

**SOFTWARE REQUIREMENTS:** Introduction of Requirement, User Requirements, System requirements, Functional and Nonfunctional Requirements, The Requirement Engineering Process, **Requirements Elicitation:** Fact finding Techniques, Data/system Analyst, **Requirement Analysis:** Structured Analysis, Data oriented Analysis, Object oriented Analysis, Prototype. **Requirement Specification:** SRS, Characteristics and Components of SRS, Requirements Validation, Requirements Management.

### REQUIREMENT ENGINEERING

#### Introduction

1. The requirements of a system are the **descriptions of the features or services** that the system exhibits within the specified constraints.
2. The requirements collected from the customer are **organized in some systematic manner** and presented in the formal document called *software requirements specification (SRS)* document.
3. *Requirements engineering is the process of gathering, analyzing, documenting, validating and managing requirements.*
4. The main goal of requirements engineering is to **clearly understand the customer requirements** and systematically organize these requirements in the SRS.

#### Software Requirements

- A requirement is a detailed, formal description of system functionalities. It specifies a function that a system or component must be able to perform for customer satisfaction.
- IEEE defines a requirement as :
  - “a condition of capability of a system required by customer to solve a problem or achieve an objective.”
  - “a capability that a product must possess or something a product must do in order to ultimately satisfy customer need, contract, constraint, standard, specification or other formally imposed documents .”
  - “a documented representation of a condition or capability as in (1) or (2).”

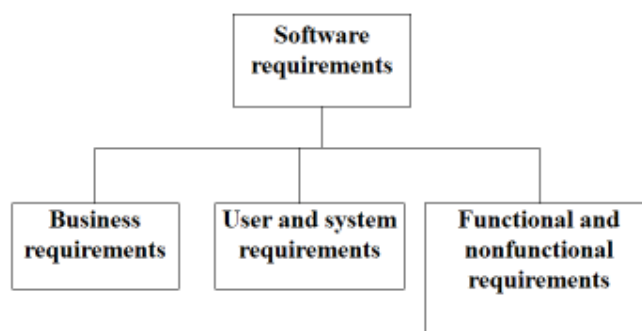


Figure 5.1: Types of requirements

#### Business Requirements:

1. Business requirements define the **project goal and the expected business benefits** for doing the project.
2. The **enterprise mission, values, priorities, and strategies** must be known to understand the business requirements that cover higher level data models and scope of the models.

3. The **business analyst** is well versed in understanding the concept of business flow as well as the **process being followed in the organization**.
4. The business analyst guides the client through the complex process that elicits the requirements of their business.

### User Requirements

1. User requirements are the **high-level abstract statements** supplied by the customer, end users, or other stakeholders.
2. These requirements are translated into system requirements keeping in mind user's views.
3. These requirements are generally represented in some natural language with pictorial representations or tables to understand the requirements.
4. User requirements **may be ambiguous or incomplete** in description with less product specification and little hardware/software configurations are stated in the user requirements.
5. There may be **composite requirements** with several complexities and confusions.
6. In an **ATM machine**, user requirements allow users to **withdraw and deposit** cash.

### System Requirements

1. System requirements are the **detailed and technical functionalities** written in a systematic manner that are implemented in the business process to achieve the goal of user requirements.
2. These are considered as a **contract between** the client and the development organization.
3. **System requirements are** often expressed as documents in a structured manner using **technical representations**.
4. The system requirements consider customer ID, account type, **bank name, consortium, PIN, communication link, hardware, and software**. Also, an ATM will service one customer at a time.

### Functional Requirements

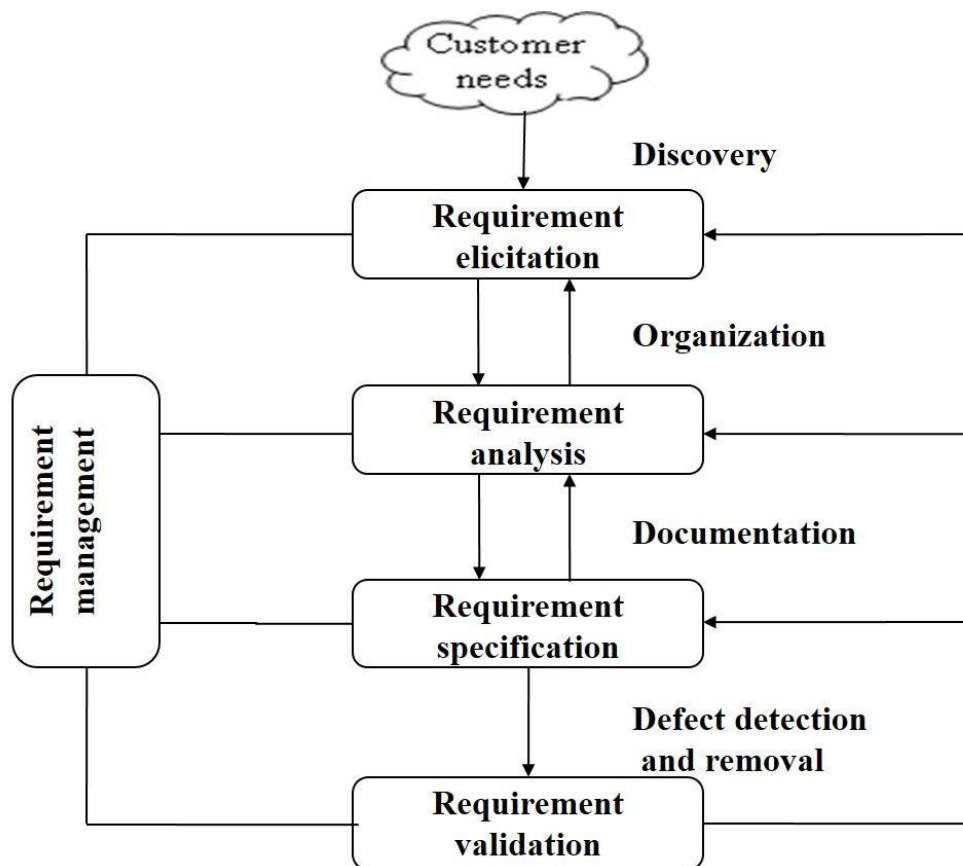
1. Functional requirements are the **behavior or functions** that the system must support.
2. These are the attributes that **characterize** what the software does to **fulfill the needs** of the customer.
3. These can be business rules, **administrative tasks**, transactions, cancellations, authentication, authorization, **external interfaces, legal or regulatory requirements**, audit tracking, certification, reporting requirements, and historical data.

