Software Engineering

Unit 2

Classification of Software Requirements

- According to IEEE standard 729, a requirement is defined as follows:

- A condition or capability needed by a user to solve a problem or achieve an objective.

- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents

- A documented representation of a condition or capability as in 1 and 2.

- **A software requirement can be of :**
  - Functional requirements

Classification of Software Requirements

- Non-functional requirements
- Interface Requirements
- Inverse Requirements
- Design and Constraints Requirements

Functional Requirements

- These are the requirements that the end user specifically demands as basic facilities that the system should offer.

- All these functionalities need to be necessarily incorporated into the system as a part of the contract.

- These are represented or stated in the form of input to be given to the system, the operation performed, and the output expected.

Functional Requirements

- They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
- Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.
- There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

Non-Functional Requirements

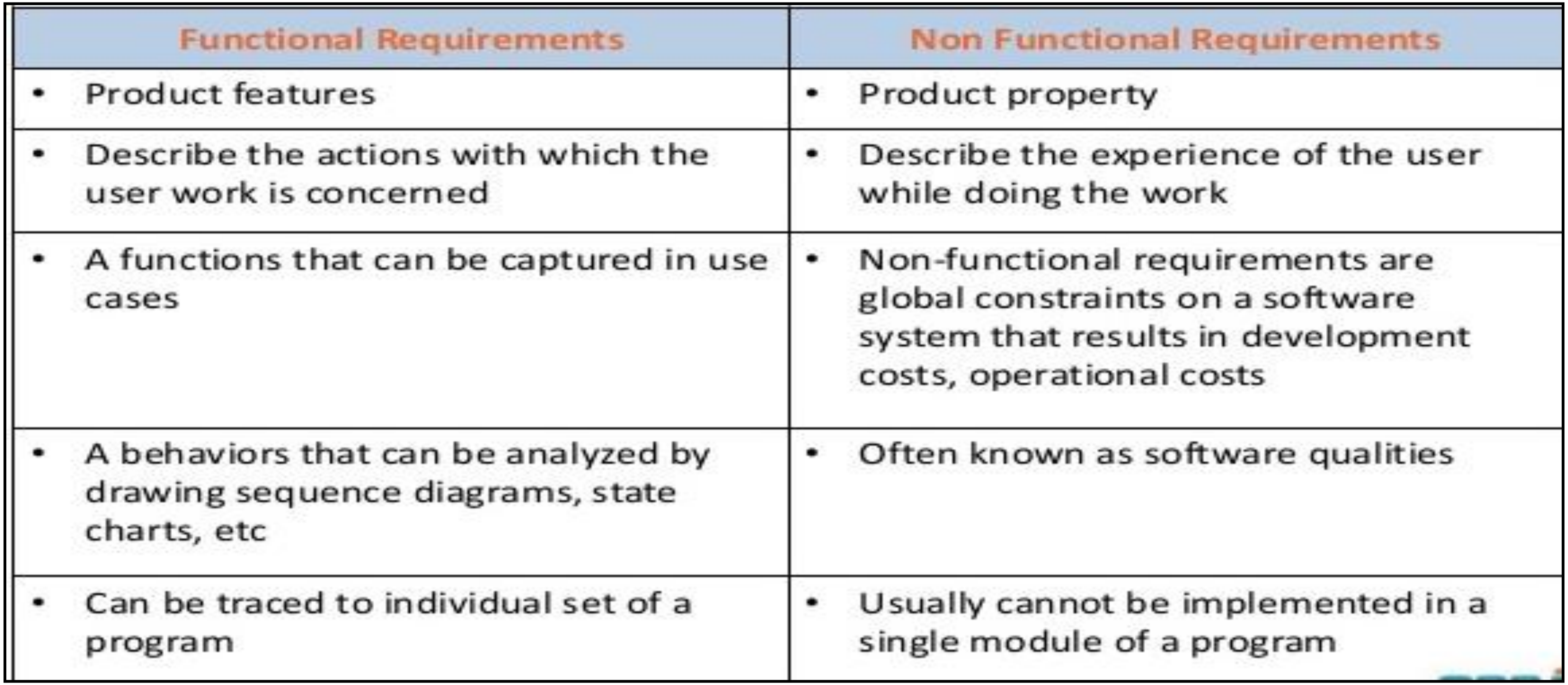
- These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:
- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Non-Functional Requirements

- NFR’s are classified into following types:
    - Interface constraints
    - Performance constraints: response time, security, storage space, etc.
    - Operating constraints
    - Life cycle constraints: maintainability, portability, etc.
    - Economic constraints

- The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Difference between Functional Requirement and Non-functional Requirement



Functional Requirements	Non Functional Requirements
• Product features	• Product property
• Describe the actions with which the user work is concerned	• Describe the experience of the user while doing the work
• A functions that can be captured in use cases	• Non-functional requirements are global constraints on a software system that results in development costs, operational costs
• A behaviors that can be analyzed by drawing sequence diagrams, state charts, etc	• Often known as software qualities
• Can be traced to individual set of a program	• Usually cannot be implemented in a single module of a program

Interface Requirements

- Systems must operate with other systems and the operating interfaces must be specified as part of requirements. It includes:

1. User Interface
2. Communication Interface

Interface Requirements

1. Software Interface
2. Hardware Interface

1. User Interface: Describe the logical characteristics of each user interface that the system needs. Example includes GUI, CLI (Command line Interface), API (Application Program Interface) etc.

2. Communication Interface: State the requirements for any communication functions the product will use, including e-mail, Web browser, network communications protocols, and electronic forms. Define any pertinent message formatting. Specify communication security or encryption issues, data transfer rates, and synchronization mechanisms.

Interface Requirements

3. Software Interface: Describe the connections between this product and other software components (identified by name and version), including databases, operating systems, tools, libraries, and integrated commercial components. State the purpose of the messages, data, and control items exchanged between the software components. Describe the services needed by external software components and the nature of the inter-component communications. Identify data that will be shared across software components. If the data-sharing mechanism must be implemented in a specific way, such as a global data area, specify this as a constraint.

4. Hardware Interface: Describe the characteristics of each interface between the software and hardware components of the system. This description might include the supported device types, the data and control interactions between the software and the hardware, and the communication protocols to be used.

Interface Requirements

5. Inverse Requirements

- Inverse requirements specify the constraints on allowable behavior. Software safety and security requirements are usually stated in this manner.
- Inverse requirements can be functional and nonfunctional.
- For example: When a customer specifies that something must not be done. For example, User ID should only contain digits.

Design and Implementation Requirements

6. Design and Implementation Requirements

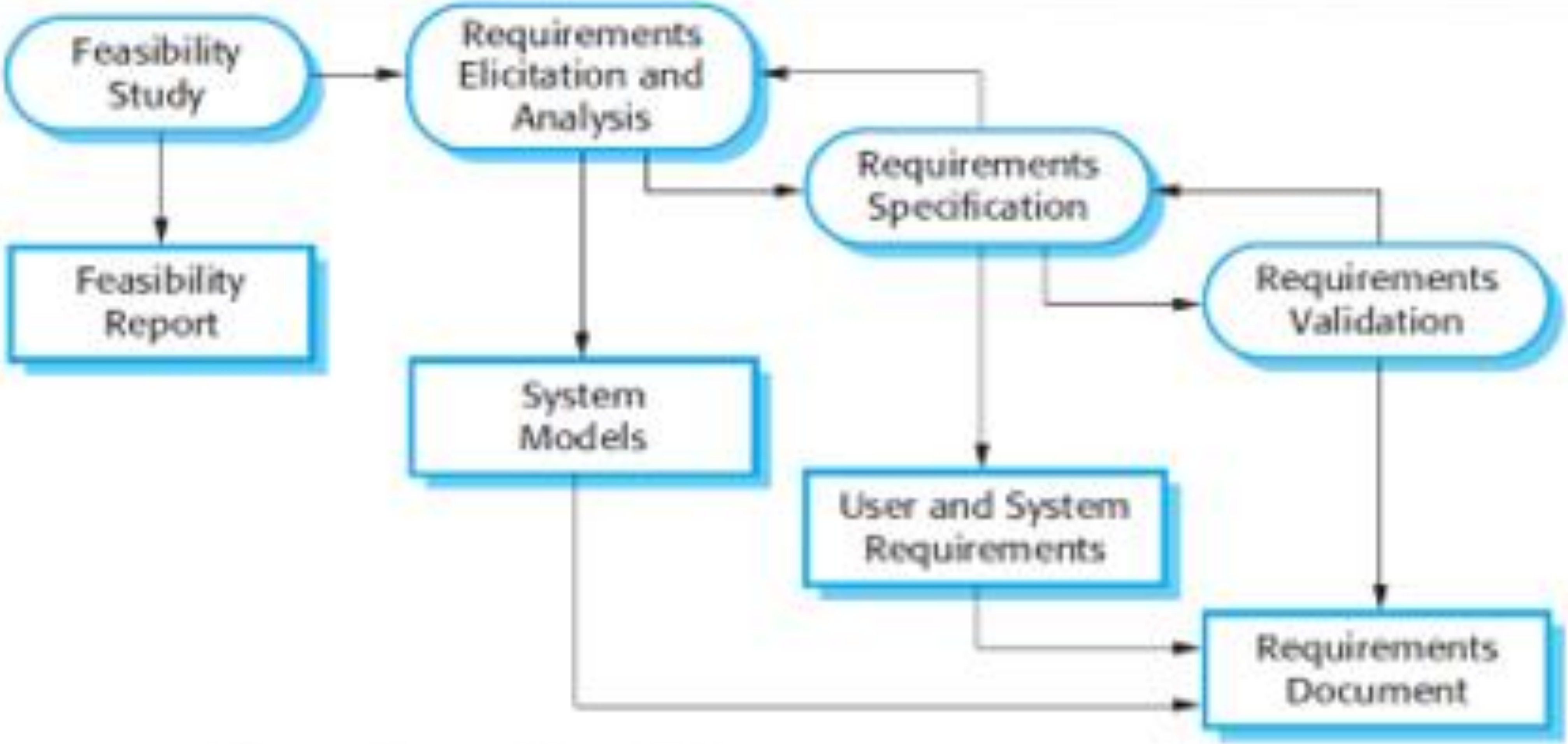
Design constraints and implementation constraints are boundary conditions on how the required software is to be constructed and implemented.

Examples of design constraints include the fact that the software must run using a certain database system or that the software must fit into the memory of a 512 Kbyte machine.

Requirement Engineering

- According to **Pohl** in 1993, “it can be defined as the process of developing requirements through an interactive, co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained”.

- Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:
  - Requirements elicitation
  - Requirement Analysis
  - Requirements specification
  - Requirements verification and validation
  - Requirements management


Feasibility Study
Requirements Elicitation and Analysis
Requirements Specification
Requirements Validation
Feasibility Report
System Models
User and System Requirements
Requirements Document

Requirements Elicitation

- It is related to the various ways used to gain knowledge about the project domain and requirements.
- The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.
- The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique (opinions are most valuable, traditionally industry experts), prototyping, etc. Elicitation does not produce formal models of the requirements understood.
- Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements Analysis

- Requirements Analysis is the activity during which the requirements gathered elicitation are analyzed for **conflicts, ambiguity, inconsistencies, missing requirements or extra requirements**.
- This activity reviews all requirements and may provide **a graphical view of the entire system**.
- **After the completion of analysis**, it is expected that the **understandability of the project** may improve significantly.

- An important part of the analysis effort is to model the system to understand what you are trying to understand.

Requirement Specification

- The purpose of the requirements analysis is to identify requirements for the proposed system. The emphasis is on the discovery of user requirements. Each requirement (or problem) must be defined and documented in the requirements catalogue.
- This activity is used to produce formal software requirement models.
- All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality.
- During specification, more knowledge about the problem may be required which can again trigger the elicitation process.
- The models used at this stage include ER diagrams, data flow diagrams(DFDs), data dictionaries, etc.
- Requirements specification is the activity during which requirements are recorded in one or more forms, usually in a Software Requirement Specification Document (SRS)

Requirement Verification and Validation

- **Concerned with demonstrating that the requirements define the system that the customer really wants**.
- Requirements error costs are high, so **validation is very important**
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
- **Verification:** It refers to the set of tasks that ensures that the software correctly implements a specific function.
-

  Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements.
  If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

Requirement Verification and Validation

- The requirements should be consistent with all the other requirements i. e. no two requirements should conflict with each other.
- The requirements should be complete in every sense.
- The requirements should be practically achievable.
- Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements Management

- Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders.

- This stage takes care of the changing nature of requirements.

- It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too.

- Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Feasibility Study

- **Feasibility Study** in Software Engineering is a study to evaluate feasibility of proposed project or system.
- Feasibility study is the feasibility analysis, or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view.
- Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.
- **Types of Feasibility Study**

1. Technical Feasibility
2. Operational Feasibility
3. Economic Feasibility
4. Schedule Feasibility
5. Legal

Technical Feasibility

- In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project.
- This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development.
- Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

Operational Feasibility

- In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment.
- Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

Economic Feasibility

- In Economic Feasibility study cost and benefit of the project is analyzed.
- In this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on.
- After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

Schedule Feasibility

- In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.

Legal Feasibility

- In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc.
- Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

Feasibility Study Process

- The below steps are carried out during entire feasibility analysis.

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

Need for Feasibility Study

- Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.
- Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and enhance success rate analyzing different parameters associated with proposed project development.

Information Modelling

- An information model is essential to provide the structure for information that is transferred in a communication.
- An information model is a formal description of a portion of interest of the real world and that provides explicit rules to enable the data items that populate the model to be interpreted without ambiguity.
- A quality information model should be complete, sharable, stable, extensible, well-structured, precise, and unambiguous.
- In general, the contents of an information model include
  - Scope
  - Information requirements

Scope

- The scope specifies the domain of discourse and the processes that are to be supported by the information model.
- It is a bounded collection of processes, information, and constraints that satisfy some industry need.
- The scope statements include the purpose as well as viewpoints of the model mentioned bellow:
  - type of product,
  - type of data requirements,
  - supporting manufacturing scenario,
  - supporting manufacturing activities,
  - supporting stage in the product life cycle.

Information Requirement

- After the scope is defined, the next phase is to conduct a requirements analysis. There is no standard method for collecting information requirements. However, requirements analysis may be accomplished by:

  - Literature surveys,
  - Standards surveys,
  - Domain experts' interviews,
  - Industrial data reviews,
  - State-of-the-art assessments.

Information Modeling

- Information modeling is a technique for specifying the data requirements that are needed within the application domain.

