UNIT-2

Requirement Engineering

Introduction

- The requirements of a system are the descriptions of the features or services that the system exhibits within the specified constraints.
- The requirements collected from the customer are organized in some systematic manner and presented in the formal document called *software requirements specification (SRS)* document.
- *Requirements engineering is the process of gathering, analyzing, documenting, validating, and managing requirements.*
- 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)."

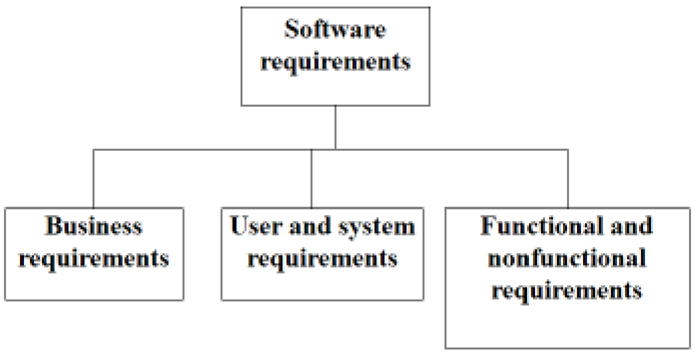

Software requirements
Business requirements
User and system requirements
Functional and nonfunctional requirements

Figure 5.1: Types of requirements

- Understanding the business rules or the processes of organization is vital to software development.

- Business requirements define the project goal and the expected business benefits for doing the project.
- 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.
- The business analyst is well versed in understanding the concept of business flow as well as the process being followed in the organization.
- The business analyst guides the client through the complex process that elicits the requirements of their business.

User Requirements

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

System Requirements

- 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.
- These are considered as a contract between the client and the development organization.
- System requirements are often expressed as documents in a structured manner using technical representations.
- 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

- Functional requirements are the behavior or functions that the system must support.
- These are the attributes that characterize what the software does to fulfill the needs of the customer.
- 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

- 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.
- 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.
- The examples of nonfunctional requirements are reliability, maintainability, performance, usability, security, scalability, capacity, availability, recoverability, serviceability, manageability, integrity, and interoperability.

Example: EtransQ

- It is a web-based information system for suppliers, transporters, and agents to share a common platform and help in gaining profit.

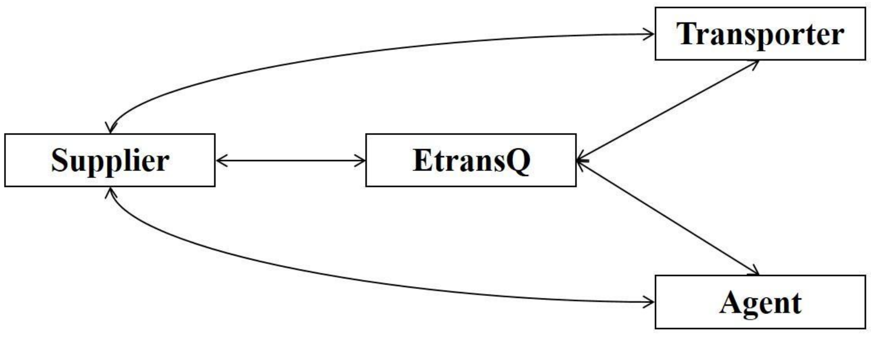


Figure 5.2: Business architecture for EtransQ

- **Business requirements for ETransQ**
- The aim of EtransQ is to shape an online information- and knowledge-based system to be a recognized, professional, innovative, and profitable service.
- The objectives of EtransQ are to
  - provide the common platform for suppliers, transporters, and agents;
  - provide an easy way to search for tenders and offers;
  - build trust and relationships between consumers and service providers;
  - provide easy and improved way of communication; and
  - provide enough options to get the best deal available in the market.

User and system requirements for ETransQ

- EtransQ system should provide a common platform for suppliers, agents, and transporters to share knowledge that will help them to grow their business.
- The system requirements in this system are:

  - The user should be able to communicate with other users connected with the system irrespective of their physical location.
  - The system should be flexible enough to provide personalized search results for every registered user.
  - The system should work in a secured manner.
  - The information available should be genuine and reliable.

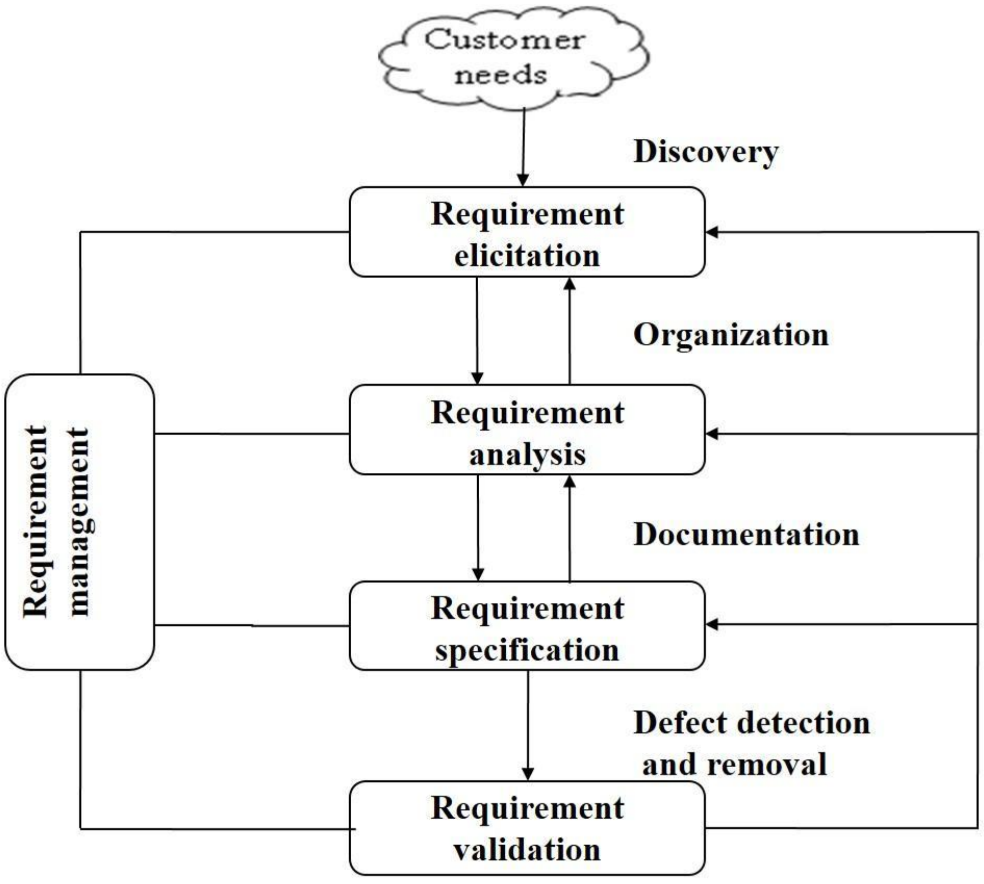
- **Functional Requirements for ETransQ**
- EtransQ system provides
  - *registration,*
  - *association,*
  - *quick search,*
  - *tendering,*
  - *offering, and*
  - *testimonial services to its user.*
  - *etc.*

Nonfunctional Requirements for ETransQ

- *Availability*
- *Recoverability*
- *Extensibility*
- *Maintainability*
- *Privacy*
- *Security*
- *Compatibility*
- *Usability*
- *Stability*
- *Reusability*
- *Robustness*
- **Requirement Engineering Process**
  - 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.