### Nonfunctional Requirements

1. Nonfunctional requirements specify **how a system must behave**. These are **qualities, standards, constraints upon the systems services** that are specified with respect to a product, organization, and external environment.
2. Nonfunctional requirements are related to functional requirements, i.e., how efficiently, by **how much volume, how fast, at what quality, how safely**, etc., a function is performed by a particular system.
3. The examples of nonfunctional requirements are reliability, maintainability, performance, **usability, security, scalability, capacity, availability, recoverability, serviceability, manageability, integrity, and interoperability**.

## Requirement Engineering Process

1. Requirement engineering is the key phase in software development which decides what to build and it provides the outline of the quality of the final product.
  2. Requirement engineering is the disciplined, process-oriented approach to the **development and management of requirements**.
- The main issues involved in requirement engineering are
    - discovering the requirements,
    - technical organization of the requirements,
    - ensuring correctness and completeness of the requirements,
    - Managing requirements that changes over time.



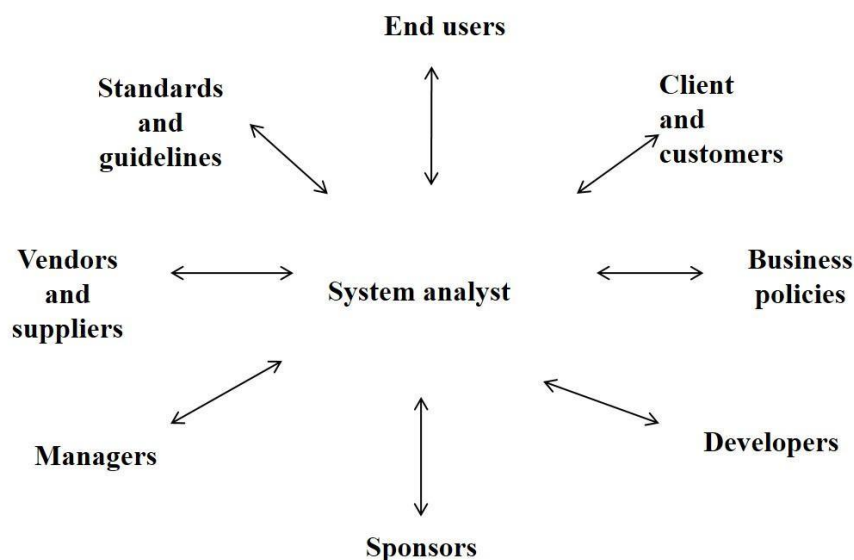
- A typical requirement engineering process has two main aspects:
  1. **Requirement development**
  2. **Requirement management**
  1. **Requirement development** includes various activities, such as elicitation, analysis, specification and validation of requirements.
  2. **Requirement management** is concerned with managing requirements that change dynamically, controlling the baseline requirements, monitoring the commitments and consistency of requirements throughout software development.

## I REQUIREMENTS ELICITATION

1. Requirement elicitation aims to **gather requirements** from different perspectives to understand the **customer needs in** the real scenario.
2. The **goals** of requirement **elicitation** are to **identify the different parties** involved in the project as sources of requirements, **gather requirement from different parties**, write requirements in their original form as collected from the parties, and integrate these requirements.
3. The **original requirements may be inconsistent, ambiguous, incomplete, and infeasible** for the business.
4. Therefore, the **system analyst** involves domain experts, software engineers, clients, end users, sponsors, managers, vendors and suppliers, and other stakeholders and follows standards and guidelines to elicit requirements.

### System analyst

- The role of the system analyst is multifunctional, fascinating, and challenging as compared to other people in the organization.
- A system analyst is the **person who interacts with different people**, understands the business needs, and has knowledge of computing.
- The **skills** of the system analyst include **programming experience, problem solving, interpersonal savvy, IT expertise, and political savvy**. He acts as a broker and needs to be a team player.
- He should also have **good communication and decision-making skills**.
- He should be able to motivate others and should have sound business awareness.



## Challenges in requirements elicitation

- The main challenges of requirements elicitation are
  1. identification of problem scope,
  2. identification of stakeholders,
  3. understanding of problem, and
  4. Volatility of requirements.
- Stakeholders are generally unable to express the complete requirement at a time and in an appropriate language.
- There may be conflicts in the views of stakeholders.
- It may happen that the analyst is not aware of the problem domain and the business scenario.

## Fact-Finding Techniques

- Fact-finding techniques are used to discover the information pertaining to system development.
- Some of the popular techniques are
  - a) *interviewing*,
  - b) *questionnaires*,
  - c) *joint applications development (JAD)*,
  - d) *onsite observation*,
  - e) *prototyping*,
  - f) *viewpoints*,
  - g) *and review records*

### a. Interviewing

1. Interview involves eliciting requirements through **face to face interaction with clients**, end-users, domain experts, or other stakeholders associated with the project.
2. They are conducted to gather facts, verify and clarify facts, identify requirements, and **determine ideas and opinions**.
3. There are **two types of interviews**: unstructured interviews and structured interviews.
4. An **unstructured interview** aims at eliciting various issues related to stakeholders and the existing business system perspectives. There is no predefined agenda and thus general questions are asked from the stakeholders.
5. A **structured interview** is conducted to elicit specific requirements and therefore the stakeholders are asked to answer specific questions.

## b. Questionnaires

1. Questionnaires are used to **collect and record large amount of qualitative** as well as **quantitative** data from a number of people.
2. The system analyst **prepares a series of questions in a logical manner**, supplemented by **multiple choice answers and instructions for filling** the questionnaires.
3. It is a **quick method of data collection** and **inexpensive** as compared to interviews.
4. It provides **standardized and uniform responses** from people as questions are designed objectively.
5. However, it is very difficult to design logical and unambiguous questions.
6. A very few people take interest in responding and carefully filling questionnaires.

## c. Joint Applications Development (JAD)

1. It is a structured group meeting **just like workshop** where the **customer, designer**, and other **experts meet** together for the purpose of identifying and understanding the problems to define requirements.
2. The JAD session has **predefined agenda and purpose**.
3. It includes various participants with their well-defined roles, such as facilitator, **sponsors, end users, managers, IT staff, designers**, etc.
4. The JRD meeting is conducted to identify and understand problems, **resolve conflicts, generate ideas, and discuss some alternatives** and the best solutions.
5. **Brainstorming** and focus group techniques are used for JRD.

