

UNIT 2

UNIT 2 REQUIREMENT ANALYSIS AND SPECIFICATIONS

9 Hrs.

Functional And Non-Functional - User - System - Requirement Engineering Process - Feasibility Studies –communication practices- Requirements - Elicitation - Validation and management - Fundamental of requirement analysis – Analysis principles – Structured System Analysis - Software prototyping - Prototyping in the Software Process - Data - Functional and Behavioral Models - Structured Analysis and Data Dictionary.

Functional Requirement:

In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.

Example of Functional Requirements

- The software automatically validates customers against the ABC Contact Management System
- The Sales system should allow users to record customers sales
- The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
- Only Managerial level employees have the right to view revenue data.

Non-Functional Requirement:

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

UNIT 2

- The software system should be integrated with banking API
- The software system should pass [Section 508](#) accessibility requirement.

Examples of Non-functional requirements

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Functional vs Non Functional Requirements

Parameters	Functional Requirement	Non-Functional Requirement
What it is	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing,	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.

UNIT 2

Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Advantages of Functional Requirement

Here, are the pros/advantages of creating a typical functional requirement document-

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activities for easy project management
- Functional requirement can be expressed in Use Case form or user story as they exhibit externally visible functional behavior.

Advantages of Non-Functional Requirement:

Benefits/pros of Non-functional testing are:

- The nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

UNIT 2

User requirements:

User requirements, often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform. User requirements are generally documented in a User Requirements Document (URD) using narrative text. User requirements are generally signed off by the user and used as the primary input for creating system requirements.

An important and difficult step of designing a software product is determining what the user actually wants it to do. This is because the user is often not able to communicate the entirety of their needs and wants, and the information they provide may also be incomplete, inaccurate and self-conflicting. The responsibility of completely understanding what the customer wants falls on the business analyst. This is why user requirements are generally considered separately from system requirements. The business analyst carefully analyzes user requirements and carefully constructs and documents a set of high quality system requirements ensuring that the requirements meet certain quality characteristics.

A system is said to be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategic use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

System Requirements:

UNIT 2

System requirements are the building blocks developers use to build the system. These are the traditional “shall” statements that describe what the system “shall do.” System requirements are classified as either functional or supplemental requirements. A functional requirement specifies something that a user needs to perform their work. For example, a system may be required to enter and print cost estimates; this is a functional requirement. Supplemental or non-functional requirements specify all the remaining requirements not covered by the functional requirements. I prefer to use the term supplemental requirements instead of non-functional requirements; who wants to be termed non-functional. Supplemental requirements are sometimes called quality of service requirements. The plan for implementing functional requirements is detailed in the system design. The plan for implementing supplemental requirements is detailed in the system architecture. The list below shows various types of supplemental requirements.

- Accessibility
- Accuracy
- Audit, control, and reporting
- Availability
- Backup and restore
- Capacity, current and forecast
- Certification
- Compliance
- Compatibility of software, tools, standards, platform, database, and the like
- Concurrency
- Configuration management
- Dependency on other parties
- Deployment
- Documentation
- Disaster recovery
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Emotional factors (like fun or absorbing)
- Environmental protection
- Error handling
- Escrow

UNIT 2

- Exploitability
- Extensibility (adding features, and carry-forward of customizations at next major version upgrade)
- Failure management
- Interoperability
- Legal and regulatory
- Licensing
- Localizability
- Maintainability
- Modifiability
- Network topology
- Open source
- Operability
- Performance/response time
- Price
- Privacy
- Portability
- Quality
- Recovery/recoverability
- Redundancy

A solution may contain only software components, or it could incorporate both software and hardware components. It may also include training and organizational change requirements that will be provided to instructional designers to develop a comprehensive training program. Software requirements represent the system's functional and supplemental requirements that define the software components of the system.

