UNIT-3
Requirement Analysis and Specification

1. Requirements Engineering
2. Fact finding Techniques
3. Introduction to Types of Requirement Modeling
4. Data Modeling Concepts- Data Objects, Data Attributes & Relationship

3.1 Requirement Engineering:

- ✓ **Requirement** analysis is also called **requirements engineering.**
- ✓ It is the process of determining user expectations for a new or modified product.
- ✓ These **requirements** must be
  - ➢ Quantifiable
  - ➢ relevant and
  - ➢ detailed.
- ✓ In software **engineering**, such **requirements** are often called functional specifications.

- The process to gather the software 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 ‘System Requirements Specification’ **(SRS)** document.

Requirement Engineering Process

Four steps of Requirement Engineering Process:

- ✓ Feasibility Study
- ✓ Requirement Gathering
- ✓ Software Requirement Specification
- ✓ Software Requirement Validation

Feasibility study

- When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

- Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

- The feasibility study is focused towards goal of the organization.
- The study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization.
- It explores technical aspects of the project and product
    - such as usability
    - maintainability
    - productivity and
    - integration ability.

- The output of the feasibility phase should be
  - a feasibility study report
    - that should contain adequate comments and
    - recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

- If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user.

- Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification (SRS)

- SRS is a document created by system analyst after the requirements are collected from various stakeholders.
- SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software 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 software development team.

SRS should have features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

- After requirement specifications are developed, the requirements mentioned in this document are validated.
- User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly.
- This results in huge increase in cost if not nipped in the bud.

Requirements can be checked against following conditions

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

3.2 Fact Finding Techniques

- Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

- There are various ways to discover requirements

Interviews

- Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:
- Structured (closed) interviews, where to gather single information is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

- A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

- A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

- Team of engineers and developers may analyze the operation for which the new system is required.

- If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

- Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

- Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product.
- It helps giving better idea of requirements.
- If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements.
- The prototype is shown to the client and the feedback is noted.
- The client feedback serves as an input for requirement gathering.

Observation

- Team of experts visits the client’s organization or workplace.
- They observe the actual working of the existing installed systems.
- They observe the workflow at client’s end and how execution problems are dealt.
- The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project.

Hence they must be clear, correct and well-defined

- A complete Software Requirement Specifications must be:
  - Clear
  - Correct
  - Consistent
  - Coherent
  - Comprehensible
  - Modifiable
  - Verifiable
  - Prioritized
  - Unambiguous
  - Traceable
  - Credible source

3.3 Introduction to Types of Requirement Modeling

The requirements model must achieve three primary objectives:

1. to describe what the customer requires,
2. to establish a basis for the creation of a software design.
3. to define a set of requirements that can be validated once the software is built.

- The analysis model bridges the gap between a system-level description that describes overall system or business functionality as it is achieved by applying software, hardware, data, human, and other system elements and a software design

The requirements modeling action results in one or more of the following types of models:

- **Scenario-based models**

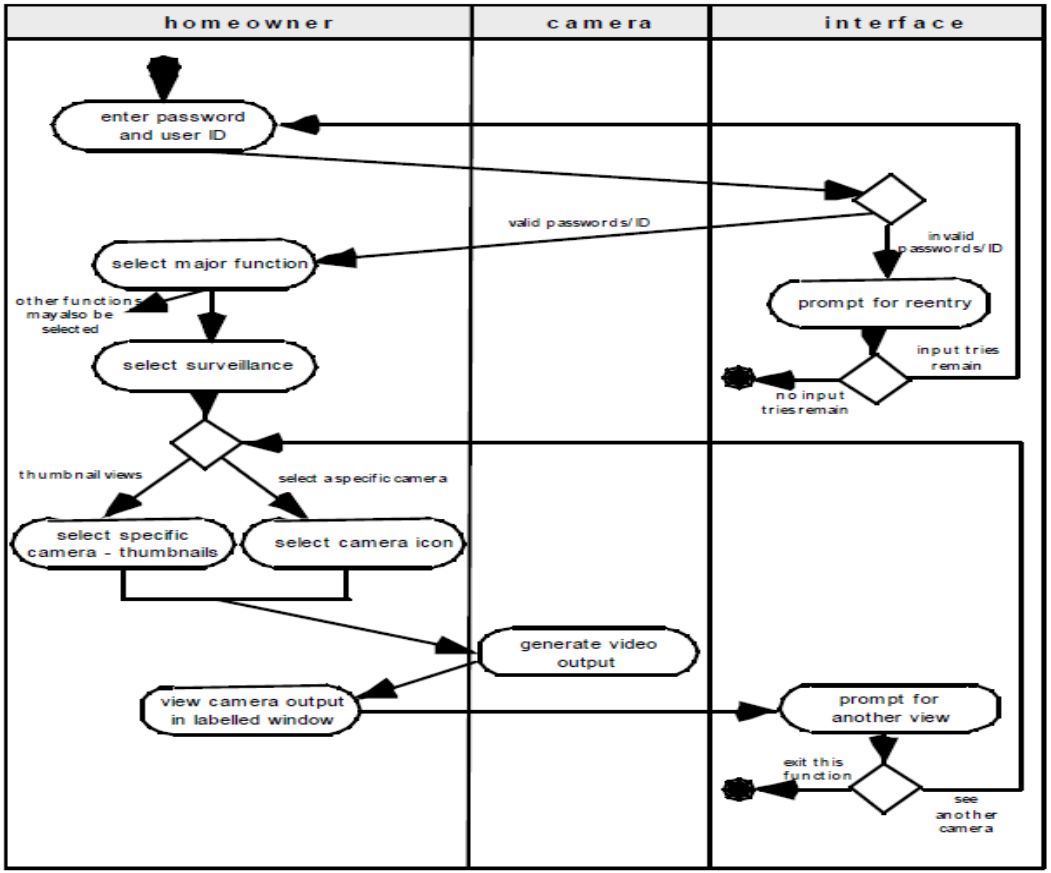
Scenario-based models

- **Scenario-based models** of requirements from the point of view of various system “actors”
- “[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases).”
- Use Cases:
- A scenario that describes a “thread of usage” for a system.
- Actors represent roles people or devices play as the system functions.
- Users can play a number of different roles for a given scenario.

- The UML swimlane diagram is a useful variation of the activity diagram and allows the modeler to represent the flow of activities described by the user-case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool

swimlane diagram



- **Data models** that depict the information domain for the problem, are described in 3.4 section
- **Class-oriented models** that represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements

Class-oriented models describes the process of developing an object-oriented analysis (OOA) model. The generic process described begins with guidelines for identifying potential analysis classes, suggestions for defining attributes and operations for those classes, and a discussion of the Class-Responsibility-Collaborator (CRC) model. The CRC card is used as the basis for developing a network of objects that comprise the object-relationship model.

Class-oriented models

- **Class-oriented models** describes the process of developing an object-oriented analysis (OOA) model.
- The generic process described begins with guidelines for identifying potential analysis classes, suggestions for defining attributes and operations for those classes, and a discussion of the Class-Responsibility-Collaborator (CRC) model.
- The CRC card is used as the basis for developing a network of objects that comprise the object-relationship model.

Identifying Analysis Classes

- Identify analysis classes by examining the problem statement
- Use a “grammatical parse” to isolate potential classes
- Identify the attributes of each class
- Identify operations that manipulate the attributes

Flow-oriented models

- **Flow-oriented models** that represent the functional elements of the system and how they transform data as it moves through the system.

- Represents how data objects are transformed as they move through the system. A data flow diagram (DFD) is the diagrammatic form that is used to complement UML diagrams. Flow-oriented modeling continues to provide a view of the system that is unique. The DFD takes an input-process-output insight into system requirements and flow. Data objects are represented by labeled arrows and transformations are represented by circles (called bubbles).

The DFD diagram

- The DFD diagram enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition of the system.