## d. Onsite observation

1. The system analyst personally visits the client site organization, observes the **functioning of the system**, understands the flow of documents and the users of the system, etc.
2. It helps the system analyst to gain insight into the working of the system **instead documentation**.
3. Through constant observation, the system analyst identifies **critical requirements** and their influence in the system development.
4. Users may feel **uncomfortable if someone observes** their work.

## e. Prototyping

1. Prototyping technique is helpful in situations **where it is difficult to discover and understand** the requirements and **where early feedback from the customer** is needed.
2. **An initial version of the final product called prototype** is developed that can give clues to the client to provide and think on additional requirements. The initial version will be changed and a new prototype is developed.
3. **Prototype** development is a **repetitive process** that **continues until the product** meets the business needs or requirements.
4. In this way, prototype is corrected, enhanced, or refined to reflect the new requirements.

#### f. Viewpoints

1. Viewpoint is a way of structuring requirements to represent the perspectives of different stakeholders as there is no single correct way to analyze the system requirements.
2. Stakeholders may be classified **under different viewpoints**:
  - a. direct (who directly interact with the system),
  - b. indirect (who indirectly interact with the system),
  - c. Domain (represent the domain standards and constraints) viewpoints.
3. The collected viewpoints are classified and evaluated under the system's circumstances and finally, it is integrated into the normal requirement engineering process.

#### g. Review records

1. **Reviewing the existing documents** is a good way of gathering requirements.
2. The information related to the system is found in the documents such as **manual of the working process, newspapers, magazines, journals**, etc.
3. The **existing forms, reports, and procedures help to understand the guidelines** of a process to identify the business rules, discrepancies, and redundancies.
4. This method is beneficial to new employees or consultants working on the project.

### II REQUIREMENTS ANALYSIS

1. In requirement analysis, we **analyze stakeholders' needs, constraints, assumptions, qualifications**, and other information relevant to the **proposed system** and organize them in a systematic manner.
2. It is performed in **several iterations** with elicitation to **clarify requirements, resolve conflicts**, and ensure their correctness and completeness.
3. It includes various activities, such as **classification, organization, prioritization, negotiation, and modeling requirements**.
4. The draft requirements are **analyzed to verify their correctness and completeness**.
5. During **requirement analysis, models are prepared to analyze the requirements**.
6. The following **analysis techniques** are generally used for the modeling of requirements:
  - a. *Structured analysis*
  - b. *Data-oriented analysis*
  - c. *Object-oriented analysis*
  - d. *Prototyping*

## A. Structured Analysis

1. Structured analysis is also referred to as *process modeling* or *data flow modeling*.
2. It focuses on **transforming** the documented **requirements in the form of processes** of the system.
3. **During transformation**, it follows a **top-down functional decomposition process** in which the system is considered a **single process** and it is further decomposed **into several sub-processes** to solve the problem.
4. Thus, the aim of structured analysis is to **understand the work flow of the system** that the user performs in the existing system of the organization.

During Structured Analysis, various tools and techniques are used for system development. They are

### A. Data Flow Diagrams

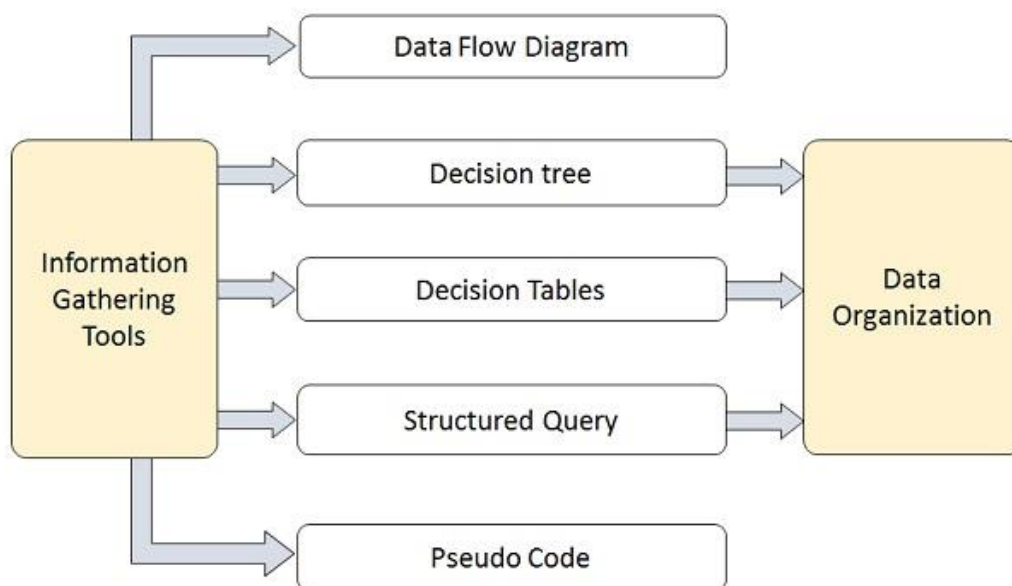
### B. Data Dictionary

### C. Decision Trees

### D. Decision Tables

### E. Structured English

### F. Pseudo code



## A. Data Flow Diagrams (DFD) or Bubble Chart





- a. A DFD is a graphical tool that describes the flow of data through a system and the functions performed by the system.
- b. It shows the processes that receive **input**, perform a series of transformations, and produce the desired **output**.
- c. It does not show the control information (time) at which processes are executed.
- d. DFD is also called a **bubble chart or process model** or information flow model.



## Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance

Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

## Types of DFDs

DFDs are of two types: **Physical DFD** and **Logical DFD**. The following table lists the points that differentiate a physical DFD from a logical DFD.

**Physical DFD** : It provides low level details of hardware, software, files, and people.

**Logical DFD**: It explains events of systems and data required by each event.

## Constructing DFD

1. The construction of the DFD starts with the **high-level functionality** of the system, which incorporates external **inputs and outputs**.
2. This abstract DFD is **further decomposed into smaller functions** with the same input and outputs.
3. The **decomposed DFD** is the **elaborated** and **nested DFD** with more concrete functionalities.
4. Decomposition of DFD at various levels to design a nested DFD is called *leveling of DFD*.
5. Sometimes, dotted lines are used to represent the control flow information. Control flow helps to decide the sequencing of the operations in the system.

