MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1111 (Management Information System)

Block 2: System Analysis and Design

Unit 5: Requirements Analysis and Modelling

Structure	
5.0	Introduction
5.1	Objectives
5.2	Requirements Analysis
5.2.1	Requirements Elicitation
5.2.2	Requirements Documentation
5.2.3	Requirements Validation
5.3	Requirements Modelling
5.3.1	Data Modelling
5.3.2	Process Modelling
5.3.3	Behavioural Modelling
5.3.4	User Interface Modelling
5.4	Tools for Requirements Analysis and Modelling
5.4.1	Requirements Management Tools
5.4.2	Diagramming and Modelling Tools
5.4.3	Prototyping Tools
5.4.4	Comprehensive Modelling Tools
5.5	Practices in Requirements Analysis and Modelling
5.5.1	Stakeholder Involvement
5.5.2	Clear and Concise Documentation
5.5.3	Iterative Refinement
5.6	Entity Relationship Diagram
5.7	Data Flow Diagram
5.8	Summary
5.9	Questions & Answers
5.10	References

5.0 Introduction

In the realm of system development, the accuracy and clarity of initial requirements are crucial determinants of a project's success. Requirements Analysis and Modelling form the cornerstone of this process, ensuring that the final product aligns with the needs and expectations of stakeholders. These activities involve identifying what the system should do, how it should perform under various conditions, and the constraints it must operate within. By methodically gathering, documenting, validating, and modelling requirements, developers can create systems that are both functional and efficient, minimizing the risk of costly revisions later in the development lifecycle.

The journey begins with Requirements Analysis, where the focus is on understanding the needs of users and stakeholders through techniques such as interviews, surveys, and observations. This phase culminates in comprehensive documentation that clearly articulates the requirements. Once gathered, these requirements undergo validation to ensure they are feasible and accurately reflect stakeholder needs. This step is vital in confirming that the requirements are both practical and aligned with the project's goals, reducing ambiguities and potential misunderstandings.

Following validation, Requirements Modelling translates these needs into structured representations, including data models, process models, behavioural models, and user interface models. These models provide a systematic and visual way to understand and communicate the system's functionality. Various tools and best practices are employed throughout these stages to enhance efficiency and accuracy. This chapter delves into the detailed processes of Requirements Analysis and Modelling, exploring the techniques, tools, and methodologies that ensure the development of robust, user-centric systems.

5.1 Objectives

- **Understand Requirements Analysis**: Gain a comprehensive understanding of the processes involved in requirements analysis, including elicitation, documentation, and validation techniques.
- **Learn Requirements Modelling**: Explore various modelling techniques such as data Modelling, process modelling, behavioural modelling, and user interface modelling to represent system requirements effectively.
- **Identify and Utilize Tools**: Familiarize with the essential tools for requirements management, diagramming, prototyping, and comprehensive modelling to facilitate efficient and accurate requirements analysis and modelling.
- **Implement Best Practices**: Learn best practices in requirements analysis and modelling, including stakeholder involvement, clear and concise documentation, and iterative refinement to ensure the development of robust systems.
- **Apply Modelling Techniques**: Understand and apply specific modelling techniques like Entity Relationship Diagrams (ERD) and Data Flow Diagrams (DFD) to visualize and analyze system requirements clearly.

5.2 Requirements Analysis

In systems engineering and software engineering, requirements analysis focuses on the tasks that determine the needs or conditions to meet the new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analysing, documenting, validating, and managing software or system requirements.

Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

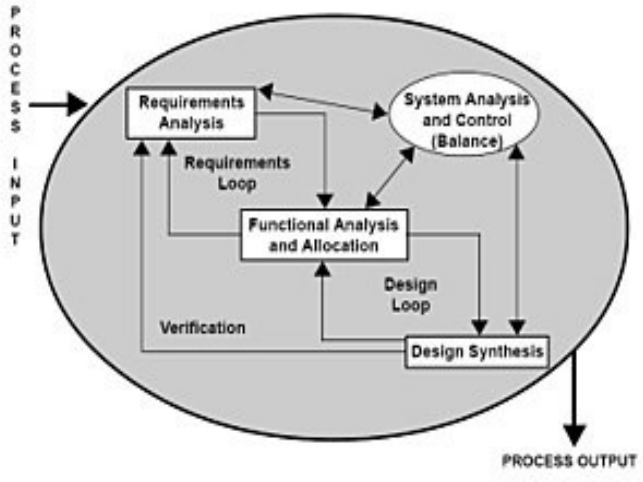

Image Source: Wikipedia

Requirements Analysis and Modelling is a critical phase in the software development process. It involves understanding and documenting what a software system should do and how it should perform under various conditions. This phase lays the foundation for designing, developing, and deploying a successful software system.

Requirements Analysis is the process of determining user expectations for a new or modified product. This process is essential in understanding what the users need and ensuring that the final product meets these needs. It involves various stages including requirements elicitation, documentation, and validation.

5.2.1 Requirements Elicitation

Elicitation is the first step in requirements analysis where information is gathered from stakeholders to understand their needs and constraints. This step is crucial for setting the foundation for all subsequent development activities.

Techniques for Requirements Elicitation:

1. **Interviews**

- **Structured Interviews:** Predefined questions are asked to gather specific information.
- **Unstructured Interviews:** Open-ended questions allow stakeholders to provide broader insights.
- **Benefits:** Provides in-depth information, clarifies misunderstandings, and builds rapport with stakeholders.
- **Challenges:** Time-consuming and may lead to biased responses if not conducted properly.

2. **Surveys/Questionnaires**

- **Surveys:** Distribute standard questions to a large audience to gather quantitative data.
- **Questionnaires:** Similar to surveys but may include open-ended questions for qualitative data.
- **Benefits:** Efficient for gathering data from a large group, cost-effective, and provides quantifiable data.

- **Challenges:** Limited depth of responses, potential low response rates, and ambiguity in responses if questions are not clear.

3. **Observation**

- **Direct Observation:** Observing users in their natural environment to understand their workflows.
- **Participant Observation:** The analyst participates in the daily activities of users to gain a deeper understanding.
- **Benefits:** Provides context and insight into actual user behavior and identifies unarticulated needs.
- **Challenges:** Time-consuming, may be perceived as intrusive, and requires careful interpretation of observed behaviors.

4. **Workshops and Focus Groups**

- **Workshops:** Interactive sessions with stakeholders to brainstorm and prioritize requirements.
- **Focus Groups:** Group discussions with selected participants to gather opinions and attitudes.
- **Benefits:** Facilitates consensus building, generates diverse ideas, and accelerates the requirements gathering process.
- **Challenges:** Requires skilled facilitation, potential for groupthink, and may be dominated by strong personalities.

5. **Prototyping**

- **Throwaway Prototyping:** Build a model to gather feedback and discard it after use.
- **Evolutionary Prototyping:** Build a prototype and refine it based on user feedback until it evolves into the final system.
- **Benefits:** Provides a tangible representation of the system, helps clarify requirements, and identifies usability issues early.
- **Challenges:** Time-consuming, may lead to unrealistic expectations, and requires careful management to avoid scope creep.

5.2.2 Requirements Documentation

Once requirements are gathered, they need to be documented in a clear and organized manner. Proper documentation ensures that all stakeholders have a common understanding of the requirements and serves as a reference throughout the project lifecycle.

Key Types of Requirements Documentation:

1. **Business Requirements Document (BRD)**

- **Content:** High-level business goals, objectives, and scope of the project.
- **Purpose:** Aligns the project with business objectives and provides a clear understanding of the business context.
- **Audience:** Business stakeholders, project managers, and senior management.

2. **Functional Requirements Specification (FRS)**

- **Content:** Detailed description of the system's functionalities, including inputs, processes, outputs, and interactions.

- **Purpose:** Defines what the system should do to meet the user needs.
- **Audience:** Development team, testers, and system architects.

3. **Non-Functional Requirements (NFR)**

- **Content:** Defines system attributes such as performance, security, usability, reliability, and scalability.
- **Purpose:** Specifies the quality attributes and constraints of the system.
- **Audience:** Development team, testers, and system architects.

