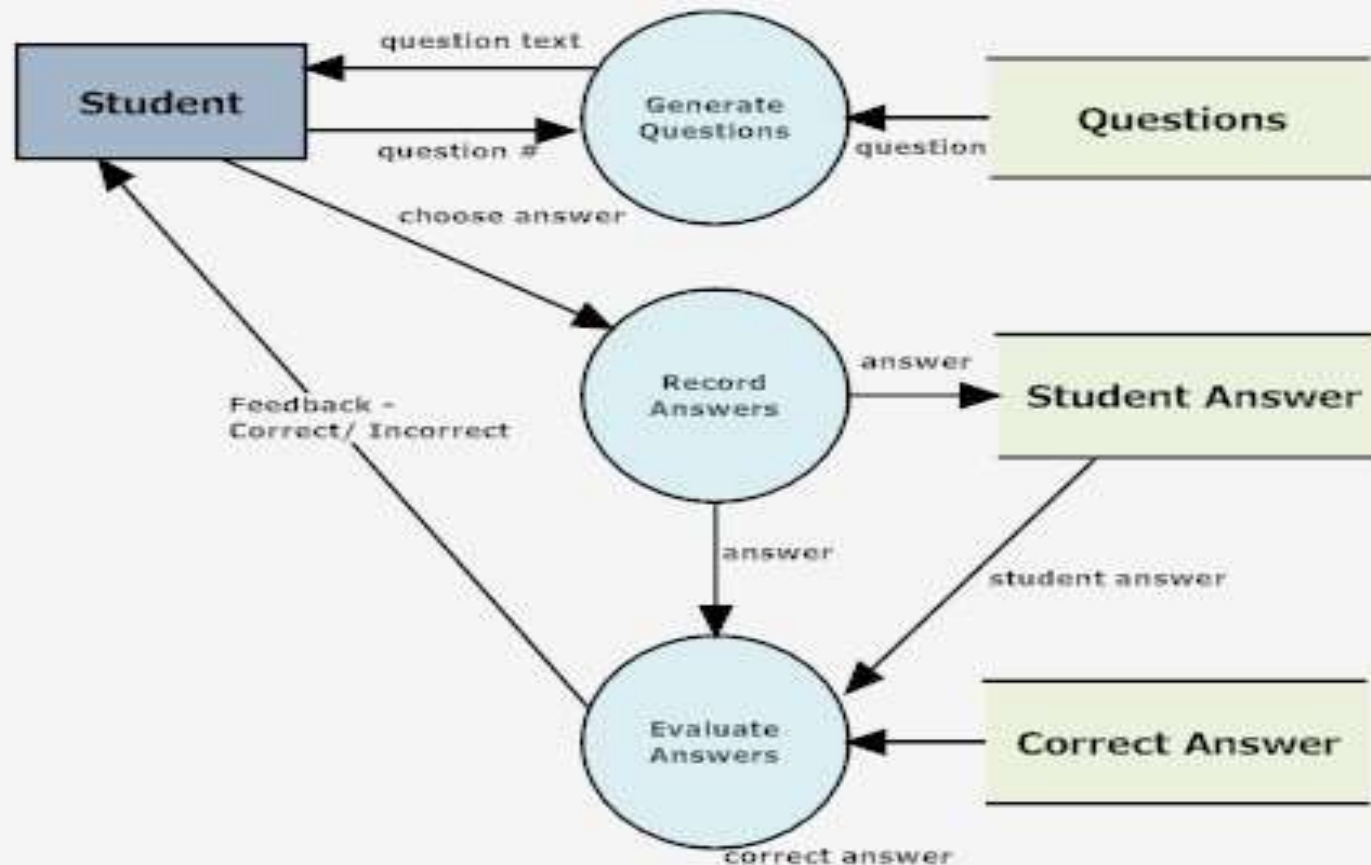
**A source of system inputs or sink of System outputs**



**Data Store**

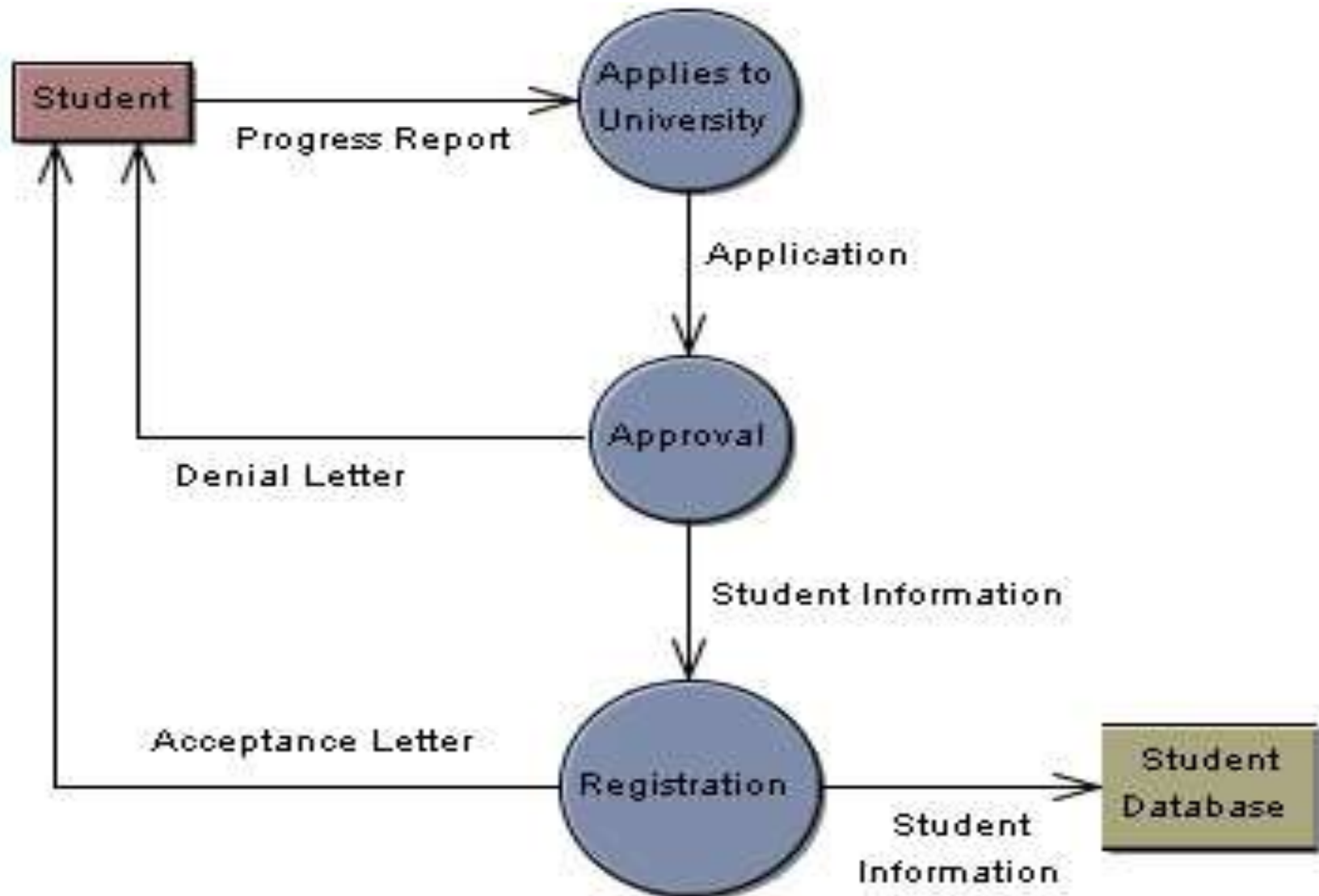
**A repository of data, the arrowheads indicate net inputs and net outputs to store.**

# DFD for QUIZ SOFTWARE



# DFD for UNIVERSITY COURSE REGISTRATION SYSTEM

---



# Data Dictionaries

---

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

At the requirement stage, the data dictionary should at least define customer data items, to ensure that the customer and developer use, the same definitions and terminologies.

Typical information stored include –

- Name of the data item
- Aliases
- Description/Purpose
- Related data items
- Range of Values
- Data Structure Definition

# Data Dictionaries

---

The data dictionary has two different kinds of items: **composite** data and **elemental** data.

- Higher-level (composite) elements may be defined in terms of lower-level items.
- Elemental data are items that cannot be reduced any further and are defined in terms of the values that it can assume or some other unambiguous base type.

The general format of a data dictionary is similar to a standard dictionary; it contains an alphabetically ordered list of entries.

Each entry consists of a formal definition and verbal description

# COMPOSITE DATA

---

Composite data can be constructed in three ways: sequence, repetition, or selection of other data types.

## sequence:

**+** A plus sign indicates one element is followed by or concatenated with another element.

## repetition:

**[ ]** Square brackets are used to enclose one or more optional elements.

**|** A vertical line stands for "or" and is used to indicate alternatives.

## selection:

**{ }** Curly braces indicate that the element being defined is made up of a series of repetitions of the element(s) enclosed in the brackets.

**{ }y** The upper limit on the number of allowable repetitions can be indicated by including an integer superscript on the right bracket. Here y represents an integer and indicates that the number of repetitions cannot be greater than y.



# Examples of COMPOSITE DATA

---

**sequence:**

**Mailing-address = name + address + city + zip-code**

\* The address at which the customer can receive mail \*

**repetition:**

**Completed-order = [ item-ordered ]**

\* A complete customer order \*

**selection:**

**Atm-transaction = { deposit | withdrawal }**

\* An customer transaction at an automated teller machine \*

In these examples asterisks are used to delimit the comment or verbal description of the item, but other notations can be used as well.

# Examples of ELEMENTAL DATA

---

Elemental data is described in terms of a data type or by listing the values that the item can assume.

## **desired-temperature = floating point**

\*Desired-temperature is the value the user sets for desired water temperature in the pool. It is a floating point number in the range from 0.0 to 100.0, inclusive. The units are Celsius.\*

## **age = non-negative integer**

\*Age is how old the customer is in years. Age is a whole number greater than or equal to zero.\*

## **performances-attended = counter**

\* Performances-attended is the number of performances this customer has attended.\*

## **counter = positive integer**

\*Counter is a whole number greater than zero that can only be incremented by one and never decremented.\*

# Mathematical Operator used within DD

---

Notations	Meaning
$x=a+b$	x consists of data elements a & b
$x=[a \mid b]$	x consists of either data element a or b
$x=a$	x consists of data element a
$x=y\{a\}$	x consists of y or more occurrences of data element a
$x = \{a\}z$	x consists of z or fewer occurrences of data element a
$x=y\{a\}z$	x consists of some occurrences of data element a which are between y and z.

# **Entity – Relationship Model (ER model)**

# Introduction

---

- An **Entity – Relationship model (ER model)** is an abstract way to **describe a database**.
- It is a visual representation of different data using conventions that **describe how these data are related to each other**.