This **approach begins** with **studying the existing system**

- a. to understand the working of the system
- b. to design the context diagram,
- c. apply functional decomposition to subdivide the functional requirements,

- d. Mark the boundary of the system to know what can be automated or what will be manual, and prepare the dictionary of the system.

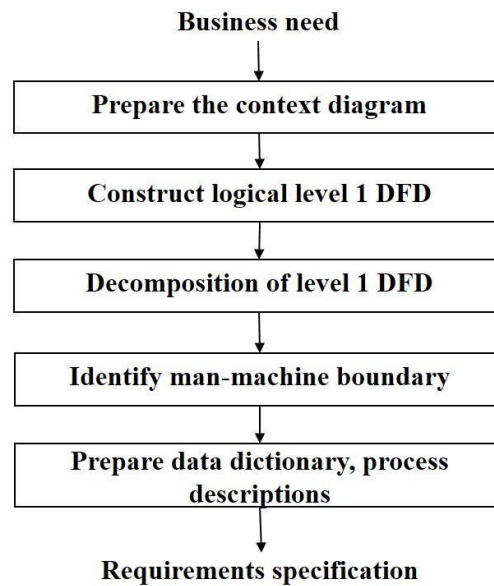


Figure 5.6: Structured analysis approach

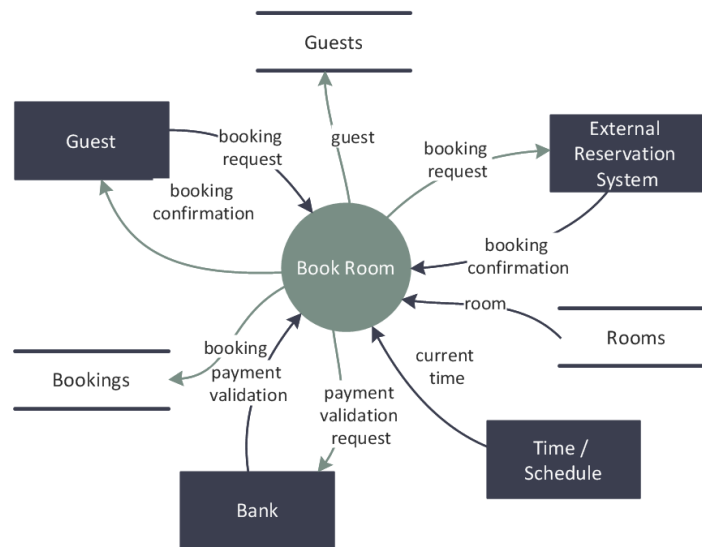
### 1. Prepare Context Diagram

The context diagram (or level 0 DFD) is the high-level representation of the proposed system after studying the physical DFD of the existing system. .

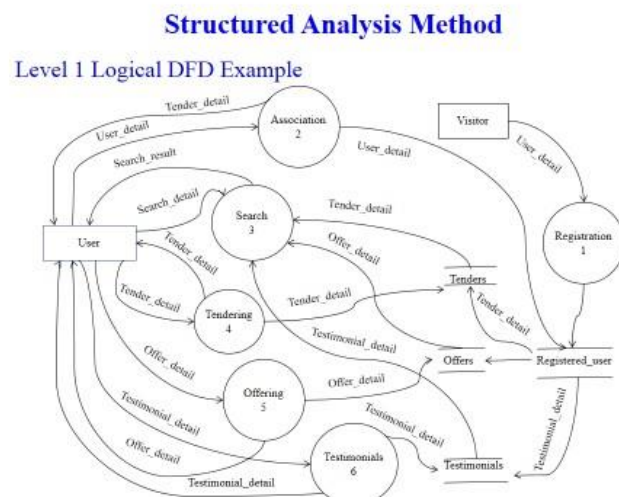
1. The entire system is treated as a single process called *bubble*, with all its external entities.
2. That is, the system is represented with main input and output data, process, and external entities.



Context Diagram for Reservation



## 2. Construct Level 1 Logical DFD



## 3. Decomposition of Level 1 DFD

- It follows a top-down analysis and functional decomposition to refine the level 1 DFD into smaller functional units.
- Functional decomposition of each function is also called **exploding the DFD** or **factoring**.
- The process of decomposition is repeated until a function needs no more subdivisions.
- Each successive level of DFD provides a more detailed view of the system.
- The goal of decomposition is to develop a *balanced* or *leveled DFD*. That is, data flows, datastores, external entities, and processes are matched between levels from the context diagram to the lowest level.
- For example, the level 2 DFDs for the decomposed processes *offers* and *tenders* are shown in Figure 5.9 and Figure 5.10, respectively.

## 4. Identify Man-Machine Boundary

- After constructing the **final DFD**, boundary conditions are identified, which ensures what will be **automated** and **what can be done manually** or by another machine.
- For example, in an ATM system, a user will select options, cancel operation, and receive cash and receipt.

## B. Data Dictionary

A data dictionary is a structured **repository of data elements** in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

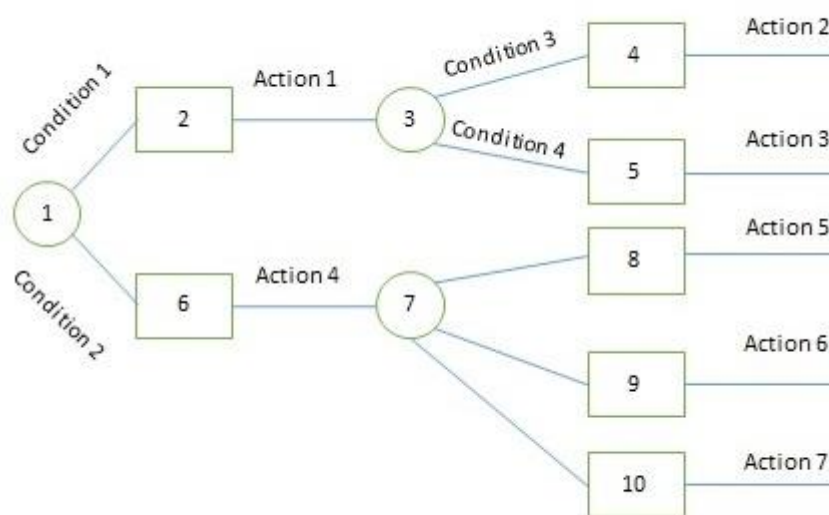
A **data dictionary improves the communication between the analyst and the user**. It plays an important role in building a database. **Most DBMSs have a data dictionary** as a standard feature. For example, refer the following table –

Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15

## C. Decision trees :

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

Decision trees 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.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree –



#### D. Decision Tables

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

1. It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
2. It is a matrix containing row or columns for defining a problem and the actions.

