Unit-II

Software Requirements

The s/w requirements are description of ~~that~~ features and functionalities of the target system. Requirements convey the expectations of users from the s/w product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the s/w requirements from client, analyze and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'Software Requirement specification' (SRS) document.

Requirement engineering Process: It is a four step process –

- Feasibility study
- Requirement Gathering
- Software Requirement specification – [Requirement Analysis, Requirement Documentation]
- Software Requirement Validation

* **Feasibility study:** The aims of feasibility study to find out whether the system is worth implementing and if it can be implemented, given the existing budget and schedule.

The purpose of feasibility study is not to solve the problem, but to determine whether the problem is worth solving. This helps to decide whether to proceed with the project or not.

The feasibility study concentrates on the following area.

- Technical feasibility
- Economical feasibility
- operational feasibility
- Social feasibility
- Schedule feasibility

Technical feasibility - It is used to identify whether the technical resources are available to form the project or system.

Economic feasibility - Given the financial resources of the company, is the project something that can be completed. It is more commonly called cost/benefit analysis.

Social feasibility - What will be the impact on both local and general cultures? what sort of environmental implications does the feasibility study have?

Schedule feasibility - Can the project be completed in the available time?

operational feasibility - It ~~analyzes~~ analysis the behaviour of the proposed system and whether the proposed system is easier than the existing system for the users of the system

Requirement Gathering:

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analyst and engineers communicate with the client and end users to know their ideas on what the s/w should provide and which features they want the s/w to include.

Meetings, one to one interviews, Group interviews, workshops, brainstorming, questionnaires and Requirement gathering tools are employed to gather requirements from client side.

Requirement Gathering:

Software Requirement specification:

SRS is a document created by system analyst after the requirement are collected from various stakeholders.

SRS defines how the intended s/w will interact with h/w, external interfaces, speed of operation, response time of system, portability of s/w across various platforms, maintainability, speed of recovery after crashing, security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the s/w development team.

Requirement Analysis - Requirements are analyzed in order to identify inconsistencies, defects, commissions etc.

Requirement Documentation - The documentation is very important as it will be the foundation for the design of the s/w. The document is known as software Requirement Specification (SRS).

Software Requirement Validation:

After requirement specifications are developed, the requirements mentioned in this document are validated. Requirements can be checked against following conditions:

- If they can be practically implemented
- If they are valid and as per functionality and domain of s/w
- If there are any ambiguities
- If they can be demonstrated
- If they are complete

Characteristics of an SRS

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

(1) An SRS is correct if every requirement included in the SRS represents something required in the final system.

(2) An SRS is complete if everything the s/w is supposed to do and the responses of the s/w to all classes of input data are specified in the SRS.

(3) An SRS is unambiguous if and only if every requirement stated has one and only one interpretation. One way to avoid ambiguities is to use some formal requirements specification language.

(4) An SRS is verifiable if and only if every stated requirement is verifiable. A requirement is verifiable if there exists some cost effective process that can check whether the final s/w meets that requirement

(5) An SRS is consistent if there is no requirement that conflicts with another. There may be logical or temporal conflict between requirements that causes inconsistencies.

(6) An SRS is ranked for importance and/or stability if for each requirement the importance and the stability of the requirement are indicated. Stability of a requirement reflects the chances of it changing in future. It can be reflected in terms of the expected change volume.

(7) An SRS is modifiable if its structure and style are such that any necessary change can be made easily while preserving completeness and consistency.

(8) An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development. Forward traceability means that each requirement should be traceable to some design and code elements. Backward traceability requires that it be possible to trace design and code elements to the requirements they support.

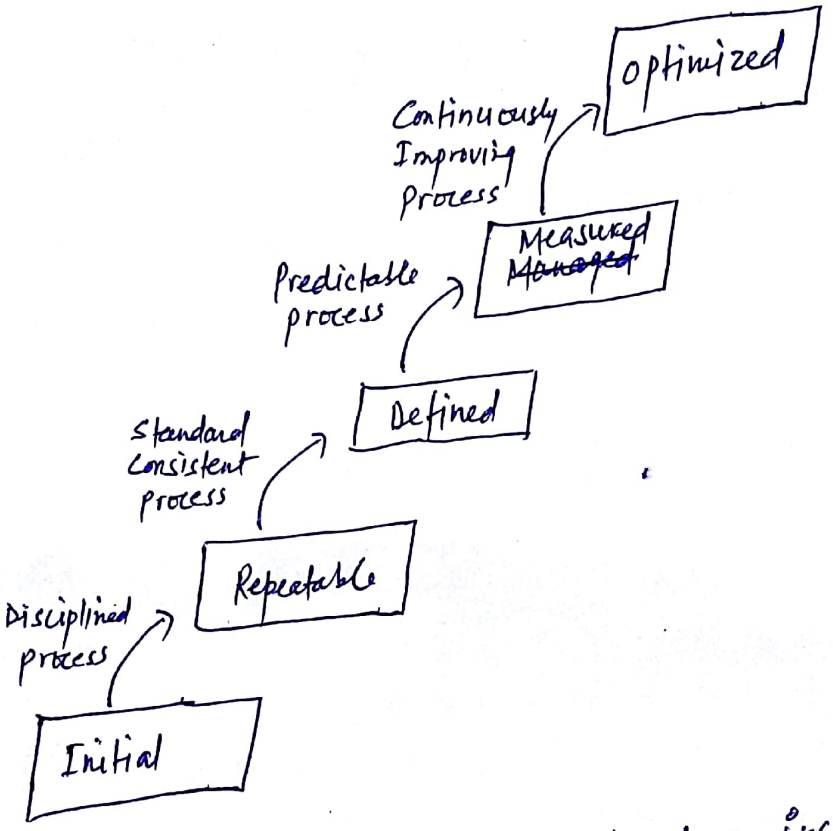
Organization of the SRS

Different projects may require their requirements to be organized differently, i.e. there is no one method that is suitable for all projects.
The general structure of an SRS given by IEEE is as:

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms and Abbreviations
   - 1.4 References
   - 1.5 Overview
2. The Overall Description
   - 2.1 Product Perspective
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 General Constraints
   - 2.5 Assumptions and Dependencies
3. Specific Requirements.
   - 3.1 External Interface Requirement
   - 3.2 Functional Requirement
   - 3.3 Performance Requirements
   - 3.4 Design constraints
   - 3.5 Attributes
   - 3.6 Other Requirements
4. Change Management Process
5. Document Approvals
6. Supporting Information

Software Engineering Institute – Capability Maturity Model (SEI-CMM)

SEI-CMM specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is expensive and time consuming process.