- 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.



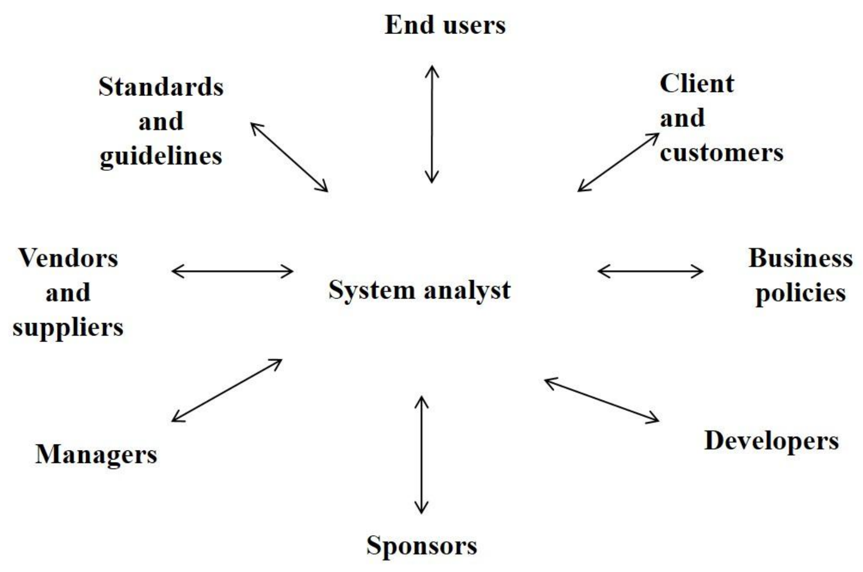
- A typical requirement engineering process has two main aspects:
  - Requirement development
  - Requirement management
  - Requirement development includes various activities, such as elicitation, analysis, specification and validation of requirements.
  - Requirement management is concerned with managing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

Requirements Elicitation

- Requirement elicitation aims to gather requirements from different perspectives to understand the customer needs in the real scenario.
- 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.
- The original requirements may be inconsistent, ambiguous, incomplete, and infeasible for the business.
- 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
  - identification of problem scope,
  - identification of stakeholders,
  - understanding of problem, and
  - 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
  - *interviewing,*
  - *questionnaires,*
  - *joint applications development (JAD),*
  - *onsite observation,*
  - *prototyping,*
  - *viewpoints,*
  - *and review records*
- Interviewing
  - Interview involves eliciting requirements through face to face interaction with clients, end-users, domain experts, or other stakeholders associated with the project.
  - They are conducted to gather facts, verify and clarify facts, identify requirements, and determine ideas and opinions.
  - There are two types of interviews: unstructured interviews and structured interviews.
  - 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.

  - A structured interview is conducted to elicit specific requirements and therefore the stakeholders are asked to answer specific questions.

- Questionnaires
  - Questionnaires are used to collect and record large amount of qualitative as well as quantitative data from a number of people.
  - The system analyst prepares a series of questions in a logical manner, supplemented by multiple choice answers and instructions for filling the questionnaires.
  - It is a quick method of data collection and inexpensive as compared to interviews.
  - It provides standardized and uniform responses from people as questions are designed objectively.
  - However, it is very difficult to design logical and unambiguous questions.
  - A very few people take interest in responding and carefully filling questionnaires.
- Joint Applications Development (JAD)
  - 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.
  - The JAD session has predefined agenda and purpose.
  - It includes various participants with their well-defined roles, such as facilitator, sponsors, end users, managers, IT staff, designers, etc.
  - The JRD meeting is conducted to identify and understand problems, resolve conflicts, generate ideas, and discuss some alternatives and the best solutions.
  - Brainstorming and focus group techniques are used for JRD.
- Onsite observation
  - 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.
  - It helps the system analyst to gain insight into the working of the system instead documentation.
  - Through constant observation, the system analyst identifies critical requirements and their influence in the system development.
  - Users may feel uncomfortable if someone observes their work.
- Prototyping
  - 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.

  - 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.
  - Prototype development is a repetitive process that continues until the product meets the business needs or requirements.
  - In this way, prototype is corrected, enhanced, or refined to reflect the new requirements.
  - Prototyping can be costly if it is repeated many times
- Viewpoints
  - 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.
  - Stakeholders may be classified under different viewpoints:
    - direct (who directly interact with the system),
    - indirect (who indirectly interact with the system),
    - domain (represent the domain standards and constraints) viewpoints.
  - The collected viewpoints are classified and evaluated under the system's circumstances and finally, it is integrated into the normal requirement engineering process.
- Review records
  - Reviewing the existing documents is a good way of gathering requirements.
  - The information related to the system is found in the documents such as manual of the working process, newspapers, magazines, journals, etc.
  - The existing forms, reports, and procedures help to understand the guidelines of a process to identify the business rules, discrepancies, and redundancies.
  - This method is beneficial to new employees or consultants working on the project.

Requirements Analysis

- 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.
- It is performed in several iterations with elicitation to clarify requirements, resolve conflicts, and ensure their correctness and completeness.
- It includes various activities, such as classification, organization, prioritization, negotiation, and modeling requirements.
- The draft requirements are analyzed to verify their correctness and completeness
- During requirement analysis, models are prepared to analyze the requirements.

- The following analysis techniques are generally used for the modeling of requirements:
  - *Structured analysis*
  - *Data-oriented analysis*
  - *Object-oriented analysis*
  - *Prototyping*

Structured Analysis

- Structured analysis is also referred to as *process modeling* or *data flow modeling*.
- It focuses on transforming the documented requirements in the form of processes of the system.
- 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.
- 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.

Structured analysis uses a graphical tool called data flow diagrams (DFD), which represent the system behavior

Data Flow Diagram (DFD)

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

A DFD has four different symbols:

- *Process:* A process is represented by a *circle* and it denotes transformations of the input data to produce the output data.
- *Data flow:* represent the movement of data, i.e., leaving one process and entering into another process. Data flows are represented by *arrows*, connecting one data transformation to another.
- *Data store:* Data store is the data at rest. It is represented in *parallel lines*.
- *Actor:* It is the external entity that represents the source or sink (destination of data). It is represented by a *rectangle*.