##### Components of a Decision Table

- a. **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- b. **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- c. **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- d. **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 it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table –

<i>CONDITIONS</i>	<b>Rule 1</b>	<b>Rule 2</b>	<b>Rule 3</b>	<b>Rule 4</b>
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N
Regular Customer	-	Y	N	-
<i>ACTIONS</i>				
Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

### E. Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

- It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
- It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

**For example, see the following sequence of actions –**

```
if customer pays advance
then
    Give 5% Discount
else
    if purchase amount >=10,000
    then
        if the customer is a regular customer
        then Give 5% Discount
        else No Discount
        end if
    else No Discount
    end if
end if
```

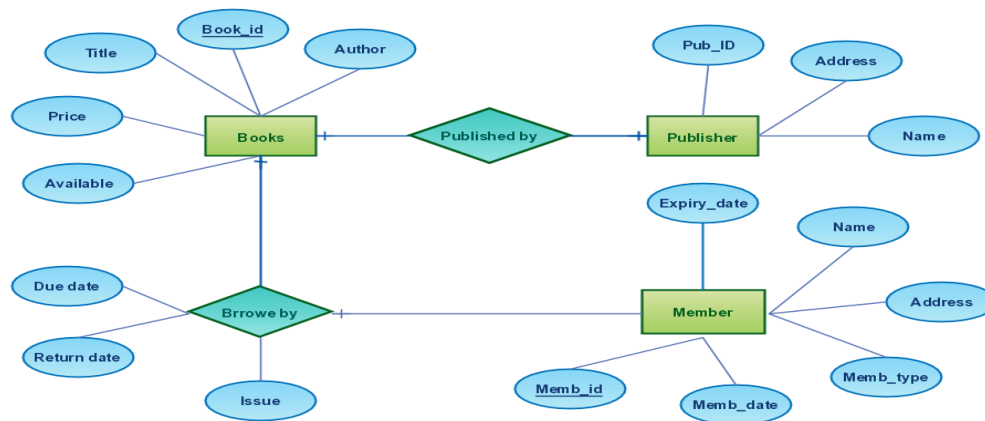
### F. Pseudo code

A pseudo code does not conform to any programming language and expresses logic in plain English.

- It may specify the physical programming logic without actual coding during and after the physical design.
- It is used in conjunction with structured programming.
- It replaces the flowcharts of a program.

## II. Data-Oriented analysis: Entity Relationship Model:

E-R Diagram for Library Management System



### What is ER Diagrams

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

### Facts about ER Diagram :

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities
- ER modeling, which aims at **conceptual representation** of the business requirements.
- Entities are classified as independent or dependent entities.
- An **independent entity** or **strong entity** is one that does not rely on another for identification.
- A dependent entity or weak entity is one that relies on another for identification.
- A **weak entity** set is represented by a doubly outlined box
- its relationship is represented by a doubly outlined diamond.
- The discriminator of a weak entity set is underlined with a dashed line.
- Attributes are the properties or descriptors of an entity. e.g., the entity course contains ID, name, credits, and faculty attributes. Attributes are represented by ellipses.
- The logically-grouped attributes are called compound attributes.
- An attribute can be **single valued** or **multi-valued**.
- A **multi-valued** attribute is represented by a **double ellipse**.
- The attribute which is used to uniquely identity an entity is called a **key attribute** or an Identifier and it is indicated by an **underline**.
- **Derived attributes** are the attributes whose values are derived from other attributes. They are indicated by a **dotted ellipse**.

## Notations of ER Model:

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E <sub>2</sub> IN R
	CARDINALITY RATIO 1:N FOR E <sub>1</sub> :E <sub>2</sub> IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

partial key

## ER Model Relationships

- A relationship represents the association between two or more entities.
- It is represented by a diamond box and two connecting lines with the name of relationship between the entities.
- Relationships are classified in terms of degree, connectivity, cardinality, direction, and participation.
- The number of entities associated with a relationship is called the *degree of relationship*. It can be recursive, binary, ternary, or n-ary.
- A *recursive relationship* occurs when an entity is related to itself.
- A *binary relationship* associates two entities.
- A *ternary relationship* involves three entities



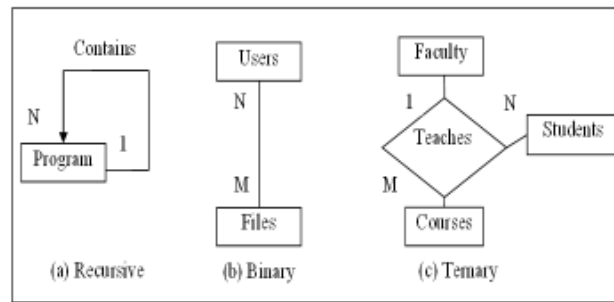


Figure 5.13: Relationships with cardinalities

### Process of Data-Oriented analysis Method or ER Model:

1. Identification of entity and relationships
2. Construct the basic E-R diagram
3. Add key attributes to the basic E-R model
4. Add non-key attributes to the basic E-R model
5. Perform normalization
6. Adding integrity rules to the model

### III. OBJECT-ORIENTED ANALYSIS:

- A. Object-oriented approach also combines both data and processes into single entities called objects.
- B. The **object-oriented** approach has **two aspects**, object-oriented analysis (**OOA**) and object-oriented design (**OOD**).
- C. The idea behind OOA is to consider the **whole system** as a **single** complex entity called **object, breaking down the system into its various objects**, and combining the data and operations in objects.
- D. OOA increases the understanding of problem domains, promotes a smooth transition from the analysis phase to the design phase, and provides a more natural way of organizing specifications.
- E. There exist various object-oriented approaches **for OOA and OOD**, e.g. object modeling technique (**OMT**)

### OBJECTS AND CLASSES:

- F. Class is the combinations of Data Members and Member functions into a single unit.
- G. Object is class variable and instance of the class. Or Objects are the real world entities that can uniquely be identified.
- H. Whatever the attributes (states) and Member functions (behavior) of class will be in object.
- I. *Attributes* are the data values of an object.
- J. *Operations* are the services or functions of an object in a class.