# Basic elements in ER model

---

There are three basic elements in ER models:

**Entities** are the “things” about which we seek information.

**Attributes** are the data we collect about the entities.

**Relationships** provide the structure needed to draw information from multiple entities.

# Symbols used in E-R Diagram

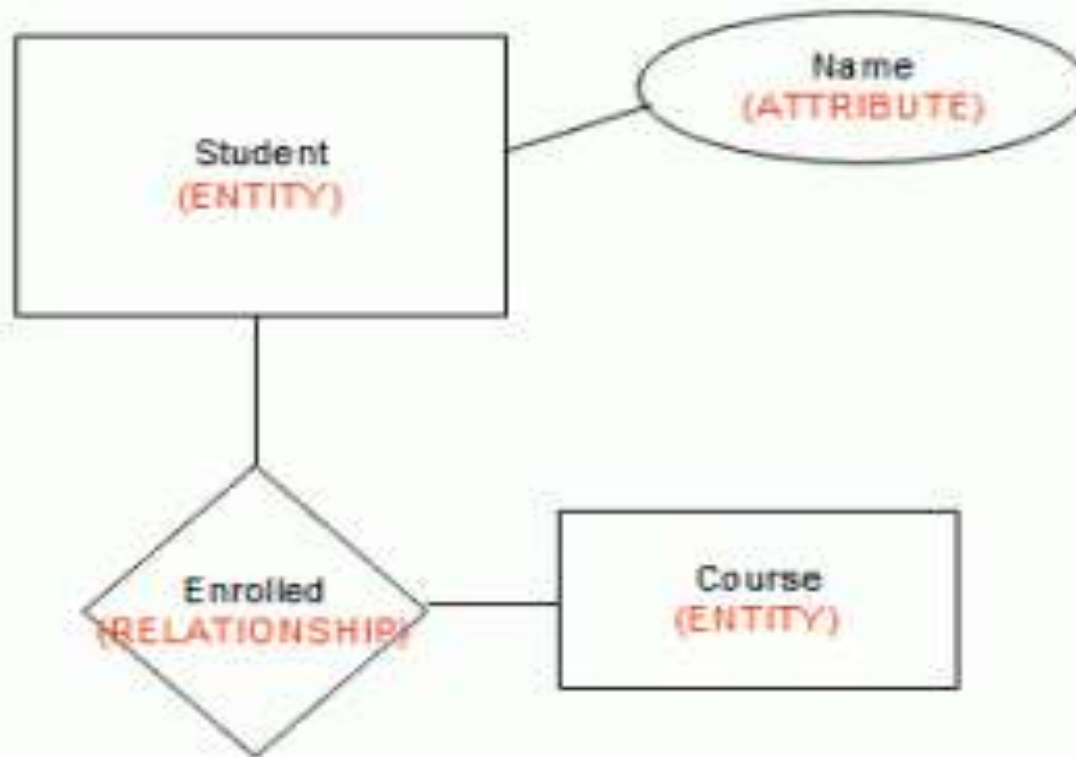
---

**Entity** – rectangle

**Attribute** -oval

**Relationship** – diamond

**Link** - line

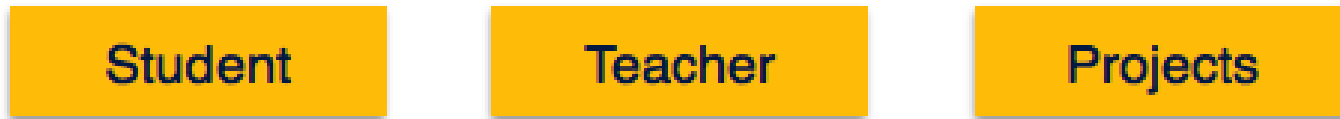


# ER Model

---

## Entity

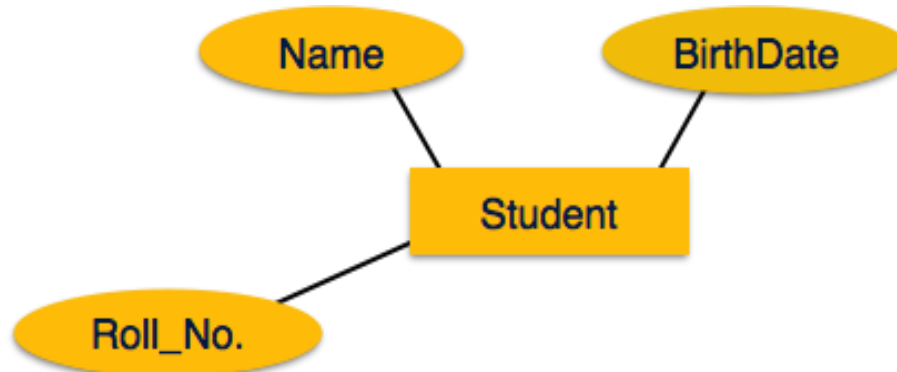
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



*[Image: Entities in a school database]*

## Attributes

Attributes are properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).





# ER Model

---

## Relationship

A relationship **describes how entities interact**. For example, the entity “carpenter” may be related to the entity “table” by the relationship “builds” or “makes”. Relationships are represented by diamond shapes.



# TYPES OF ATTRIBUTES

---

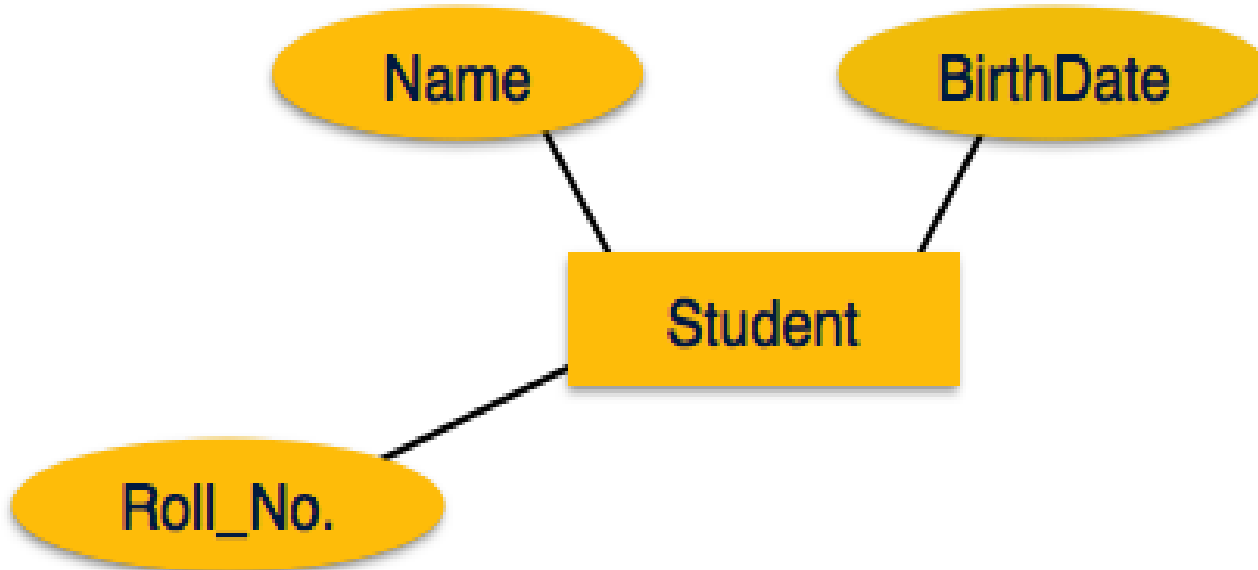
**There are total six types of attributes :-**

- 1. Simple attribute**
- 2. Composite attribute**
- 3. Derived attribute**
- 4. Stored attribute**
- 5. Single valued attribute**
- 6. Multi valued attribute**

# TYPES OF ATTRIBUTES

---

- **Simple attribute:** Cannot be split in to further attributes(indivisible). Also known as **Atomic** attribute. Ex: Ssn(Social Security Number) or Roll No



# TYPES OF ATTRIBUTES

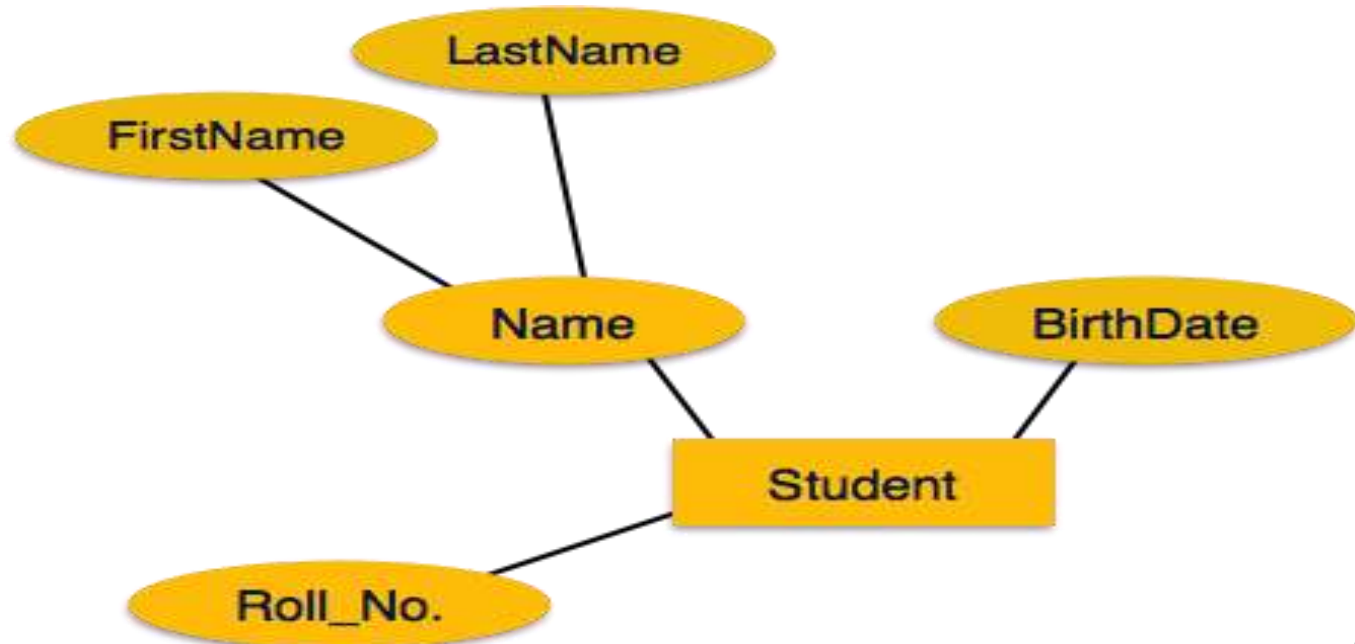
---

- **Composite attribute:**

- Can be divided into smaller subparts which represent more basic attributes with independent meaning

- Even form hierarchy

Ex: **Address, Name**



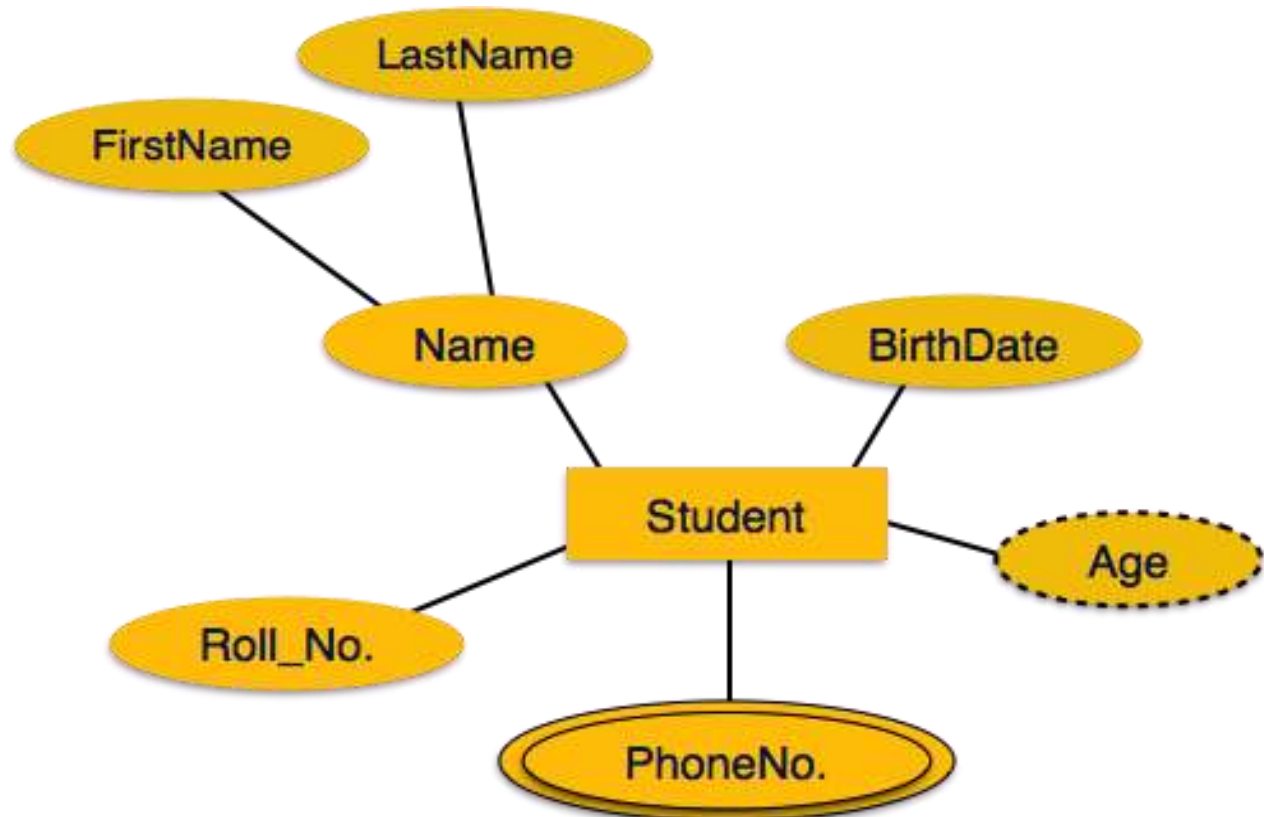
# TYPES OF ATTRIBUTES

---

- **Derived attribute:**

- Attribute values are derived from another attribute.
- Denoted by dotted oval

Ex - Age



# TYPES OF ATTRIBUTES

---

## **Stored attribute:**

Attributes from which the values of other attributes are derived.