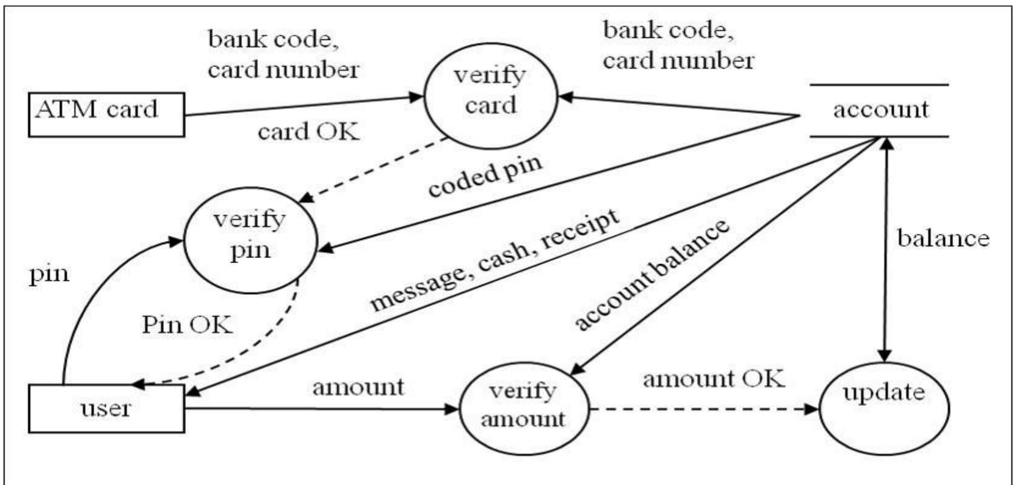


Figure 5.5: DFD for Cash withdrawal from ATM

Constructing DFD

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

Conventions in constructing DFD

- Data flow diagrams at each level must be numbered for reference purposes, for example, level 0, level 1 etc.
- Multiple data flow can be shown on single data flow line. A bidirectional arrow can be used as input and outflow (if same data is used) data or separate line can be used for input and output.
- External agents/actors and data flow are represented using nouns; for example, stock, pin, university, transporter, etc.
- Processes should be represented with verbs followed by nouns. Longer names must be connected with underscores ("_") and these should be short but meaningful, e.g. sales_detail.
- Avoid representation of control logics in the DFD.
- Each process and data store must have at least one incoming data flow into it and one outgoing data flow leaving it.
- There should not be any black hole in the system.

DFD Vs. Flowcharts

- DFDs represent the flow of data through the system while flowcharts represent the flow of control in a program.
- DFDs do not have branching and iteration of data whereas flowcharts have conditional and repetitive representation of processes.
- A flowchart describes the sequence of processes in a program whereas a DFD does not show the procedural details.
- Flowcharts represent the flow of control through an algorithm.
- Flowcharts do not show the input, output, and storage of data in the system.
- The DFD can be used to show the parallel processes executing in the system while flowcharts have only one process active at a time.

Data Dictionary

- Data flows are used by the programmers in designing data structures and also it is used by testers to design test cases.
- Such data structures may be primitive or composite.
- Composite data structures may consist of several primitive data structures.
- Also, longer composite data structures are difficult to write on the line of data flow in DFD.
- Therefore, data flow and data structures are described in the data dictionary.
- *Data dictionary* is metadata that describe composite data structures defined in the DFD.
- Data dictionary is written using special symbols, such as “+” for composition, “*” for repetition, and “|” for selection.
- The combinations of repeated data are written using “* +” symbol.
- For example, a data dictionary for a restaurant bill is as follows:

  restaurant_bill = dish_price + sales_tax + service_charge + seat_charge + grand_total

  *dish_price = [dish_name + quantity + price]**

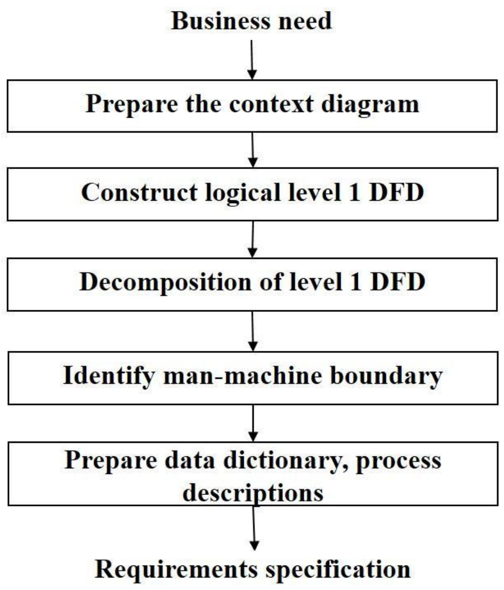
  seat_charge = [reserved_table | common_table]

Structured Analysis Method

- Structured analysis is a systematic approach for requirement analysis that employs techniques and graphical tools to represent the scenario of the working system in an organization.
- Structured analysis uses data flow diagrams and data dictionary for requirement analysis.
- This approach begins with studying the existing system
  - to understand the working of the system
  - to design the context diagram,

– apply functional decomposition to subdivide the functional requirements,

– mark the boundary of the system to know what can be automated or what will be manual, and prepare the dictionary of the system.



Business need
Prepare the context diagram
Construct logical level 1 DFD
Decomposition of level 1 DFD
Identify man-machine boundary
Prepare data dictionary, process descriptions
Requirements specification

Figure 5.6: Structured analysis approach

1. Prepare Context Diagram

    1. 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. .
    2. The entire system is treated as a single process called *bubble,* with all its external entities.
    3. That is, the system is represented with main input and output data, process, and external entities.
    4. A *Physical DFD* is the representation of the real physical environment of a non-automated system.
    5. *Logical DFD* describes logical rather than physical entities. Processes might be implemented as programs, data might be considered as file or database, etc.

Context Diagram Example

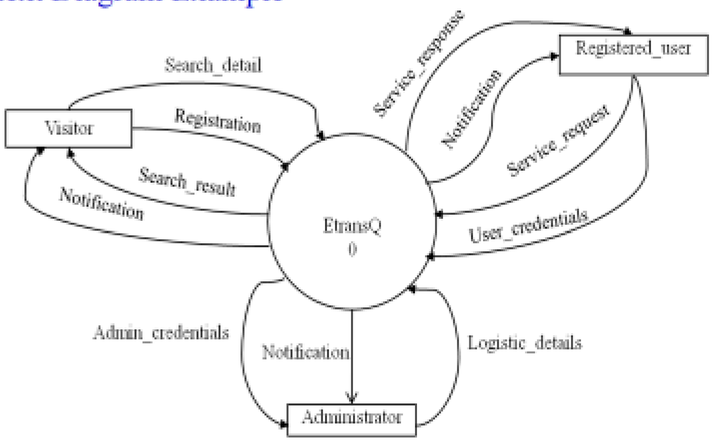


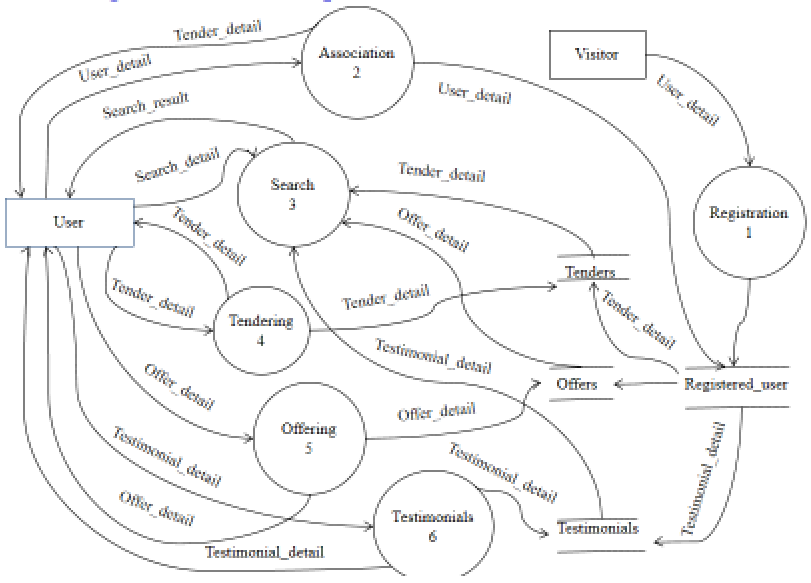
Figure 5.7: Context diagram (level 0 DFD) for EtransQ

2. Construct Level 1 Logical DFD

- Level 1 logical DFD includes the main functions supported by the proposed system.
- The equivalent logical DFD for the existing physical DFD of the system is designed with additional services required by the customer.
- The system will provide equivalent resources to the customer as these are in the non-automated or existing system.
- Sometimes, common functional requirements are merged together to represent the abstract function.