- K. An object communicates to other objects through **message passing** or signatures.
- L. In OMT, a class is represented through **class diagram**, which is indicated by a **rectangle** with its three parts: first part for class name, second part for its attributes, and third part for operations
- M. Objects are shown through **object diagrams**, which are also represented with **rounded rectangles**.
- N. For example, the class diagram for an employee is shown in Figure 5.16.

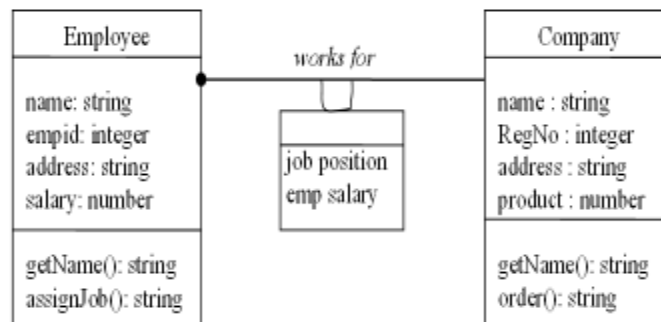


Figure 5.16: Class diagram and association of employee in a company

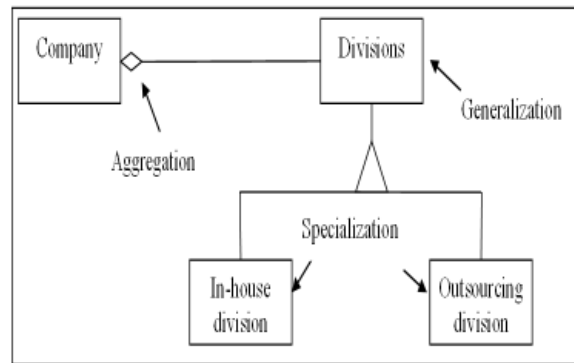
#### Association:

- A. **Relationship** between **classes** is known as **association**.
- B. Association can be **binary, ternary, or n-ary**.
- C. Multiplicity defines how many instances of a class may relate to a single instance of another class. It can be one-to-one, one-to-many, or many-to-many.
- D. An association can also have attributes and they are represented by a box connected to the association with a loop.
- E. Role names are also attached at the end of the association line.
- F. Sometimes, a qualifier can be attached to the association at the many side of the association line.

#### Aggregation, Generalization, and Specialization

1. An *aggregation* models a “whole-part” relationship between objects. In aggregation, a single object is treated as a collection of objects.
2. *Generalization* represents the relationship between the super class and the sub-class.
  - a. The most general class is at the top, with the more specific object types shown as the sub-class.
3. Generalization and specialization are helpful to describe the systems that should be implemented using inheritance in an object-oriented language.

Figure 5.17: Aggregation, generalization, and specialization



### Object-Oriented Analysis Techniques:

- OOA in the OMT approach starts with the problem statement of the real world situation expressed by the customer.
- Based on the problem statement, following three kinds of modeling are performed to produce the object-oriented analysis model:
- Object modeling*
- Dynamic modeling*
- Functional modeling*
- The object model describes the structural aspects of a system;
- The dynamic model represents the behavioral aspects; and
- The functional model covers the transformation aspects of the system.

### Process of Object Oriented modeling:

- Identifying object and classes
- Prepare a data dictionary
- Identifying associations
- Identifying attributes
- Refining with inheritance
- Grouping classes into modules

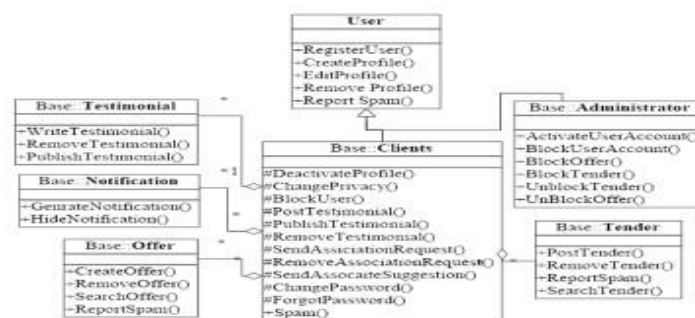


Figure 5.18: Class diagram for EtransQ system

## IV Prototyping Analysis

- a. Prototyping is **more suitable** where **requirements are not known in advance**, rapid delivery of the product is required, and the customer involvement is necessary in software development.
- b. It is an iterative approach that **begins** with the **partial development of an executable model** of the system. It is then demonstrated to the customer for the collection of feedback.
- c. The development of prototype is repeated until it satisfies all the needs and until it is considered the final system.
- d. Prototype can be developed either using automated tools, such as Visual Basic, PHP (Hypertext Preprocessor), 4GL (fourth generation languages), or paper sketching.
- e. There **are two types of prototyping approaches** widely used for elicitation, analysis, and requirement validation:
  1. Throwaway prototyping
  2. Evolutionary prototyping
- f. In **Throwaway prototyping**, a prototype is built as quickly as possible for the purpose of observing the product's viability.
- g. If the prototype is not acceptable to the customer, then it is totally discarded and the project begins from scratch.
- h. The various versions of the prototype are developed from customer requirements until the customer is satisfied.

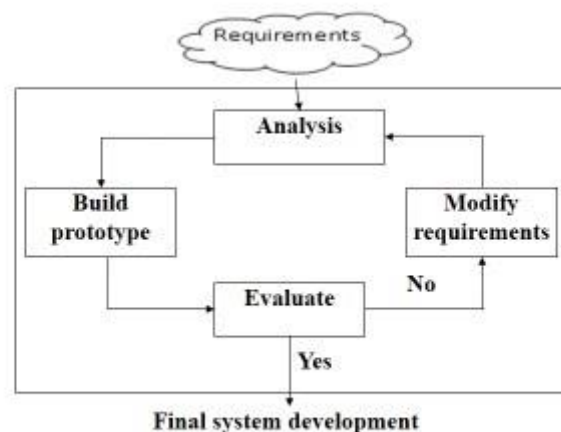
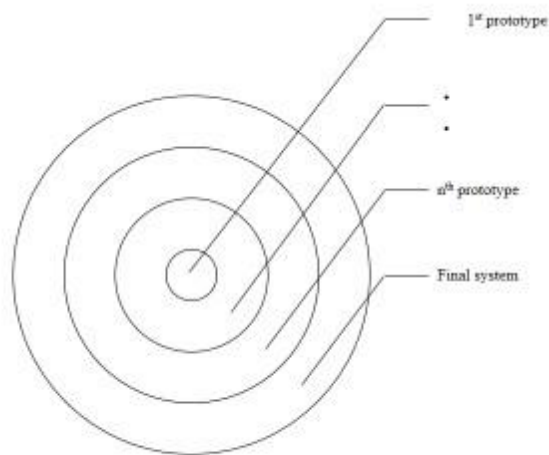


Figure 5.22: Throwaway Prototyping

- i. In **Evolutionary prototyping**, a prototype is built with the focus that the **working prototype will be considered the final system**.
- j. The process begins with the customer requirements.
- k. The **prototypes** are produced in **several iterations**.
- l. They are shown to the customers for their acceptance and customer suggestions are incorporated until the final prototype as the final product is constructed.
- m. This type of prototyping uses the rapid application development (RAD) approach

in which automated tools and CASE tools are used for prototype development.