Level 1 : Initial – The s/w process is characterized as inconsistent. Essentially an adhoc process that has no formalized method for any activity. In crisis the project plans and development processes are abandoned in favor of a code and test type of approach. Sucess in such organizations depends solely on the quality and capability of individuals.

Repeatable - Policies for making a software project and procedures to implement those policies exist. That is, project management is well developed in a process at this level. project commitments are realistic and based on past experience with similar projects, cost and schedule are tracked and problems resolved when they arise.

Defined - The organization has standardized a software process, which is properly documented. A software process group exists in the organization that owns and manages the process. Each step is carefully defined with verifiable entry and exit criteria, methodologies for performing the step and verification mechanism for the output of the step.

Manged - Quantitative goals exist for process and products. Measurement plays an important role in a process at this level. The results of using such a process can be predicted in quantitative terms

optimized - The focus of the organization is on continuous process improvements. Data is collected and routinely analyzed to identify areas that can be strengthend to improve quality or productivity. New technologies and tools are introduced. Best software engineering and management practices are used throughout the organization.

SEI-CMM encourage organizations to standardize methodologies, use them consistently and improve them based on experience

Software Quality Attributes

(1) Reliability : The extent to which a software performs its intended functions without failures.

(2) Correctness : The extent to which a software meets its specifications

(3) Robustness : The extent to which a software tolerates the unexpected problems.

(4) Efficiency : The amount of computing resources and code required by software to perform a function.

(5) Usability : The extent of effort required to learn, operate and understand the functions of the software.

(6) Maintainability : The effort required to locate and fix an error during maintenance phase

(7) Testability : The effort required to test a s/w to ensure that it performs its intended functions.

(8) Flexibility : The effort required to modify an operational program.

(9) Portability : The effort required to transfer a program from one platform to another platform.

(10) Reusability : The extent to which a program can be reused in other applications.

(11) Interoperability : The effort required to couple one system with another.

(12) Integrity: The extent to which access to software or data by the unauthorized persons can be controlled.

(13) Traceability: The extent to which an error is traceable in order to fix it.

(14) Completeness: The extent to which a software has specified functions

(15) Testability: The effort required to test a software to ensure that it performs its intended functions.

(16) Adaptability: The extent to which a software is adaptable to new environments (platforms & technologies)

Components of an SRS

Having guidelines about what different things an SRS should specify will help in completely specifying the requirements. The basic issues an SRS must address are:

- Functionality
- Performance
- Design constraints imposed on an implementation
- External interfaces

Functional Requirements: Functional requirements specify which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system.
For each functional requirement, a detailed description of all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.
The functional requirement must clearly state what the system should do if abnormal situations occurs. Furthermore behaviour for situations where the input is valid but the normal operation can not be performed should also be specified

Performance Requirements: All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic.
static requirement are those that do not impose constraint on the execution characteristics of the system. These include requirements like the no of terminals to be supported, the number of simultaneous users to be supported and the no of files that the system has to process and their sizes.
Dynamic requirements specify constraints on the execution behaviour of the system. These typically include response time and throughput constraints on the system.
All of these requirements should be stated in measurable terms.

Design constraints: There are a number of factors in the client's environment that may restrict the choices of a designer. Such factors include standards that must be followed, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

External Interface Requirements: All the interactions of the s/w with people, h/w and other s/w should be clearly specified.
A preliminary user manual should be appear to the user, and feedback and error messages.
For h/w interface requirements, the SRS should specify the logical characteristics of each interface between the s/w product and the h/w components.
The interface requirements should specify the interface with other s/w. This includes the interface with the operating system and other application. The message content and format of each interface should be specified.

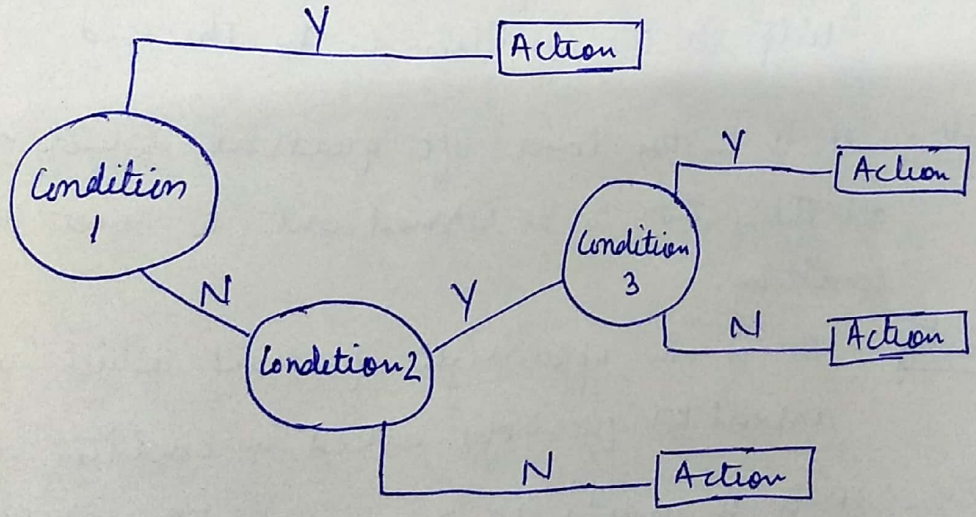
Need for SRS

* Establishes the basis for agreement between the client and the supplier on what the s/w product will do.
* An SRS provides reference for validation of the final product
* A high quality SRS is prerequisite to high quality s/w
* A high quality SRS reduces development cost.

Decision Tree

Decesion Tree are method for defining complex relationships by describing decising and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and condetions within horizontal tree framework. Thus, it depicts which conditions to consider first, second and so on.

Decision Tree depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



Decision Tables -

Decision Tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

It is useful in situations where the resulting actions depend on the occurence of one or several combinations of independent conditions.

It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table :

Condition Stub - It is in the upper left quadrant which lists all the condition to be checked.

Action Stub - It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.