For example 'Date of birth' of a person is a stored attribute.

# TYPES OF ATTRIBUTES

---

## Single-valued attribute:

- A single valued attribute can **have only a single value**.
- For example a person can have only one **'date of birth', 'age' , Social\_Security\_Number.etc.**
- That is a single valued attributes can have only single value. But it can **be simple or composite attribute**.
- That is **'date of birth' is a composite attribute , 'age' is a simple attribute. But both are single valued attributes.**

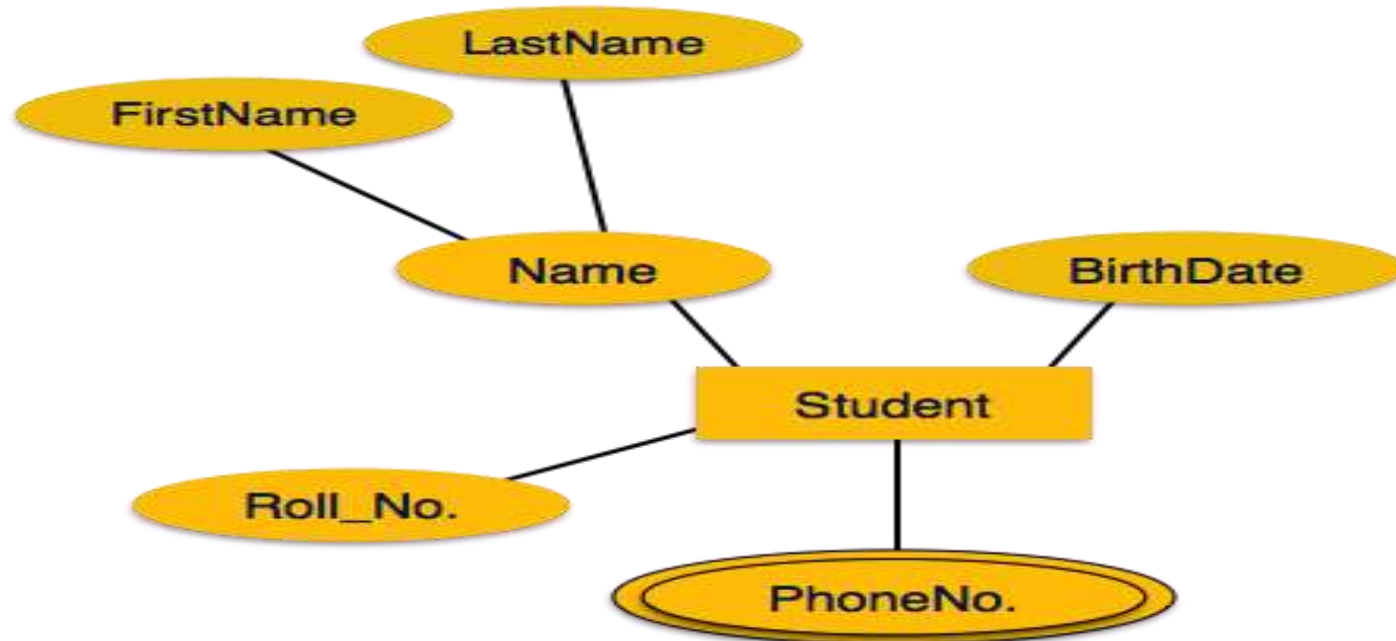
# TYPES OF ATTRIBUTES

---

## Multi-value attribute:

Attribute may contain more than one values. Denoted **by double circled oval** line connecting to the entity in the ER diagram.

Ex: Phone-number, College-degree, email addresses etc





# KEYS

---

Keys are of following types:-

1. Candidate Key
2. Composite Key
3. Primary key
4. Foreign Key

The name of each primary key attribute is underlined.

# CANDIDATE KEY

---

- a simple or composite key that is unique and minimal
- unique – no two rows in a table may have the same value at any time
- minimal – every column must be necessary for uniqueness
- For example, for the entity **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)
- Possible candidate keys are EID, SIN

# COMPOSITE KEY

---

- **Composed of more than one attribute**
- For example, First Name and Last Name – assuming there is no one else in the company with the same name, Last Name and Department ID – assuming two people with the same last name don't work in the same department.
- A composite key can have two or more attributes, but it must be minimal.

# PRIMARY KEY

---

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
- A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. **This key is indicated by underlining the attribute in the ER model.**
- For example **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID) – EID is the Primary key.

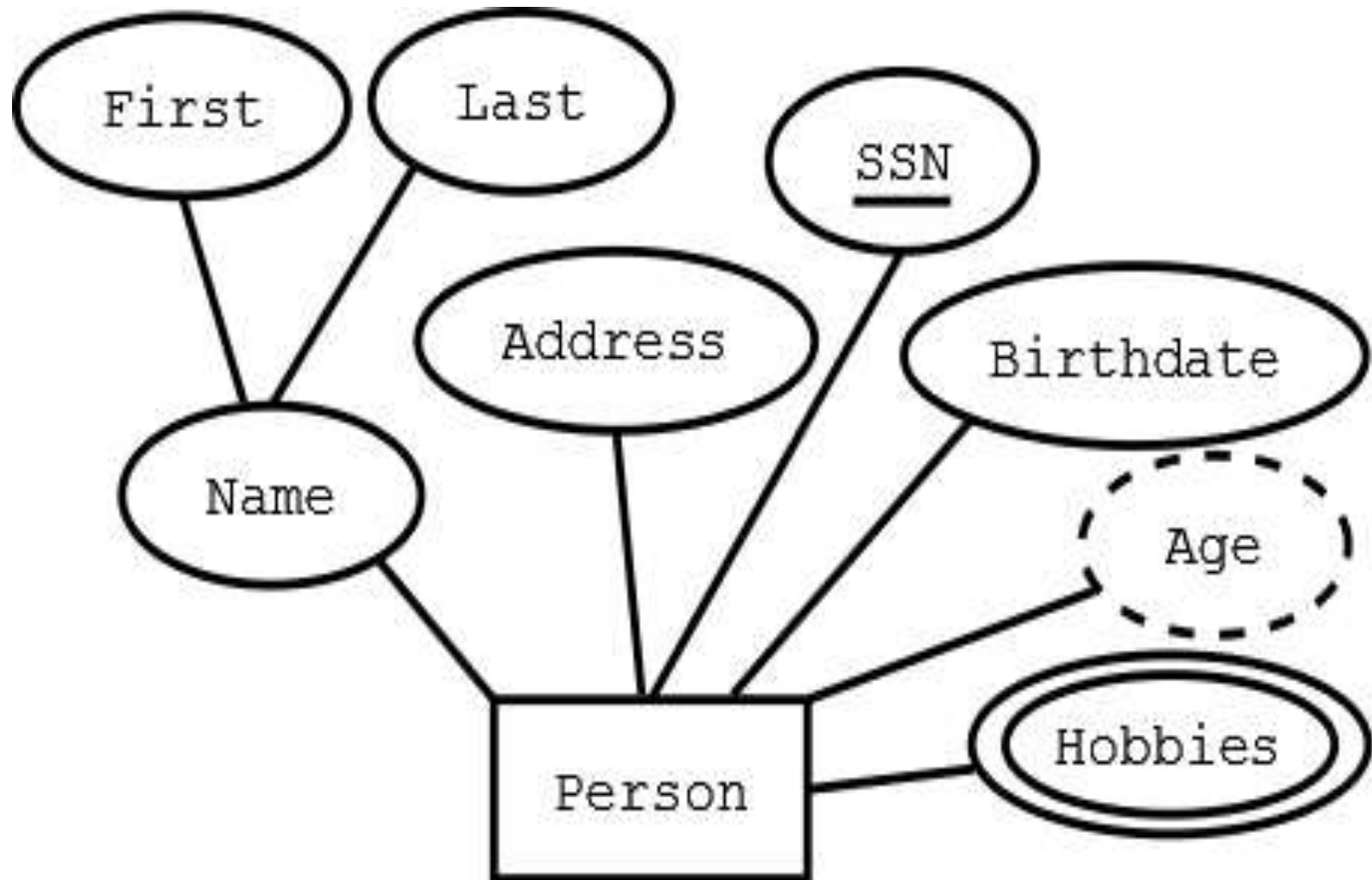
# FOREIGN KEY

---

- An attribute in one table that references the primary key of another table OR it can be null.
- Both foreign and primary keys must be of the same data type
- For example: **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID) – DepartmentID is the Foreign key.

# Graphical Representation in E-R diagram

---



# DEGREE OF A RELATIONSHIP

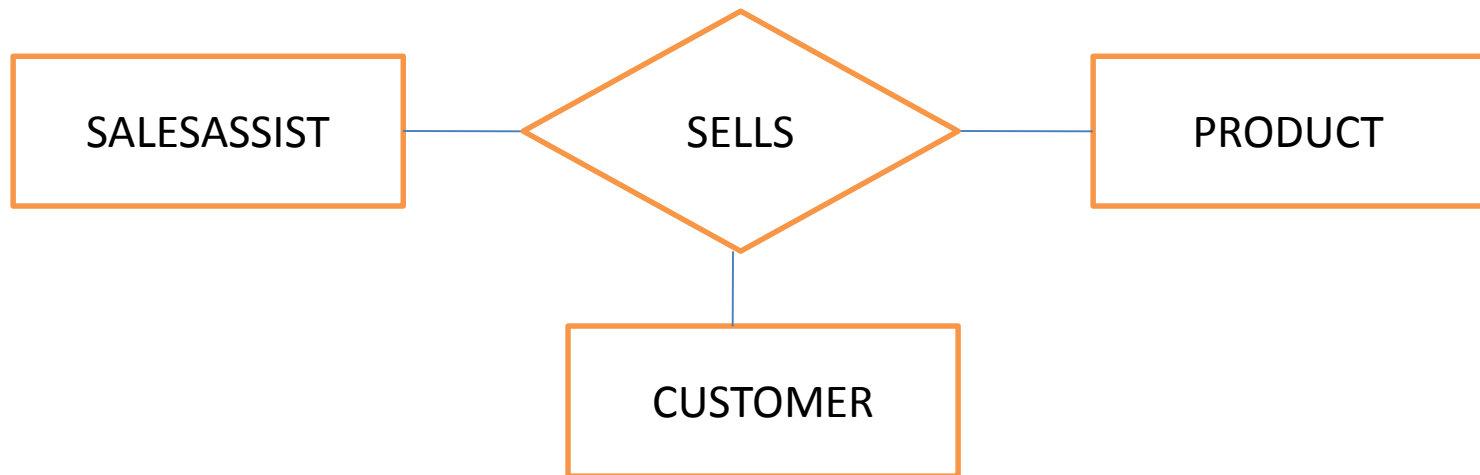
---

It is the number of entity types that participate in a relationship

If there are two entity types involved it is a *binary* relationship type

If there are three entity types involved it is a *ternary* relationship type

It is possible to have a n-array relationship (quaternary)



# CARDINALITY CONSTRAINTS

---

The number of instances of one entity that can or must be associated with each instance of another entity.