**Figure 5.23: Evolutionary prototyping**

### III REQUIREMENTS SPECIFICATION(SRS)

1. The main focus of the problem analysis approaches is to **understand the internal behavior** of the software
2. Requirements are described in a formal document called *software requirement specification(SRS)*.
3. *Software requirement specification (SRS) document is a formal document that provides the complete description of the proposed software, i.e., what the software will do without describing how it will do so.*
4. Software requirements specification is one of the important documents required in the software development.

#### **SRS is needed for a variety of reasons:**

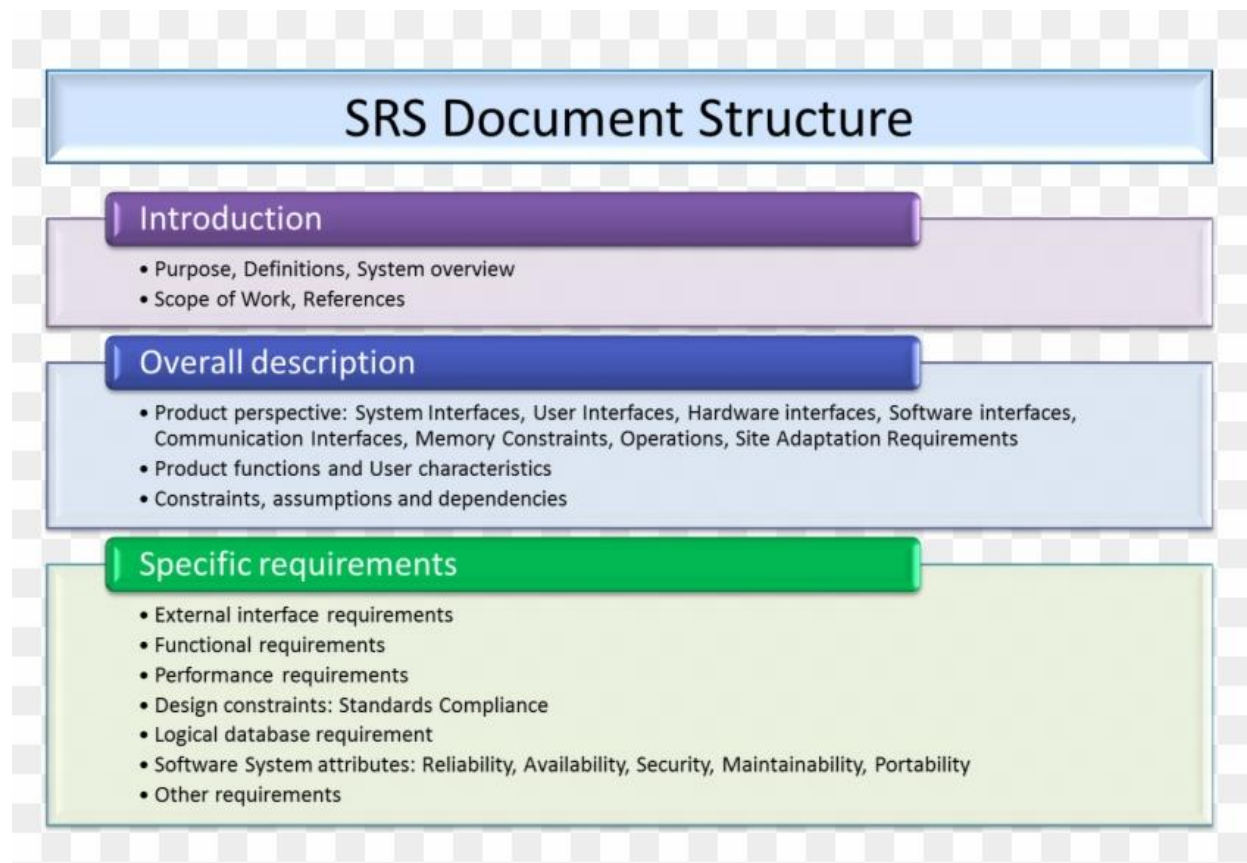
- a. Customers and users rely more on and better understand a written formal document than some technical specification.
- b. It provides basis for later stages of software development, viz., design, coding, testing, standard compliances, delivery, and maintenance.
- c. It acts as the reference document for the validation and verification of the work products and final software.
- d. It is treated as an agreement on the features incorporated in the final system of project between customer and the supplier.
- e. A good quality SRS ensures high quality software product.
- f. A high quality SRS reduces development effort (schedule, cost, and resources) because unclear requirements always lead to unsuccessful projects.

## Characteristics of the SRS

- g. **Completeness** : SRS defines precisely all the go-live situations that will be encountered and the system's capability to successfully address them.
- h. **Consistent** : SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions. For example, the only electric hedge trimmer that is safe is one that is stored in a box and not connected to any electrical cords or outlets.
- i. **Accurate** : SRS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it. This aspect of requirements is a significant problem area for many SRSs.
- j. **Modifiable** : The logical, hierarchical structure of the SRS should facilitate any necessary modifications (grouping related issues together and separating them from unrelated issues makes the SRS easier to modify).
- k. **Ranked** : Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.
- l. **Testable** : An SRS must be stated in such a manner that unambiguous assessment criteria (pass/fail or some quantitative measure) can be derived from the SRS itself.
- m. **Traceable** : Each requirement in an SRS must be uniquely identified to a source (use case, government requirement, industry standard, etc.)
- n. **Unambiguous** : SRS must contain requirements statements that can be interpreted in one way only. This is another area that creates significant problems for SRS development because of the use of natural language.
- o. **Valid** : A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it. This is one of the main reasons SRSs are written using natural language.
- p. **Verifiable** : A verifiable SRS is consistent from one level of abstraction to another. Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts. Using indicators of strength and weakness provide some evidence that preferred attributes are or are not present.