- There are different practices in developing an information model:
  - Data Flow Diagram
  - Data Dictionaries
  - E-R Diagram
  - Decision Tables

Data Flow Diagram

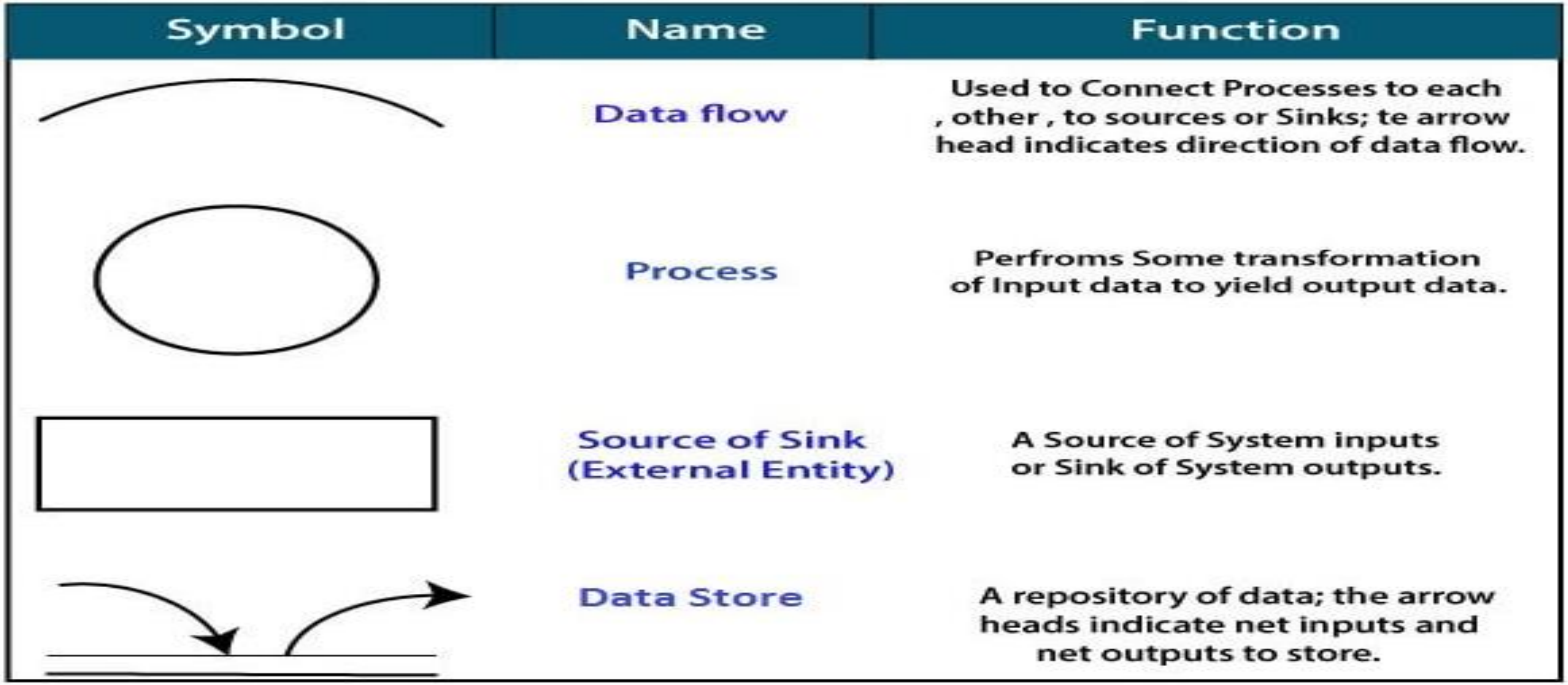
- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

Data Flow Diagram

- **The following observations about DFDs are essential:**

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Symbols used in DFD



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	Perfroms 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

Levels in Data Flow Diagrams (DFD)

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

0-level DFD

- It is also known as context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.
- Then the system is decomposed and described as a DFD with multiple bubbles.
- Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs.
- This process may be repeated at as many levels as necessary until the program at hand is well understood.
- It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

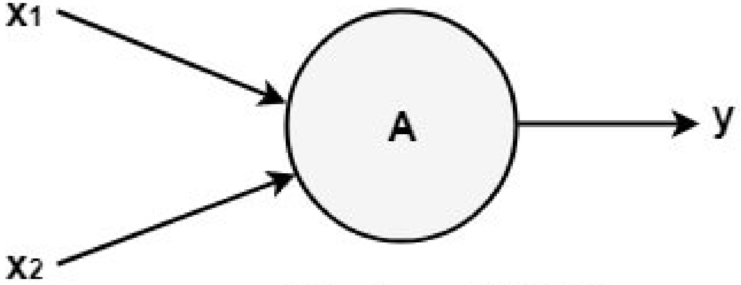


Fig: Level-0 DFD.

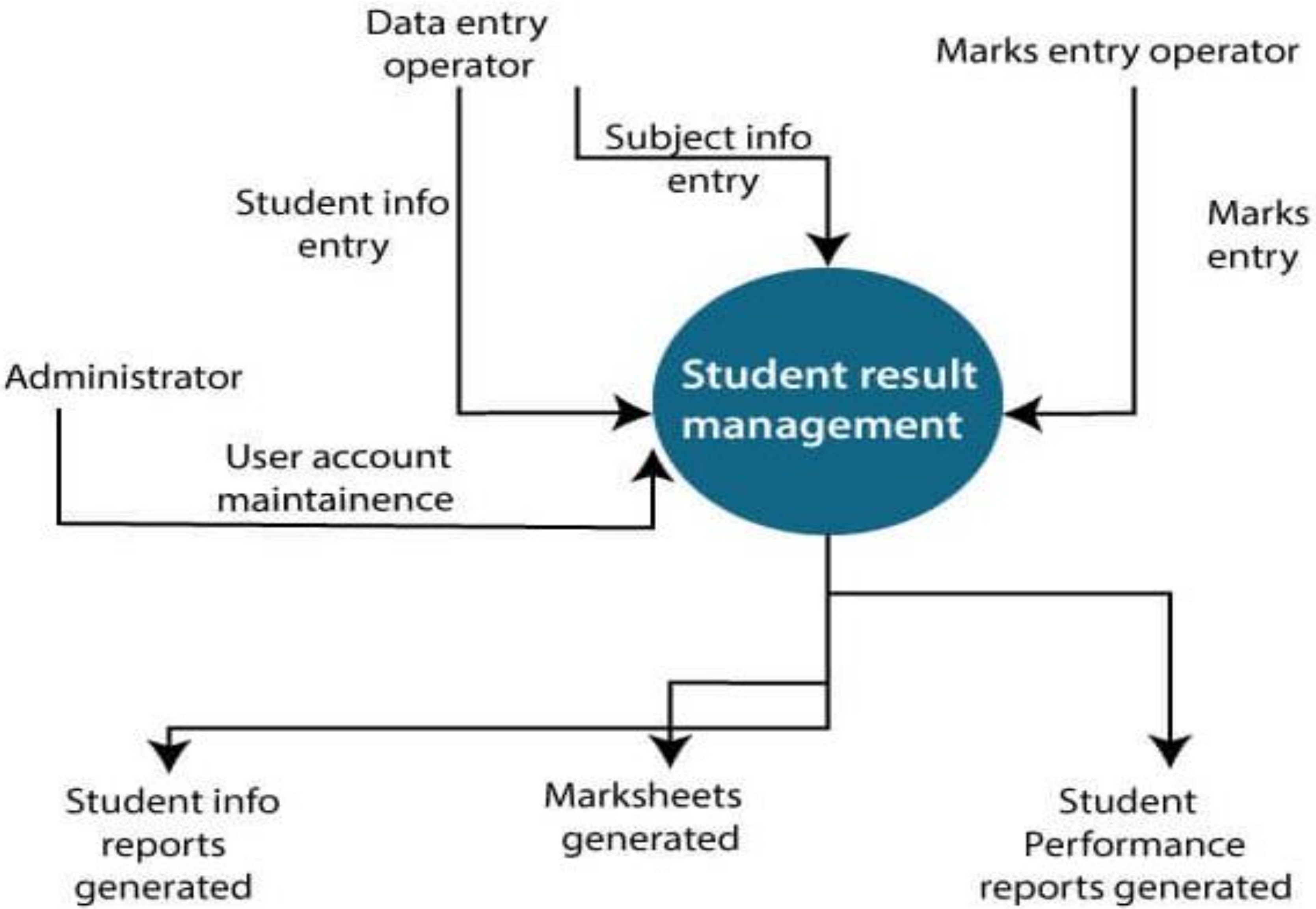
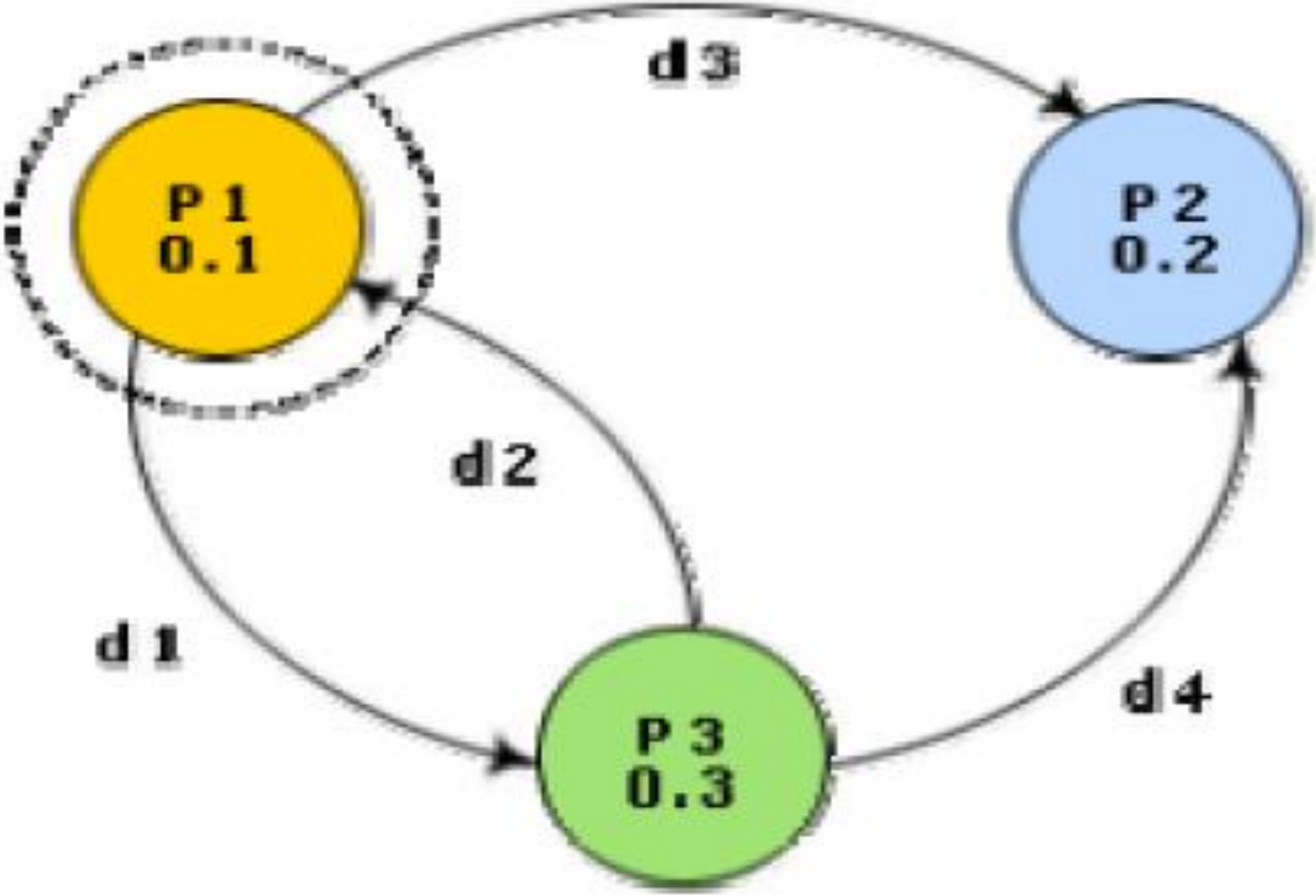
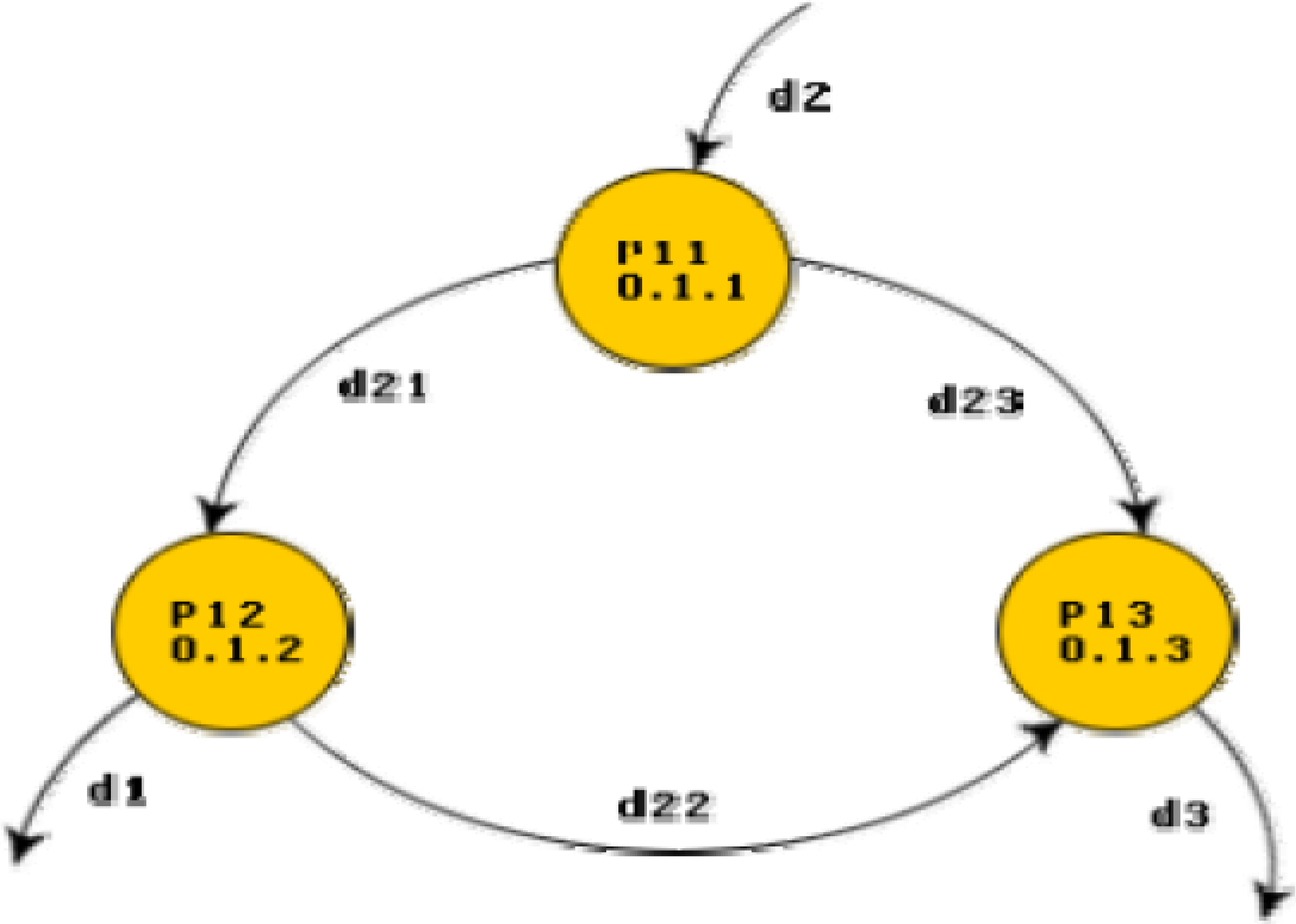