If we have two entity types A and B, the cardinality constraint specifies the number of instances of entity B that can (or must) be associated with entity A.

Four possible categories are

***One to one (1:1) relationship***

***One to many (1:m) relationship***

***Many to one (m:1) relationship***

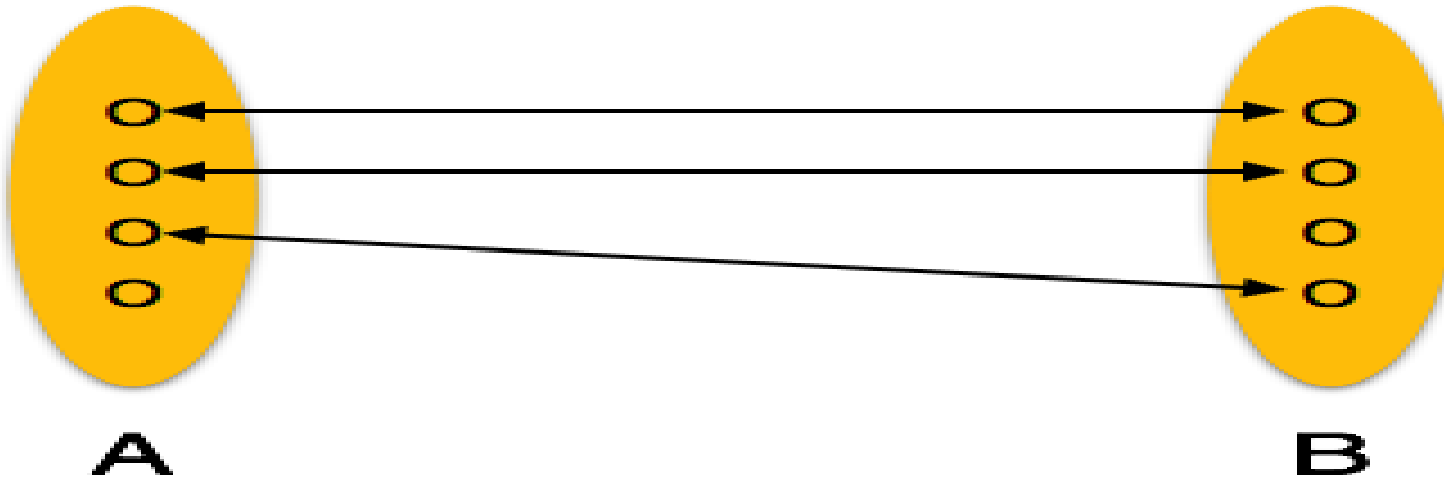
***Many to many (m:n) relationship***



# CARDINALITY CONSTRAINTS

---

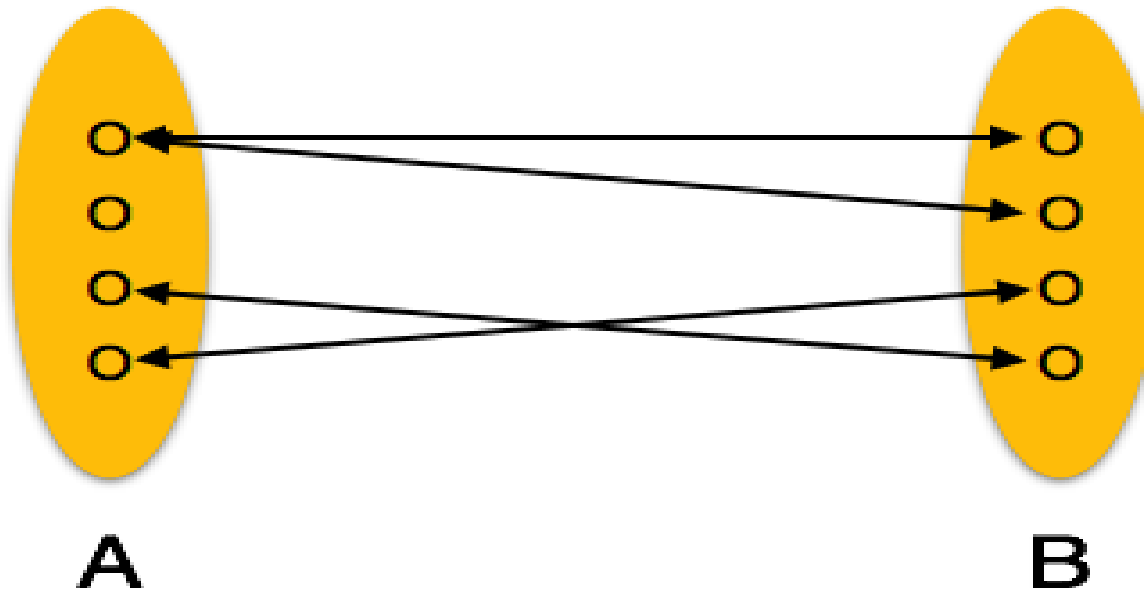
one-to-one



# CARDINALITY CONSTRAINTS

---

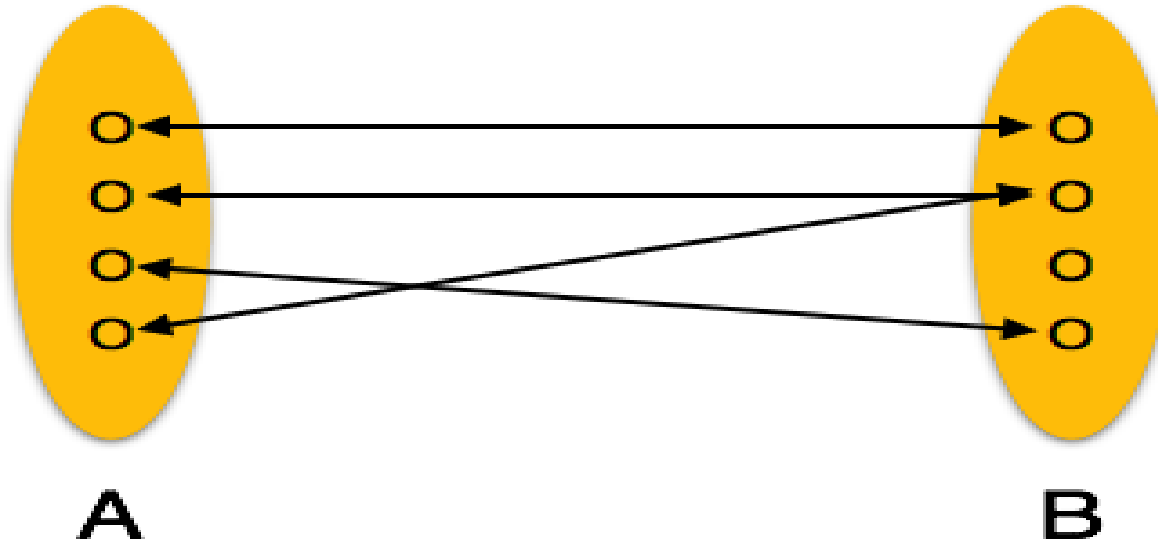
one to many



# CARDINALITY CONSTRAINTS

---

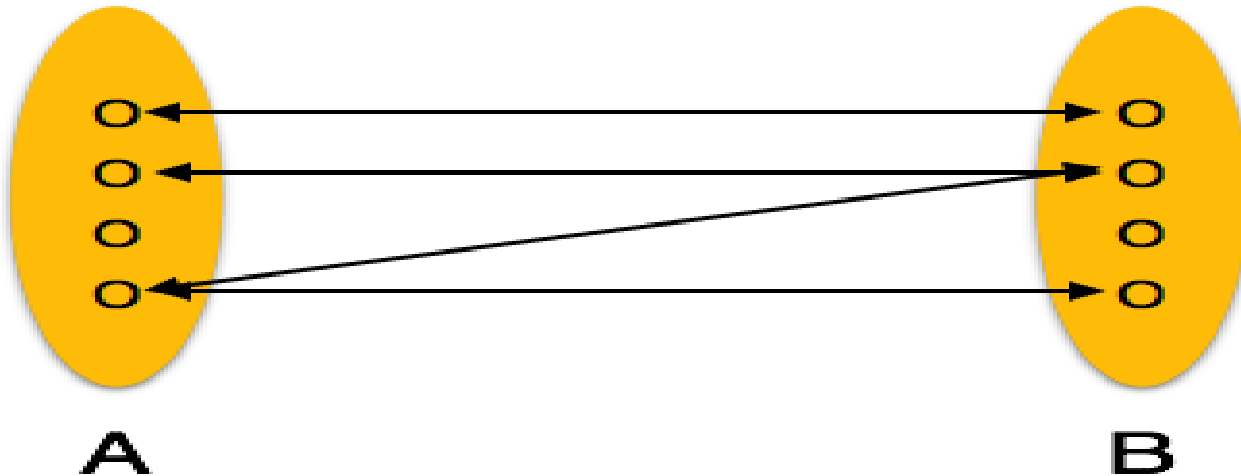
many-to-one



# CARDINALITY CONSTRAINTS

---

many-to-many

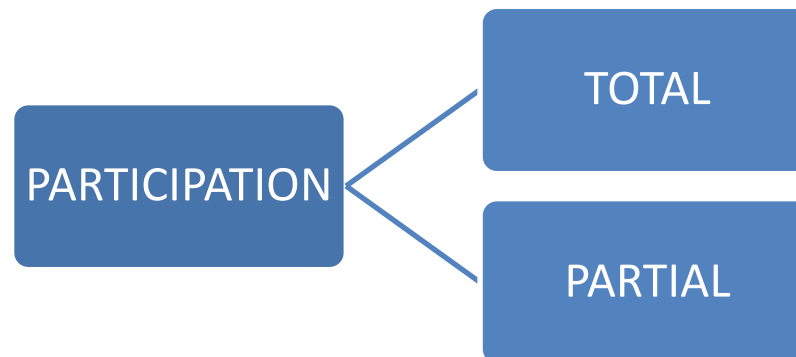


# PARICIPATION CONSTRAINTS (OPTIONALITY)

---

*Specifies if existence of an entity depends on it being related to another entity via relationship.*

- Specifies minimum number of relationship instances each entity can participate in .
- This is called ***minimum cardinality constraint***.
- Two type of the participation are : Total And Partial



- Ex: if company policy says that every employee must work for the department then participation of employee in work-for is total.