## Structure of an SRS

- The structure of the SRS describes the organization of the software requirement document.
- The best way to specify requirements is to use predefined requirements specification templates.
- The specific requirement subsection of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.
- It includes functional requirements, performance requirements, design constraints, and external interface requirements.



### Components of an SRS

- The main focus for specifying requirements is to cover all the specific levels of details that will be required for the subsequent phases of software development and these are agreed upon by the client.
- The specific aspects that the requirement document deals with are as follows:
  - *Functional requirements*
  - *Performance requirements*
  - *Design constraints (hardware and software)*
  - *External interface requirements*

### Functional Requirements

- Functional requirements describe the behavior of the system that it is supposed to have in the software product.
- They specify the functions that will accept inputs, perform processing on these inputs, and produce outputs.
- Functions should include descriptions of the validity checks on the input and output data, parameters affected by the operation and formulas, and other operations that must be used to transform the inputs into corresponding outputs.
- For example, an ATM machine should not process transaction if the input amount is greater than the available balance. Thus, each functional requirement is specified with valid/invalid inputs and outputs and their influences.

## **Performance Requirements**

- This component of the SRS specifies the performance characteristics or the nonfunctional aspects of the software system.
- The performance outcomes depend on the operation of the functional requirements.
- Performance requirements are classified as
  - Static requirements: These are fixed and they do not impose constraint on the execution of the system.
  - Dynamic requirements: specify constraints on the execution behavior of the system. These typically include system efficiency, response time, throughput, system priorities, fault recovery, integrity, and availability constraints on the system.

## **Design Constraints**

- There are certain design constraints that can be imposed by other standards, hardware limitations, and software constraints at the client's environment.
- These constraints may be related to the standards compliances and policies, hardware constraints (e.g., resource limits, operating environment, etc.), and software constraints.
- The hardware constraints limit the execution of software on which they operate.
- Software constraints impose restrictions on software operation and maintenance.

## **External Interface Requirements**

- The external interface specification covers all the interactions of the software with people, hardware, and other software.
- The working environment of the user and the interface features in the software must be specified in the SRS.
- The external interface requirement should specify the interface with other software.
- It includes the interface with the operating system and other applications.

## **Program Design Language (PDL)**

- Program Design Language (PDL) is a method of specifying procedures of functional requirements in the software.
- It represents the requirements in the form of pseudo code and structured languages. It adds more abstract constructs to increase its expressive power.
- Various special purpose requirements specification languages have been designed, such as the Problem Statement Language (PSL)/Problem Statement Analyzer (PSA) and Requirements Statement Language (RSL).
- Requirements expressed in PDL are easy to understand and translate into programming languages.
- In PDL, it becomes difficult to express the domain concepts in an understandable manner.
- Also, the specification is considered a design rather than a specification.



## REQUIREMENTS VALIDATION

- The SRS document may contain errors and unclear requirements and it may be the cause of human errors.
- The most common errors that occur in the SRS documents are omission, inconsistency, incorrect fact, and ambiguity.
- Requirement validation is an iterative process in the requirements development process that ensures that customer requirements are accurately and clearly specified in the SRS document.
- There are various methods of requirements validation, such as *requirements review*, *inspection*, *test case generation*, *reading*, and *prototyping*.

### Requirements Review

- It is a formal process of requirement validation, which is performed by a group of people from both sides, i.e. clients and developers.
- Requirements review is one of the most widely used and successful techniques to detect errors in requirements.
- Requirements review helps to address problems at early stages of software development.
- Although requirement reviews take time but they pay back by minimizing the changes and alteration in the software.

### Requirement Inspection

- It is an effective way for requirement validation that detects defects at early stages.
- Inspection is a costly and time-consuming process because large number of requirement artifacts are analyzed, searched, and sorted.
- Inspection can detect up to 90% defects [95].
- The inspection process correctly gives results for small and less complex projects.
- It is not a common practice in industry due to the cost which is associated to it.

### Test Case Generation

- The purpose of this technique is to ensure that requirements are good enough for the product and planning activities.
- It removes the defects before a project starts and during the project execution and development.
- In this technique, the product manager and the tester select and review the high priority requirements and least priority requirements are discarded in the initial specification.
- Writing test cases early may result in some rework if the requirements change, but this rework cost will be lower than finding and fixing defects in the later stages.

### Reading

- Reading is a technique of reviewing requirements in which the reader applies his knowledge to find the defects.
- There are various reading techniques, such as ad-hoc based, checklist based, etc..

- Detection of the defect depends upon the knowledge and experience of the reviewer.
- Checklist-based reading is one of the commonly used techniques in which a set of questions is given to the reviewer as a checklist.

### **Prototyping**

- Prototyping is mostly used to understand, analyze, and validate the requirements of a system.
- In requirements validation, prototyping helps to identify the errors like missing, incomplete, and incorrect requirements from the requirements document.
- Prototyping works with developing an executable model of the proposed system. This is developed in an incremental manner.
- In each increment, defects are identified and discussed with the stakeholders and corrective actions are taken accordingly.

### **REQUIREMENTS MANAGEMENT**

- During the requirements engineering process and later stages of development, requirements are always changing.
- Customer requirements are unclear even at the final stage of system development, which is one of the important causes of project failures.
- Therefore, it becomes necessary for project managers to monitor to effect any changes that may be necessary as the project work advances.
- *Requirements management is the process of systematically collecting, organizing, documenting, prioritizing, and negotiating on the requirements for a project.*
- Requirements management planning is a continuous and cross-sectional process that continues throughout the project life span.
- It is performed after development as well as during maintenance.
- The main activities of requirements management are as follows:
  - *Planning for the project requirements*
  - *Focusing on the requirements identification process*
  - *Managing the requirements changes*
  - *Controlling and tracking the changes*
  - *Agreeing on the requirements among stakeholders*
  - *Performing regular requirements reviews*
  - *Performing impact analysis for the required changes*

### **Conclusion:**

- Requirements analysis and specification is an extremely important phase of software development because it is the basis of later stages.
- Requirements ultimately reflect in the quality of the final product.
- Requirements development focuses on preparing validated requirements, which includes activities such as elicitation, analysis, specification, and validation of requirements.
- Requirements management is concerned with managing requirements that change dynamically, controlling the baseline requirements, and monitoring the commitments and consistency of requirements throughout software development.