Fig: Level-0 DFD of result management system



d3
P1
0.1
P2
0.2
d2
d1
d4
P3
0.3


d2
P11
0.1.1
d21
d23
P12
0.1.2
P13
0.1.3
d1
d22
d3

1-level DFD

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

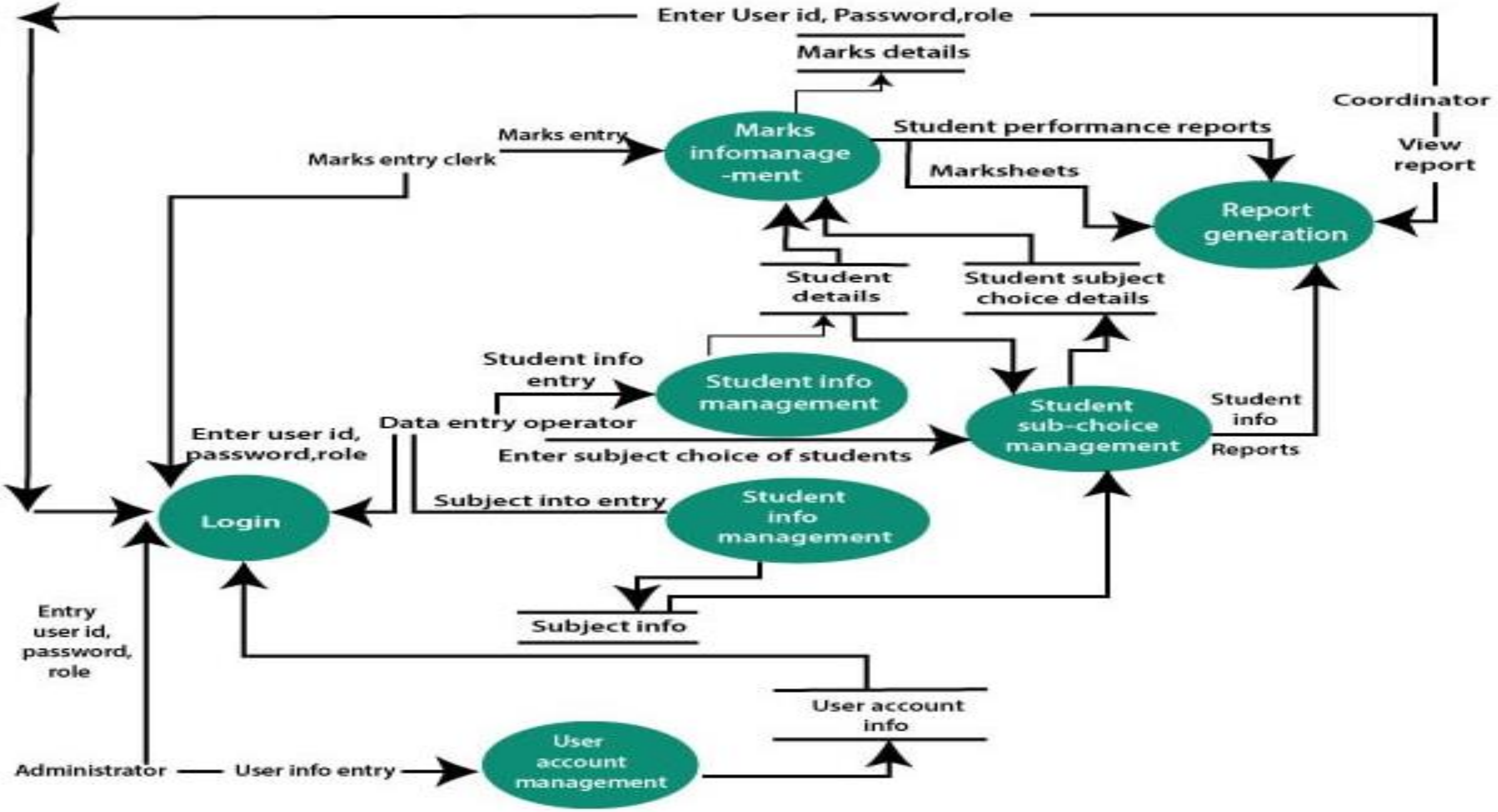
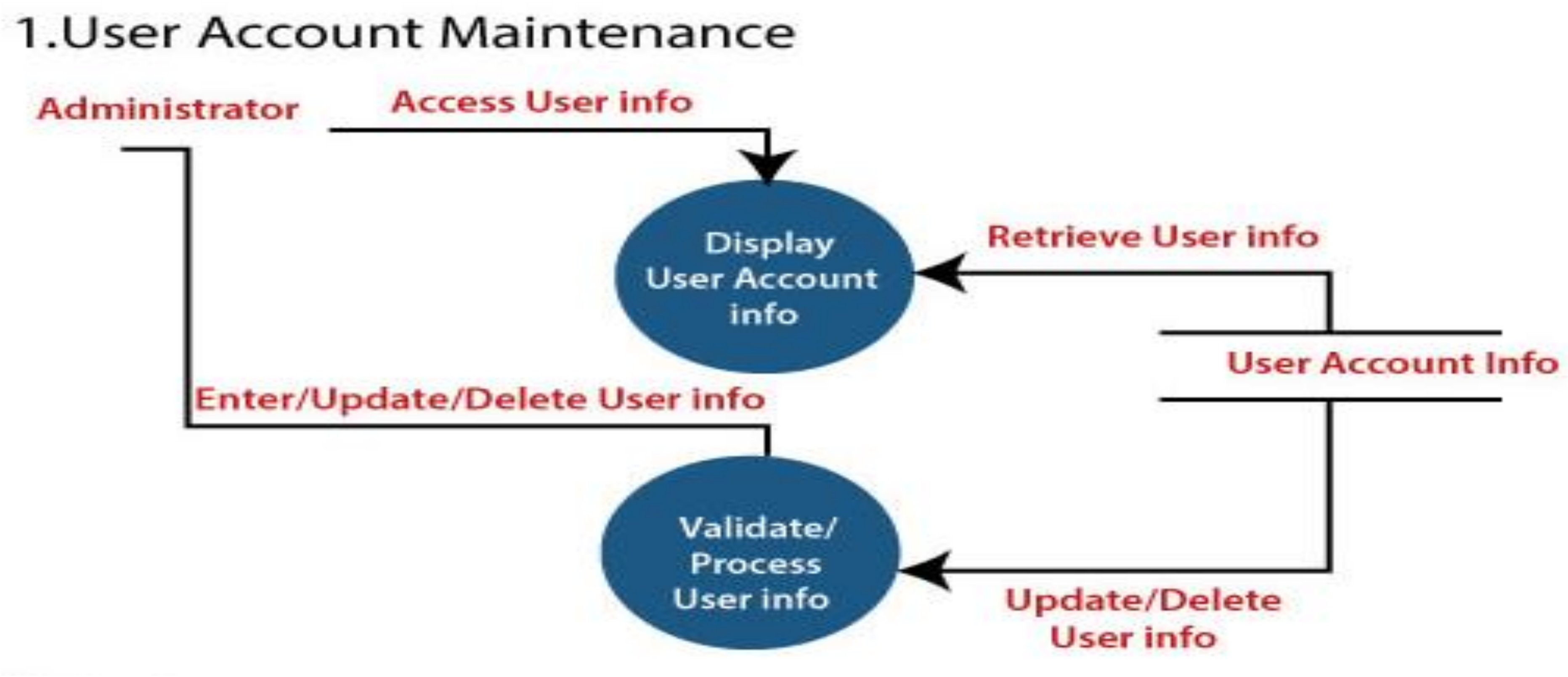


Fig: Level-1 DFD of result management system

2-Level DFD

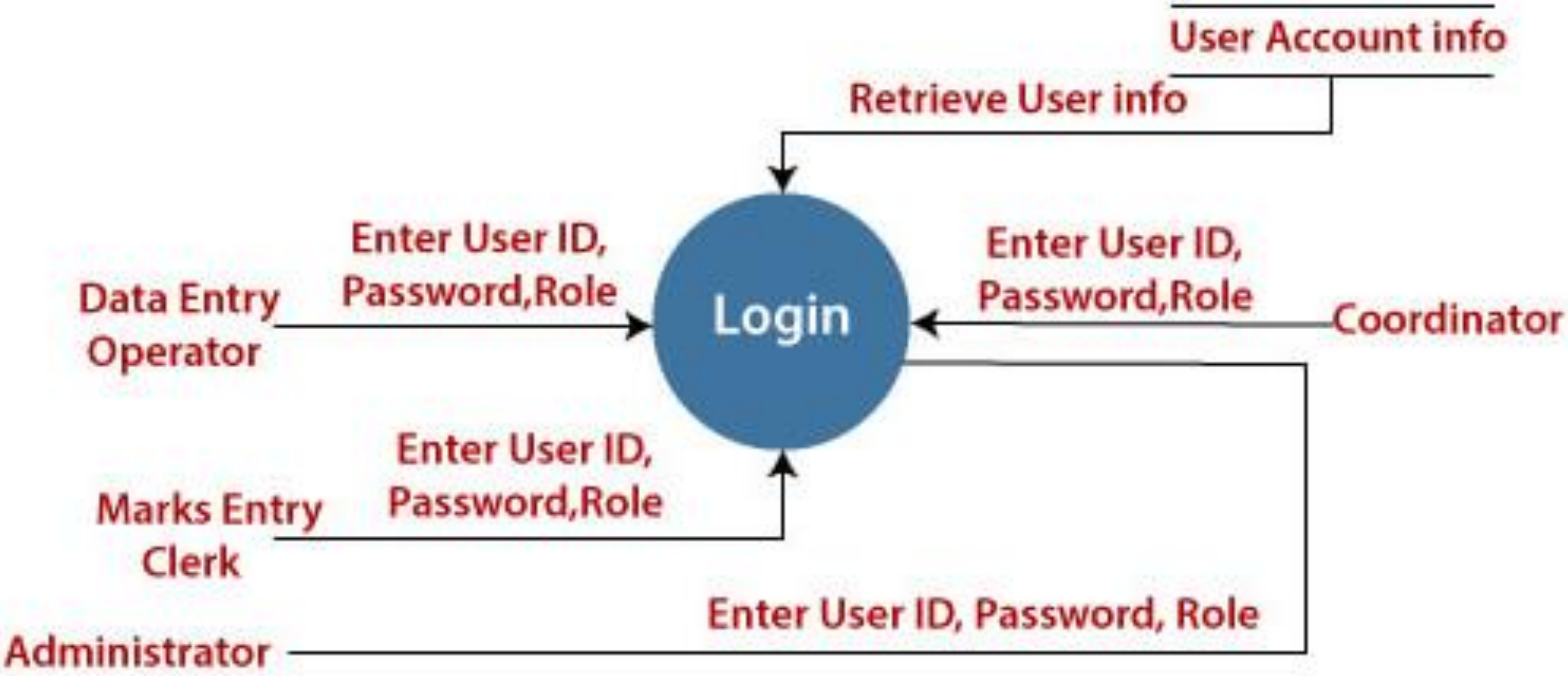
- 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.



1.User Account Maintenance

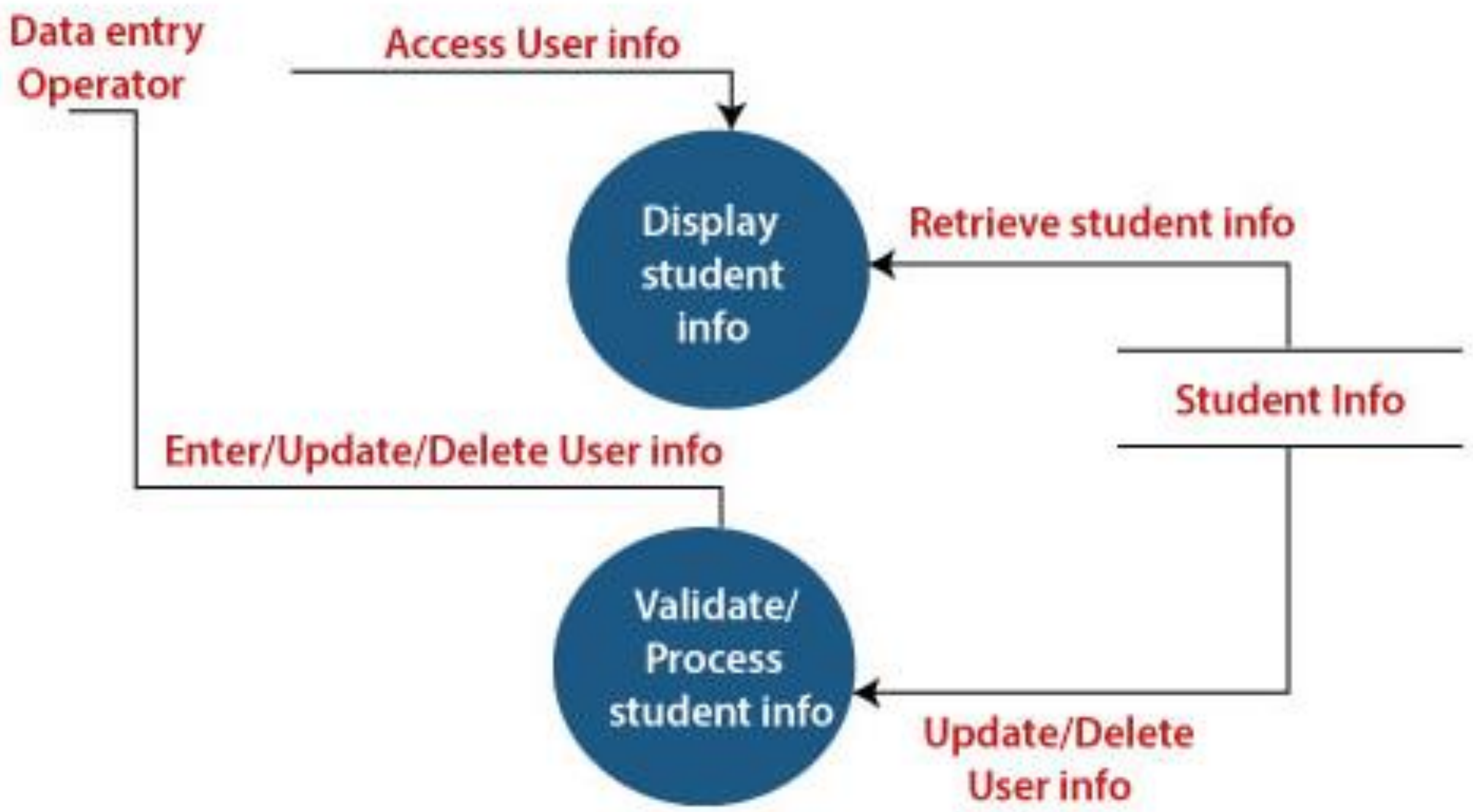
2. Login

The level 2 DFD of this process is given below:



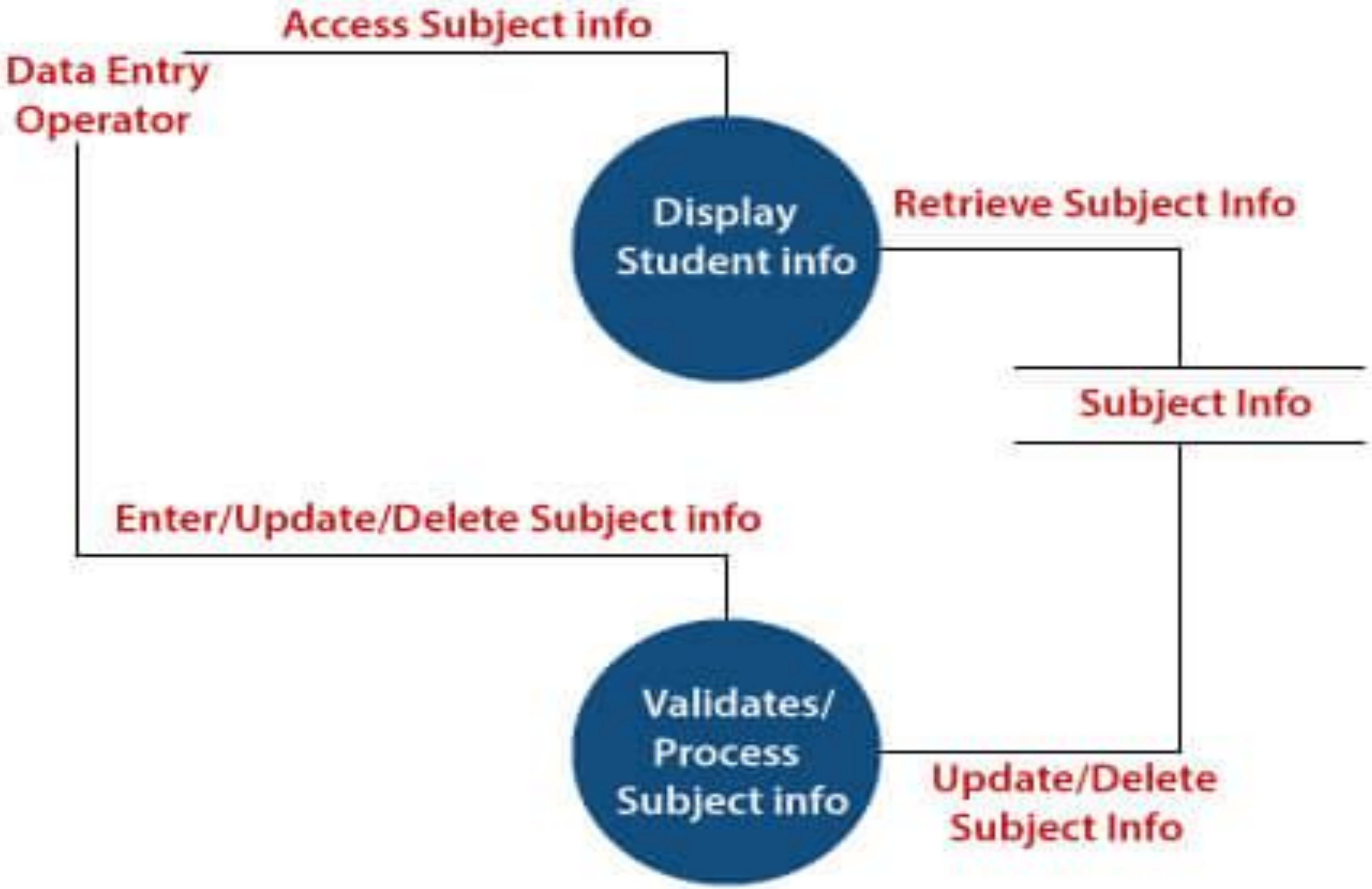
User Account info
Retrieve User info
Enter User ID, Password,Role
Data Entry Operator
Login
Enter User ID, Password,Role
Coordinator
Enter User ID, Password,Role
Marks Entry Clerk
Enter User ID, Password, Role
Administrator