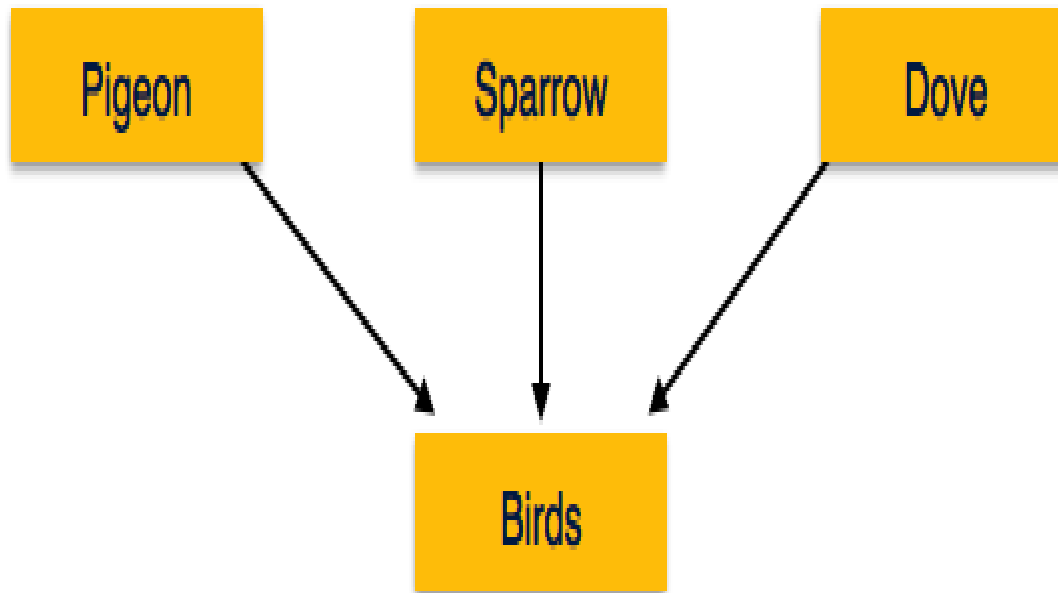


- Total participation is also called *existence dependencies*.
- Every entity in total set of employee must be related to a department via WORKS-FOR
- But **we can't say that every employee must MANAGE a department**
- Hence relationship is *partial*.
- **Total participation is indicated by double line and partial participation by single line.**

# GENERALIZATION

---

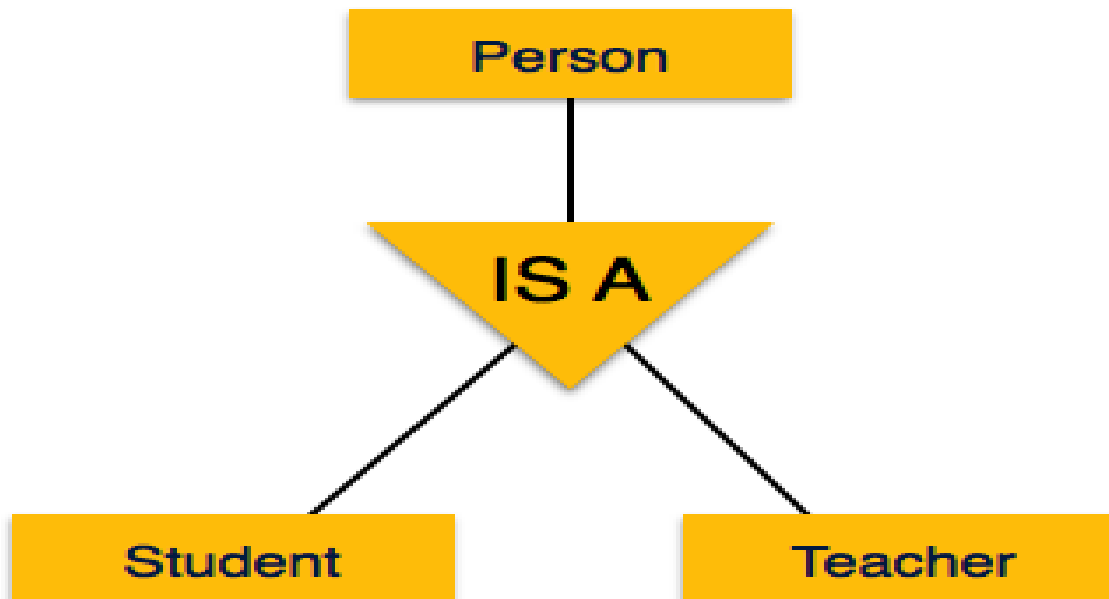
The process of generalizing entities, where the generalized entities contain the properties of all the generalized entities is called Generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For an example, pigeon, house sparrow, crow and dove all can be generalized as Birds.



# SPECIALIZATION

---

Specialization is a process, which is opposite to generalization, as mentioned above. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group Person for example. A person has name, date of birth, gender etc. These properties are common in all persons, human beings. But in a company, a person can be identified as employee, employer, customer or vendor based on what role do they play in company

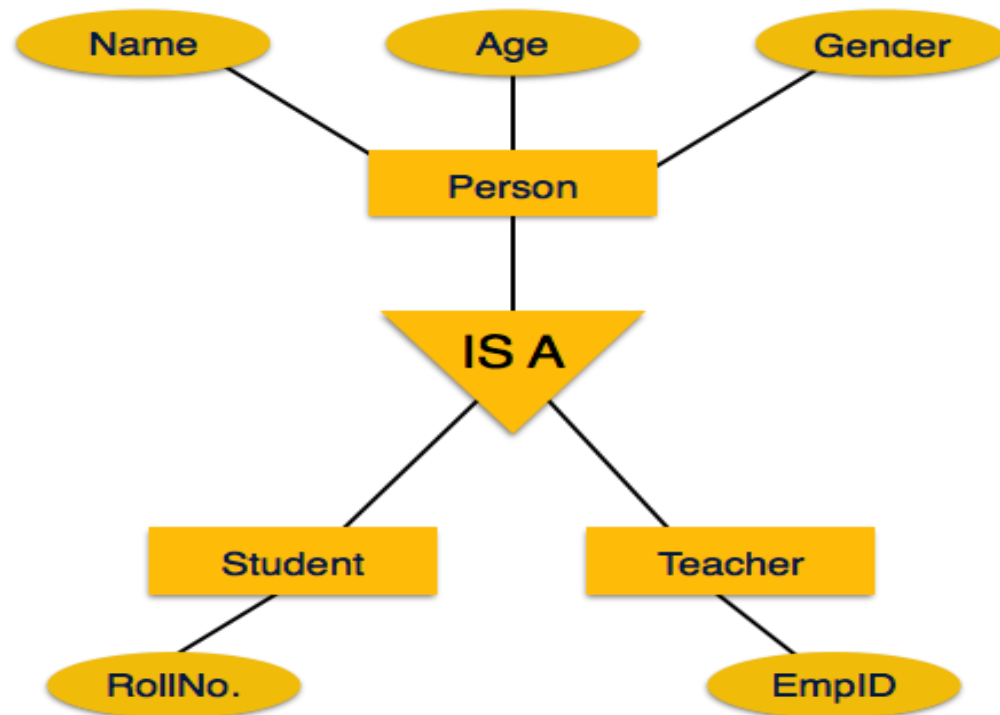




# INHERITANCE

---

One of the important features of Generalization and Specialization, is inheritance, that is, the attributes of higher-level entities are inherited by the lower level entities.



For example, attributes of a person like name, age, and gender can be inherited by lower level entities like student and teacher etc.

# Benefits of ER diagrams

---

ER diagrams constitute a very **useful framework for creating and manipulating databases.**

First, ER diagrams are easy to understand and **do not require a person to undergo extensive training to be able to work with it efficiently and accurately.** This means that designers can use ER diagrams to easily communicate with developers, customers, and end users, regardless of their IT proficiency.

Second, ER diagrams are **readily translatable into relational tables which can be used to quickly build databases.** In addition, ER diagrams can directly be used by database developers as the blueprint for implementing data in specific software applications.

Lastly, ER diagrams may be applied in other contexts such as **describing the different relationships and operations within an organization.**

# CONSTRUCTING AN ER MODEL

- Before beginning to draw the ER model, read the requirements specification carefully.
- Document any assumptions you need to make.

## 1. Identify entities

- list all potential entity types. These are the object of interest in the system. It is better to put too many entities in at this stage and then discard them later if necessary.

## 2.Remove duplicate entities

- Ensure that they really separate entity types or just two names for the same thing
- Also do not include the system as an entity type
- e.g. if modelling a library, the entity types might be books, borrowers, etc.
- The library is the system, thus should not be an entity type.

### 3. List the attributes of each entity

- Ensure that the entity types are really needed.
- Are any of them just attributes of another entity type?
- If so keep them as attributes and cross them off the entity list.
- Do not have attributes of one entity as attributes of another entity!

## 4. Mark the primary keys

- Which attributes uniquely identify instances of that entity type?

## 5. Define the relationships

- Examine each entity type to see its relationship to the others.

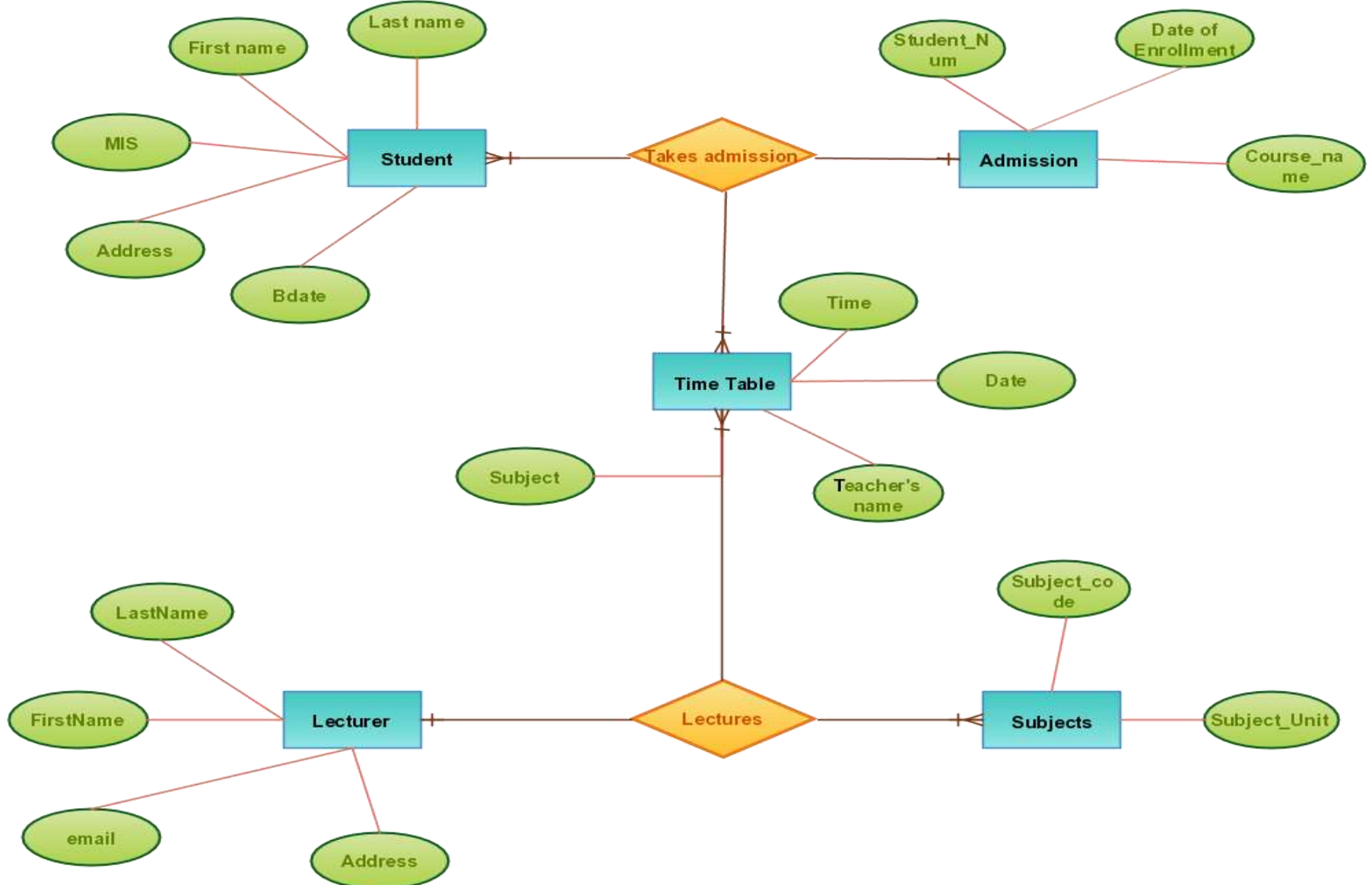
## 6. Describe the cardinality and optionality of the relationships

- Examine the constraints between participating entities.