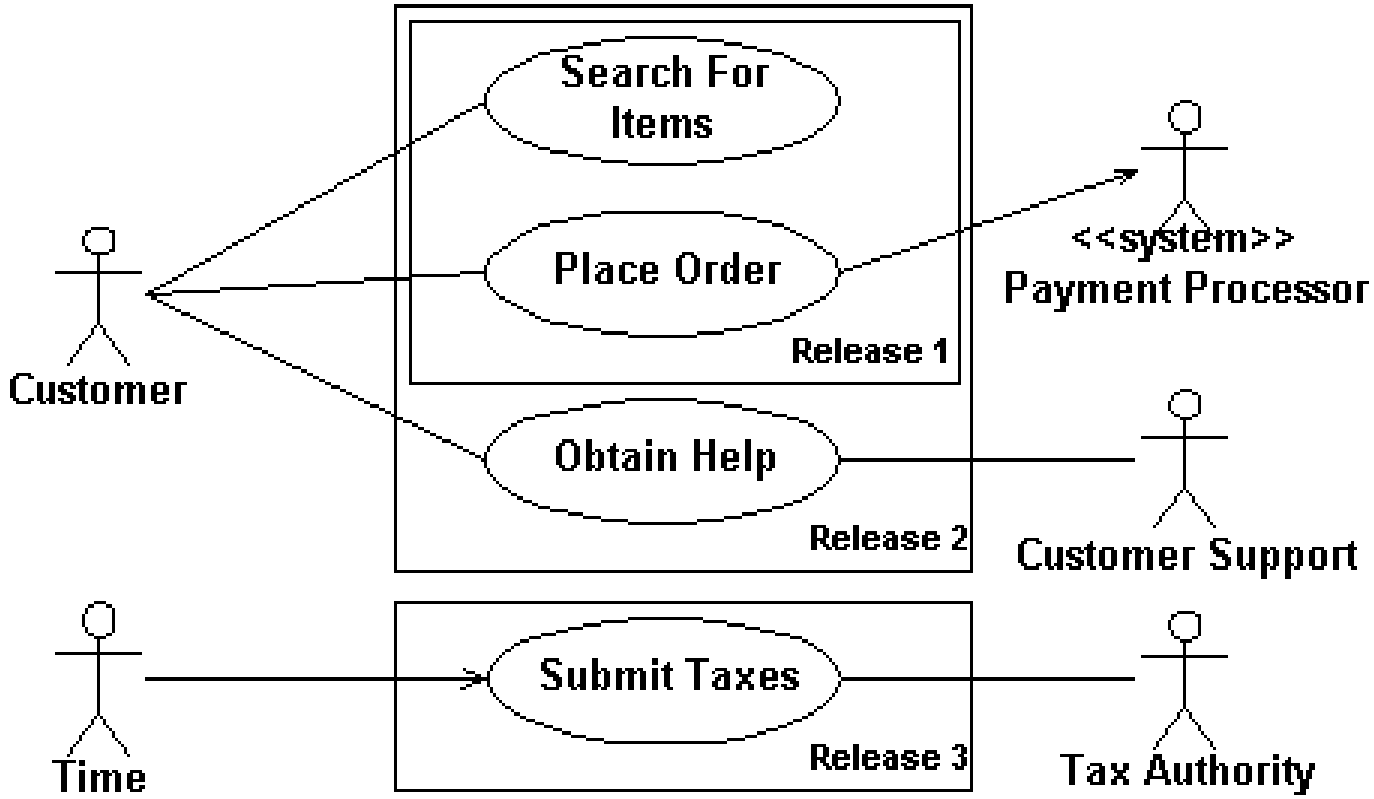
- DFD diagrams are described in chapter 4 , in 4.5 Data Flow Diagram

- **Behavioral models** that depict how the software behaves as a consequence of external “events”
- Behavioral models are used to describe the overall behavior of a system. system. Two types of behavioral model are: Two types of behavioral model are:
- Data processing models that show how data is processed as it moves through the system;
- State machine models that show the systems response to events.
- These models show different perspectives so both of them are required to required to describe the system’s behavior.
- A behavioral model shows the interactions between objects to produce some particular system behavior that is specified as a use-case. • Sequence diagrams, Activity diagrams, state diagrams, collaboration diagrams are used to model interaction between objects. • Where are people coming from? • Where are they going? • How do they move from one space to the other.

use-case diagram for online shopping

- A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.
- A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.
- The use cases are represented by either circles or ellipses.