3. Student Information Management



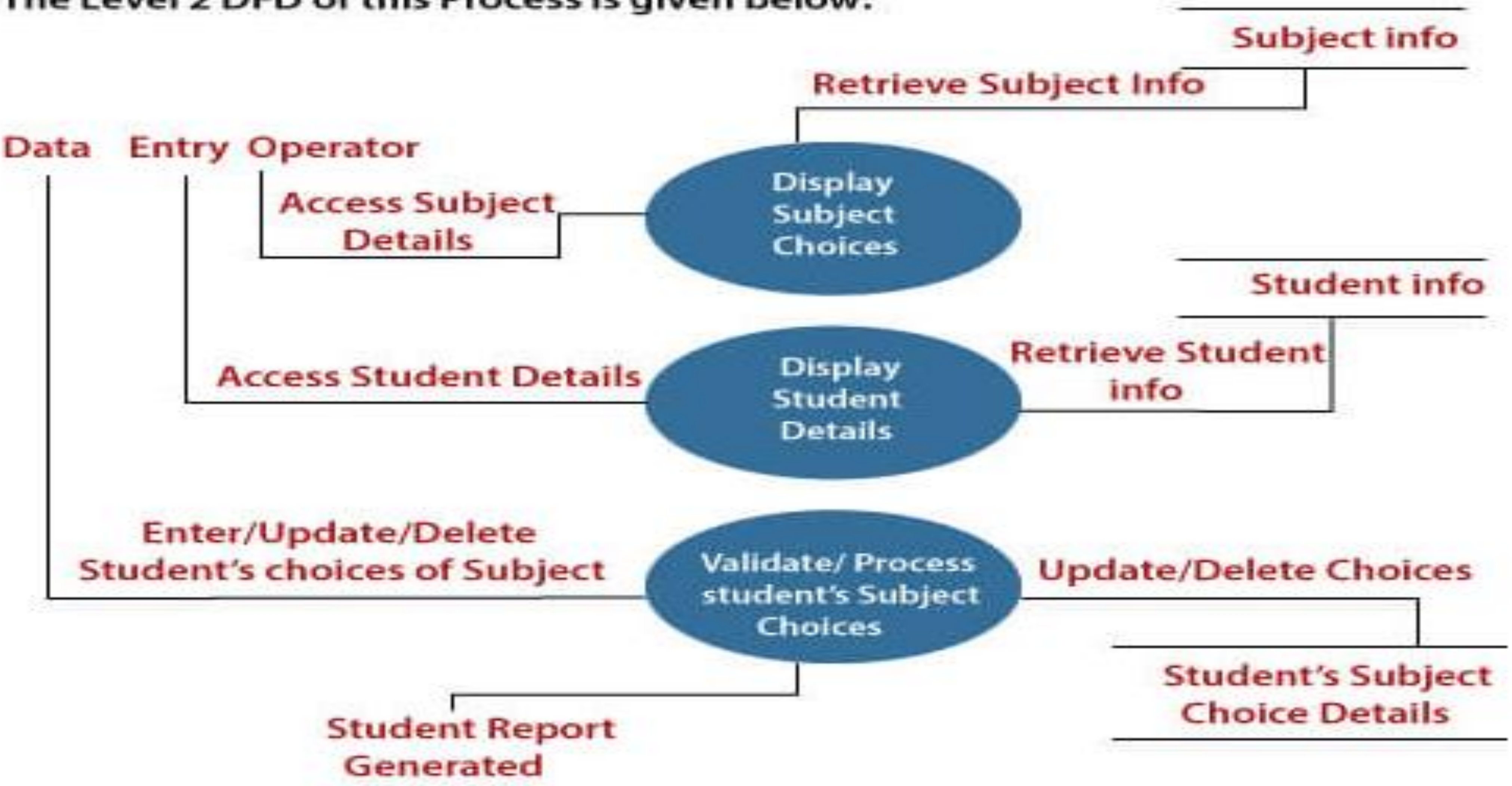
4. Subject Information Management

The level 2 DFD of this process is given below:



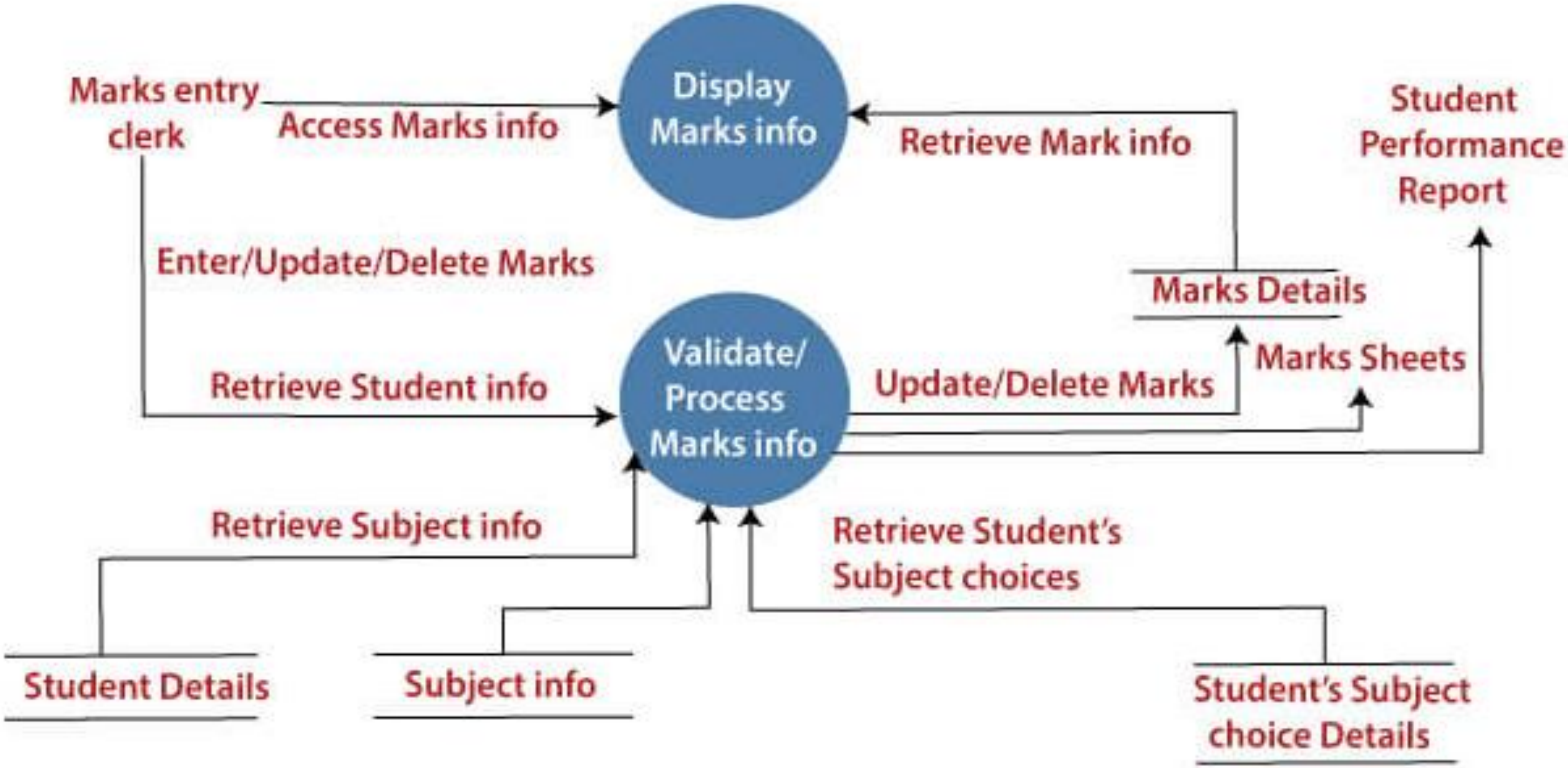
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Managment

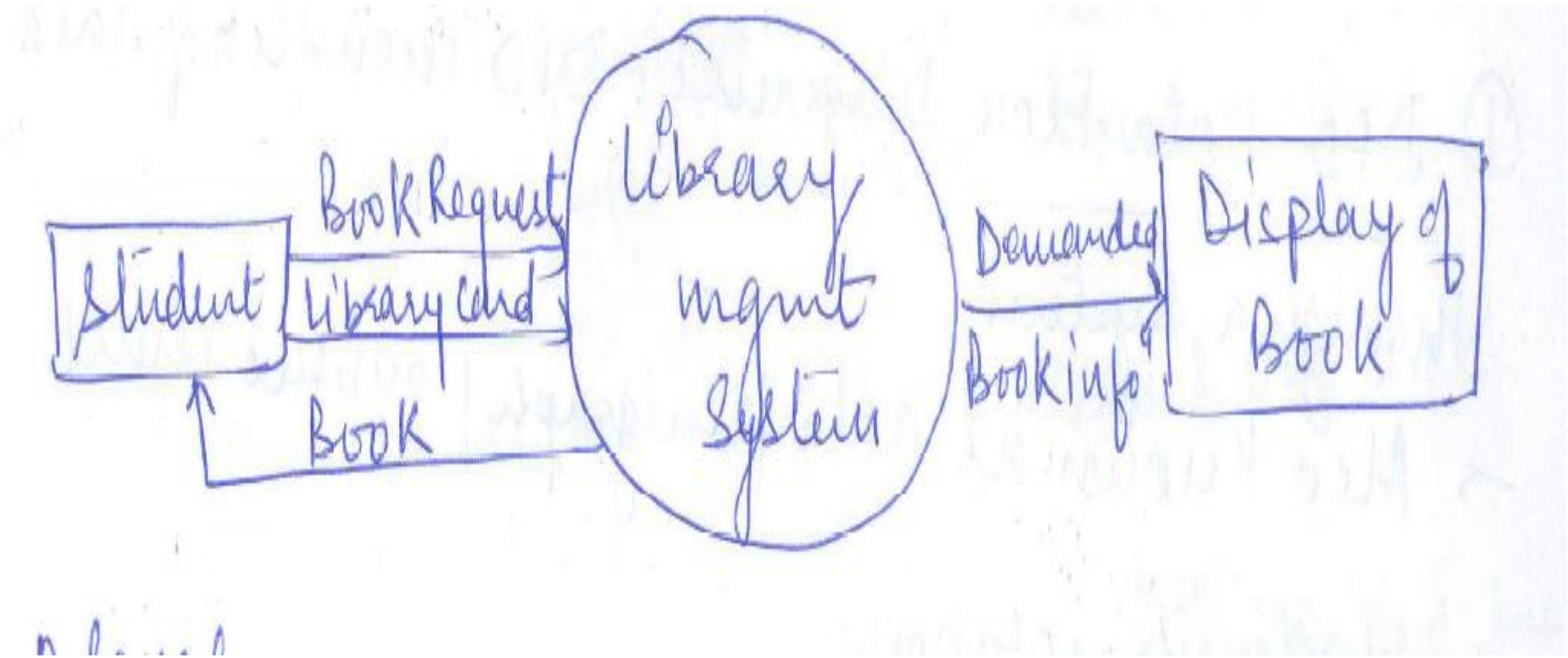
The Level 2 DFD of this Process is given below:

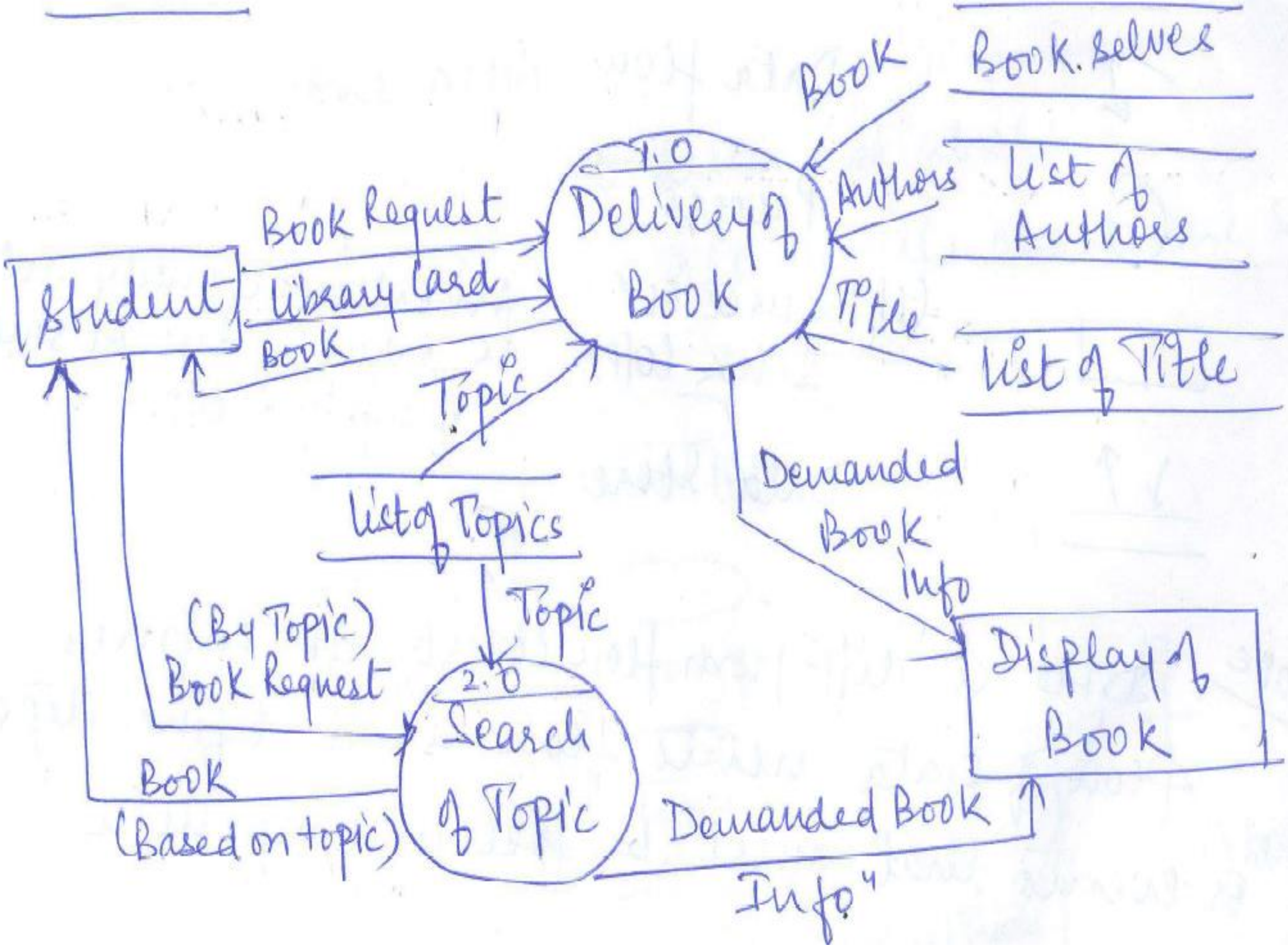


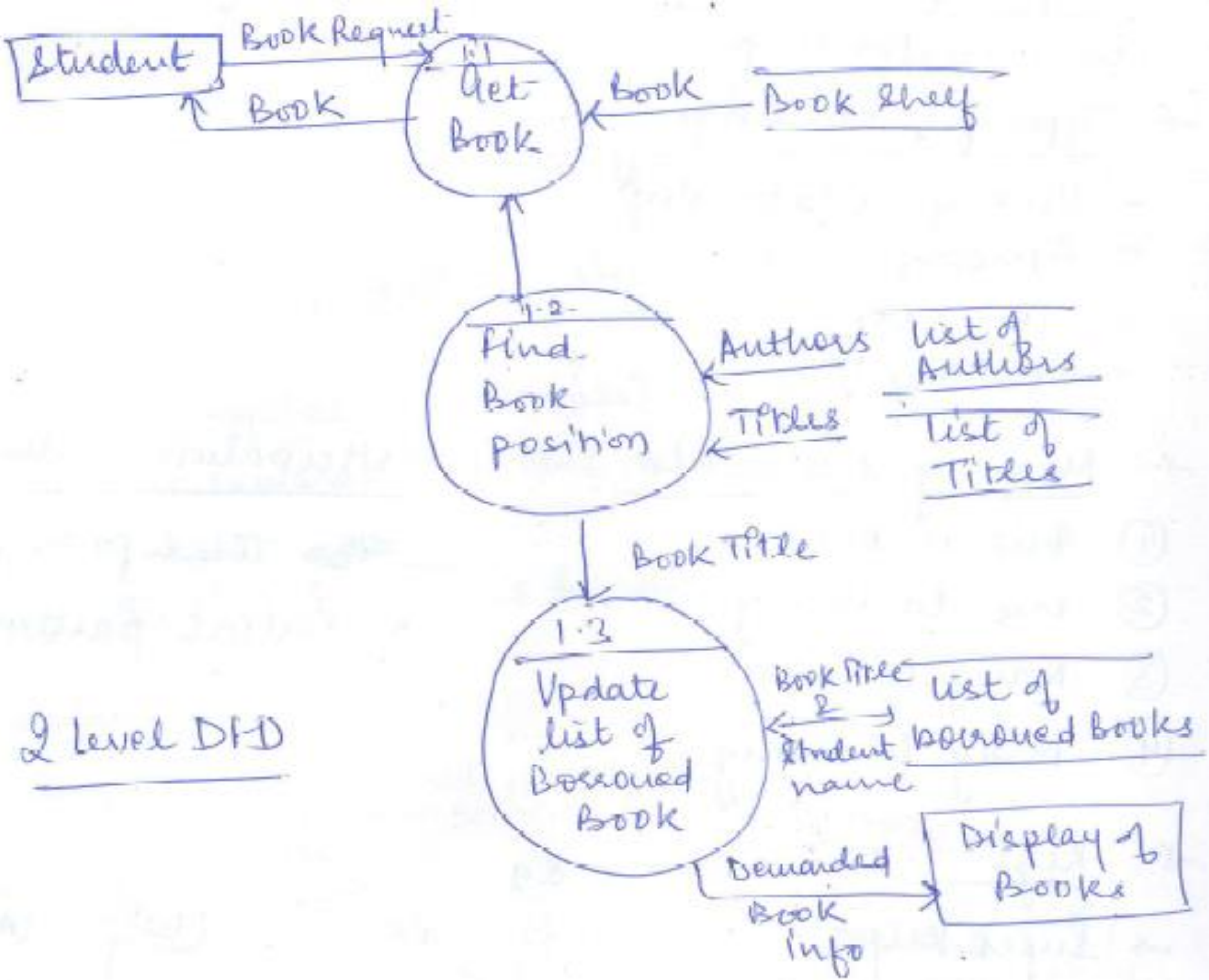
Questions

- Draw overall DFD for Issue/Return of a books in a library

Zero level DFD




Book
Book. selves
1.0
Delivery of Book
Authors
List of Authors
Title
List of Title
Book Request
Library Card
Book
Student
Topic
List of Topics
Demanded Book info
Topic
(By Topic) Book Request
2.0
Search of Topic
Display of Book
Book (Based on topic)
Demanded Book Info"