Structured Analysis Method

Level 1 Logical DFD Example



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, data stores, 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.

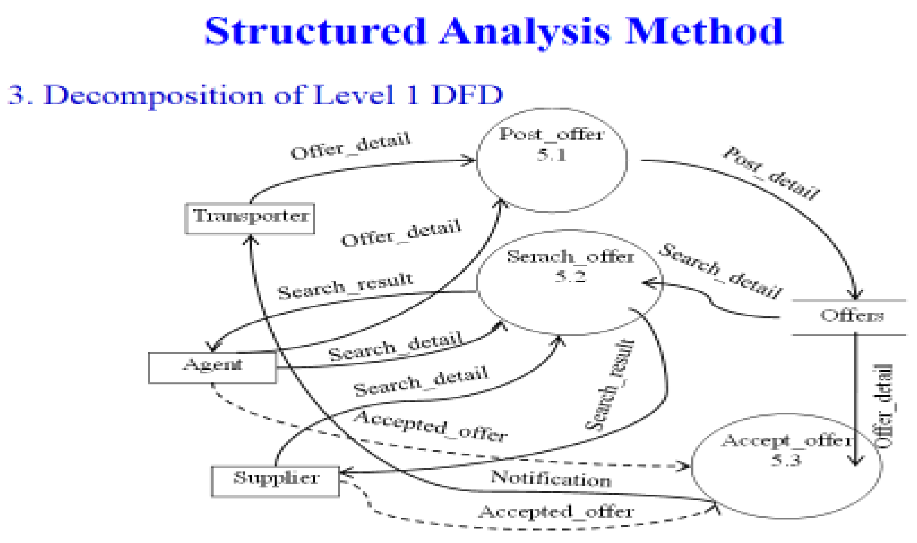


Figure 5.9: Level 2 DFD for *Offers* in EtransQ

Structured Analysis Method

3. Decomposition of Level 1 DFD

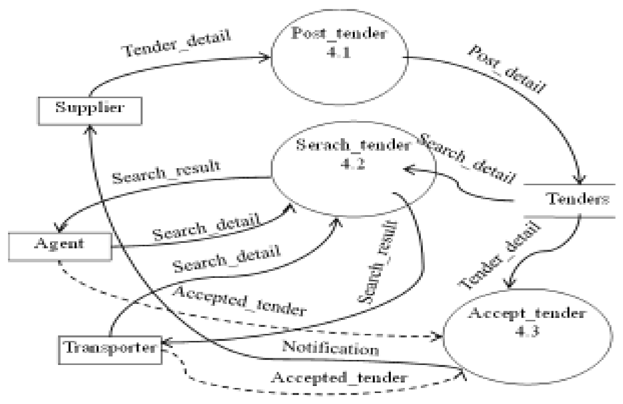


Figure 5.10: Level 2 DFD for *Tenders* in EtransQ

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.
    - In the EtransQ system, user will access information, add information, make association, select operational details, and perform the verification process.

5. Prepare Data Dictionary and Process Descriptions

- The data flows and processes are defined in the data dictionary with proper structures and formats.
- Data dictionary and process for the EtransQ system are shown in Figure 5.11.

Data dictionary for the EtransQ system

User_credential = Username + Password

Offer_detail = Offer_id + Offer_owner + Cost + Offer_date + Branch_id + Summary

Tender_detail = Tender_id + Start_date + End_date + Cost + City

User_detail = Name + Password + Organization_name + Address + [Office_number + Mobile_number + Fax_number]*

Search_detail = [Offer_detail | Tender_details | User_detail]*

Registration_detail = Name + Password + Organization_name + Address + [Office_number + Mobile_number + Fax_number]*

Search_result = [Offer_detail | Tender_detail | User_detail]*

Registered_user = User_id + Password

User = [Supplier | Transporter | Agent | Visitor | Administrator | Client]

Admin_credential = Admin _name + Password

Logistic_detail = [News | Event]*

Testimonial_detail = Testimonial_id + Testimonial_for + Testimonial_by + Publish_status + Request_date

- Benefits of Structured Analysis
  - A data-flow-based structured analysis is easy to present the customer problems in a pictorial form that can easily be converted into structured design.
  - The top-down decomposition approach enables producing a model in an organized manner.
  - The DFD-based approachcan be used to support other methodologies.
  - It does not require much technical expertise
  - It helps to understand the system scope and its boundaries.
  - It provides proper communication of system knowledge to the users.
- Limitations of Structured Analysis
  - It is difficult to understand the final DFD and also it does not reveal the sequence in which processes are performed in the system.
  - The structured analysis methodology sometimes becomes a time-consuming and tedious job to manage such voluminous data and to produce several levels of DFD.
  - Although a step-by-step approach is suitable for the waterfall model but the system requirements and user requirements must be frozen at the early in the life cycle.
  - A complete DFD is constructed if all the requirements are available at the beginning of the structured analysis.
-

Data-Oriented analysis

- Data-oriented analysis is also referred to as *data-oriented modeling,* which aims at conceptual representation of the business requirements.
- A data model is the abstraction of the data structures required by a database rather than the operations on those data structures.
- A data model is independent of hardware or software constraints that are used to implement the database.
- Without analyzing data models, it is not possible to design database.
- Data models ensure that all data objects required by the database are completely and accurately represented.
- Data models are composed of data entities, associations among different entities, and the rules which govern operations on the data.
- Data models are accompanied by functional models that describe how the data will be processed in the system.
- Thus, producing data models and functional models together is called *conceptual database design.*
- Data-oriented analysis is performed using entity relationship modeling (ERM).

Entity Relationship Modeling (ERM)

- Entity relationship modeling (ERM) is a pictorial method of data representation.
- ERM is represented by the *E-R diagram* that represents data and organizes them in such a graphical manner that helps to design the final database.
- An entity or entity class is analogous to the class in object orientation, which represents a collection of similar objects.
- An entity may represent a group of people, places, things, events, or concepts.
- For example, student, course, university, fees, etc., represent entities.
- An *entity instance* is the single instance of an entity class.
- *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.

Entity and Attributes Examples

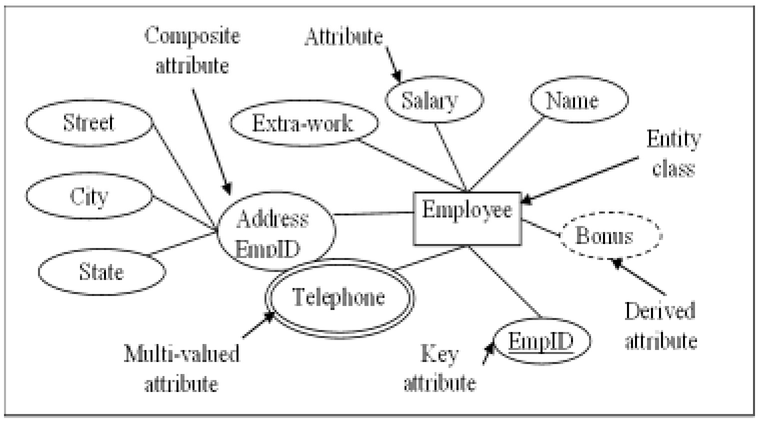


Figure 5.12: Entity and attributes

ERM 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
- An n-ary relationship involves many entities in it.