Traditionally, functional and supplemental requirements for software were documented in the software requirements specification (SRS). The SRS is the principal deliverable that analysts use to communicate detailed requirements

UNIT 2

information to developers, testers, and other project stakeholders. Many modern systems, such as the Enfocus Requirement Suite,TM place requirements into multiple requirement bundles to support iterative development and to break requirements into various components required by teams to develop the solution (e.g., software, hardware, organizational change, etc.) Requirement bundles provide more flexibility than URD and SRS. Requirement bundles may be developed iteratively and incrementally with both stakeholders and project team members being able to validate and review the requirements as they evolve. User needs are always mapped to system requirements so that the developer sees not only the requirement but the source from where it originated.

Requirement Engineering:

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' document.

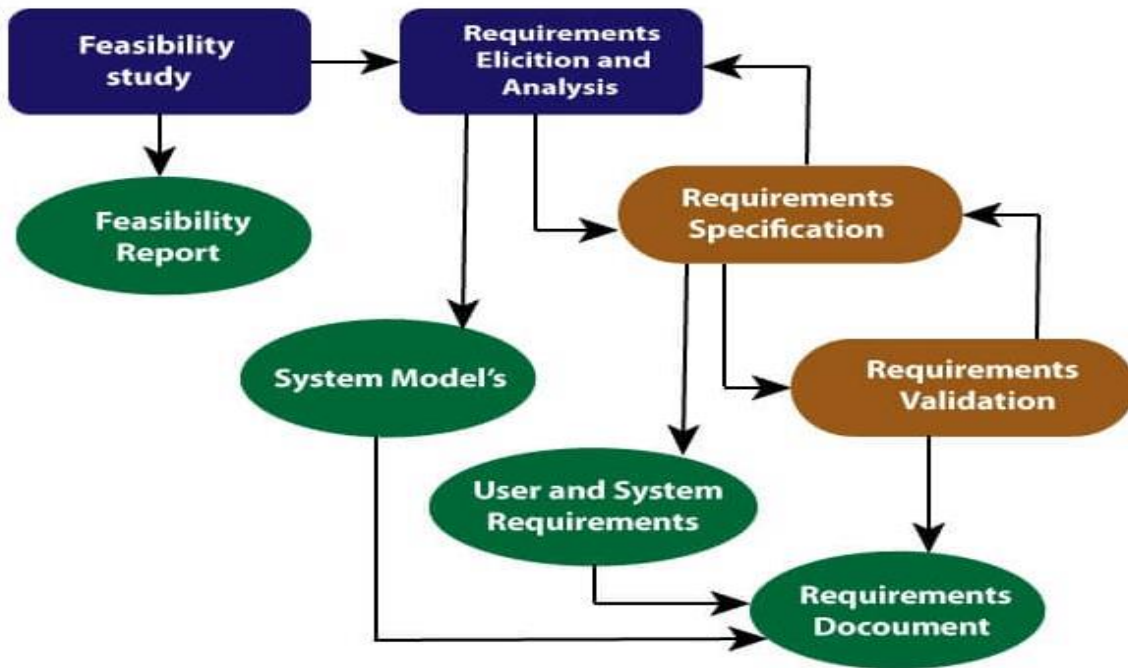
Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a four step process, which includes –

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

UNIT 2



Requirement Engineering Process

Feasibility study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

Technical Feasibility - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

Operational Feasibility - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.

Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

UNIT 2

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.

This feasibility study is focused towards goal of the organization. This 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 this 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 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 come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.