Condition Entry - It is in upper right quadrant which provides answers to questions asked in condition stub.

Action Entry - It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by decision rules which define the relationships between combinations of conditions and courses of action.

In rules section,

→ Y shows the existence of a condition

→ N represents the condition, which is not satisfied

→ A blank - against action states is to be ignored

→ X (or check mark) against action states is to be carried out.

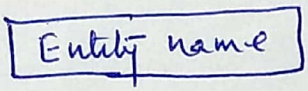
Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Advance Payment made	Y	N	N	N
Purchase amount ≥ 10000	-	Y	Y	N
Regular Customer	-	Y	N	-
Actions				
Give 5% discount	X	X	-	-
Give No discount	-	-	X	X

ER Diagram

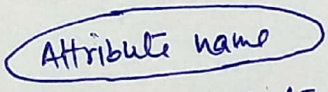
An ER diagram is composed of entity types and specifies relationships that can exist between instances of those entity types.

ER Diagram Components:

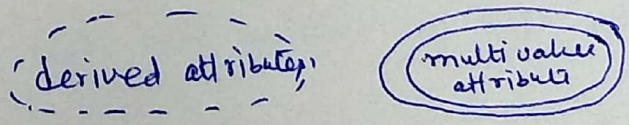
Entity – Entities are represented by means of rectangles



Attributes – Attributes are properties of entities, and they are represented by means of ellipses.



There are composite attribute, multivalued attributes and derived attributes. Derived attributes are depicted by dashed line ellipses.



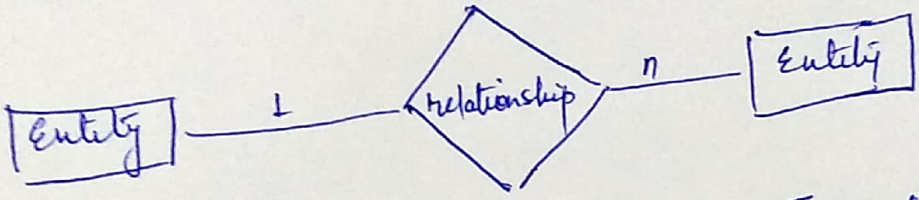
Relationships – Relationship are represented by diamond shaped box.

Binary Relationship and Cardinality – A relationship where two entities are participating is called a binary relationship.

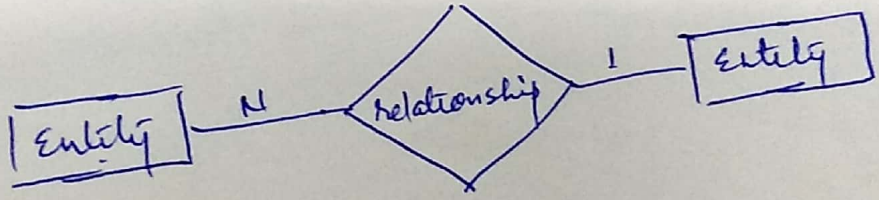
Cardinality is the number of instances of an entity from a relation that can be associated with the relation.

One-to-One – When only one instance of an entity is associated with the relationship, it is marked as 1:1.

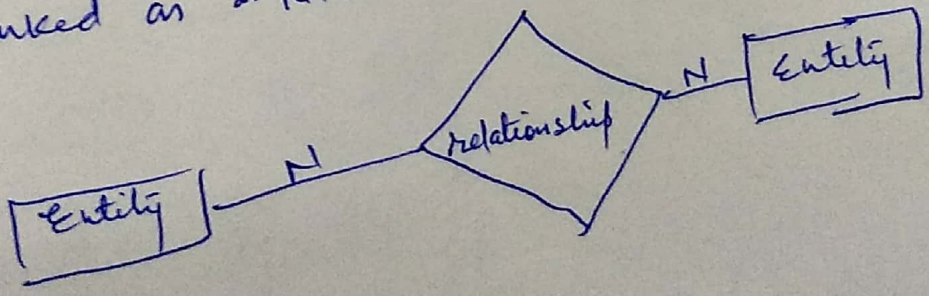
One-to-many – When more than one instance of an entity is associated with a relationship, it is marked as 1:n