Student
Book Request
1.1
Get Book
Book
Book
Book Shelf
1.2
Find Book position
Authors
List of Authors
Titles
List of Titles
Book Title
1.3
Update list of Borrowed Book
Book Title & Student name
List of borrowed books
Demanded Book info
Display of Books

2 Level DFD

Decision Table

- **Decision table** is a brief visual representation for specifying which actions to perform depending on given conditions.
- The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.
- Decision tables specify:
  - Which variables are to be tested
  - What actions are to be taken if the conditions are true,
  - The order in which decision making is performed.
- A **Decision table** is a table of rows and columns, separated into four quadrants and is designed to illustrate complex decision rules
  - **Condition Stub** – upper left quadrant
  - **Rules Stub** – upper right quadrant
  - **Action Stub** – bottom left quadrant
  - **Entries Stub** - bottom right quadrant

Decision Table

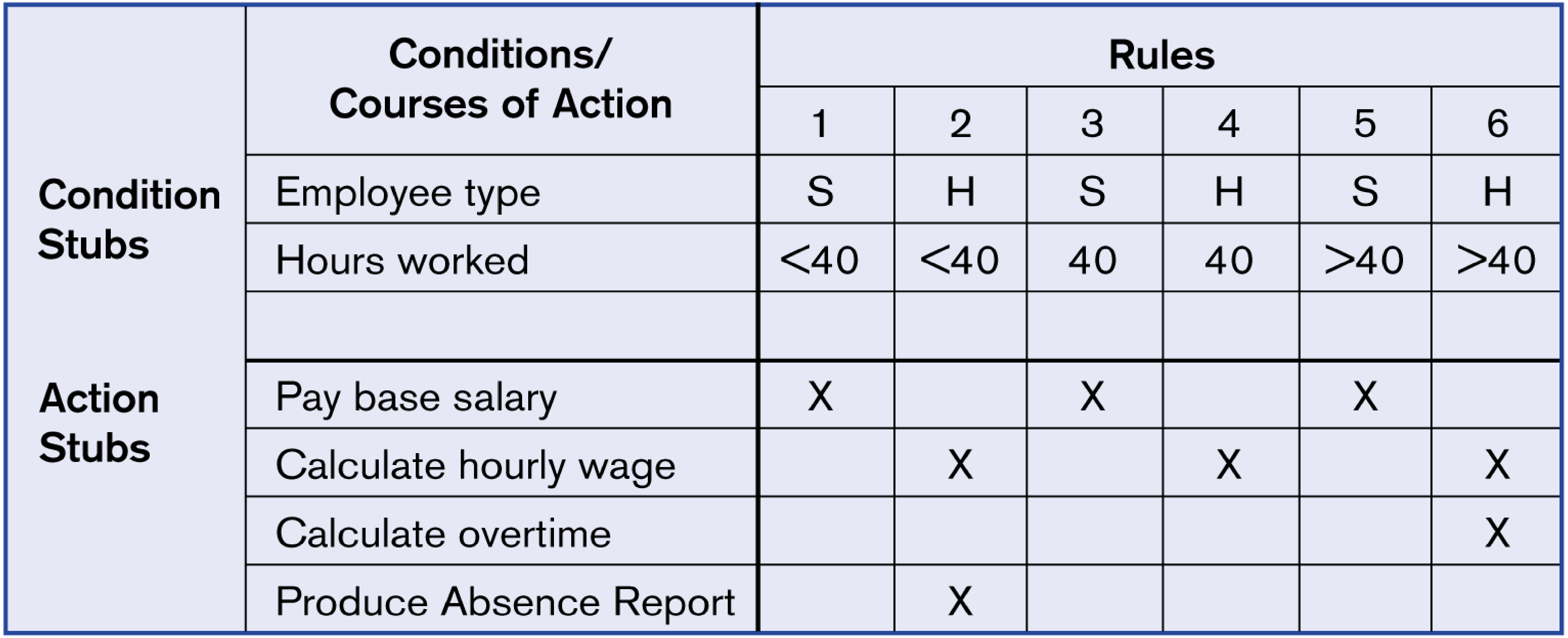
- In technical terminology,
    - A column of the table is called a rule.
    - A rule implies:
    - If a condition is true, then execute the corresponding action.

Decision Table Layout

- Standard format used for presenting decision tables.

Condition Stub	***Rules Stub***
Action Stub	***Entries Stub***

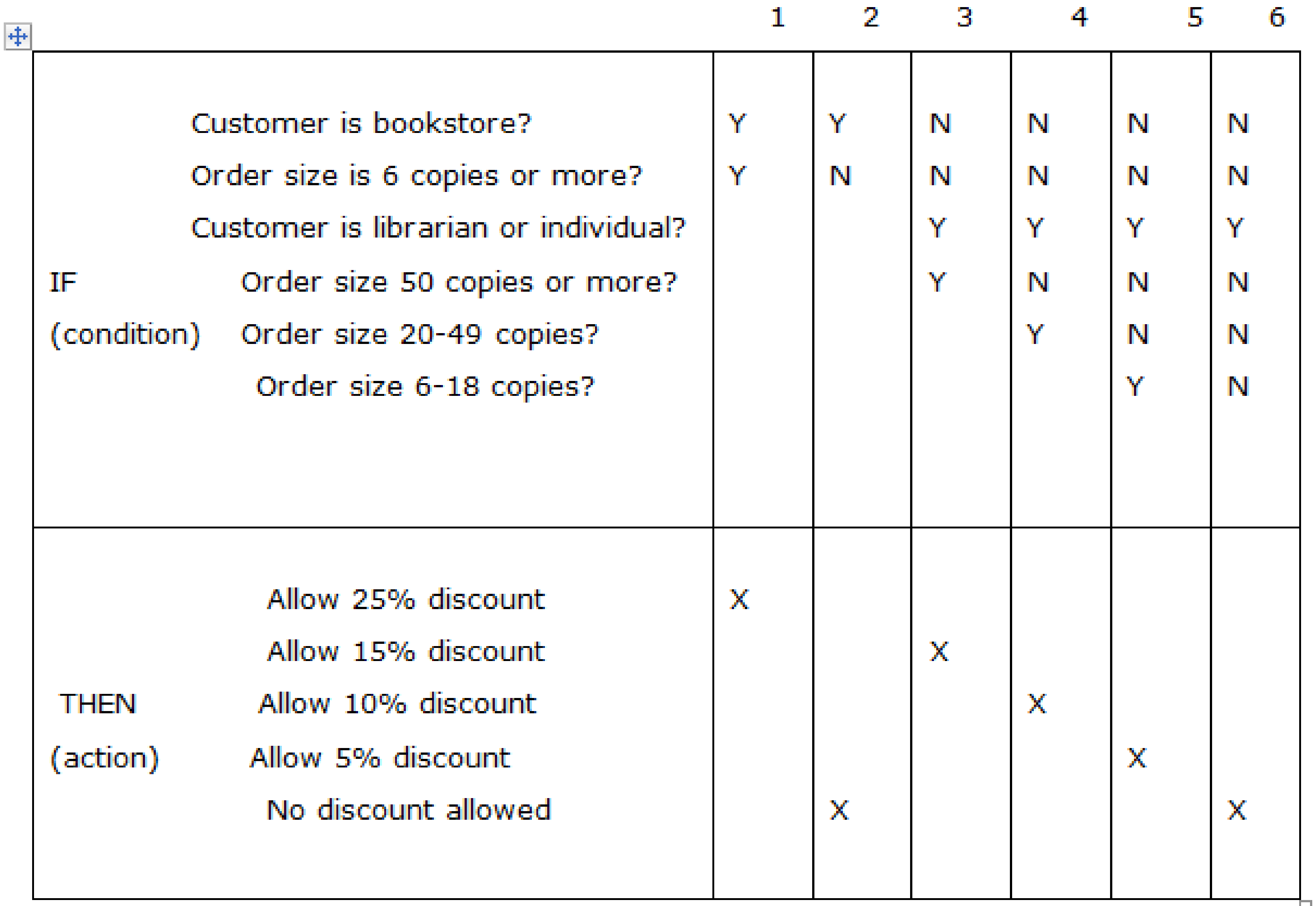
Payroll System example



	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Decision Table

- For example: A bookstore get a trade discount of 25% for order more then 6 books; for order from libraries and individuals, 5% allowed on orders of 6-19 copies per book title; 10% on orders for 20-49 copies per book title; 15% on orders for 50 copies or more per book title.



		1	2	3	4	5	6
	Customer is bookstore?	Y	Y	N	N	N	N
	Order size is 6 copies or more?	Y	N	N	N	N	N
	Customer is librarian or individual?			Y	Y	Y	Y
IF	Order size 50 copies or more?			Y	N	N	N
(condition)	Order size 20-49 copies?				Y	N	N
	Order size 6-18 copies?					Y	N
	Allow 25% discount	X					
	Allow 15% discount			X			
THEN	Allow 10% discount				X		
(action)	Allow 5% discount					X	
	No discount allowed		X				X

Action Stub **Action Entry**

Developing Decision Tables

- Process requires the determination of the number of conditions (inputs) that affect the decision.
- The set of possible actions (outputs) must likewise be determined
- The number of rules is computed
- Each rule must specify one or more actions

Number of Rules

- Each condition generally has two possible alternatives (outcomes): ***Yes*** or ***No***
  - In more advanced tables, multiple outcomes for each condition are permitted
- The total number of rules is equal to
  - no. of condition1 values * no. of condition2 values
  - If no of values for each condition is T/F, then $\mathbf{2}^{\textbf{no. of conditions}}$
- Thus, if there are four conditions, there will be sixteen possible rules

Building the Table

- For each rule, select the appropriate action and indicate with an ‘X’
- Identify rules that produce the same actions and attempt to combine those rules; for example:

Condition 1	*Y*	*Y*	*Condition 1*	*Y*
Condition 2	*Y*	*N*	*Condition 2*	*-*
Action 1	*X*	*X*	*Action 1*	*X*

Cleaning Things Up

- Check the table for any impossible situations, contradictions, and redundancies and eliminate such rules

- Rewrite the decision table with the most reduced set of rules; rearranging the rule order is permissible if it improves user understanding

Decision Table example: combine and reduce

Conditions and Actions	1	2	3	4	5	6	7	8
Order from Fall Catalog	Y	Y	Y	Y	N	N	N	N
Order from Christmas Catalog	Y	Y	N	N	Y	Y	N	N
Order from Special Catalog	Y	N	Y	N	Y	N	Y	N
Mail Christmas Catalog		X		X		X		X
Mail Special Catalog			X				X	
Mail Both Catalogs	X				X			

The four gray columns can be combined into a single rule. Note that four each, there was NO order placed from the Special Catalog.

In addition, Rules 1&5 and Rules 3&7 can be combined. Each pair produces the same action and each pair shares two common conditions.

1	5	1&5	2	4	2&4	3	7	3&7	6	8	6&8
Y	N		Y	Y		Y	N		N	N	
Y	Y		Y	N		N	N		Y	N	
Y	Y		N	N		Y	Y		N	N	

1	5	1&5	2	4	2&4	3	7	3&7	6	8	6&8
Y	N	-	Y	Y	Y	Y	N	-	N	N	N
Y	Y	Y	Y	N	-	N	N	N	Y	N	-
Y	Y	Y	N	N	N	Y	Y	Y	N	N	N

2&4	6&8	2,4,6 & 8
Y	N	-
-	-	-
N	N	N
X	X	X

Conditions and Actions	1& 5	3 & 7	2,4	6,8
Order from Fall Catalog	-	-	Y	N
Order from Christmas Catalog	Y	N	-	-
Order from Special Catalog	Y	Y	N	N
Mail Christmas Catalog			X	X
Mail Special Catalog		X		
Mail Both Catalogs	X			

Conditions and Actions	1& 5	3 & 7	2,4,6,8
Order from Fall Catalog	-	-	-
Order from Christmas Catalog	Y	-	-
Order from Special Catalog	Y	Y	N
Mail Christmas Catalog			X
Mail Special Catalog		X	
Mail Both Catalogs	X		

Decision Table example ~ Final Version

Conditions and Actions	1	2	3
Order from Fall Catalog	--	--	--
Order from Christmas Catalog	Y	--	N
Order from Special Catalog	Y	N	Y
Mail Christmas Catalog		X	
Mail Special Catalog			X
Mail Both Catalogs	X		

Eliminates the need to check for every possible case.

Constructing a Decision Table

- PART 1. FRAME THE PROBLEM.
  - Identify the conditions (decision criteria). These are the factors that will influence the decision.
    - E.g., We want to know the total cost of a student’s tuition. What factors are important?
  - Identify the range of values for each condition or criteria.
    - E.g. What are they for each factor identified above?
  - Identify all possible actions that can occur.
    - E.g. What types of calculations would be necessary?
- PART 2. CREATE THE TABLE.
  - Create a table with 4 quadrants.
    - Put the conditions in the upper left quadrant. One row per condition.
    - Put the actions in the lower left quadrant. One row per action.
  - List all possible rules.
    - Alternate values for first condition. Repeat for all values of second condition. Keep repeating this process for all conditions.
    - Put the rules in the upper right quadrant.
  - Enter actions for each rule
    - In the lower right quadrant, determine what, if any, appropriate actions should be taken for each rule.
  - Reduce table as necessary.

Example

- If you are a new customer and you want to open a credit card account then there are three conditions first you will get a 15% discount on all your purchases today, second if you are an existing customer and you hold a loyalty card, you get a 10% discount and third if you have a coupon, you can get 20% off today (but it can’t be used with the ‘new customer’ discount). Discount amounts are added, if applicable.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
New Customer (15%)	Y	Y	N	N	N	N
Loyality Card (10%)	N	N	Y	Y	N	N
Coupon (20%)	Y	N	Y	N	Y	N
ACTIONS						
NO DISCOUNT						X
20%	X					
15%		X				
30%			X			
10%				X		
20%					X	

Consider the payroll system of a person

(a) If the salary of a person is less than equal to Rs. 70,000 and expenses do not exceed Rs. 30,000 then 10% tax is charged by IT department.