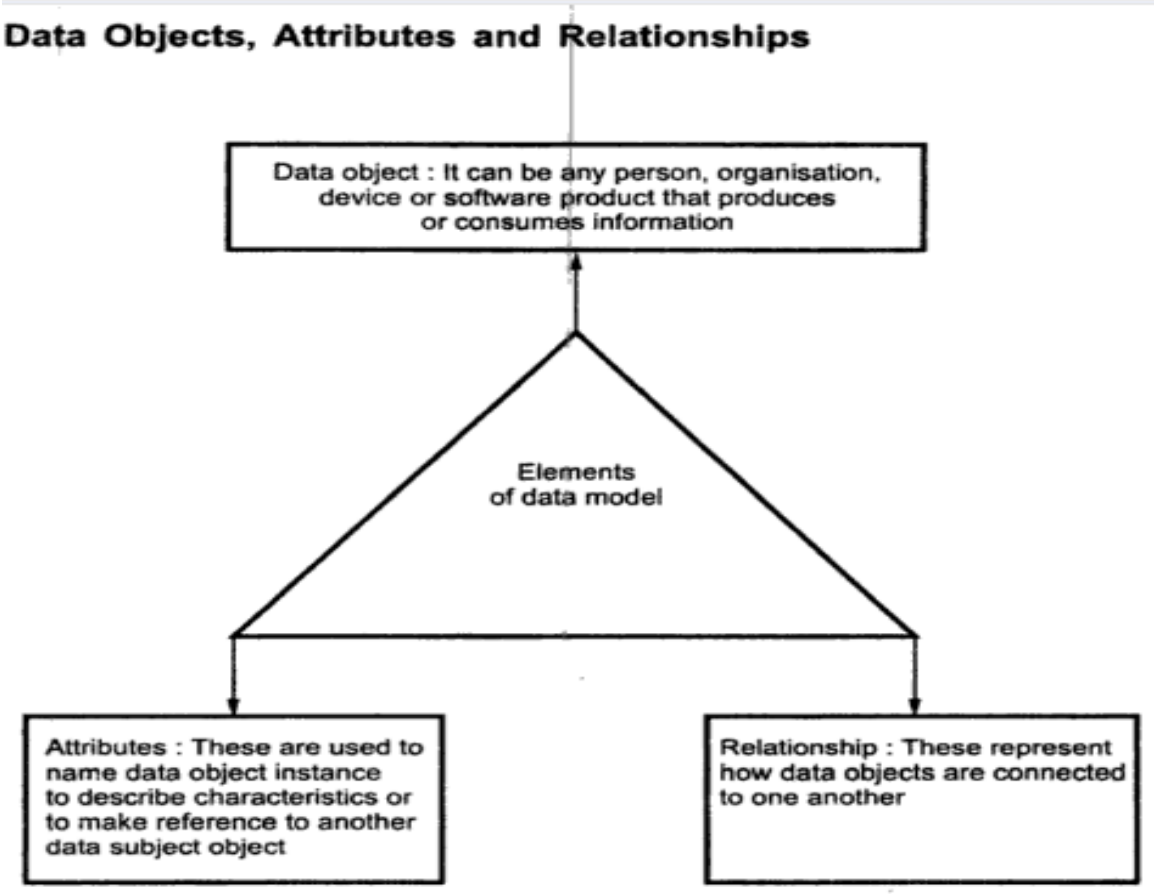
use-case diagram for online shopping



3.4 Data Modeling Concepts

- Data modeling is the analysis modeling. In data modeling the data objects are examined independently of processing.
- The data domain is focused and model is created at the customer's level of abstraction.
- The data model represents how data objects are related with one another.