4. **Use Cases and User Stories**

- **Use Cases:** Describe interactions between a user (actor) and the system to achieve a specific goal.
- **User Stories:** Short, simple descriptions of a feature from the perspective of the end user.
- **Purpose:** Helps in understanding user interactions and prioritizing features.
- **Audience:** Development team, testers, and product owners.

5.2.3 Requirements Validation

Validation ensures that the documented requirements accurately reflect the needs of stakeholders and are feasible for implementation.

Techniques for Requirements Validation:

1. **Reviews and Inspections**

- **Formal Reviews:** Structured meetings to evaluate requirements documents.
- **Peer Reviews:** Colleagues review each other's work for accuracy and completeness.
- **Benefits:** Identifies defects early, ensures requirements are clear and complete, and involves multiple perspectives.
- **Challenges:** Can be time-consuming and may require significant coordination.

2. **Prototyping**

- **Usage:** Creating prototypes to validate requirements through user feedback.
- **Benefits:** Provides early validation of requirements, reduces misunderstandings, and allows for iterative refinement.
- **Challenges:** May lead to unrealistic expectations if not managed properly.

3. **Requirements Traceability Matrix (RTM)**

- **Purpose:** Ensures all requirements are covered through development and testing phases.
- **Benefits:** Tracks requirements throughout the project lifecycle, ensures coverage and compliance, and facilitates impact analysis.
- **Challenges:** Maintaining traceability can be labor-intensive and requires diligent documentation practices.

5.3 Requirements Modelling

Requirements Modelling involves representing the gathered requirements in a systematic and visual manner. This helps in understanding, analysing, and communicating the requirements effectively.

5.3.1 Data Modelling

Data Modelling focuses on defining the structure, relationships, and constraints of data within the system. It provides a clear picture of how data is organized and managed.

1. **Entity-Relationship Diagrams (ERD)**

- **Components:** Entities (objects of interest), attributes (properties of entities), and relationships (associations between entities).
- **Purpose:** Visualizes the data structure and relationships in a system.
- **Usage:** Used in database design and to understand data requirements.
- **Example:** An ERD for a library system might include entities such as Book, Member, and Loan, with relationships showing how members borrow books.

2. **Class Diagrams**

- **Components:** Classes (blueprints for objects), attributes (properties), methods (functions), and relationships (associations, inheritance, aggregation).
- **Purpose:** Represents the static structure of an object-oriented system.
- **Usage:** Used in object-oriented design to define system classes and their interactions.
- **Example:** A class diagram for an e-commerce system might include classes like Customer, Order, Product, with attributes and methods defining their behaviours and properties.

5.3.2 Process Modelling

Process Modelling represents the workflows and processes within the system. It helps in understanding how data flows and processes are executed.

1. **Data Flow Diagrams (DFD)**

- **Components:** Processes (activities), data stores (repositories), data flows (arrows indicating data movement), and external entities (sources or sinks of data).
- **Purpose:** Illustrates how data moves through a system and how it is processed.
- **Usage:** Used to model system processes and data flow at different levels of detail.
- **Example:** A DFD for an order processing system might show how customer orders are received, processed, stored, and shipped.

2. **Flowcharts**

- **Components:** Symbols representing processes, decisions, inputs/outputs, and flow lines.
- **Purpose:** Visualizes the sequence of steps and decisions in a process.
- **Usage:** Used to model algorithms, workflows, and system processes.
- **Example:** A flowchart for a login process might include steps like entering username and password, verifying credentials, and granting access.

3. **Activity Diagrams**

- **Components:** Activities (tasks), transitions (flow), start and end points, decisions, and parallel activities.
- **Purpose:** Represents the flow of activities and their dependencies.
- **Usage:** Used in UML to model dynamic aspects of the system, including workflows and business processes.
- **Example:** An activity diagram for an online shopping process might include activities like browsing products, adding to cart, checkout, and payment.

5.3.3 Behavioural Modelling

Behavioural Modelling focuses on how the system behaves in response to different inputs or events. It helps in understanding the dynamic aspects of the system.