UNIT 2

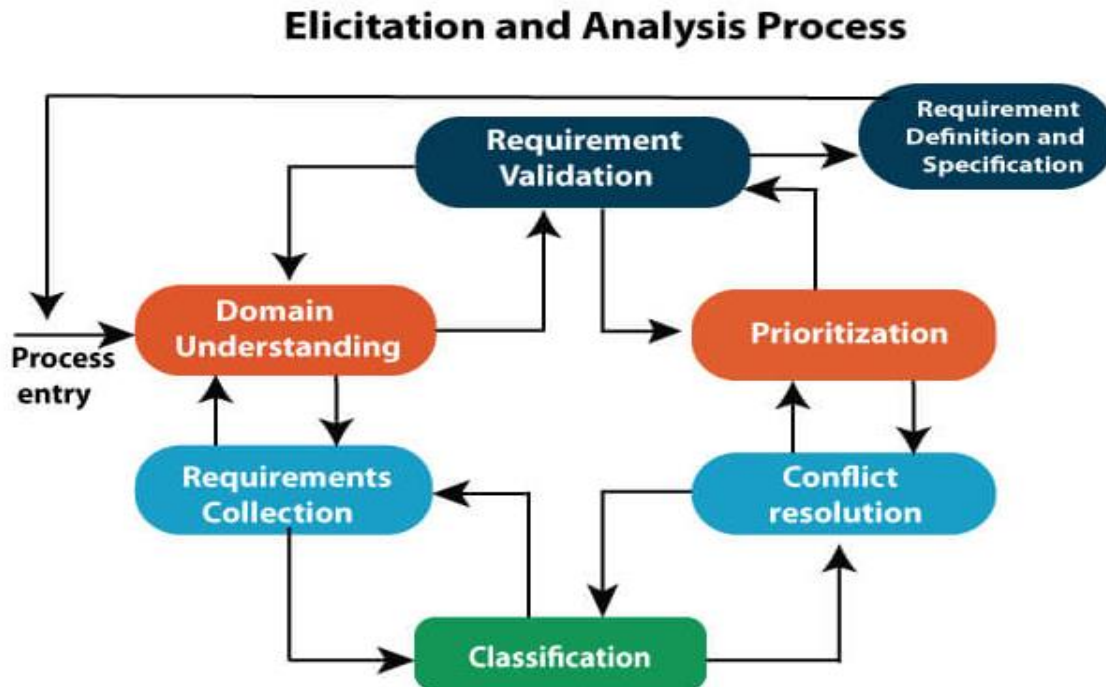
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

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

Requirement Elicitation Process



UNIT 2

Requirement elicitation process can be depicted using the following diagram:

- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation 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 every single information to gather 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.

UNIT 2

- 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 visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at

UNIT 2

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

Validation and Management:

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.

In the requirements validation process, we perform a different type of test to check the requirements mentioned in the **Software Requirements Specification (SRS)**, these checks include:

UNIT 2

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**

Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.

2. **Prototyping:**

In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.

UNIT 2

3. Requirements Reviews:

In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.

4. Automated Consistency Analysis:

This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

5. Walk-through:

A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.

- Checking early whether the idea is feasible or not.
- Obtaining the opinions and suggestion of other people.
- Checking the approval of others and reaching an agreement.

Requirements analysis process

Requirements Analysis is the process of defining the expectations of the users for an application that is to be built or modified. Requirements analysis involves all the tasks that are conducted to identify the needs of different stakeholders. Therefore requirements analysis means to analyze, document, validate and manage software or system requirements. High-quality requirements are documented, actionable, measurable, testable, traceable, helps to identify business opportunities, and are defined to facilitate system design.

UNIT 2

Requirements analysis process

The requirements analysis process involves the following steps:

Eliciting requirements. The process of gathering requirements by communicating with the customers is known as eliciting requirements.

Analyzing requirements

This step helps to determine the quality of the requirements. It involves identifying whether the requirements are unclear, incomplete, ambiguous, and contradictory. These issues resolved before moving to the next step.

Requirements modeling

In Requirements modeling, the requirements are usually documented in different formats such as use cases, user stories, natural-language documents, or process specification.

Review and retrospective

This step is conducted to reflect on the previous iterations of requirements gathering in a bid to make improvements in the process going forward.

Requirements Analysis Techniques

There are different techniques used for Requirements Analysis. Below is a list of different Requirements Analysis Techniques:

Business process modeling notation (BPMN)

This technique is similar to creating process flowcharts, although BPMN has its own symbols and elements. Business process modeling and notation is used to create graphs for the business process. These graphs simplify understanding the business process. BPMN is widely popular as a process improvement methodology.

UML (Unified Modeling Language)

UML consists of an integrated set of diagrams that are created to specify, visualize, construct and document the artifacts of a software system. UML is a useful technique while creating object-oriented software and working with the software

UNIT 2

development process. In UML, graphical notations are used to represent the design of a software project. UML also help in validating the architectural design of the software.