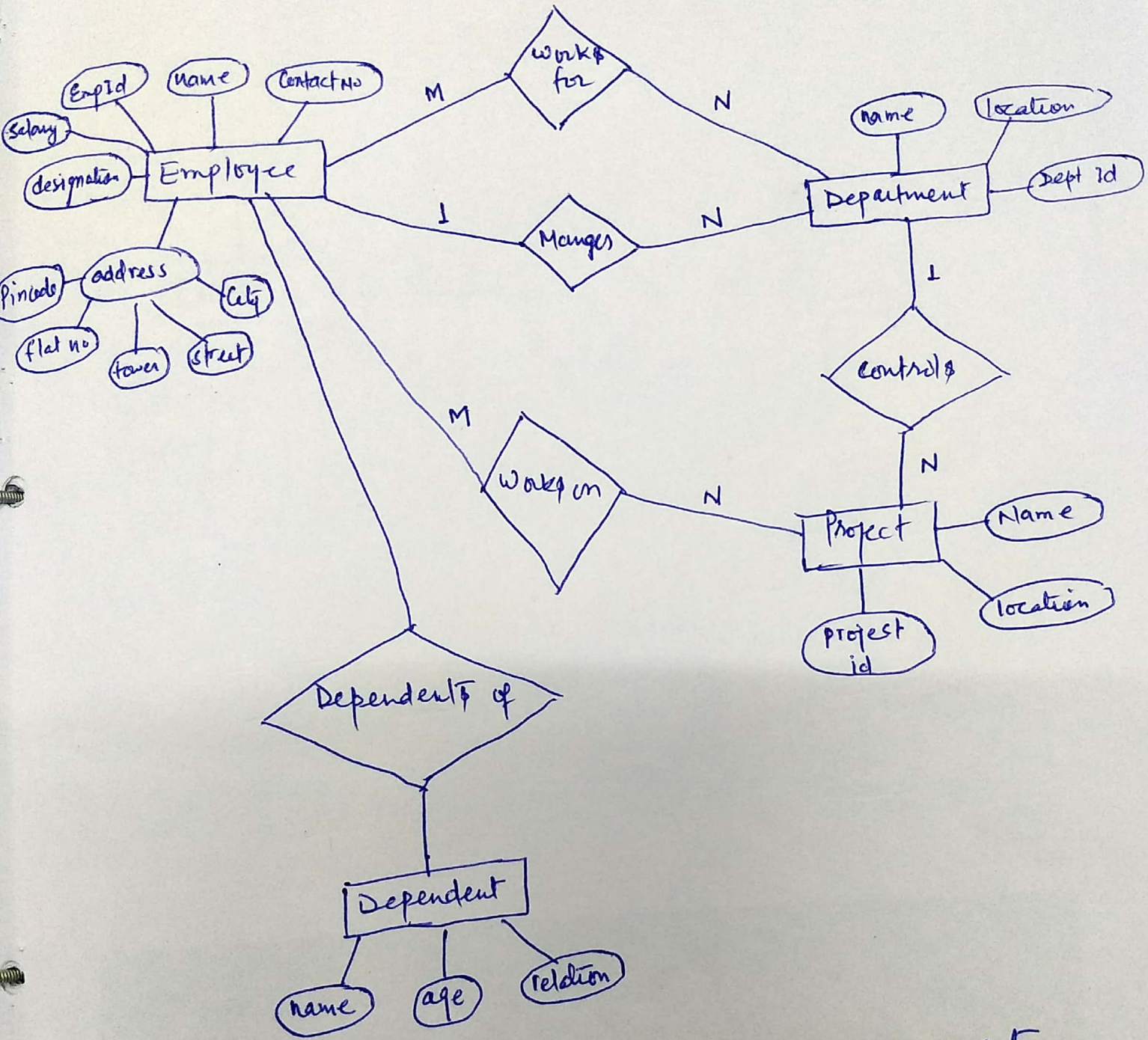


Many-to-One – When more than one instance of entity is associated with the relationship, it is marked as N:1.

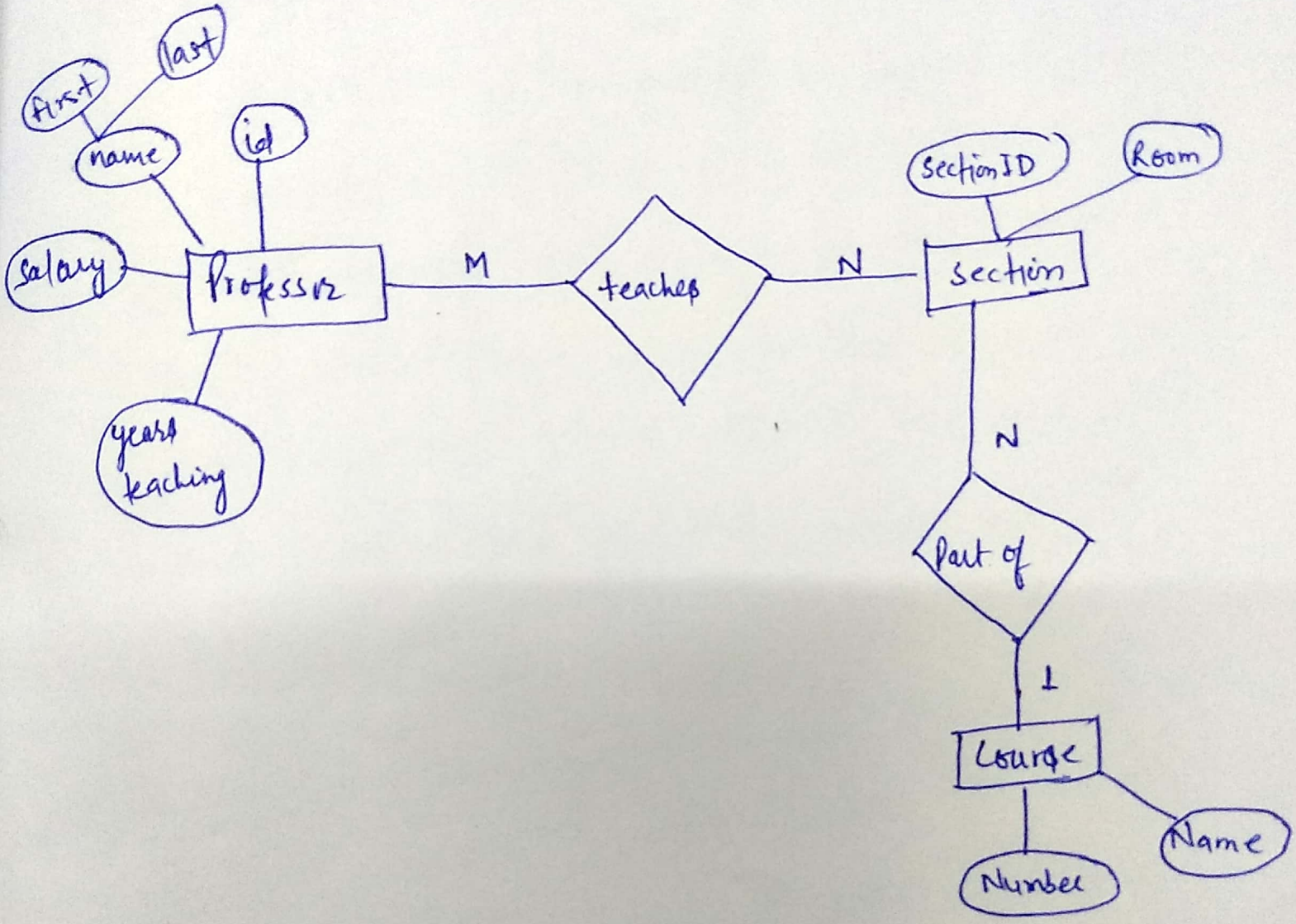


Many-to-many – when more than one instance of both entities are associated with the relationship; it is marked as N:N





ER diagram of Employee Management



ER diagram of Professor Management

Data Flow Diagram (DFD)

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of ~~data~~ control in program module. DFD depicts flow of data in the system at various Levels. DFD does not contain any control or branch elements.

DFD Components :

Entities :- Entities are source & destination of information data. Entities are represented by a rectangle with their respective name.

Process :- Activities and action taken on the data are represented by circle or round edged rectangle.