(b) If the salary is greater than Rs.60,000 and less than equal to Rs 2lakhs and expenses don’t exceed Rs. 40,000 than 20% tax is charged by IT department.

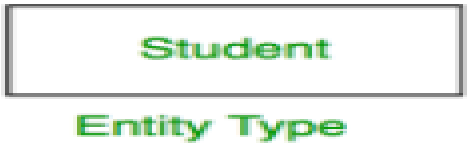
(c) For salary greater than Rs 2 lakhs, 5% additional surcharge is also charged. (d) If expenses are greater than Rs. 40,000 surcharge is 9%.

Importance of Decision Tables

1. Decision tables are very much helpful in test design technique.
2. It helps testers to search the effects of combinations of different inputs and other software states that must correctly implement business rules.
3. It provides a regular way of stating complex business rules, that is helpful for developers as well as for testers.
4. It assists in development process with developer to do a better job. Testing with all combination might be impractical.
5. A decision table is basically an outstanding technique used in both testing and requirements management.
6. It is a structured exercise to prepare requirements when dealing with complex business rules.
7. It is also used in model complicated logic.

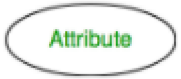
E-R Model

- ER Model is used to model the logical view of the system from data perspective which consists of these components:
  - Entity
  - Entity Type
  - Entity Set
- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.
- An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:

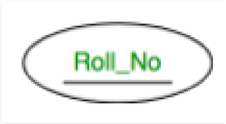


E-R Model

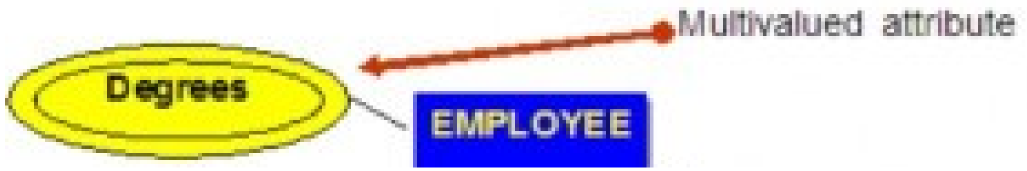
- Attributes are the **properties which define the entity type**. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.



- An attribute which **uniquely identifies each entity** in the entity set is called key attribute. For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

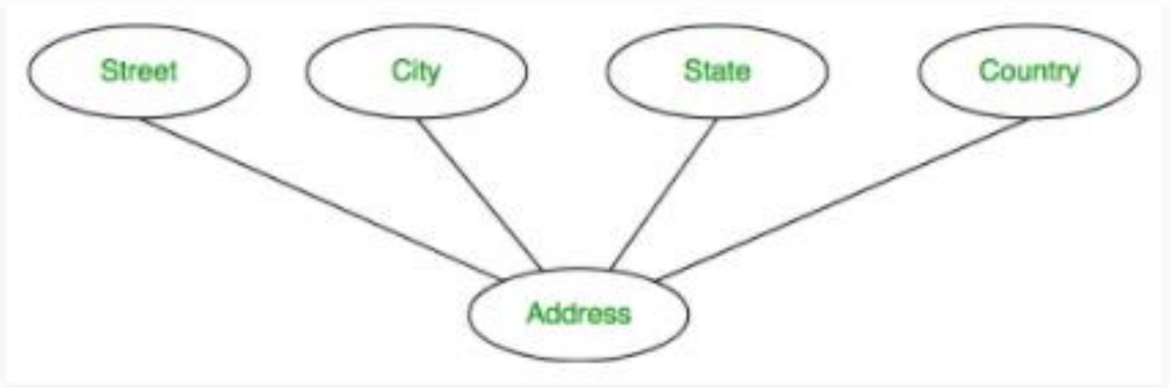


- The multi-valued attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.



E-R Model

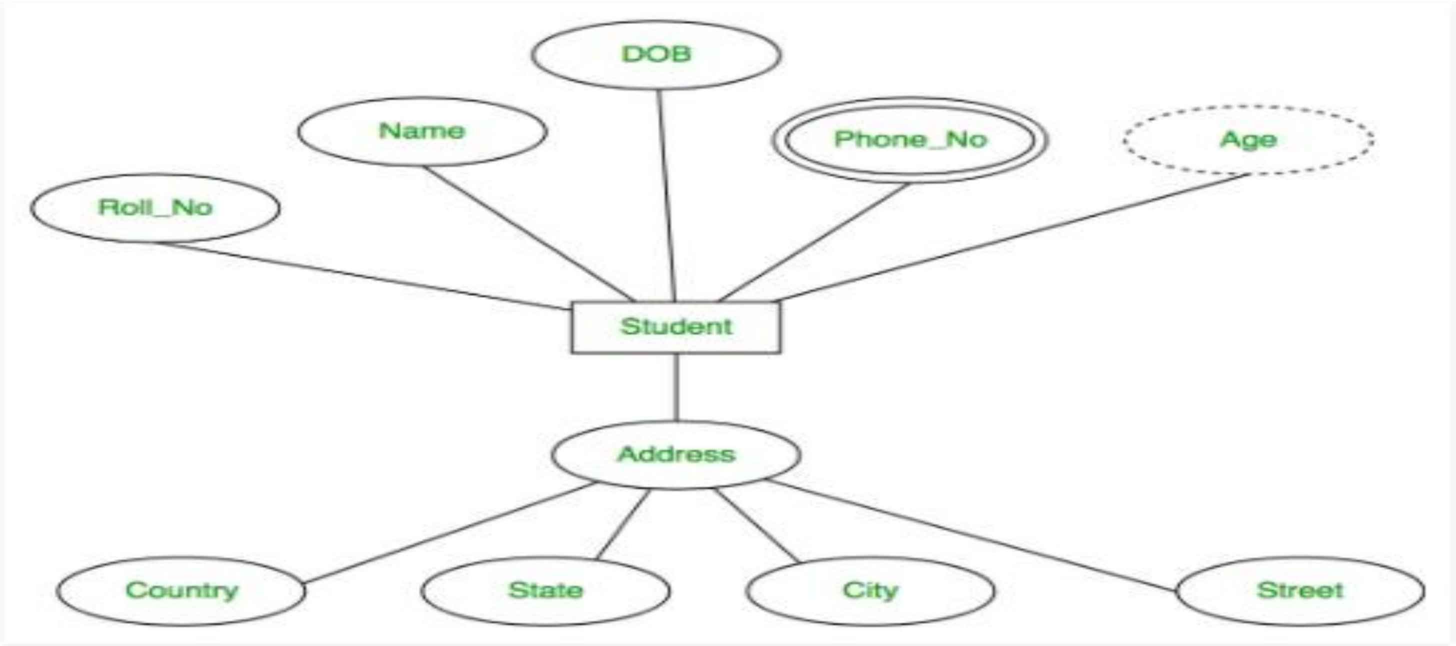
- An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



E-R Model

- An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

- The complete entity type **Student** with its attributes can be represented as:

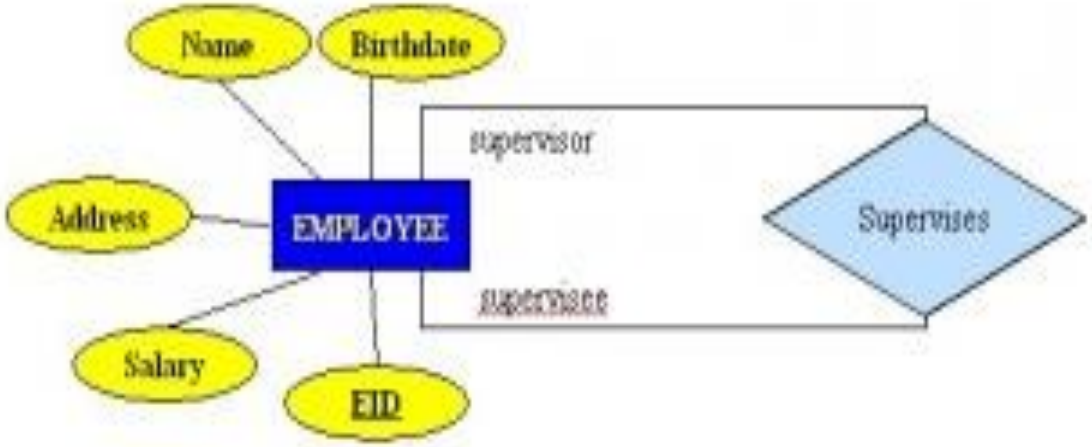


Relationships in E-R Model

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

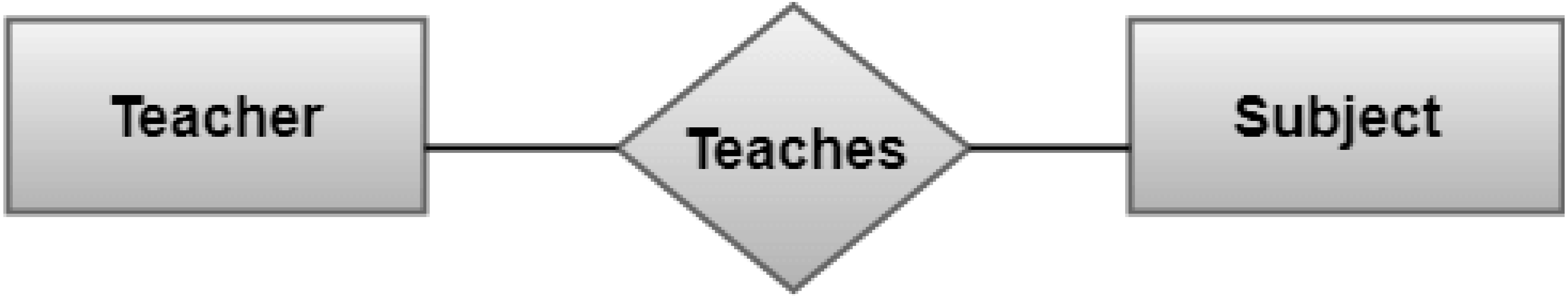
1. Unary (degree1)
2. Binary (degree2)
3. Ternary (degree3)

1. Unary relationship: A *unary relationship,* also called *recursive,* is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles.



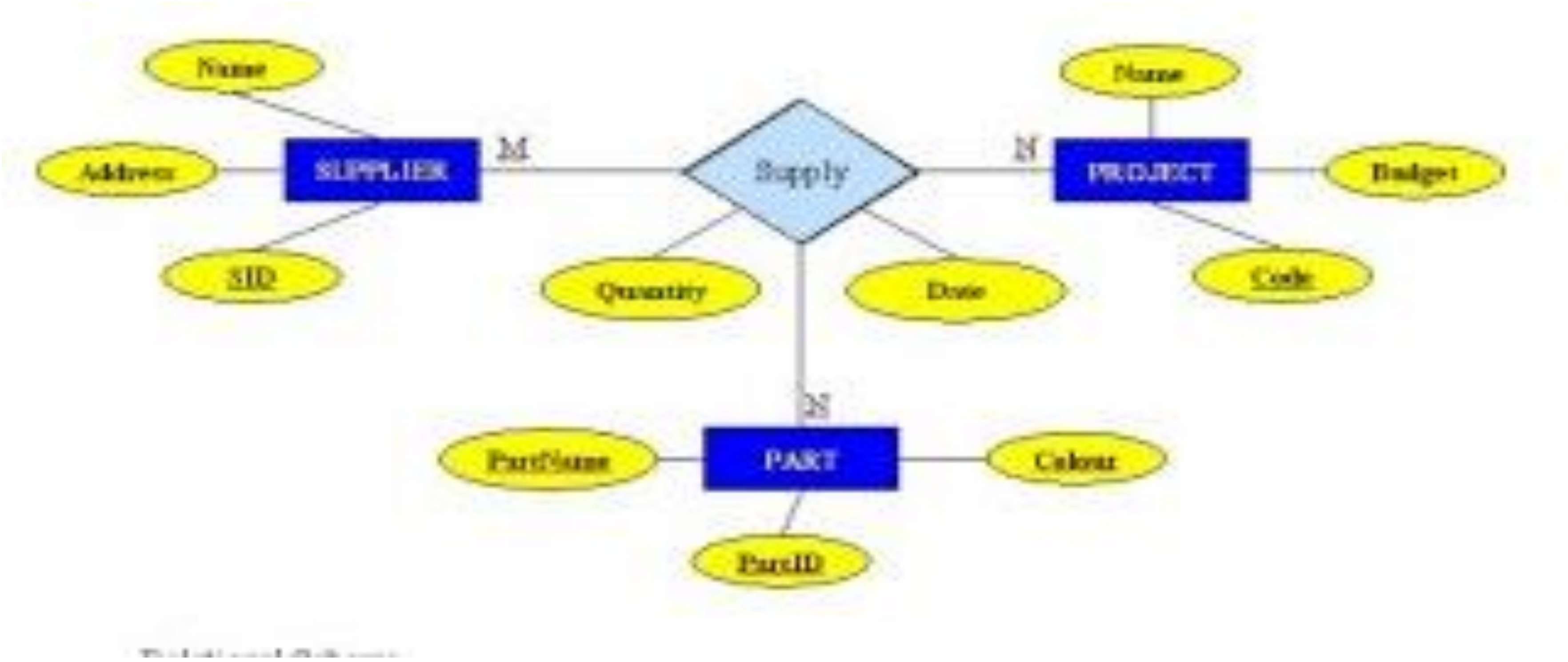
Relationships in E-R Model

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



Relationships in E-R Model

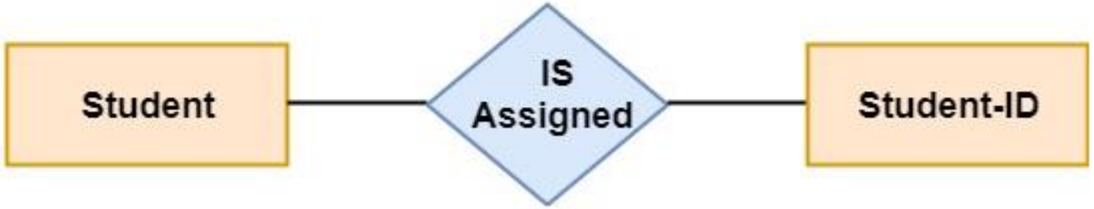
- Ternary Relationships
  - A *ternary relationship* is a relationship type that involves many to many relationships between three tables.