Flowchart technique

A flowchart depicts the sequential flow and control logic of a set of activities that are related. Flowcharts are in different formats such as linear, cross-functional, and top-down. The flowchart can represent system interactions, data flows, etc. Flow charts are easy to understand and can be used by both the technical and non-technical team members. Flowchart technique helps in showcasing the critical attributes of a process.

Data flow diagram

This technique is used to visually represent systems and processes that are complex and difficult to describe in text. Data flow diagrams represent the flow of information through a process or a system. It also includes the data inputs and outputs, data stores, and the various subprocess through which the data moves. DFD describes various entities and their relationships with the help of standardized notations and symbols. By visualizing all the elements of the system it is easier to identify any shortcomings. These shortcomings are then eliminated in a bid to create a robust solution.

Role Activity Diagrams (RAD)

Role-activity diagram (RAD) is a role-oriented process model that represents role-activity diagrams. Role activity diagrams are a high-level view that captures the dynamics and role structure of an organization. Roles are used to grouping together activities into units of responsibilities. Activities are the basic parts of a role. An activity may be either carried out in isolation or it may require coordination with other activities within the role.

Gantt Charts

Gantt charts used in project planning as they provide a visual representation of tasks that are scheduled along with the timelines. The Gantt charts help to know what is scheduled to be completed by which date. The start and end dates of all the tasks in the project can be seen in a single view.

UNIT 2

IDEF (Integrated Definition for Function Modeling)

Integrated definition for function modeling (IDEFM) technique represents the functions of a process and their relationships to child and parent systems with the help of a box. It provides a blueprint to gain an understanding of an organization's system.

Gap Analysis

Gap analysis is a technique which helps to analyze the gaps in performance of a software application to determine whether the business requirements are met or not. It also involves the steps that are to be taken to ensure that all the business requirements are met successfully. Gap denotes the difference between the present state and the target state. Gap analysis is also known as need analysis, need assessment or need-gap analysis.

Structured Analysis:

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

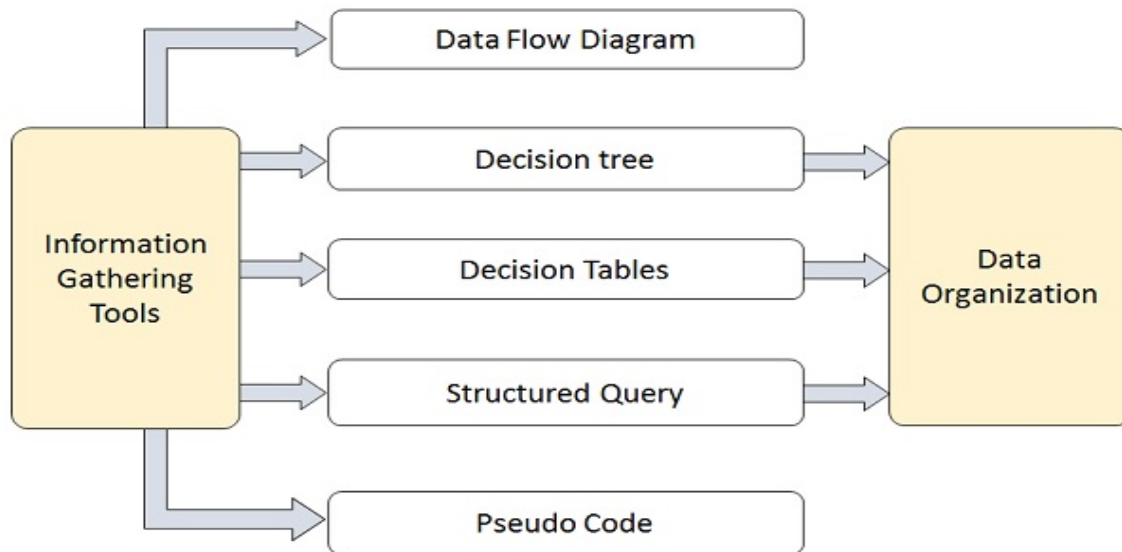
Structured Analysis Tools

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

- Data Flow Diagrams
- Data Dictionary

UNIT 2

- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.