ERM Cardinality

- *Cardinality* defines the number of occurrences of entities which are related to each other.

- It can be one-to-one (1:1), one-to-many (1:M), or many-to-many (N:M).
- *Direction* of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is called the parent entity and the entity where the relationship terminates is called the child entity.
- Figure 5.13 (a), (b), and (c) illustrate recursive, binary, and turnery relationships with cardinalities.

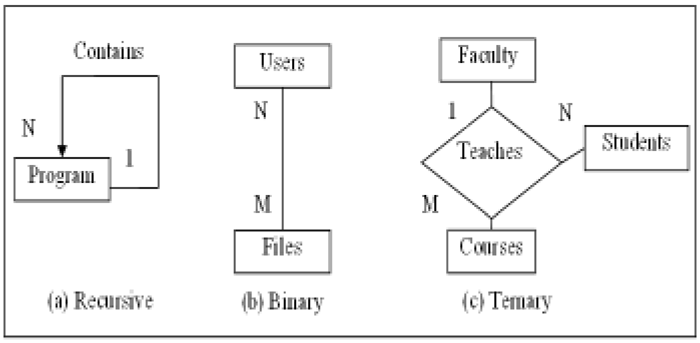


Figure 5.13: Relationships with cardinalities

ERM: Aggregation, Generalization, and Specialization

- *Participation* denotes whether the existence of an entity instance is dependent upon the existence of another related entity instance.
- *Aggregation* is an abstraction of entities that represents the “part-of” or “part whole” relationship between entities.
- *Specialization* represents the “is-a” relationship. It designates entities in two levels, viz., a high-level entity set and low-level entity set.
- *Generalization* is the relationship between an entity and one or more refined sets of it. It combines entity sets that share the common features into a higher-level entity set.

ERM: Aggregation, Generalization, and Specialization

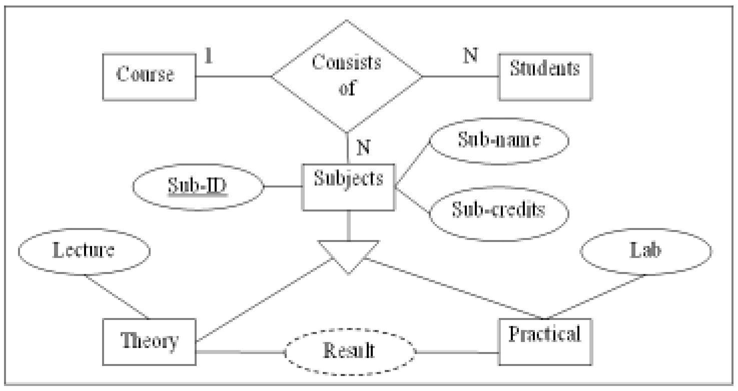


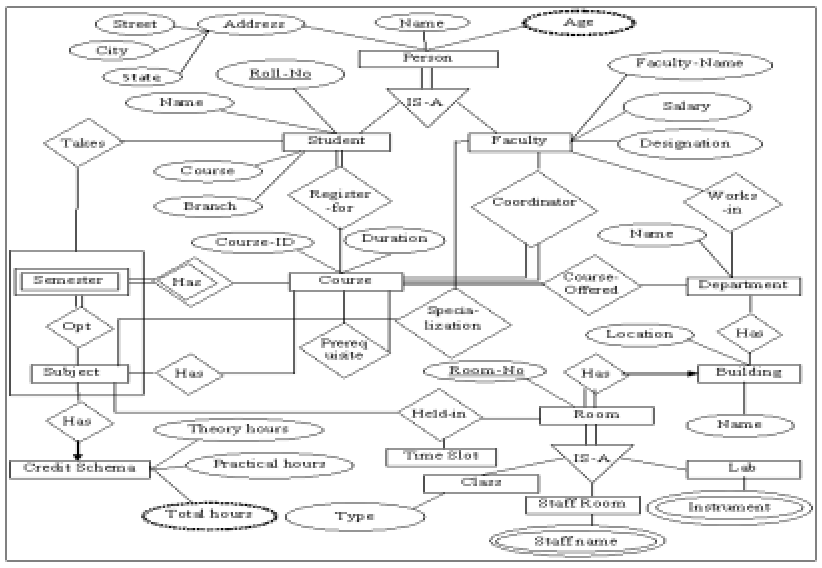
Figure 5.14: Aggregation, generalization, and specialization

Data-Oriented analysis Method

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. *Apply hierarchical relation*
6. *Perform normalization*
7. *Adding integrity rules to the model*

Data Oriented Model

Figure 5.15: Data-oriented model for course registration system in a department



Object-Oriented Analysis

- Object-oriented approach also combines both data and processes into single entities called objects.
- The object-oriented approach has two aspects, object-oriented analysis (OOA) and object-oriented design (OOD).
- 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.
- 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.
- There exist various object-oriented approaches for OOA and OOD, e.g. object modeling technique (OMT) .
- Objects and classes
- Objects are the real world entities that can uniquely be identified and distinguished from other objects.
- Each object has certain attributes (state) and operations (behavior).
- Similar objects are grouped together to form a class.
- An instance or individual object has identity and can be distinguished from other objects in a class.
- *Attributes* are the data values of an object.
- *Operations* are the services or functions of an object in a class.
- An object communicates to other objects through message passing or signatures.
- 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
- Objects are shown through *object diagrams*, which are also represented with rounded rectangles.
- For example, the class diagram for an employee is shown in Figure 5.16.