Data storage :- It can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

Data Flow :- Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD :

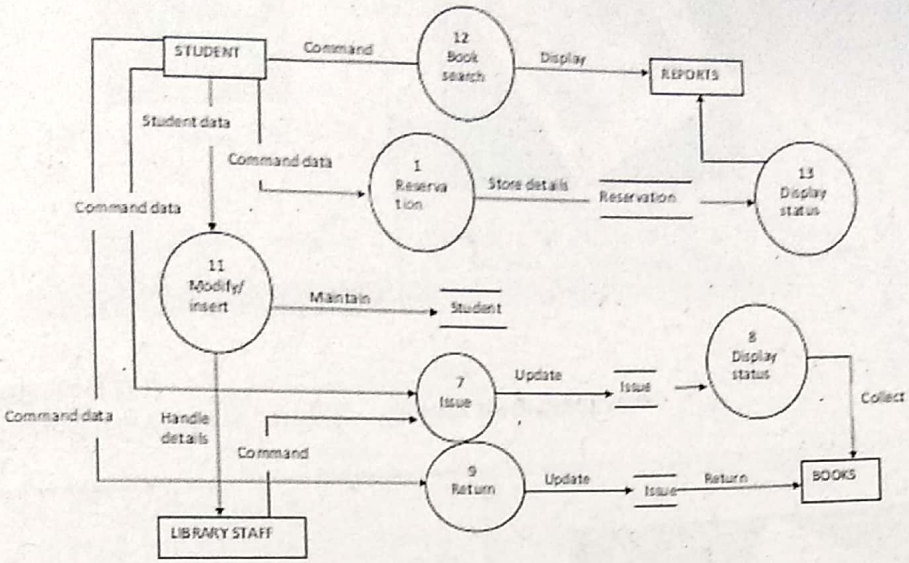
Level 0 — Highest abstraction level DFD is known as level 0 DFD, which depicts the entire information system as one diagram ~~concealing~~ concealing all the underlying details. Level 0 DFDs are also known as Context level DFDs.

Level 1 — The level 0 DFD is broken down into more specific, level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information

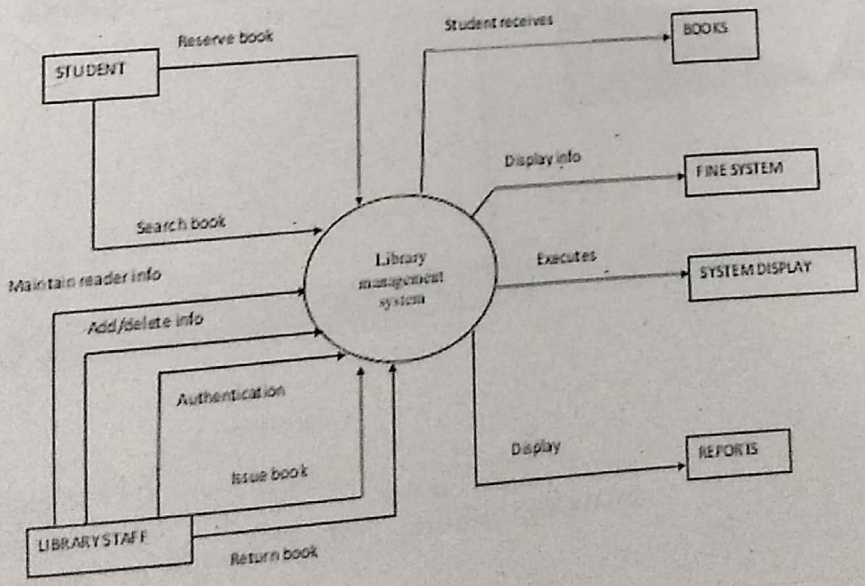
Level 2 — At this level, DFD shows how data flows inside the modules mentioned in level 1.

Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

DATA FLOW DIAGRAM (DFD)