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

UNIT 2

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 DFD

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.

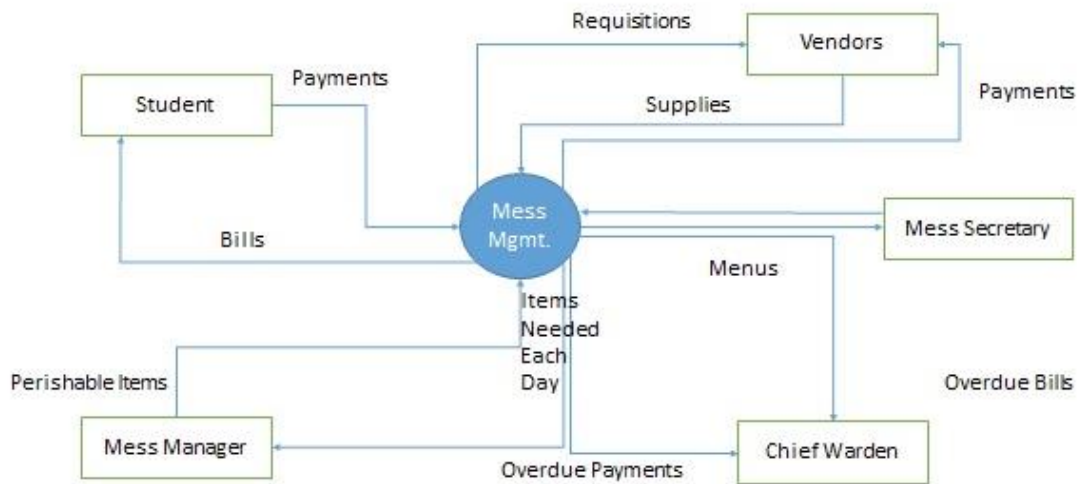
Physical DFD	Logical DFD
It is implementation dependent. It shows which functions are performed.	It is implementation independent. It focuses only on the flow of data between processes.
It provides low level details of hardware, software, files, and people.	It explains events of systems and data required by each event.
It depicts how the current system operates and how a system will be implemented.	It shows how business operates; not how the system can be implemented.

Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.

UNIT 2



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 –

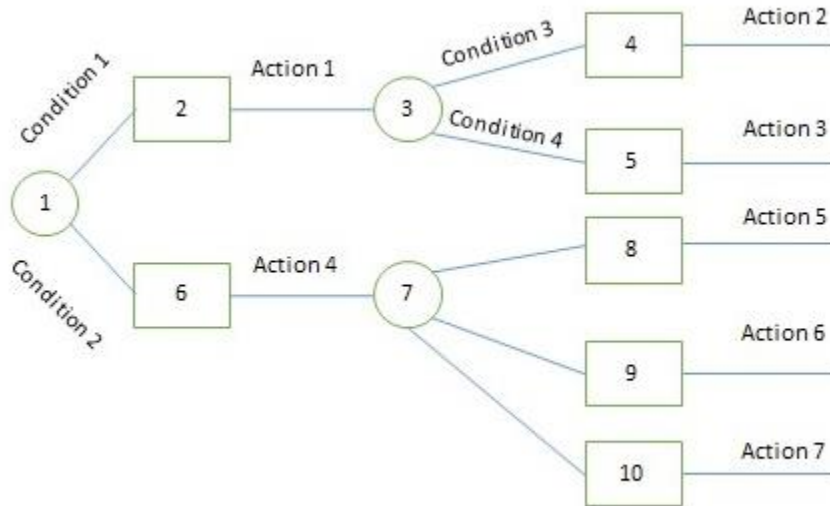
Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15

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.