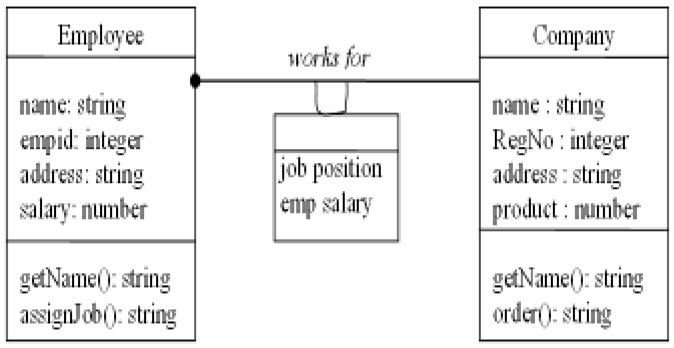


Figure 5.16: Class diagram and association of employee in a company

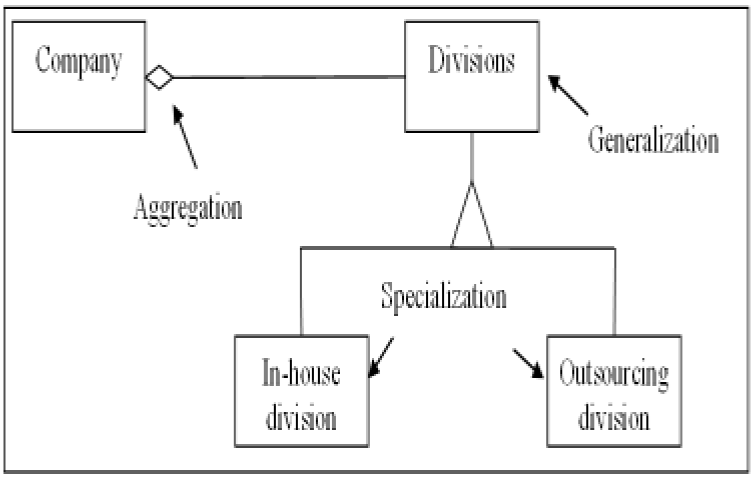
Association

- Relationship between classes is known as *association*.
- Association can be binary, ternary, or n-ary.
- *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.
- An association can also have attributes and they are represented by a box connected to the association with a loop.
- *Role names* are also attached at the end of the association line.
- Sometimes, a *qualifier* can be attached to the association at the many side of the association line.

Aggregation, Generalization, and Specialization

- An *aggregation* models a “whole-part” relationship between objects. In aggregation, a single object is treated as a collection of objects.
- *Generalization* represents the relationship between the super class and the sub-class.
- The most general class is at the top, with the more specific object types shown as the sub-class.
- 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 Method

- 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.

Object Modeling

- Object modeling begins with the problem statement of the application domain.
- The problem statement describes the services that will be performed by the system.
- The object model captures the static structure of object in the system.
- It describes the identity, attributes, relationships to other objects, and the operations performed on the objects.
- Object models are constructed using class diagrams after the analysis of the application domain.

Object modeling

1. *Identifying object and classes*
2. *Prepare a data dictionary*
3. *Identifying associations*
4. *Identifying attributes*
5. *Refining with inheritance*
6. *Grouping classes into modules*

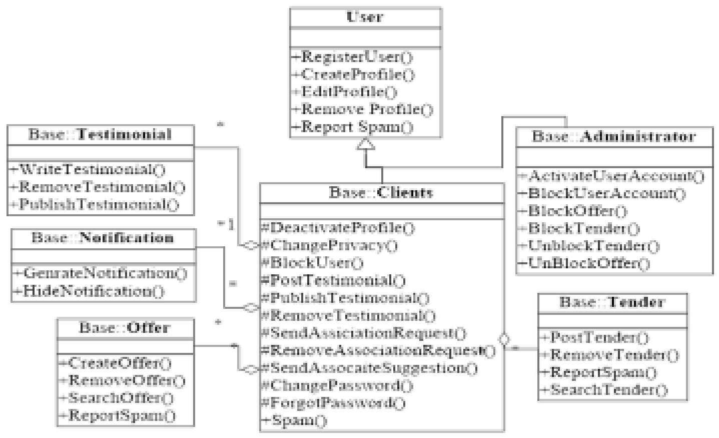


Figure 5.18: Class diagram for EtransQ system

Dynamic Modeling

- Once the structure of an object is found, its dynamic behavior and relationships over time in the system are modeled in a dynamic model.
- During dynamic modeling, *state diagrams* are modeled, which consist of states and transitions caused by events.
- The state diagram is constructed after analyzing the system behavior using the event sequence diagram.
- An *event sequence diagram* is composed of participating objects drawn as vertical lines and events passing from one object to another drawn as horizontal lines between the object lines.
- For example, the sequence diagram for the EtransQ system is shown in Figure 5.19 and Figure 5.20 for the offer and tender services, respectively.
- The respective data flow diagrams are shown in Figure 5.9 and Figure 5.10 in a previous sub-section of this chapter.
- The state diagram for the EtransQ system is shown in Figure 5.21.

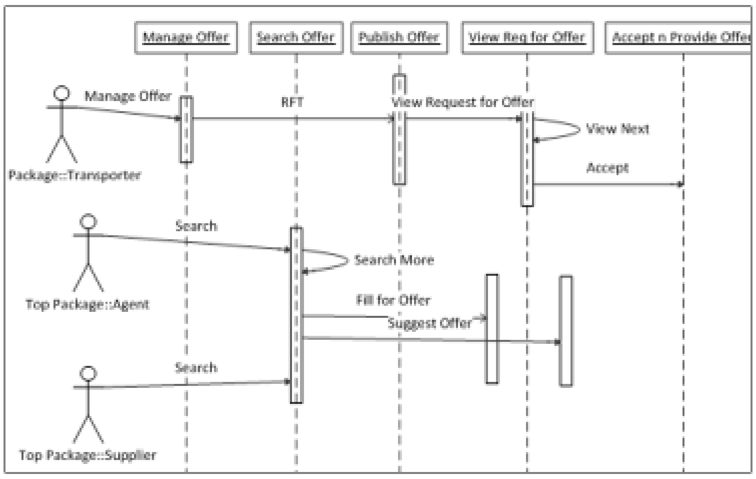


Figure 5.19: Sequence diagram for offer service in the EtransQ

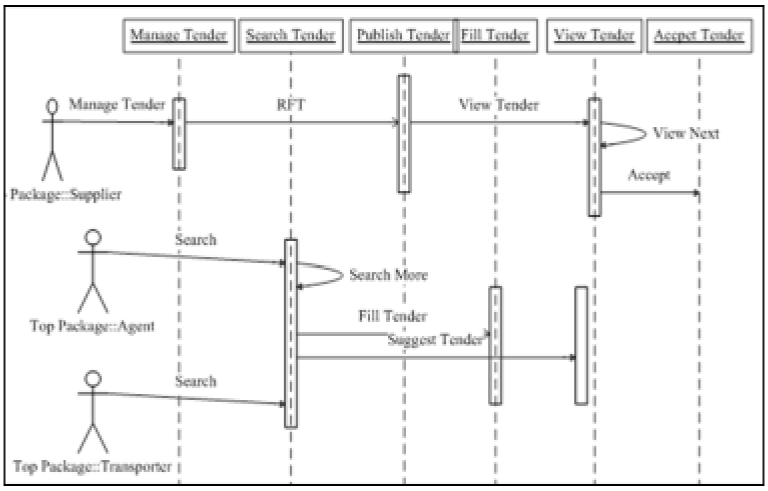


Figure 5.20: Sequence diagram for the tender service in the EtransQ

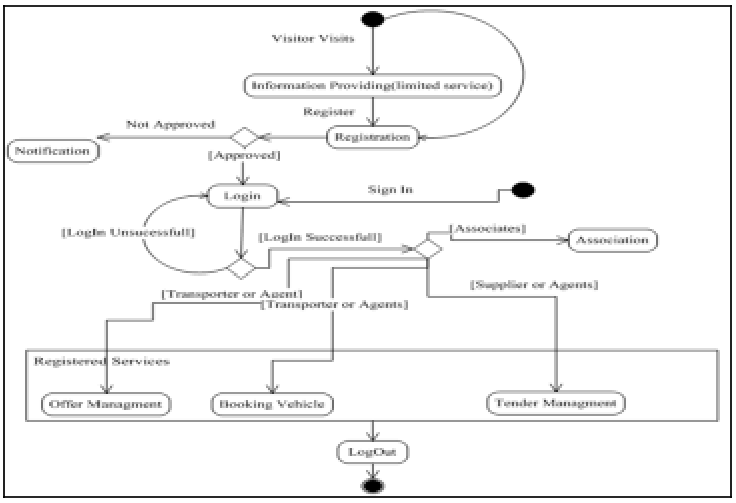


Figure 5.21: State chart diagram for the EtransQ system

Functional Modeling

- The functional model describes what are the actions performed without specifying how or when they are performed.
- A functional model involves inputs, transformations, and outcomes. Outputs are generated on some input after applying certain transformations to it.
- The functional model is represented through data flow diagrams (DFD).