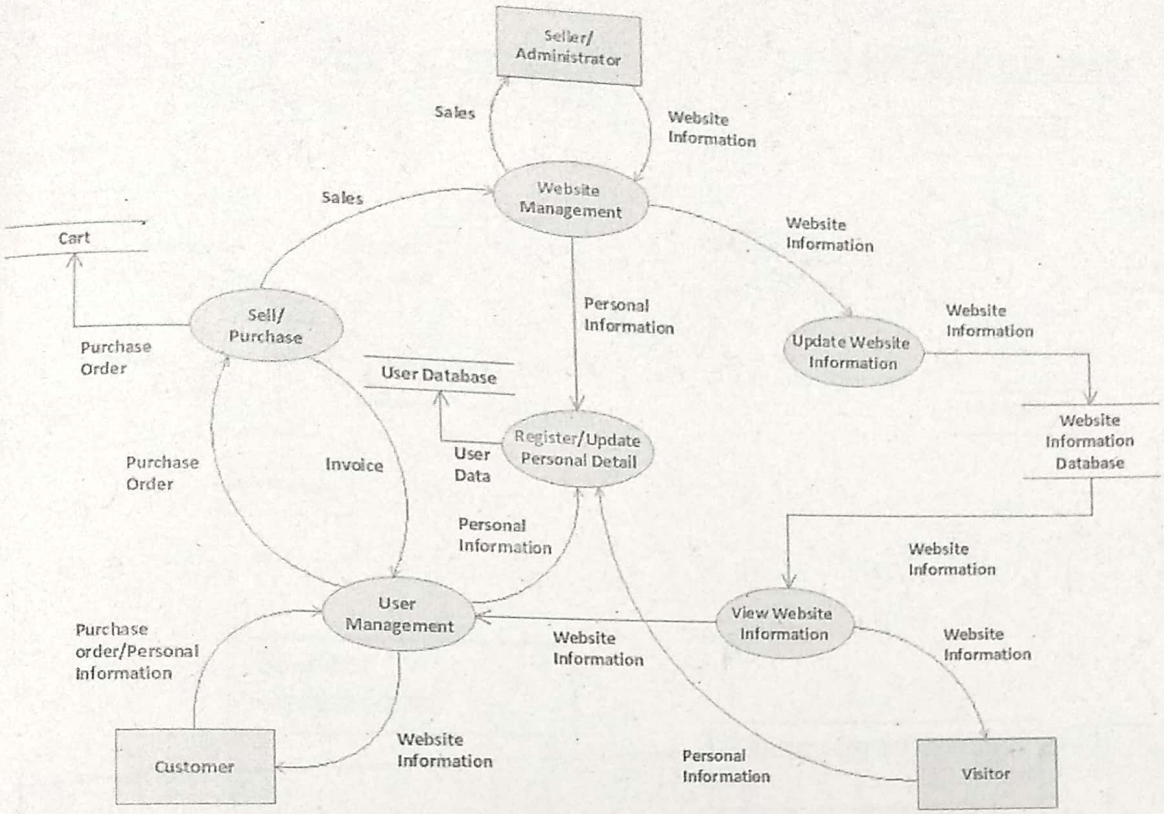


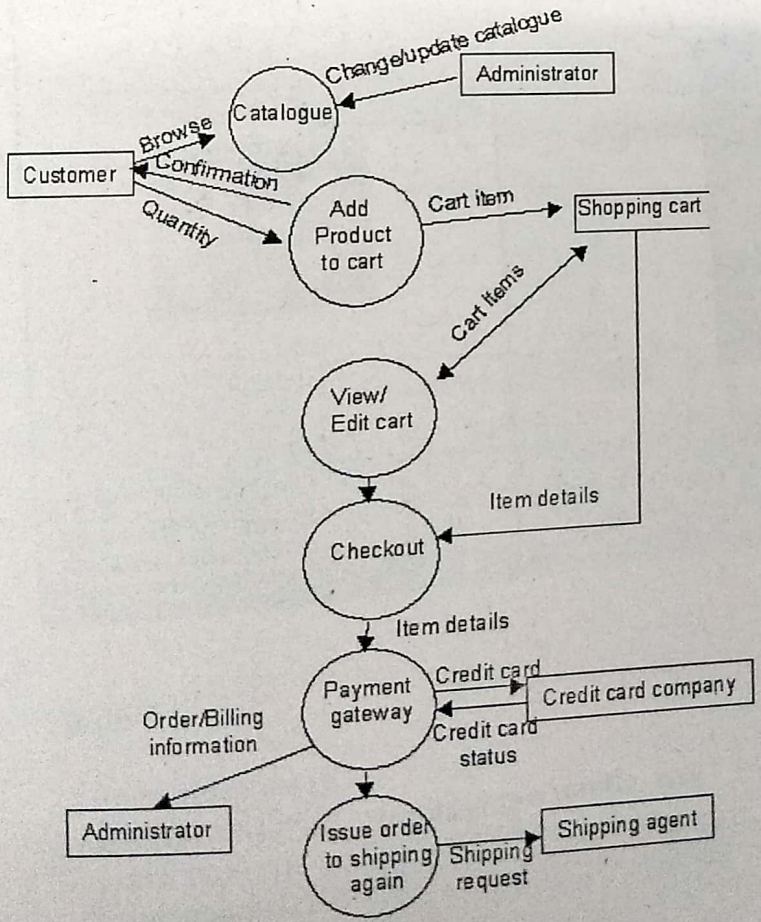
CONTEXT LEVEL DIAGRAM



Library Management System

Online Shopping

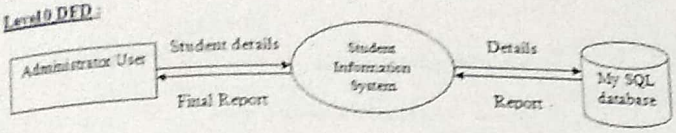




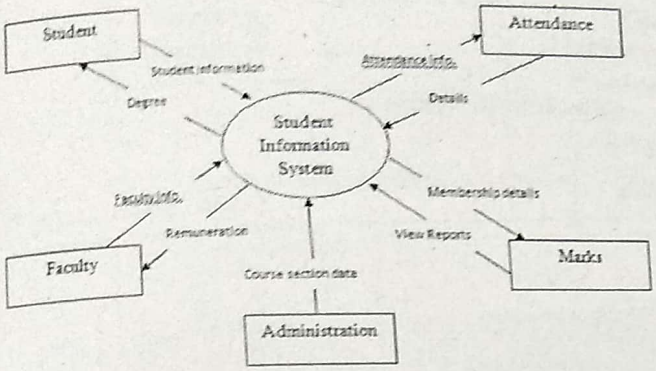
25

College Management System

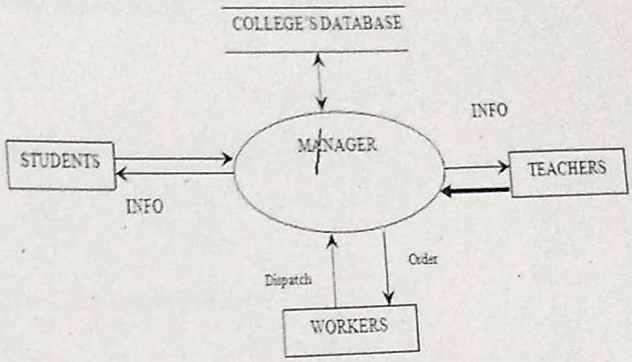
Level 0 DFD :