UNIT 2

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:



Decision Tables

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

UNIT 2

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

Components of a Decision Table

- **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **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 –

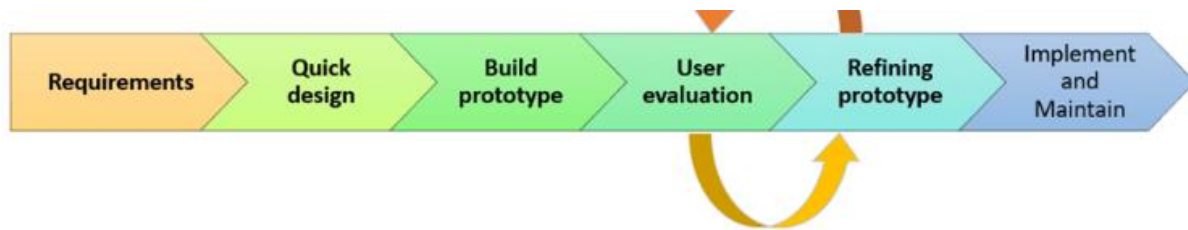
CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N
Regular Customer	-	Y	N	-
ACTIONS				
Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

UNIT 2

Software Prototyping:

Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.

Software prototyping model works best in scenarios where the project's requirement are not known. It is an iterative, trial, and error method which take place between the developer and the client.



Prototyping Model has following six SDLC phases as follow:

Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

UNIT 2

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

Types of Prototyping Models

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

Rapid Throwaway Prototype

Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.

In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

UNIT 2

Evolutionary Prototyping

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.

This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

Incremental Prototyping

In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

Extreme Prototyping:

Extreme prototyping method is mostly used for web development. It consists of three sequential phases.

1. Basic prototype with all the existing page is present in the HTML format.
2. You can simulate data process using a prototype services layer.
3. The services are implemented and integrated into the final prototype.

Data modeling:

Data modeling (data modelling) is the process of creating a data model for the data to be stored in a Database. This data model is a conceptual representation of Data objects, the associations between different data objects and the rules. Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

UNIT 2

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

The two types of **Data Models techniques** are

- Entity Relationship (E-R) Model
- UML (Unified Modelling Language)

Goal of Data Model:

The primary goal of using data model are:

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.
- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

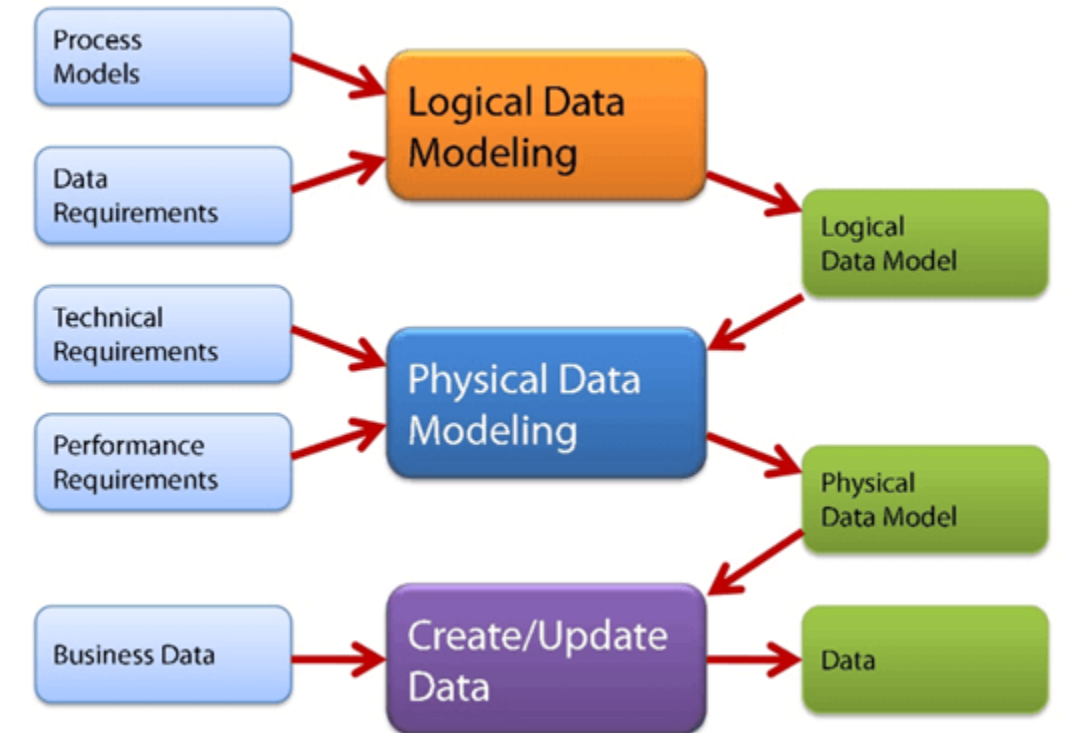
Types of Data Models

There are mainly three different types of data models:

1. **Conceptual:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.