Prototyping Analysis

- 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.
- 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.
- The development of prototype is repeated until it satisfies all the needs and until it is considered the final system.
- Prototype can be developed either using automated tools, such as Visual Basic, PHP (Hypertext Preprocessor***)***, 4GL (fourth generation languages), or paper sketching.
- There are two types of prototyping approaches widely used for elicitation, analysis, and requirement validation:
    - *Throwaway prototyping*
    - *Evolutionary prototyping*
- In *throwaway prototyping*, a prototype is built as quickly as possible for the purpose of observing the product's viability.
- If the prototype is not acceptable to the customer, then it is totally discarded and the project begins from scratch.

- The various versions of the prototype are developed from customer requirements until the customer is satisfied.

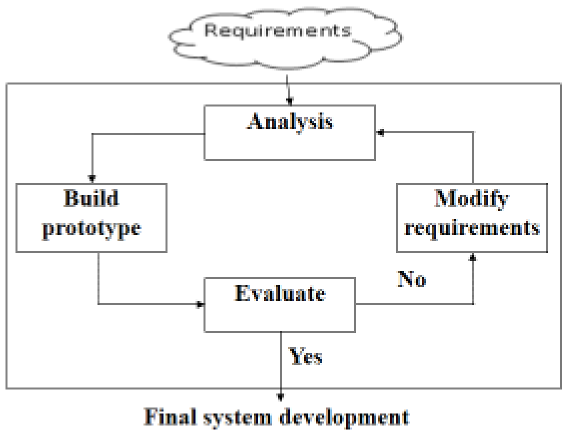


Figure 5.22: Throwaway Prototyping

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

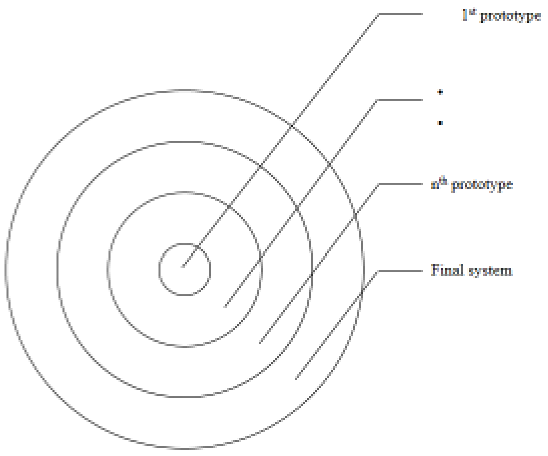


Figure 5.23: Evolutionary prototyping

- **Requirements Specification**

- The main focus of the problem analysis approaches is to understand the internal behavior of the software
- Requirements are described in a formal document called *software requirement specification (SRS)*.
- *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.*
- Software requirements specification is one of the important documents required in the software development.

SRS

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

Characteristics of the SRS

- *Correctness*
- *Unambiguity*
- *Completeness*
- *Consistency*
- *Should be ranked for importance and/or stability*
- *Verifiability*
- *Modifiability*
- *Testability*
- *Validity*
- *Traceability*
- **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.

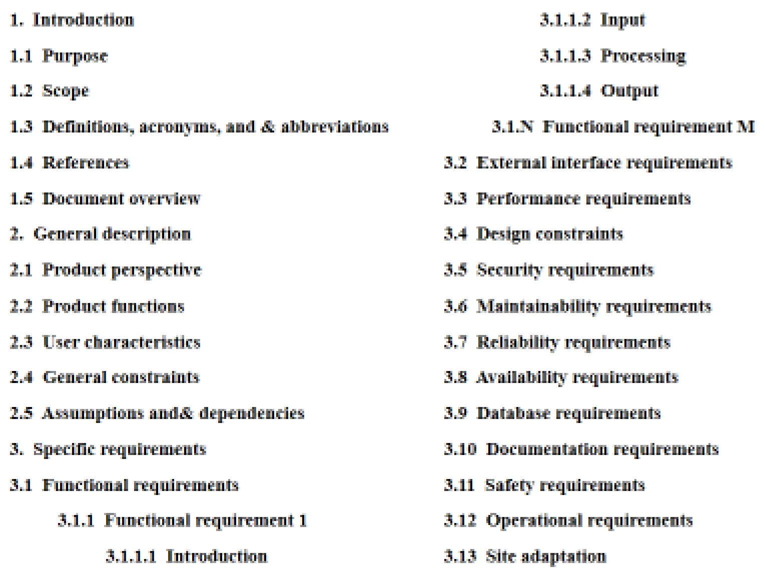
External Interface Requirements

- 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.

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.

IEEE structure of SRS



1. Introduction
1.1 Purpose
1.2 Scope
1.3 Definitions, acronyms, and & abbreviations
1.4 References
1.5 Document overview
2. General 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 Functional requirements
    3.1.1 Functional requirement 1
        3.1.1.1 Introduction
        3.1.1.2 Input
        3.1.1.3 Processing
        3.1.1.4 Output
    3.1.N Functional requirement M
3.2 External interface requirements
3.3 Performance requirements
3.4 Design constraints
3.5 Security requirements
3.6 Maintainability requirements
3.7 Reliability requirements
3.8 Availability requirements
3.9 Database requirements
3.10 Documentation requirements
3.11 Safety requirements
3.12 Operational requirements
3.13 Site adaptation

Requirements Specification Methods

- Requirements specified in natural languages are ambiguous and confusing since they may be interpreted differently by different people.
- There are some complex and difficult statements in the user requirements, which need to be specified using technical tools or other formal methods.
- Most commonly used methods are:

– *Decision tree and tables,*

– *Regular expressions,*

– *Program Design Language (PDL),*

– *Graphical methods,*

– *Mathematical methods.*

Decision Tree and Decision Table

- A decision tree is a tree-like representation of a complex logic.
- In a decision tree, edges are labeled by conditions and the leaf nodes are represented with actions.
- Decision tables are made just like tables with rows and columns.
- Decision tables are heavily used for representing the decision logic.
- There are certain advantages of using decision tables over decision trees and vice versa.

Example 5.2: Student promotion system (SPS)

- A student grading system promotes the students to the next session on the basis of the university rules.
- A student has to complete subjects of 52 credits in a session and maintain 4.00 SGPA to take admission in the next session.
- Otherwise, a fresh admission will be allowed in the same course. Represent this information in the decision table and decision tree.