Relationships in E-R Model

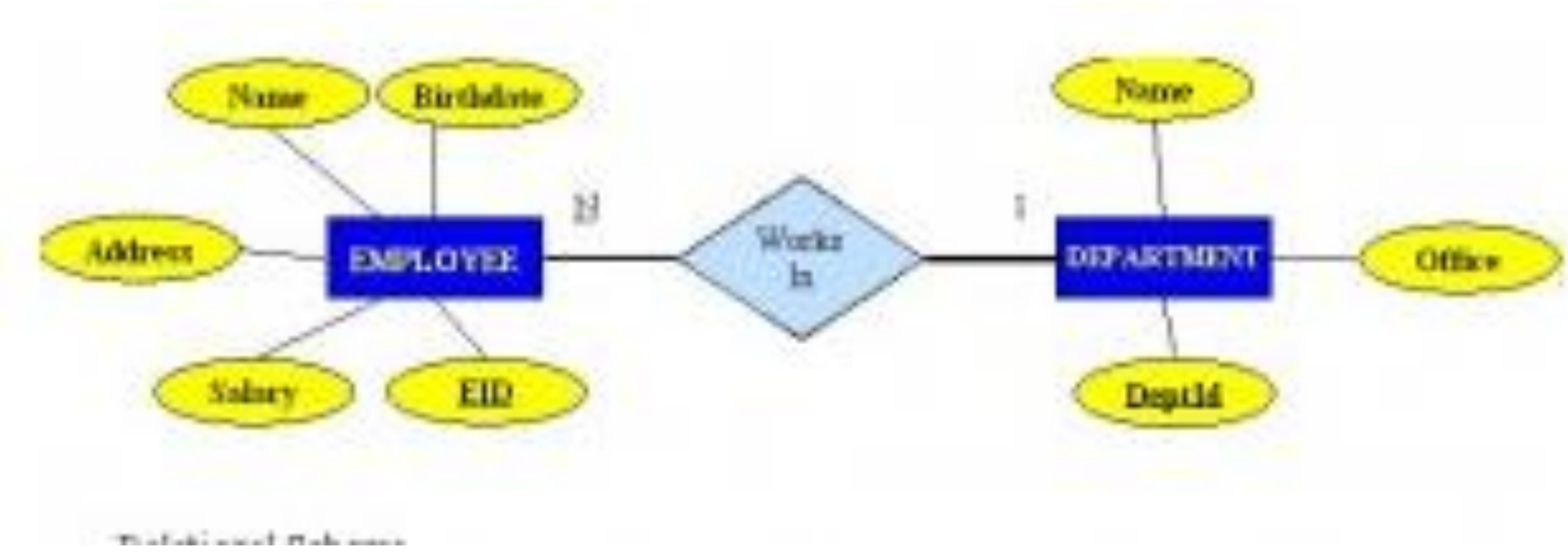
- **Cardinality**
- Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.
- **Types of Cardinalities**

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



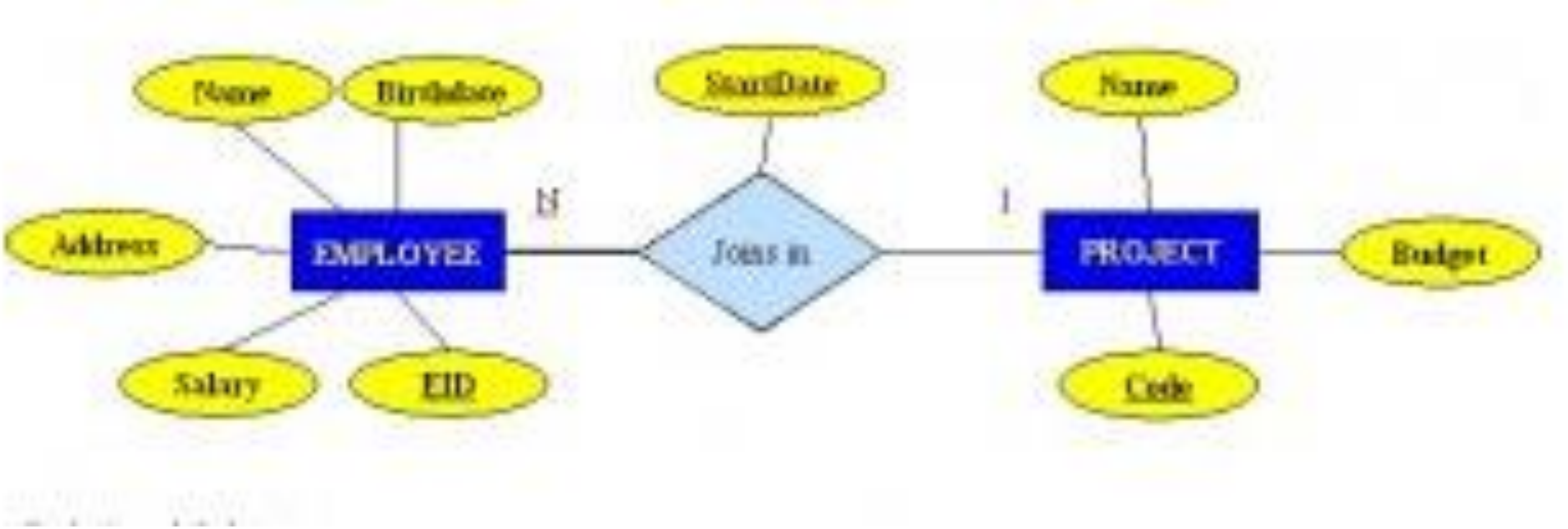
Relationships in E-R Model

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

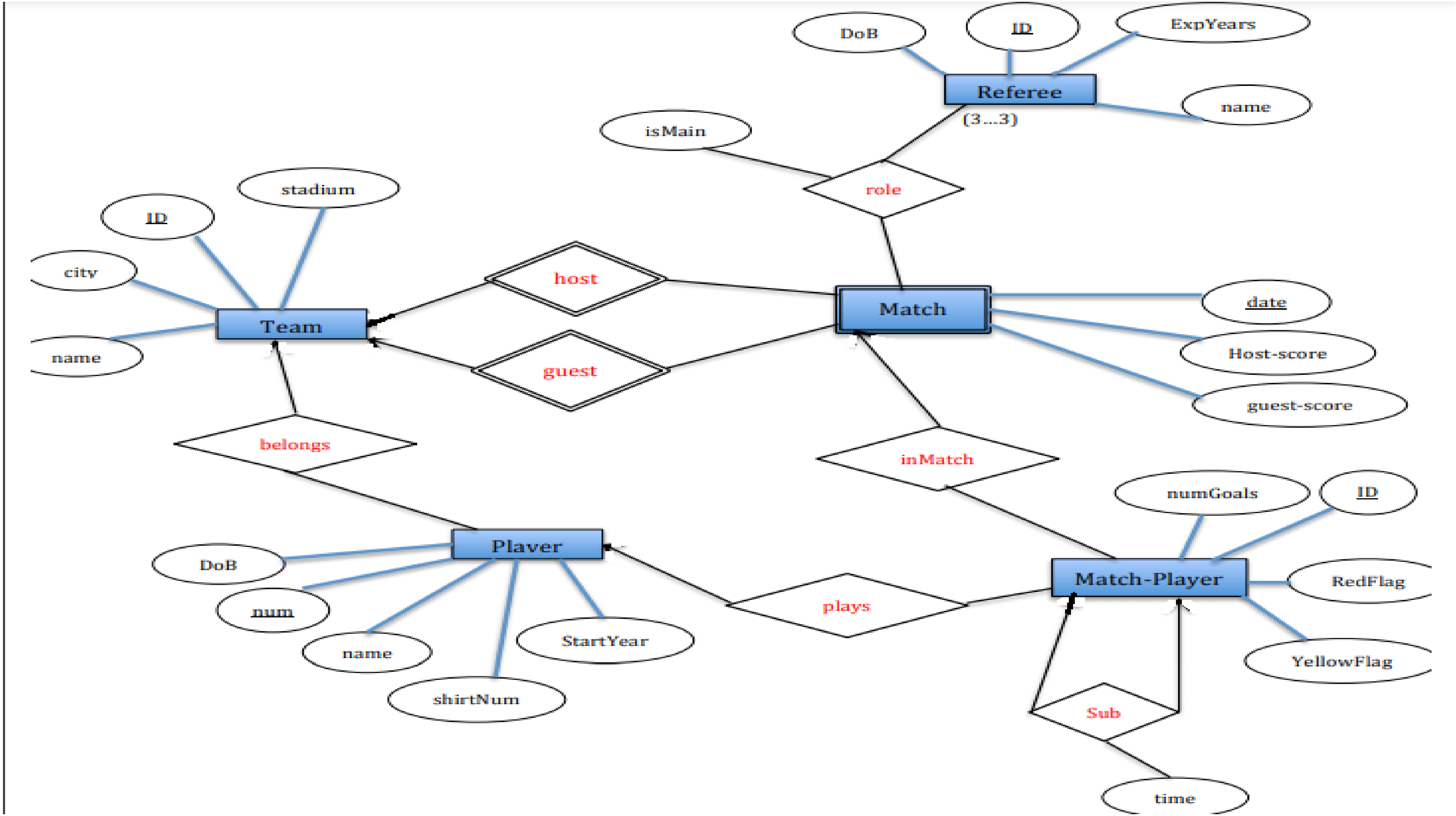


Relationships in E-R Model

ii. Many to Many : One entity from A can be associated with more than one entity from B and vice-versa.



- Assume we have the following application that models soccer teams, the games they play, and the players in each team. In the design, we want to capture the following:
- We have a set of teams, each team has an ID (unique identifier), name, main stadium, and to which city this team belongs.
- Each team has many players, and each player belongs to one team. Each player has a number (unique identifier), name, DoB, start year, and shirt number that he uses.
- Teams play matches, in each match there is a host team and a guest team. The match takes place in the stadium of the host team.
- For each match we need to keep track of the following: o The date on which the game is played o The final result of the match o The players participated in the match. For each player, how many goals he scored, whether or not he took yellow card, and whether or not he took red card. o During the match, one player may substitute another player. We want to capture this substitution and the time at which it took place.
- Each match has exactly three referees. For each referee we have an ID (unique identifier), name, DoB, years of experience. One referee is the main referee and the other two are assistant referee.
- Design an ER diagram to capture the above requirements. State any assumptions you have that affects your design (use the back of the page if needed). Make sure cardinalities and primary keys are clear.


DoB
ID
ExpYears
Referee
name
(3…3)
isMain
role
stadium
ID
city
host
Match
date
Team
name
Host-score
guest
guest-score
belongs
inMatch
numGoals
ID
Plaver
Match-Player
RedFlag
DoB
plays
num
YellowFlag
name
StartYear
shirtNum
Sub
time

Software Requirement Specification

- A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements.
- The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system.
- The software requirement specification document consists of all necessary requirements required for project development.
- To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.

Characteristics of Good SRS

1. **Correctness:**

User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

2. Completeness:

Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

3. Consistency:

Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

Characteristics of Good SRS

4. **Unambiguousness:**

An SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

5. **Ranking for importance and stability:**

There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6. Modifiability:

SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

Characteristics of Good SRS

7. **Verifiability:**

An SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

8. **Traceability:**

One should be able to trace a requirement to a design component and then to a code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

9. Design Independence:

There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

Characteristics of Good SRS

10. Testability:

An SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

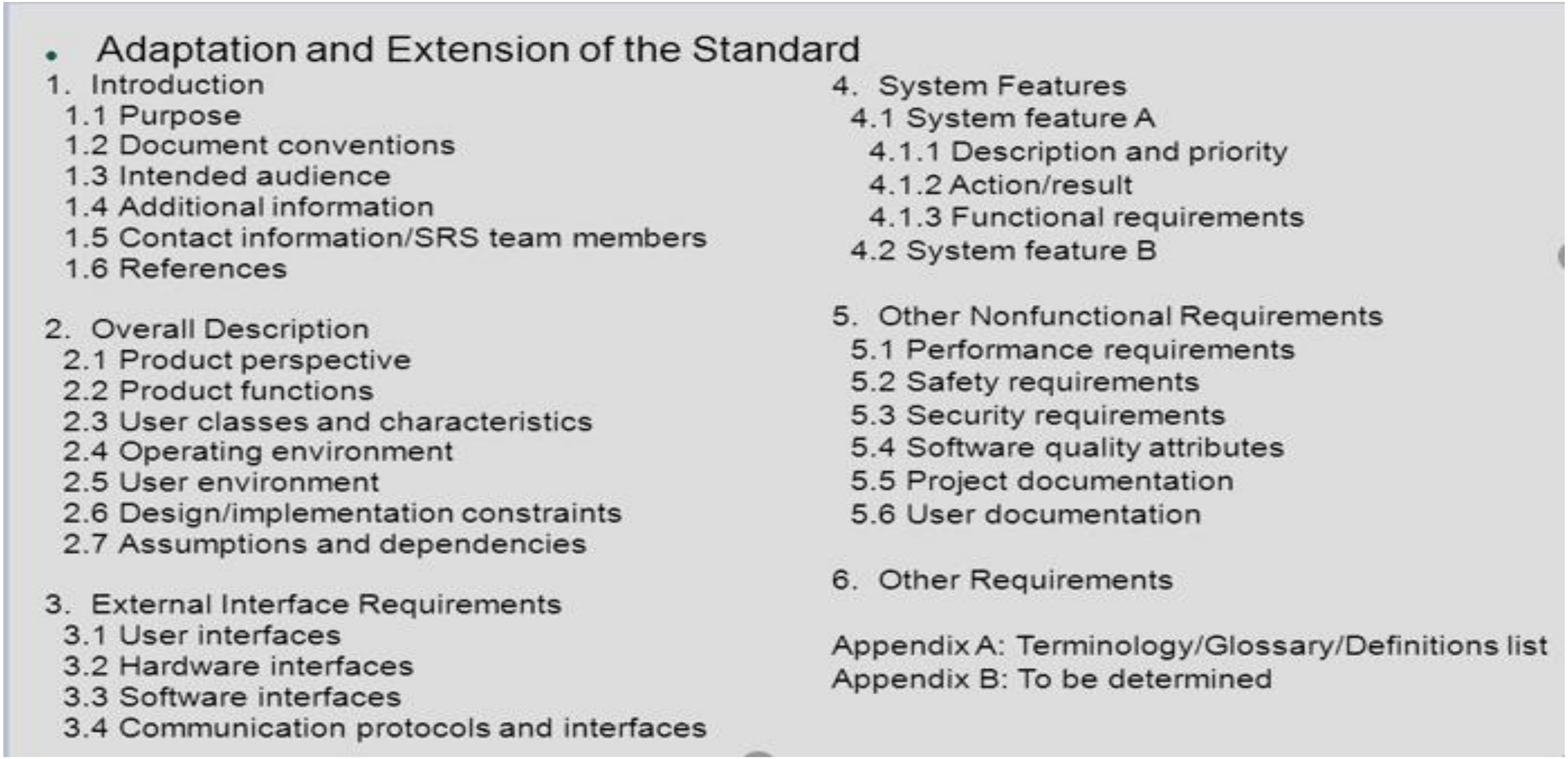
11. **Understandable by the customer:**

An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

12. Right level of abstraction:

If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

IEEE format for SRS



- Adaptation and Extension of the Standard

1. Introduction
   1.1 Purpose
   1.2 Document conventions
   1.3 Intended audience
   1.4 Additional information
   1.5 Contact information/SRS team members
   1.6 References

2. Overall Description
   2.1 Product perspective
   2.2 Product functions
   2.3 User classes and characteristics
   2.4 Operating environment
   2.5 User environment
   2.6 Design/implementation constraints
   2.7 Assumptions and dependencies

3. External Interface Requirements
   3.1 User interfaces
   3.2 Hardware interfaces
   3.3 Software interfaces
   3.4 Communication protocols and interfaces

4. System Features
   4.1 System feature A
      4.1.1 Description and priority
      4.1.2 Action/result
      4.1.3 Functional requirements
   4.2 System feature B

5. Other Nonfunctional Requirements
   5.1 Performance requirements
   5.2 Safety requirements
   5.3 Security requirements
   5.4 Software quality attributes
   5.5 Project documentation
   5.6 User documentation

6. Other Requirements

Appendix A: Terminology/Glossary/Definitions list
Appendix B: To be determined

Example SRS of Academic Administration Software
SPECIFIC REQUIREMENTS

- 3.1 Functional Requirements
- 3.1.1 Subject Registration
  - The subject registration requirements are concerned with functions regarding subject registration which includes students selecting, adding, dropping, and changing a subject.
- F-001:
  - The system shall allow a student to register a subject.
- F-002:
  - It shall allow a student to drop a course.
- F-003:
  - It shall support checking how many students have already registered for a course.

Example SRS of Academic Administration Software
Design Constraints

- C-001:
  - AAS shall provide user interface through standard web browsers.
- C-002:
  - AAS shall use an open source RDBMS such as Postgres SQL.
- C-003:
  - AAS shall be developed using the JAVA programming language

Example SRS of Academic Administration Software Non-Functional Requirements

- N-001:
  - AAS shall respond to query in less than 5 seconds.
- N-002:
  - AAS shall operate with zero down time.
- N-003:
  - AAS shall allow upto 100 users to remotely connect to the system.
- N-004:
  - The system will be accompanied by a well-written user manual.

Software Quality Assurance

- **Software quality assurance (SQA)** is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model etc.
- SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

Software Quality Assurance Plan

- The software quality assurance plan comprises of the procedures, techniques, and tools that are employed to make sure that a product or service aligns with the requirements defined in the SRS(software requirement specification).
- **The SQA plan document consists of the below sections:**

1. Purpose section
2. Reference section
3. Software configuration management section
4. Problem reporting and corrective action section
5. Tools, technologies and methodologies section
6. Code control section
7. Records: Collection, maintenance and retention section
8. Testing methodology

Software Quality Assurance (SQA) Activities)

1. Creating an SQA Management Plan:

- This is the first activity of SQA Plan to do a proper planning for SQA will be carried out for project. Along with what SQA approach are going to follow, what engineering activities will be carried out, and it also includes ensuring about the team.

2. Setting the Checkpoints:

- The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

3. Apply software Engineering Techniques:

- Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Software Quality Assurance (SQA) Activities)

4. Executing Formal Technical Reviews:

- An FTR is done to evaluate the quality and design of the prototype. In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

5. Having a Multi- Testing Strategy:

- By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

Software Quality Assurance (SQA) Activities)

6. Enforcing Process Adherence:

This activity insists the need for process adherence during the software development process. The development process should also stick to the defined procedures. **This activity is a blend of two sub-activities which are explained below in detail:**

(i) Product Evaluation:

This activity confirms that the software product is meeting the requirements that were discovered in the project management plan. It ensures that the set standards for the project are followed correctly.