UNIT 2

2. **Logical:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical:** This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.



Conceptual Model

The main aim of this model is to establish the entities, their attributes, and their relationships. In this Data modeling level, there is hardly any detail available of the actual Database structure.

The 3 basic tenants of Data Model are

Entity: A real-world thing

Attribute: Characteristics or properties of an entity

Relationship: Dependency or association between two entities

UNIT 2

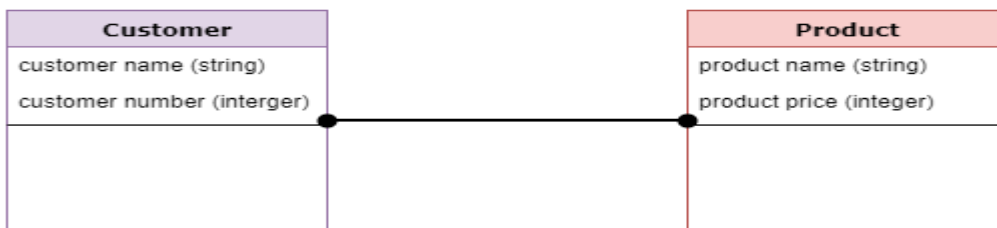
For example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity
- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product



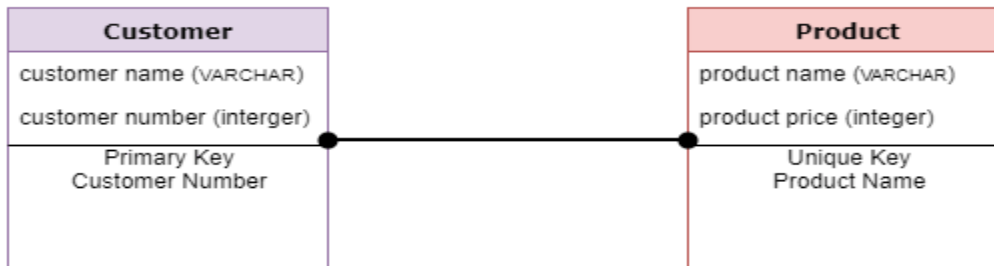
Logical Data Model

Logical data models add further information to the conceptual model elements. It defines the structure of the data elements and set the relationships between them.



Physical Data Model

A Physical Data Model describes the database specific implementation of the data model. It offers an abstraction of the database and helps generate schema. This is because of the richness of meta-data offered by a Physical Data Model.



UNIT 2

Behavioural Model :

All behavioural models really do is describe the control structure of a system. This can be things like:

- Sequence of operations
- Object states
- and Object interactions

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

Representation:

we represent them with dynamic diagrams. For example:

- (Design) Sequence Diagrams
- Communication Diagrams or collaboration diagram
- State Diagrams or state machine diagram or state chart

For consistency use communication diagram and state diagram

If we have both a sequence diagram AND and communication diagram, then together these are known as interaction diagrams, this is because they both represent how objects interact with one another using messages.

Description:

A behavioural model describes when the system is changing.

The key feature (subject) of a behavioural model is – Objects.

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 .

UNIT 2

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

A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.

A data dictionary lists the objective of all data items and the definition of all composite data elements in terms of their component data items. For example, a data dictionary entry may contain that the data *grossPay* consists of the parts *regularPay* and *overtimePay*.

$$\text{grossPay} = \text{regularPay} + \text{overtimePay}$$

For the smallest units of data elements, the data dictionary lists their name and their type.

A data dictionary plays a significant role in any software development process because of the following reasons:

- A Data dictionary provides a standard language for all relevant information for use by engineers working in a project. A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.