## 7. Remove redundant relationships

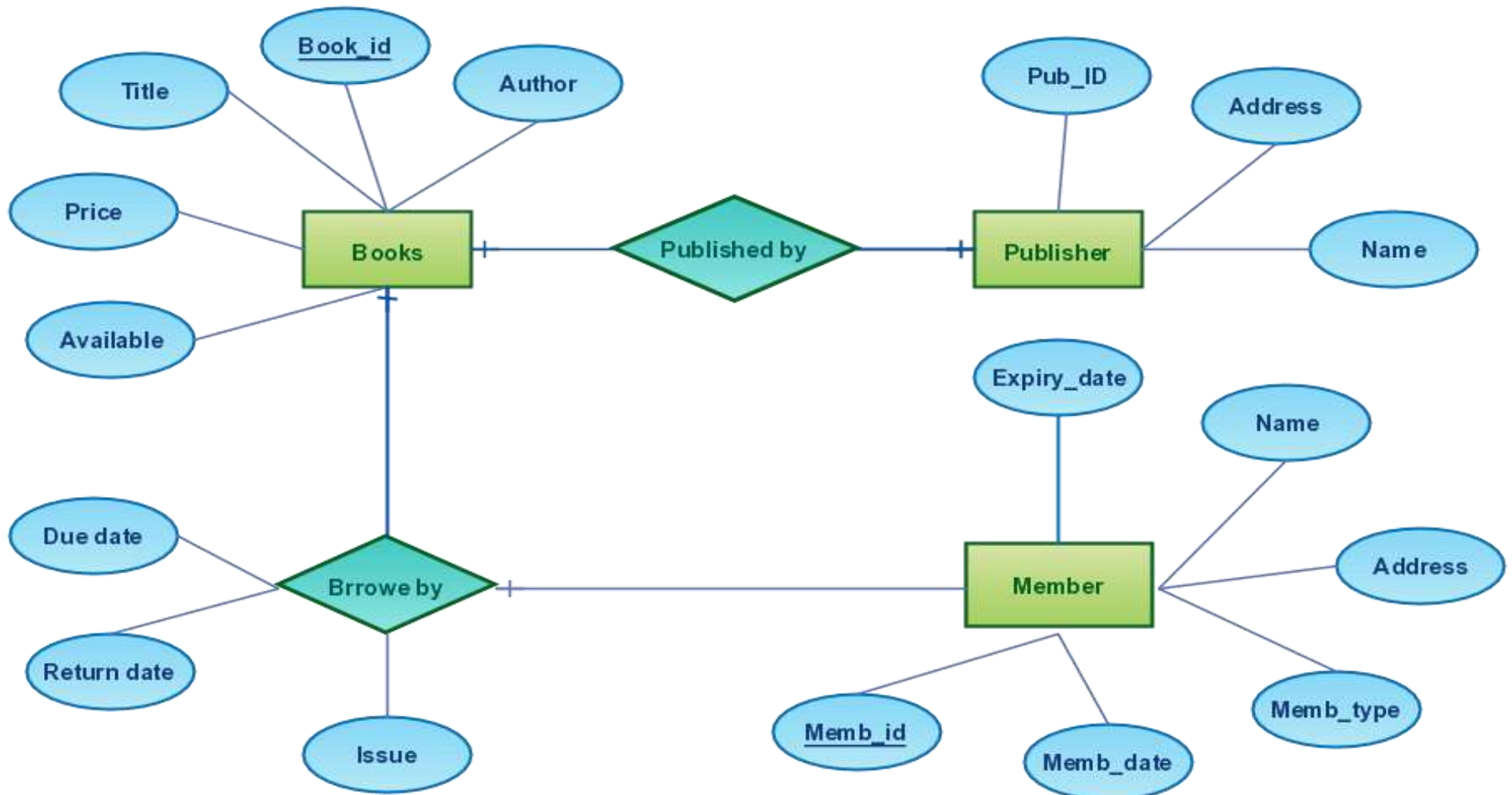
- Examine the ER model for redundant relationships.
- ER modelling is an iterative process, so draw several versions, refining each one until you are happy with it. Note that there is no one right answer to the problem, but some solutions are better than others!

## ER DIAGRAM FOR COLLEGE MANAGEMENT SYSTEM





## E-R Diagram for Library Management System



# EXAMPLE

- *“A Country Bus Company owns a number of busses. Each bus is allocated to a particular route, although some routes may have several busses. Each route passes through a number of towns. One or more drivers are allocated to each stage of a route, which corresponds to a journey through some or all of the towns on a route. Some of the towns have a garage where busses are kept and each of the busses are identified by the registration number and can carry different numbers of passengers, since the vehicles vary in size and can be single or double-decked. Each route is identified by a route number and information is available on the average number of passengers carried per day for each route. Drivers have an employee number, name, address, and sometimes a telephone number.”*

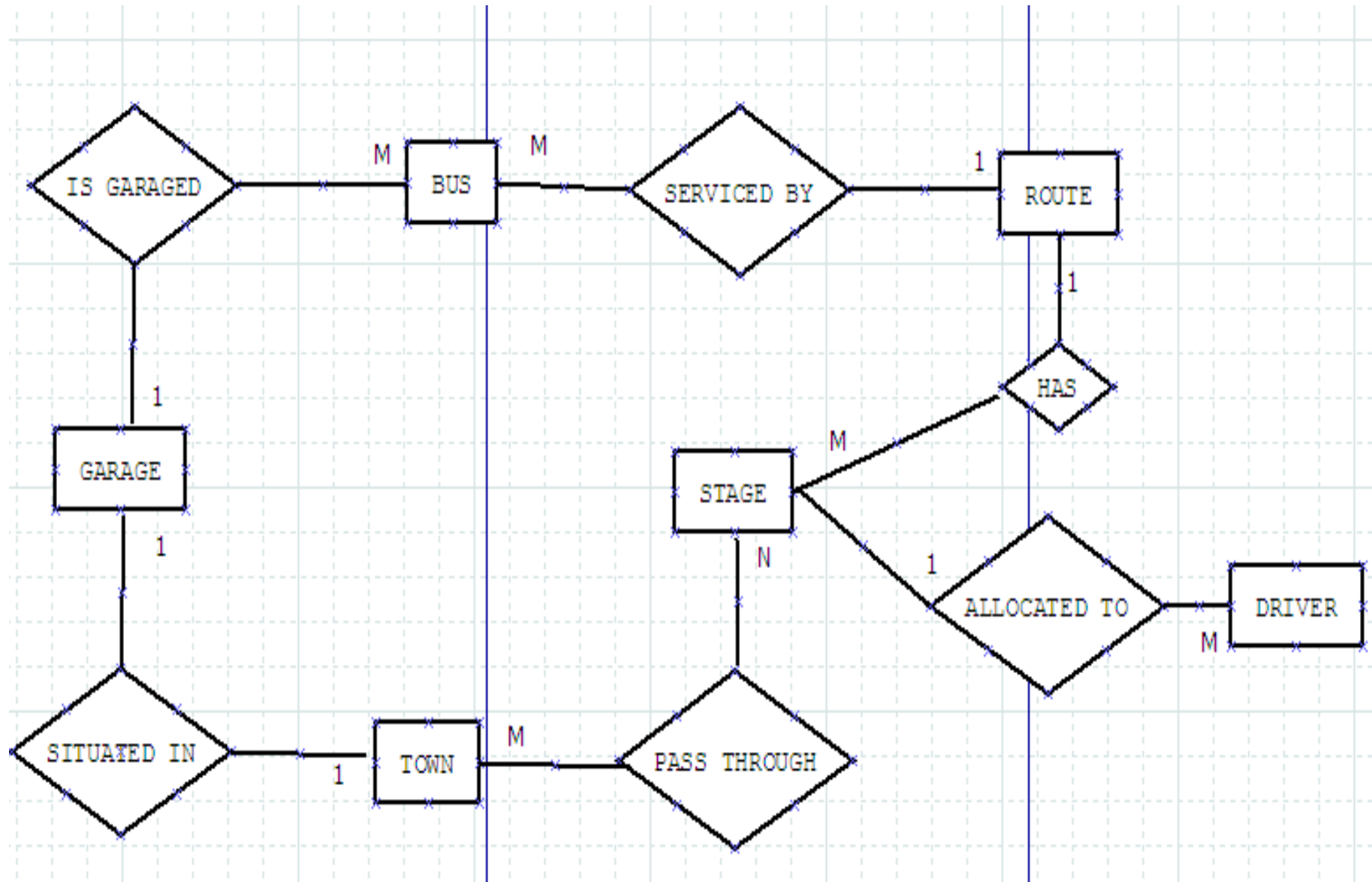
# ENTITIES

- ***Bus*** - Company owns busses and will hold information about them.
- ***Route*** - Buses travel on routes and will need described.
- ***Town*** - Buses pass through towns and need to know about them
- ***Driver*** - Company employs drivers, personnel will hold their data.
- ***Stage*** - Routes are made up of stages
- ***Garage*** - Garage houses buses, and need to know where they are.

# RELATIONSHIPS

- A bus is allocated to a route and a route may have several buses.  
Bus-route (m:1) - *is serviced by*
- A route comprises of one or more stages.  
route-stage (1:m) *comprises/has*
- One or more drivers are allocated to each stage.  
driver-stage (m:1) *is allocated*
- A stage passes through some or all of the towns on a route.  
stage-town (m:n) *passes-through*
- A route passes through some or all of the towns  
route-town (m:n) *passes-through*
- Some of the towns have a garage  
garage-town (1:1) *is situated*
- A garage keeps buses and each bus has one 'home' garage  
garage-bus (m:1) *is garaged*

# ER DIAGRAM



# ATTRIBUTES

- Bus (reg- no,make,size,deck,no-pass)
- Route (route-no,avg-pass)
- Driver (emp - no,name,address,tel-no)
- Town (name)
- Stage (stage - no)
- Garage (name,address)

MORE TECHNIQUES

# **More techniques for data analysis**

---

There are two main techniques available to analyze and represent complex processing logic:

- 1. Decision trees and**
- 2. Decision tables.**



# DECISION TREES

---

A decision tree gives a **graphic view** of the processing logic involved in decision making and the corresponding actions taken.

The **edges** of a decision tree represent **conditions** and the **leaf nodes** represent the **actions** to be performed depending on the outcome of testing the condition.

# EXAMPLE

---

Consider **Library Membership Automation Software** (LMS) where it should support the following three options:

1. New member
2. Renewal
3. Cancel membership

## New member option

**Decision:** When the 'new member' option is selected, the software asks details about the member like member's name, address, phone number etc.