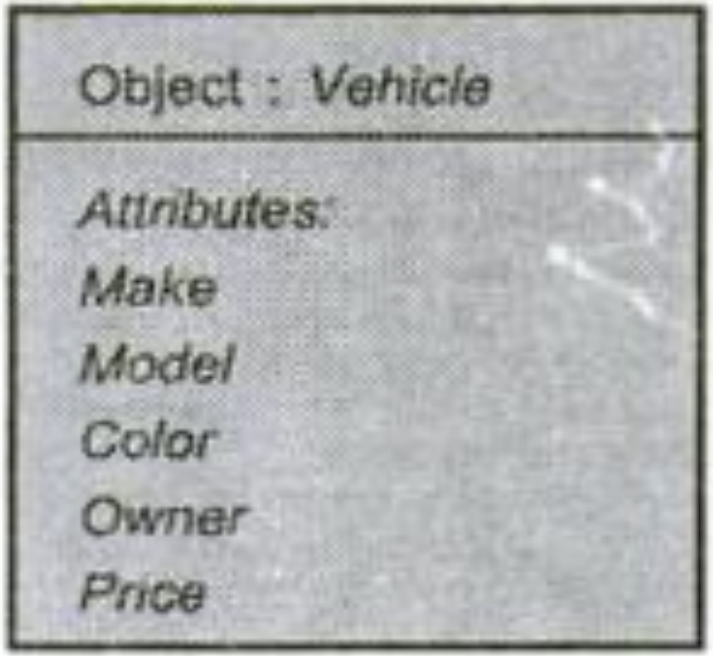
Data modeling



Data Object

- Data Object is a set of attributes(data items) that will be manipulated within the software)
- Each instance of data object can be identified with the help of unique identifier.
- Example: A student can be identified by the unique Roll number.
- The system can not perform without accessing to the instances of object.
- Each data object is described by the attributes which themselves are data items.

Data object is a collection of attributes that act as aspect, characteristics, quality or descriptor of the object.



Object Types

Typical data objects are

- External entities such as printer, user, speakers
- Things such as reports, displays, signals
- Occurrences or events such as interrupts, alarm, telephone call
- Roles such as manager, engineer, customer
- Organizational units such as division, departments
- Places such as manufacturing floor, workshops
- Structures such as student records, accounts, file

Attributes

Attributes define properties of data object

Typically there are three types of attributes –

1. **Naming attributes** – These attributes are used to name an instance of data object. For example : In a *vehicle* data object *make* and *model* are naming attributes.
2. **Descriptive attributes** – These attributes are used to describe the characteristics or features of the data object. For example : In a *vehicle* data object *color* is a descriptive attribute.
3. **Referential attribute** – These are the attributes that are used in making the reference to another instance in another table. For example : In a *vehicle* data object *owner* is a referential attribute.

Relationship

Relationship represents the connection between the data objects.For example The relationship between a shopkeeper and a toy is as shown below

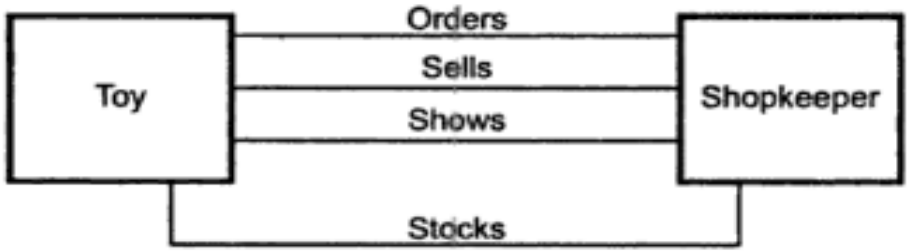


Fig. 3.11 Relationship

Here the toy and shopkeeper are two objects that share following relationships-

- Shopkeeper orders toys
- Shopkeeper sells toys
- Shopkeeper shows toys.
- Shopkeeper stocks toys.