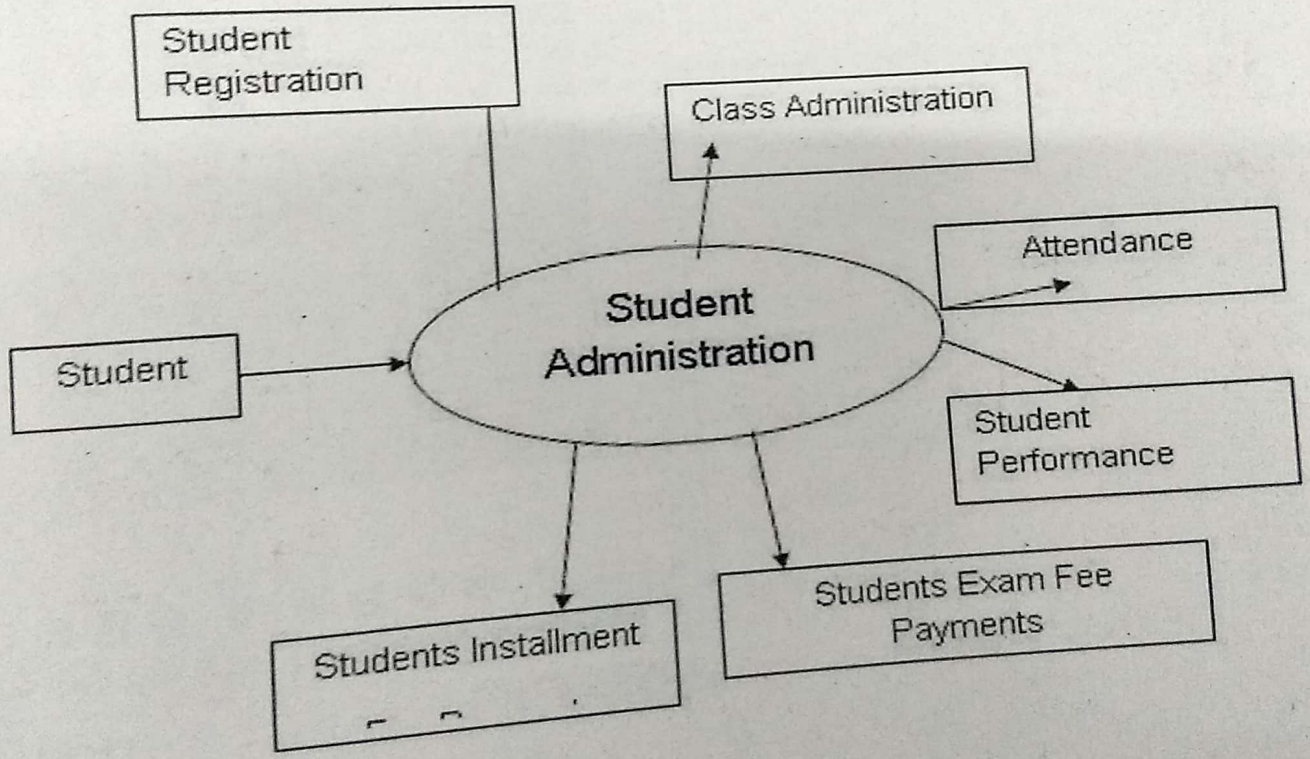


Level 1 DFD:

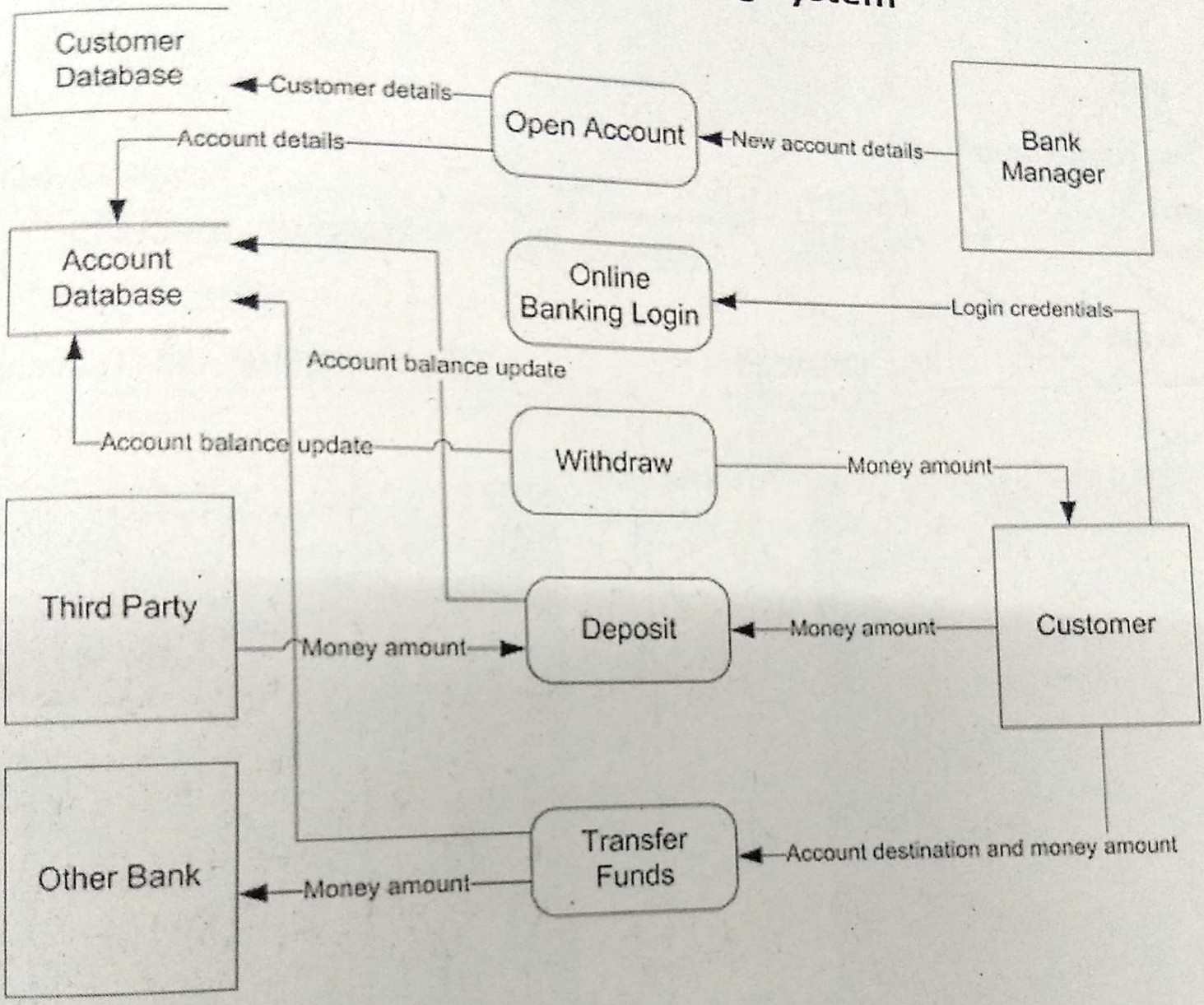


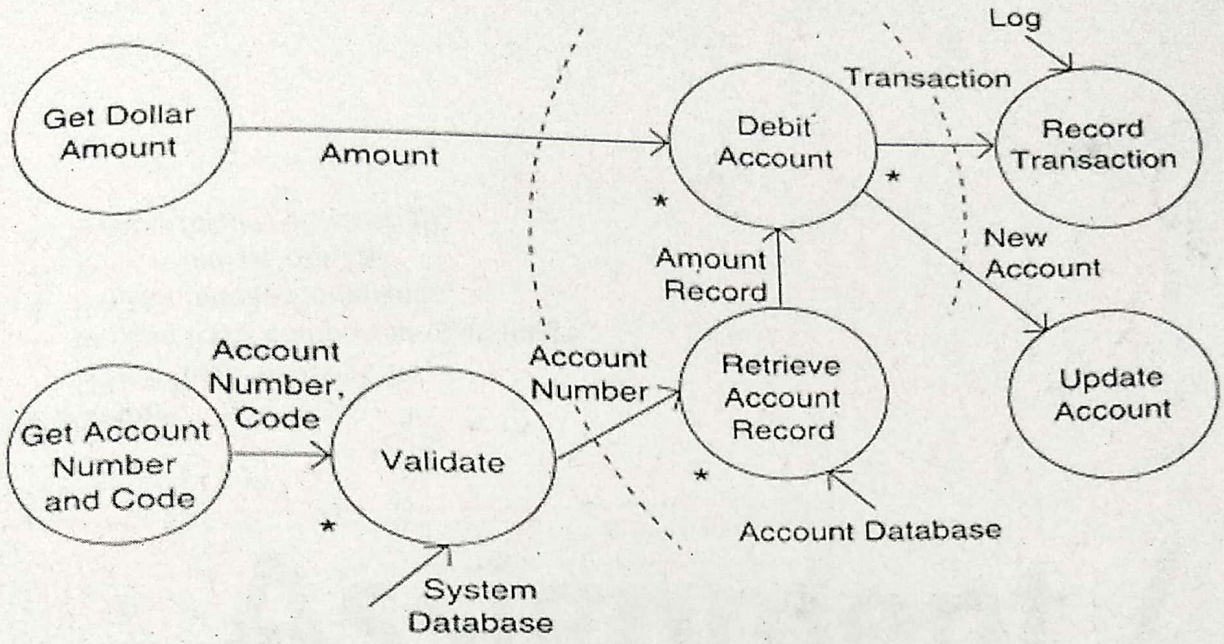
DATA FLOW DIAGRAM



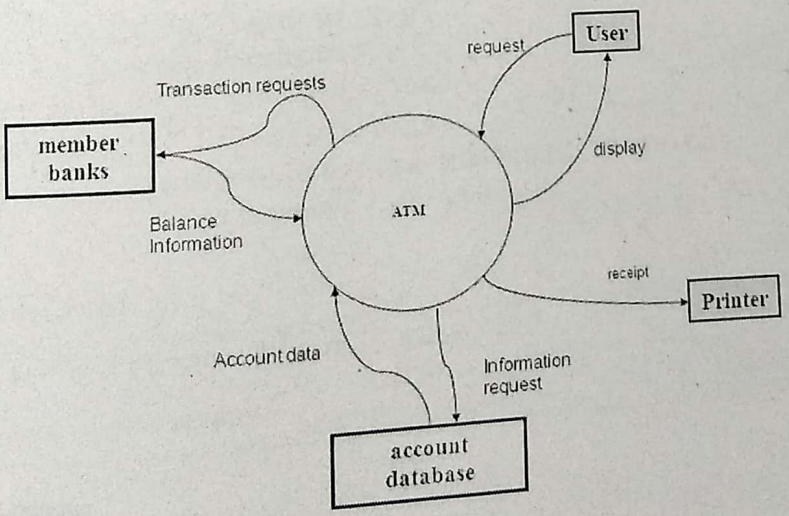


Banking System





Level 0 – DFD ATM



ATM System

ISO - 9000 (International Organization for Standardization)

ISO 9000 is a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements to be implemented to maintain an efficient quality system.

ISO 9000 can help a company satisfy its customers, meet regulatory requirements and achieve continual improvements

Types of ISO 9000 quality standards:

ISO 9000 is a series of 3 standards: ISO-9001, ISO 9002 and ISO 9003.

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

ISO 9002 applies to those organizations which do not design products but are only involved in production.

ISO 9003 applies to organizations that are involved only in installation and testing of the products.

Need for ISO 9000 certification :

1) Confidence of customers in an organizations increases when organization qualifies for ISO certification.
2) ISO 9000 makes the development process focused, efficient and cost effective.
3) ISO 9000 requires a well documented software production process to be in place which enable repeatable and high quality of the software.
4) ISO 9000 sets the basic framework for the development of an optimal process and total quality management (TQM).
5) ISO 9000 certification points out the weak points and recommends remedial action.

ISO 9000 Quality Management Principles

The ISO 9000 series are based on seven quality management principles (QMP)

QMP 1 - Customer focus

QMP 2 - Leadership

QMP 3 - Engagement of people

QMP 4 - Process approach

QMP 5 - Improvement

QMP 6 - Evidence based decision making

QMP 7 - Relationship management.

Verification and Validation

Verification - The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Verification: Are we building the product right?

It is a process oriented approach.

Methods of verification: Static Testing

- Walk Through
- Inspection
- Review

Validation - The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Validation: Are we building the right product?

It is a product oriented approach.

Methods of validation: Dynamic Testing

- Testing
- End users

Verification	Validation
① Are we building the product right?	① Are we building the right product?
② Evaluates the intermediary products to check whether it meets the specific requirements of the particular phase	② evaluates the final product to check whether it meets the business needs.
③ checks whether the product is built as per the specified requirement and design specification.	③ It determines whether the software is fit for use and satisfy the business needs.
④ This is done without executing the software	④ Is done with executing the software.
⑤ Involves static Testing techniques	⑤ Includes all the dynamic testing techniques.

The SQA Plan

The SQA plan is a document that specifies the process to be followed in each step of the software development and the procedures to be followed in each activity of such a process.

The objective of SQA plan is to ensure that the development of the software is based on a course of action and that from time to time the development can be measured, controlled and monitored with respect to such a course of action so that the end product is as per the specifications

SQA Benefits : It ensures that software built as per SQA procedures are of specified quality.

SQA helps to

1) Eliminate errors when they are still inexpensive to correct
2) Improves the quality of the software
3) Improving the process of creating software.
4) Create a mature software process.