**Action:** If proper information is entered, then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

# Example Contd..

---

## Renewal option

**Decision:** If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

**Action:** If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

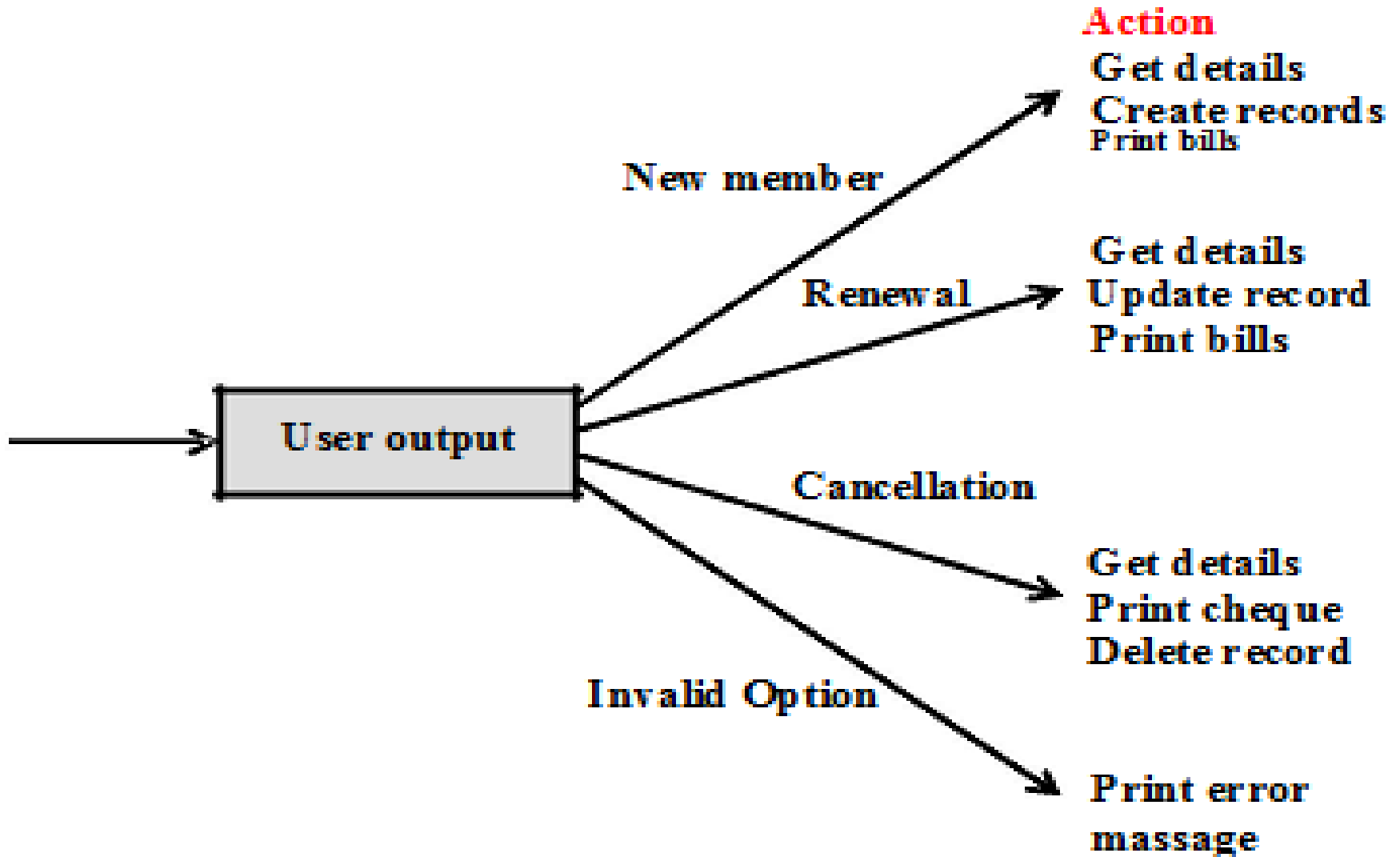
## Cancel membership option

**Decision:** If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

**Action:** The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.

# Decision Tree Representation

---



# DECISION TABLES

---

Decision tables are used mainly because of their visibility, clearness, coverage capabilities, low maintenance and automation fitness.

## STRUCTURE

### The four quadrants

Conditions	Condition alternatives
Actions	Action entries

# Decision Table for LMS

Conditions				
Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

## EXAMPLE

---

Suppose a technical support company writes a decision table to **diagnose printer problems based upon symptoms described to them over the phone from their clients.**

# Decision Table

---

Printer troubleshooter

		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognised	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				



# BENEFITS

---

- Decision tables, especially when coupled with the use of a [domain-specific language](#), **allow developers and policy experts to work from the same information**, the decision tables themselves.
- Tools to render nested if statements from traditional programming languages into decision tables can also be **used as a debugging tool**.
- Decision tables have proven to be **easier to understand and review than code**, and have been used extensively and successfully to produce specifications for complex systems.

# REQUIREMENTS DOCUMENTATION

# REQUIREMENTS DOCUMENTATION

---

This is the way of representing requirements in a consistent format.

It is called **Software Requirement Specification(SRS)**.

SRS serves many purpose depending upon who is writing it.

- written by customer
- written by developer

**Serves as contract between customer & developer.**

# Nature of the SRS

---

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

- a) Functionality
- b) External Interfaces
- c) Performance
- d) Attributes
- e) Design Constraints imposed on an implementation.

SRS Should

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

# Characteristics of a good SRS

---

SRS should be

1. Correct
2. Unambiguous
3. Complete
4. Consistent
5. Ranked for importance and/or stability
6. Verifiable
7. Modifiable
8. Traceable

# Advantages of a SRS

---

Software SRS establishes the basic for agreement between the client and the supplier on what the software product will do.

1. A SRS provides a reference for validation of the final product.
2. A high-quality SRS is a prerequisite to high-quality software.
3. A high-quality SRS reduces the development cost.

# Problems without a SRS Document

---

The important problems that an organization would face if it does not develop an SRS document are as follows:

- Without developing the SRS document, **the system would not be implemented according to customer needs.**
- **Software developers would not know whether what they are developing is what exactly is required by the customer.**
- Without SRS document, **it will be very difficult for the maintenance engineers to understand the functionality of the system.**
- **It will be very difficult for user document writers to write the users' manuals properly without understanding the SRS document.**

# Organization of the SRS

---

IEEE has published guidelines and standards to organize an SRS.

## 1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definition, Acronyms and abbreviations
- 1.4 References
- 1.5 Overview

## 2. The Overall Description

- 2.1 Product Perspective
  - 2.1.1 System Interfaces
  - 2.1.2 Interfaces
  - 2.1.3 Hardware Interfaces
  - 2.1.4 Software Interfaces
  - 2.1.5 Communication Interfaces
  - 2.1.6 Memory Constraints
  - 2.1.7 Operations
  - 2.1.8 Site Adaptation Requirements



# Organization of the SRS

---

- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Assumptions for dependencies
- 2.6 Apportioning of requirements

## 3. Specific Requirements

- 3.1 External Interfaces
- 3.2 Functions
- 3.3 Performance requirements
- 3.4 Logical database requirements
- 3.5 Design Constraints
- 3.6 Software System attributes
- 3.7 Organization of specific requirements
- 3.8 Additional Comments.

## 4. Change Management Process

## 5. Document Approvals

## 6. Supporting Information

# **REQUIREMENTS VALIDATION**

# REQUIREMENTS VALIDATION

---

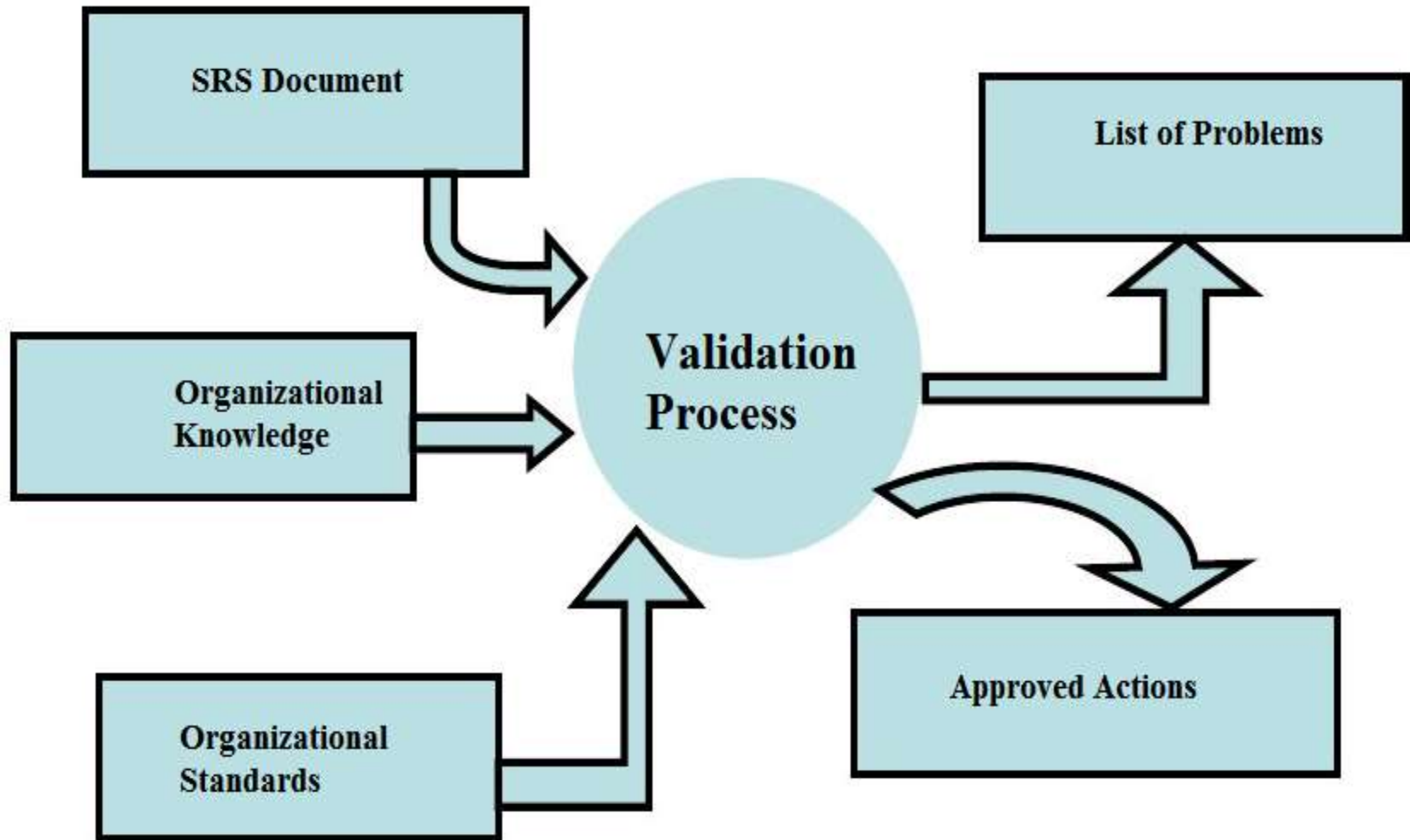
After the completion of SRS document, we would like to check the document for:

- Completeness & consistency
- Conformance to standards
- Requirements conflicts
- Technical errors
- Ambiguous requirements

The objective of requirements validation is **to certify that the SRS document is an acceptable document of the system to be implemented.**

# REQUIREMENTS VALIDATION

---



# VALIDATION TECHNIQUES

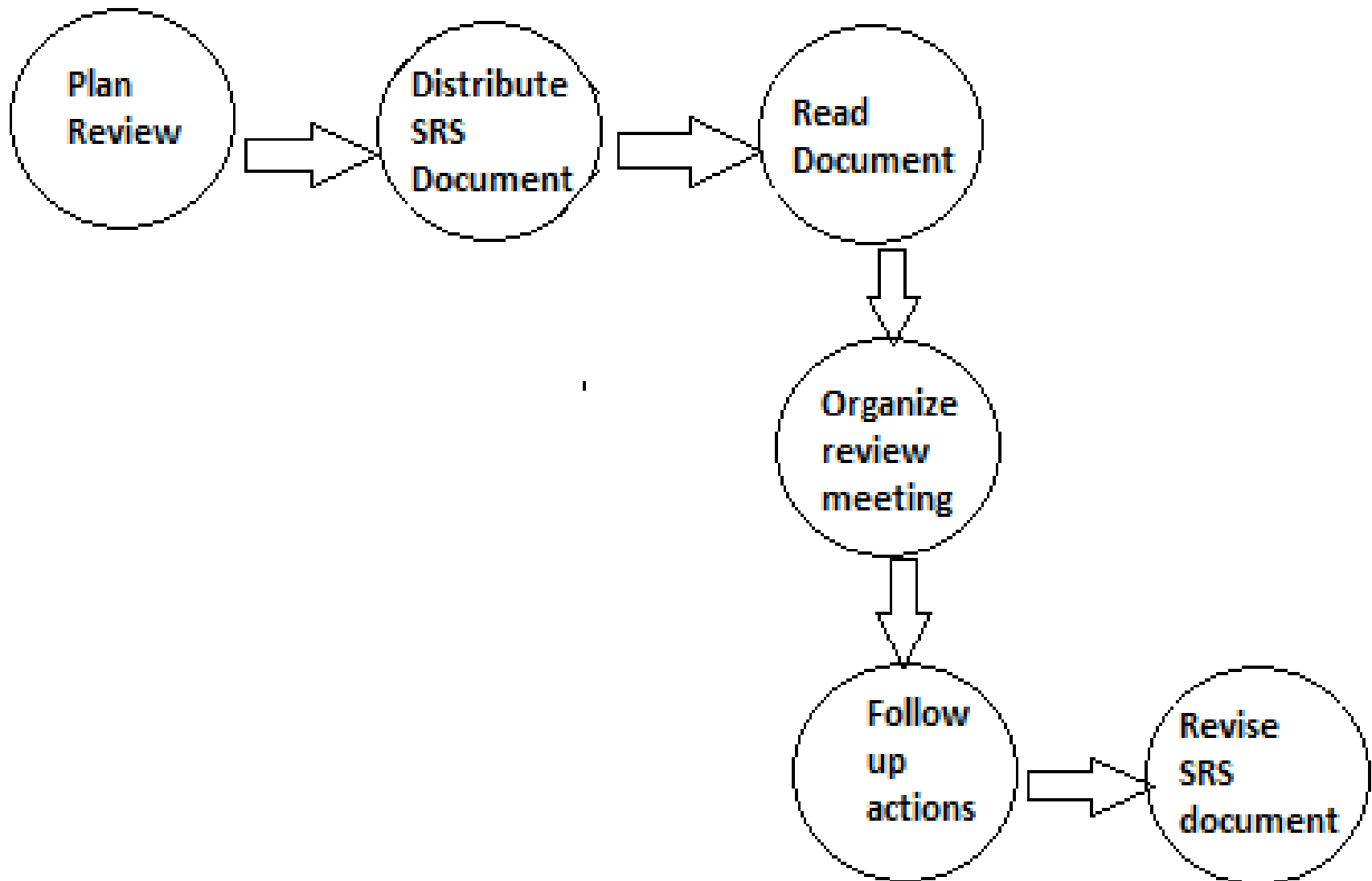
---

**There are two requirements validation techniques:-**

- 1. Requirements Reviews**
- 2. Prototyping**

# REQUIREMENTS REVIEWS

---



# REQUIREMENTS VALIDATION

---

## **Problem actions**

- Requirements clarification
- Missing information (find this information from stakeholders)
- Requirements conflicts (Stakeholders must negotiate to resolve this conflict)
- Unrealistic requirements
- Security issues