(ii) Process Monitoring:

This activity verifies if the correct steps were taken during software development. This is done by matching the actually taken steps against the documented steps.

7. Controlling Change:

In this activity, we use a mix of manual procedures and automated tools to have a mechanism for change control. By validating the change requests, evaluating the nature of change and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

Software Quality Assurance (SQA) Activities)

8. Measure Change Impact:

- If any defect is reported by the QA team, then the concerned team fixes the defect. After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project.
- For this purpose, we use software quality metrics which allows managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC and initiate corrective action wherever required.

9. Performing SQA Audits:

- The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process.
- It also checks whatever reported by the team in the status reports were actually performed or not. This activity also exposes any non-compliance issues.

Software Quality Assurance (SQA) Activities)

10) Maintaining Records and Reports:

- It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

11) Manage Good Relations:

- In fact, it is very important to maintain harmony between the QA and the development team.
- We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

SQA Techniques

- **Auditing:** Auditing involves inspection of the work products and its related information to determine if the set of standard processes were followed or not.
- **Reviewing**: A meeting in which the software product is examined by both the internal and external stakeholders to seek their comments and approval.
- **Code Inspection:** It is the most formal kind of review that does static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer and is based on rules, checklist, entry and exit criteria. The reviewer should not be the author of the code.
- **Design Inspection:** Design inspection is done using a checklist that inspects the below areas of software design:
  - General requirements and design
  - Functional and Interface specifications
  - Conventions
  - Requirement traceability
  - Structures and interfaces
  - Logic
  - Performance
  - Error handling and recovery
  - Testability, extensibility
  - Coupling and cohesion

SQA Techniques

- **Simulation:** Simulation is a tool that models the real-life situation in order to virtually examine the behavior of the system under study.
- **Functional Testing:** It is a QA technique which verifies what the system does without considering how it does. This type of black box testing mainly focuses on testing the system specifications or features.
- **Standardization:** Standardization plays a crucial role in quality assurance. It decreases the ambiguity and guesswork, thus ensuring quality.
- **Static Analysis:** It is a software analysis that is done by an automated tool without actually executing the program. This technique is highly used for quality assurance in medical, nuclear and aviation software. Software metrics and reverse engineering are some popular forms of static analysis.

SQA Techniques

- **Walkthroughs:** Software walkthrough or code walkthrough is a kind of peer review where the developer guides the members of the development team to go through the product and raise queries, suggest alternatives, make comments regarding possible errors, standard violations or any other issues.
- **Path Testing:** It is a white box testing techniques where the complete branch coverage is ensured by executing each independent path at least once.
- **Stress Testing:** This type of testing is done to check how robust a system is by testing it under heavy load i.e. beyond normal conditions.
- **Six Sigma:** Six Sigma is a quality assurance approach that aims at nearly perfect products or services. It is widely applied in many fields including software. The main objective of six sigma is process improvement so that the produced software is 99.76 % defect free.

ISO

- ISO 9000 is defined as a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements needed to maintain an efficient quality system.
- They are not specific to any one industry and can be applied to organizations of any size.
- ISO 9000 can help a company satisfy its customers, meet regulatory requirements, and achieve continual improvement.
- It should be considered as first step or the base level of a quality system.

The ISO Approach to Quality Assurance Systems

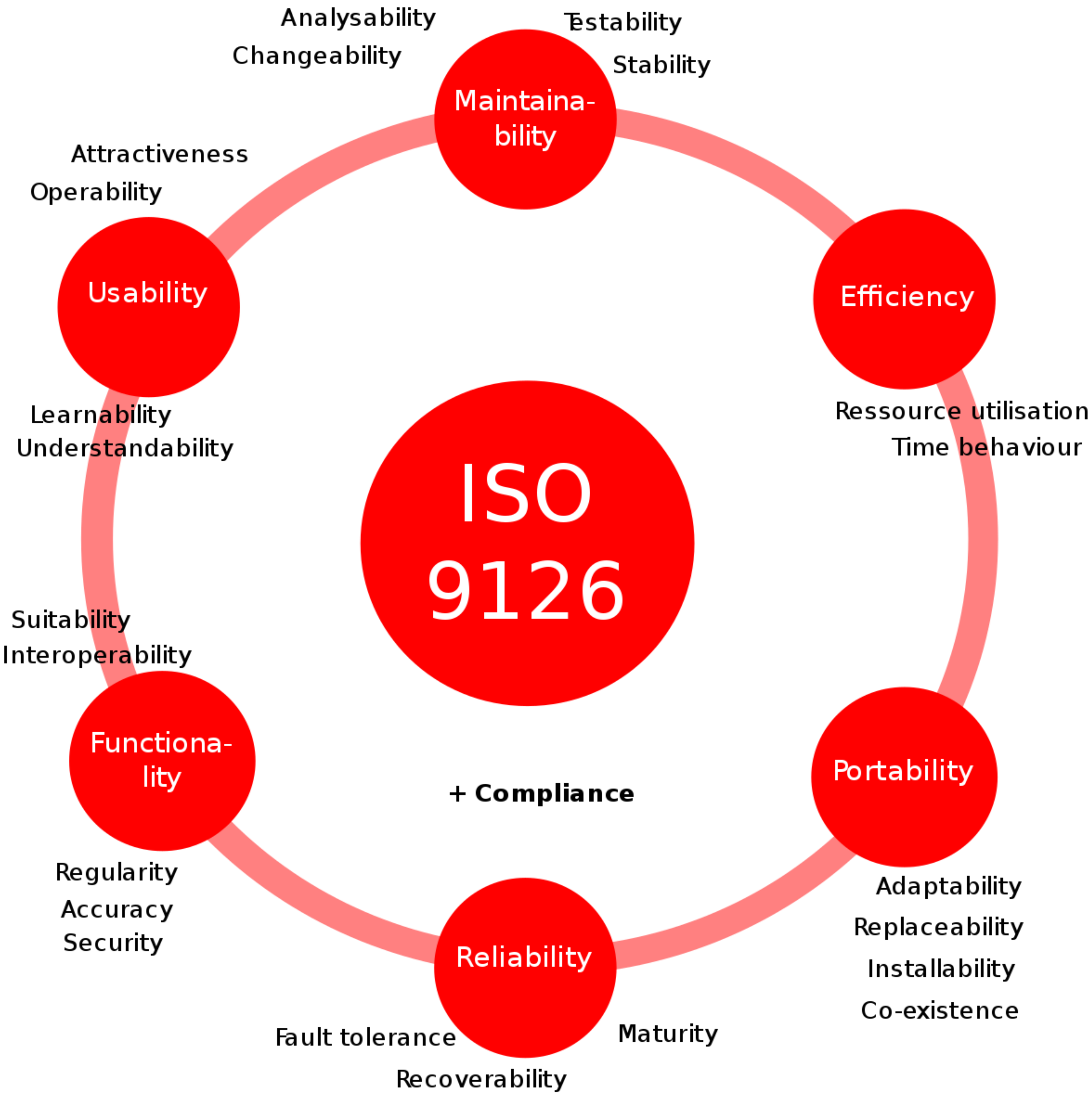
- ISO 9000 describes the elements of a quality assurance system in general terms.
- These elements include the organizational structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance, and quality improvement.
- However, ISO 9000 does not describe how an organization should implement these quality system elements.
- Consequently, the challenge lies in designing and implementing a quality assurance system that meets the standard and fits the company’s products, services, and culture.

The ISO 9001 Standard

- ISO 9001 is the quality assurance standard that applies to software engineering.
- The standard contains 20 requirements that must be present for an effective quality assurance system.
- Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guidelines have been developed to help interpret the standard for use in the software process.
- The requirements defined by ISO 9001 address topics such as
  - management responsibility,
  - quality system,
  - contract review,
  - design control,
  - document and data control,
  - product identification and
  - traceability,
  - process control,
  - inspection and testing,
  - corrective and preventive action,
  - control of quality records,
  - internal quality audits,
  - training, servicing, and
  - statistical techniques.

The ISO 9001 Standard

- The software organization registered to ISO 9001, it must establish policies and procedures to address each of the requirements just noted (and others) and then be able to demonstrate that these policies and procedures are being followed.
- **ISO/IEC (**International Electrotechnical Commission**) 9126** *Software engineering — Product quality* was an international standard for the evaluation of software quality. It has been replaced by **ISO/IEC 25010:2011**.



ISO 9126

Maintaina-bility

Analysability

Changeability

Testability

Stability

Efficiency

Ressource utilisation

Time behaviour

Portability

Adaptability

Replaceability

Installability

Co-existence

Reliability

Maturity

Fault tolerance

Recoverability

Functiona-lity

Suitability

Interoperability

Regularity

Accuracy

Security

Usability

Attractiveness

Operability

Learnability

Understandability

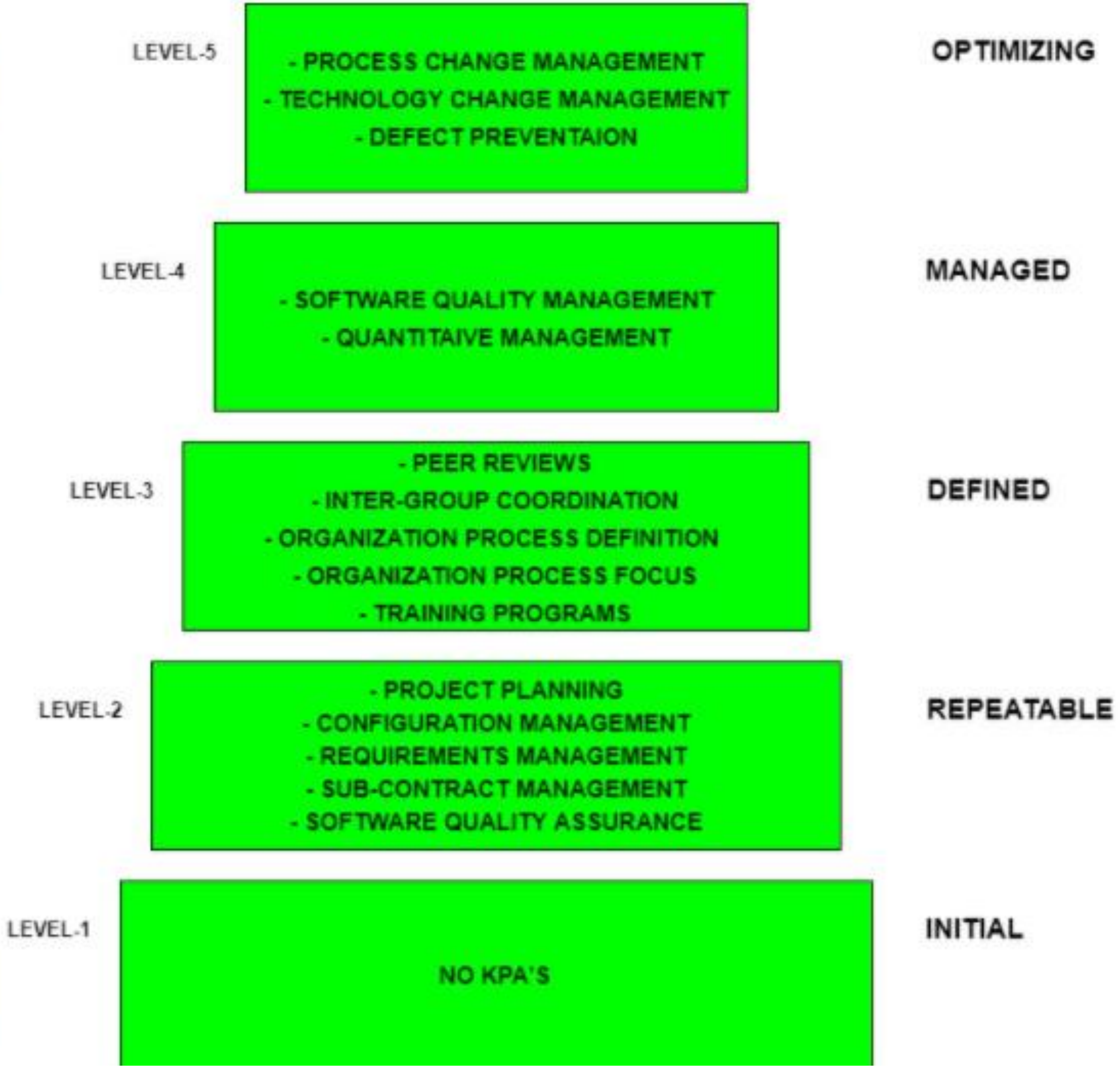
+ Compliance

Capability Maturity Model

- CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
  - It is not a software process model. It is a framework which is used to analyze the approach and techniques followed by any organization to develop a software product.
  - It also provides guidelines to further enhance the maturity of those software products.
  - It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
  - This model describes a strategy that should be followed by moving through 5 different levels.
  - Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA’s).

Key Process Area

- Each of these KPA’s defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.
- Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured, and change is properly managed.
- The 5 levels of CMM are as follows:
  - **Initial**
  - **Repeatable**
  - **Defined**
  - **Managed**
  - **Optimized**



LEVEL-5

- PROCESS CHANGE MANAGEMENT
- TECHNOLOGY CHANGE MANAGEMENT
- DEFECT PREVENTAION

OPTIMIZING

LEVEL-4

- SOFTWARE QUALITY MANAGEMENT
- QUANTITAIVE MANAGEMENT

MANAGED

LEVEL-3

- PEER REVIEWS
- INTER-GROUP COORDINATION
- ORGANIZATION PROCESS DEFINITION
- ORGANIZATION PROCESS FOCUS
- TRAINING PROGRAMS

DEFINED

LEVEL-2

- PROJECT PLANNING
- CONFIGURATION MANAGEMENT
- REQUIREMENTS MANAGEMENT
- SUB-CONTRACT MANAGEMENT
- SOFTWARE QUALITY ASSURANCE

REPEATABLE

LEVEL-1

NO KPA'S

INITIAL

Level-1: Initial

- No KPA’s defined.
- Processes followed are adhoc and immature and are not well defined.
- Unstable environment for software dvelopment.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- KPA's:
  - Project Planning- It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for successful completion of a good quality software.
  - Configuration Management- The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
  - Requirements Management- It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
  - Subcontract Management- It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
  - Software Quality Assurance- It guarantees a good quality software product by following certain rules and quality standard guidelines while development.

Level-3: Defined

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project specific software engineering and management processes.
- KPA’s:
  - Peer Reviews- In this method, defects are removed by using several review methods like walkthroughs, inspections, buddy checks, etc.
  - Intergroup Coordination- It consists of planned interactions between different development teams to ensure efficient and proper fulfilment of customer needs.
  - Organization Process Definition- It’s key focus is on the development and maintenance of the standard development processes.
  - Organization Process Focus- It includes activities and practices that should be followed to improve the process capabilities of an organization.
  - Training Programs- It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- KPA’s:
    - Software Quality Management- It includes the establishment of plans and strategies to develop a quantitative analysis and understanding of the product’s quality.
    - Quantitative Management- It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.
- KPA’s:
    - Process Change Management- Its focus is on the continuous improvement of organization’s software processes to improve productivity, quality and cycle time for the software product.
    - Technology Change Management- It consists of identification and use of new technologies to improve product quality and decrease the product development time.
    - Defect Prevention- It focuses on identification of causes of defects and to prevent them from recurring in future projects by improving project defined process.

ISO	CMM
ISO 9000 is a set of international standarads on quality management and quality assurance developed to help companies effectively document the quality system elements needed to an efficient quality system.	SEI (Software Engineering Institute), Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization.
Focus is customer supplier relationship, attempting to reduce customer's risk in choosing a supplier.	Focus on the software supplier to improve its interval processes to achieve a higher quality product for the benefit of the customer.
It is created for hard goods manufacturing industries.	It is created for software industry.
ISO9000 is recognized and accepted in most of the countries.	SEICMM is used in USA, less widely elsewhere.
It specifies concepts, principles and safeguards that should be in place.	CMM provides detailed and specific definition of what is required for given levels.
This establishes one acceptance level.	It assesses on 5 levels.
Its certification is valid for three years.	It has no limit on certification.
It focuses on inwardly processes.	It focus outwardly.
It has no level.	It has 5 levels:**(a).** Initial **(b).** Repeatable **(c).** Defined **(d).** Managed **(e).** Optimized
It is basically an audit.	It is basically an appraisal.
It is open to multi sector.	It is open to IT/ITES.
Follow set of standards to make success repeatable.	It emphasizes a process of continuous improvement.