Figure 5.25: Decision tree for Student Promotion System

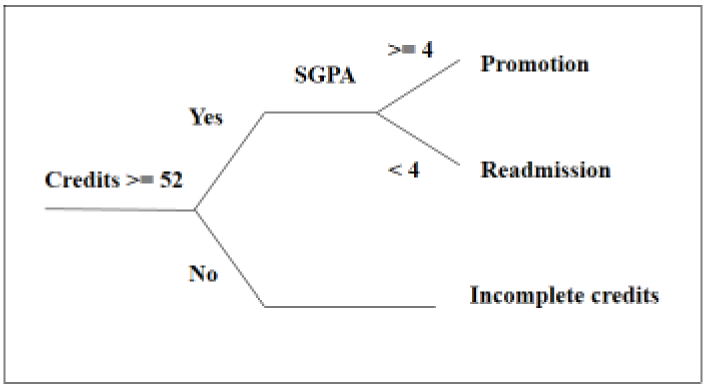
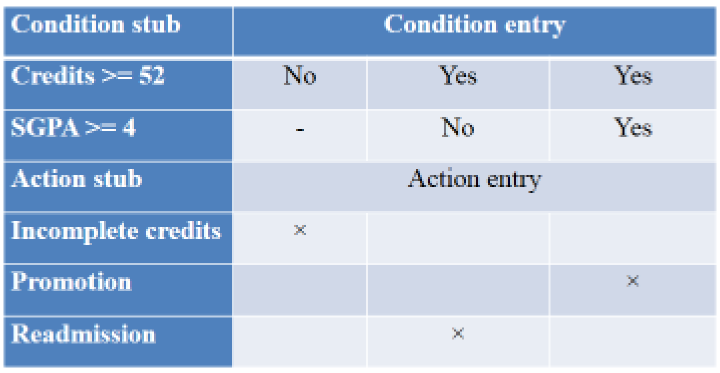


Table 5.1: Decision table for student promotion system



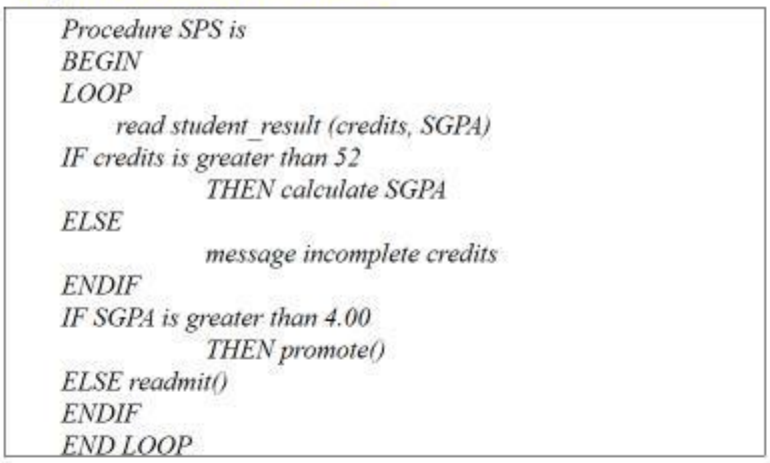
Condition stub	Condition entry		
Credits >= 52	No	Yes	Yes
SGPA >= 4	-	No	Yes
Action stub	Action entry		
Incomplete credits	×		
Promotion			×
Readmission		×	

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.

PDL for SPS

Figure 5.26: A PDL for SPS



```
Procedure SPS is
BEGIN
LOOP
      read student_result (credits, SGPA)
IF credits is greater than 52
                  THEN calculate SGPA
ELSE
                  message incomplete credits
ENDIF
IF SGPA is greater than 4.00
                  THEN promote()
ELSE readmit()
ENDIF
END LOOP
```

Regular Expressions

- A regular expression is a pattern that describes the syntactical structure of a set of strings.
- Most of the software involves the processing of strings. The common patterns of requirements are described by regular expressions.
- Regular expressions are used for describing patterns of requirements, searching a substring within a string, for lexical analysis, describing metadata, etc.
- The building blocks for regular expressions are the single characters or digits that match themselves.
- If r_1 and r_2 are regular expressions of languages L (r_1) and L (r_2), then
  - (r_1) is a regular expression denoting L (r_1).
  - $r_1 | r_2$ is a regular expression denoting L (r1) U L (r2).
  - $r_1 r_2$ is a regular expression denoting L (r_1) L (r_2).
  - r_1^* is a regular expression denoting L $(r_1)^{*.}$
- For example, the language ,a*ba*- denotes the set ,b, aba, aabaa, …-.

Graphical Methods

- Graphical methods provide more comprehensible representation than their textual counterpart.
- There exist various methods, such as data flow diagrams (DFD), E-R modeling, finite state machines (FSM), etc.
- Finite state machines (FSM)
  - are used to represent the behavior of a system or a complex object with certain conditions.
  - It consists of four elements: *states, state transitions, rules or conditions, input events.*

Finite state machines (FSM)

- A FSM has an initial state and a set of final states.
- FSM can be expressed as a state table to represent the transition function and the output function for all types of states and inputs.
- FSMs are useful for real time software.
- But in case of a large system, number of nodes and transitions become unwieldy and unmanageable.

- **Example 5.3:** FSM for reading even number of 0s (e.g., 00, 0000, 000000, etc.).

Table 5.2: A state transition table

Current state	Current input
	0
S_0	S_1
S_1	S_2
S_2	S_1

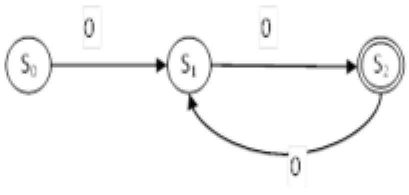


Figure 5.27: FSM for even number of 0s

Mathematical Methods

- Mathematical methods are formal requirement specification approaches that are used to model complex systems in mathematical terminologies.
- The proof of correctness is the goal of mathematical specification, which can help to identify faulty reasoning far earlier than in the traditional design.
- The specifications are clear, unambiguous and provable but these are difficult to understand for the customer.
- Formal specifications can automatically be processed using automated tools.
- Mathematical specifications are programming independent.
- There are various formal mathematical methods, such as *algebraic specifications, axiomatic specifications*, etc.
- Algebraic and axiomatic specification consists of the following three constituent elements:

Specification = (Sort, Operation, Axioms);

- *Sort* is a set of values that describes the type or class (e.g., integers, stacks of integers, strings, complex, etc.).
- *Operation* specifies the name of the operation, its parameters and return type; for example, push: stack x element -> stack.
- *Axioms* are the rules that must hold true in any legal implementation of sort.
- Example 5.4: A string supports various operations like *create, append, concatenate, length, empty,* and *equal* operations. Specify the string operations in algebraic specification.
- The algebraic specification of the string operations is:
    - **Sorts:** *string, char, size, boolean*.
    - **Operations :** *create: () -> string*

append: string, string -> string

concatenate: string, char -> string

length: string -> size isEmpty:

string -> boolean equal: string,

string -> boolean

Example 5.4:

Axioms: *s1, s2: string; c: char*

isEmpty (new()) = true

isEmpty (concatenate(s1,c)) = false

length (new()) = 0

length (concatenate(s1,c)) = length (s1) + 1

append (s1, new()) = s1

append (s1, concatenate(s2,c)) = concatenate(append(s1,s2), c)

equal (new(), new()) = true

equal (new(), concatenate(s1,c)) = false

equal (concatenate(s1,c), new()) = false

equal (concatenate(s1,c), concatenate(s2,c)) = equal(s1,s2)

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.
-