# REVIEW CHECKLISTS

---

- Understandability
- Redundancy
- Completeness
- Ambiguity
- Consistency
- Organization
- Conformance to standards
- Traceability



# Requirement Management

---

Process of understanding and controlling changes to system requirements.

## **ENDURING & VOLATILE REQUIREMENTS**

- o Enduring requirements: They are core requirements & are related to main activity of the organization.

Example: issue/return of a book, cataloging etc.

- o Volatile requirements: likely to change during software development life cycle or after delivery of the product

# Requirement Management Planning

---

- Very critical.
- Important for the success of any project.

# Requirement Change Management

---

**Requirements change process should include the following activities to be carried out when a change is needed in the requirements:-**

- Assignment of responsibilities
- Management of changes
- Documentation
- Requirements traceability
- Communication of change
- Establishment of baseline

# **SOFTWARE QUALITY ASSURANCE**

# Software Quality

---

Our objective of software engineering is to produce good quality maintainable software in time and within budget.

Here, quality is very important.

Different people understand different meaning of quality like:

- Conformance to requirements
- Fitness for the purpose
- Level of satisfaction

When user uses the product, and finds the product fit for its purpose, he/she feels that product is of good quality.

# Software Quality Assurance

---

**Software quality assurance (SQA)** consists of a means of monitoring the software engineering processes and methods used to ensure quality.

Every software developers will agree that high-quality software is an important goal. Once said, "Every program does something right, it just may not be the thing that we want it to do."

# Software Quality Assurance

---

The definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. **Lack of conformance to requirements** is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. **If the criteria are not followed**, lack of quality will almost surely result.
3. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). **If software conforms to its explicit requirements but fails to meet implicit requirements**, software quality is suspect.

# Verification and Validation

---

It is the name given to the **checking** and **analysis** process.

The **purpose** is to ensure that the software conforms to its specifications and meets the need of the customer.

**Verification** represents the set of activities that are carried out to confirm that the software correctly implements the specific functionality.

**Validation** represents set of activities that ensure that the software that has built is satisfying the customer requirements.



# Verification and Validation

---

Verification	Validation
Are we building the product right?	Are we building the right product?
Ensure that the software system meets all the functionality	Ensure that the functionalities meet the intended behavior.
Verifications take place first and include the checking for documentation, code etc	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers	Done by testers
Have static activities as it includes the reviews, walk-throughs and inspections to verify that software is correct or not	Have dynamic activities as it includes executing the software against the requirements.

# Verification and Validation

---

In the verification and validation, two techniques of system checking and analysis may be used:-

## 1. Software Inspection

## 2. System testing

The testing can be carried out using following tests:-

- i. Unit testing
- ii. Module testing
- iii. System testing
- iv. Acceptance testing

# SQA Plans

---

The SQA Plan provides a **road map** for instituting software quality assurance. Developed by the SQA group, the plan serves as a template for SQA activities that are instituted for each software project.

The documentation section describes (by reference) each of the work products produced as part of the software process. These include –

- project documents (e.g., project plan)
- models (e.g., ERDs, class hierarchies)
- technical documents (e.g., specifications, test plans)
- user documents (e.g., help files)

# Methods for SQA

---

The methods by which quality assurance is accomplished are many and varied, out of which, two are mentioned as follows:

1. ISO 9000
2. Capability Maturity Model

# ISO 9000

---

- “ISO” in greek means “equal”, so the association wanted to convey the idea of equality.
- It is an attempt to improve software quality based on ISO 9000 series standards.
- It has been adopted by over 130 countries including India and Japan.
- One of the **problem** with ISO-9000 series standard is that it is not industry specific.
- It can be interpreted by the developers to diverse projects such as hair dryers, automobiles, televisions as well as software.

# Notes

---

- ISO-9000 applies to all types of organizations.
- After adopting the standards, a country typically permits only ISO registered companies to supply goods and services to government agencies and public utilities.
- ISO-9000 series is not just software standard, but are applicable to a wide variety of industrial activities including design/development, production, installation and servicing.

# ISO 9000 Certification

---

This process consists of following stages:-

1. Application
2. Pre-assessment
3. Document review and adequacy of audit
4. Compliance audit
5. Registration
6. Continued surveillance

# ISO 9000 Series

---

The types of software industries to which the different ISO standards apply are as follows:

**ISO 9001** : This standard applies to the organization engaged in **design, development, production & servicing of goods**. This is the standard that is applicable to most **software development organization**.

**ISO 9002** : This standard applies to the organization which **do not design products but only involved in production**. E.g, Car manufacturing industries.

**ISO 9003** : This standard applies to the organization **involved only in installation and testing** of the products.



# Benefits of ISO 9000 Certification

---

1. Continuous Improvement
2. Improved Customer Satisfaction
3. Eliminate Variations
4. Better product and Services
5. Improved Profit levels
6. Improved Communication
7. Reduced Cost

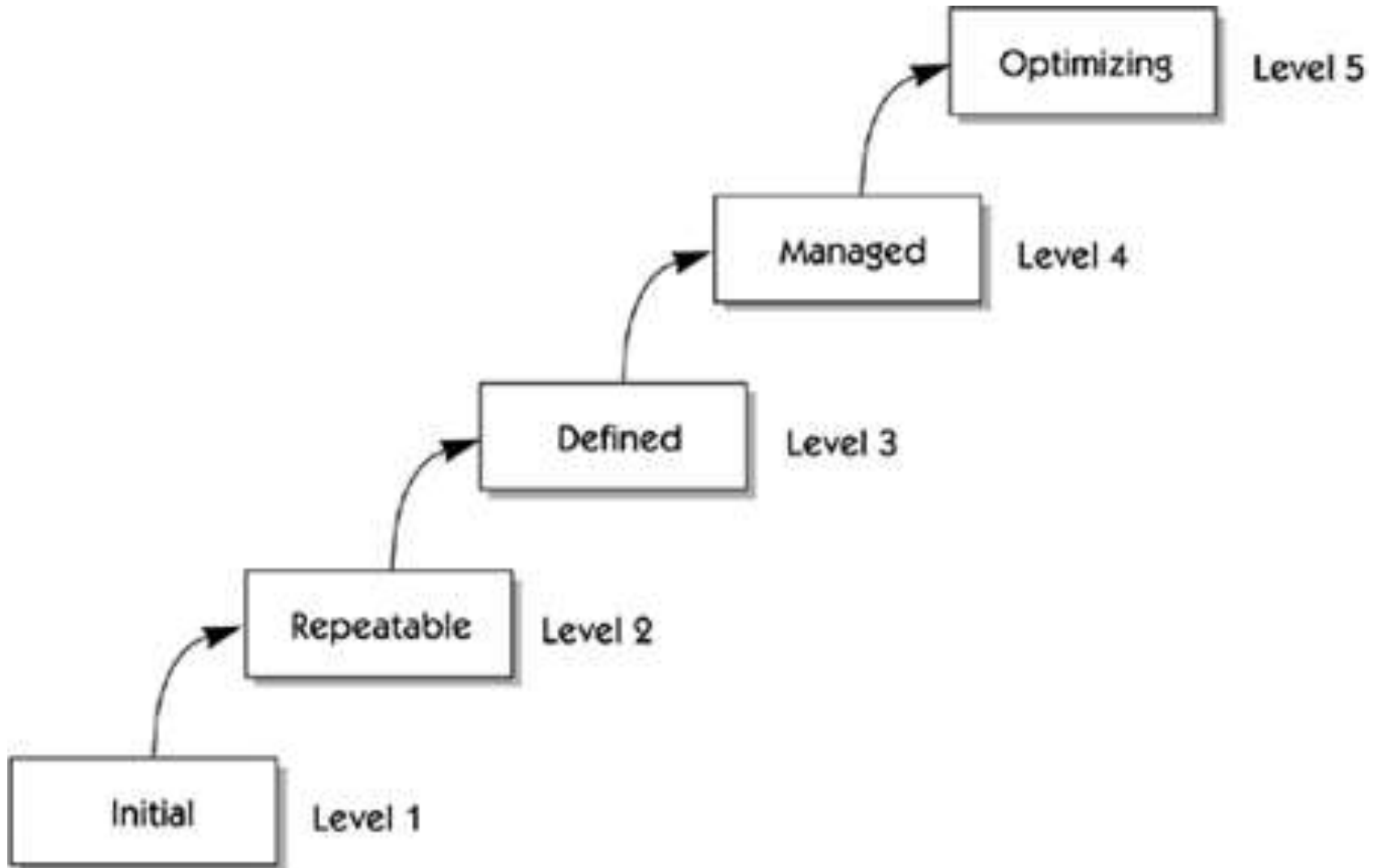
# Capability Maturity Model

---

- It was developed by Software Engineering Institute (SEI) of **Carnegie-Mellon University** in 1986.
- It specifies an increasing series of levels of a **software development organization**. The higher the level, the better the software development process.
- It can be used in two ways: **Capability evaluation** and **Software process assessment**.

# CMM Levels

---



# CMM Levels

---

- **Level One :Initial** - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.
- **Level Two: Repeatable** - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.
- **Level Three: Defined** - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing, testing and maintaining the application.

# CMM Levels

---

- **Level Four: Managed** - Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.
- **Level Five: Optimizing** - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

# Key Process Areas of each Level

---

CMM Level	Focus	Key Process Areas
Initial	Competent people	-
Repeatable	Project Management	Software Project Planning Software Configuration Management
Defined	Definition of Processes	Process Definition Training Program Peer Reviews
Managed	Product and process quality	Quantitative Process Metrics Software Quality Management
Optimizing	Continuous Process Improvement	Defect Prevention Process Change Management Technology Change Management

# Comparison of ISO-9000 Certification & SEI-CMM

---

ISO focus on the “customer-supplier relationship”, whereas CMM focus on software supplier to improve its processes to achieve a higher quality product for the benefit of the customer.

ISO 9000 standard is written for a wide range of industries whereas CMM framework is specifies for the software industry.

CMM is a five level framework for measuring software engineering practices, and ISO 9000 defines a minimum level of attributes for a quality management program.

The ISO 9000's concept is to follow a set of standards to make **success repeatable**. The CMM emphasizes a process of **continuous improvement**.

Once an organization has met the criteria to be ISO certified by an independent audit, the next step is only to maintain that level of certification. CMM is an on-going process of evaluation and improvement, moving from one level of achievement to the next. Even at the highest level of maturity, the focus is on continuous improvement.

**END OF UNIT-2**