1. **Use Case Diagrams**

- **Components:** Actors (users or other systems), use cases (functionalities), and associations (relationships between actors and use cases).
- **Purpose:** Visualizes user interactions with the system.
- **Usage:** Used to identify and document system functionalities from the user's perspective.
- **Example:** A use case diagram for a banking system might include use cases like deposit money, withdraw money, transfer funds, with actors such as customer and bank teller.

2. **Sequence Diagrams**

- **Components:** Objects (entities), lifelines (existence over time), messages (interactions), and activation bars (object activity).
- **Purpose:** Shows how objects interact in a sequence of events.
- **Usage:** Used in UML to model the flow of control and data between objects.
- **Example:** A sequence diagram for an ATM transaction might show interactions between the customer, ATM machine, and bank server.

3. **State Diagrams**

- **Components:** States (statuses), transitions (state changes), events (triggers), and actions (responses).
- **Purpose:** Represents the states an object can be in and the transitions between those states.
- **Usage:** Used to model the lifecycle of an object and its responses to events.
- **Example:** A state diagram for a traffic light system might include states like red, yellow, green, with transitions based on time events.

5.3.4 User Interface Modelling

User Interface (UI) Modelling focuses on designing the visual and interactive aspects of the system. It helps in understanding how users will interact with the system.

1. **Wireframes**

- **Components:** Basic layout of UI elements such as buttons, menus, and text fields.

- **Purpose:** Provides a skeletal framework of the user interface.
- **Usage:** Used in early stages of design to visualize and validate the UI layout and structure.
- **Example:** A wireframe for a mobile app might include the layout of the home screen, navigation menu, and key interaction points.

2. **Mockups**

- **Components:** High-fidelity visual representation of the UI, including colors, fonts, and detailed design elements.
- **Purpose:** Provides a detailed and realistic view of the final user interface.
- **Usage:** Used to validate design choices and gather feedback before development.
- **Example:** A mockup for a website might include the design of the homepage, product pages, and checkout process with all visual details.

5.4 Tools for Requirements Analysis and Modelling

Several tools assist in requirements analysis and modelling, enhancing efficiency, accuracy, and collaboration. These tools support various stages of the requirements process and help create visual representations of the system.

5.4.1 Requirements Management Tools

1. **JIRA**

- **Features:** Requirements tracking, project management, issue tracking, and agile development support.
- **Usage:** Used to manage and track requirements, tasks, and issues throughout the project lifecycle.
- **Benefits:** Provides a centralized platform for collaboration, supports agile methodologies, and integrates with other development tools.

2. **Confluence**

- **Features:** Collaborative documentation, knowledge management, and integration with JIRA.
- **Usage:** Used to document requirements, create collaborative spaces, and manage project documentation.
- **Benefits:** Facilitates collaboration, maintains version control, and integrates seamlessly with JIRA for comprehensive project management.

5.4.2 Diagramming and Modelling Tools

1. **Lucidchart**

- **Features:** Diagramming, flowchart creation, ERD, UML modelling, and real-time collaboration.
- **Usage:** Used to create various types of diagrams and models, including ERDs, flowcharts, and process models.
- **Benefits:** Easy to use, supports collaboration, and integrates with other tools like Google Drive and Microsoft Office.

2. **Microsoft Visio**

- **Features:** Diagramming, flowchart creation, data visualization, and various templates for different types of models.
- **Usage:** Used to create professional diagrams, including process models, data models, and network diagrams.
- **Benefits:** Versatile, supports a wide range of diagram types, and integrates with Microsoft Office Suite.

5.4.3 Prototyping Tools

1. **Balsamiq**

- **Features:** Wireframing, low-fidelity prototyping, and UI design.
- **Usage:** Used to create wireframes and low-fidelity prototypes to visualize and validate UI designs.
- **Benefits:** Simple and intuitive interface, supports quick iterations, and facilitates early-stage design discussions.

2. **Axure RP**

- **Features:** Prototyping, wireframing, interactive prototypes, and documentation.
- **Usage:** Used to create high-fidelity interactive prototypes to simulate user interactions.
- **Benefits:** Supports complex interactions, allows for detailed documentation, and facilitates user testing and feedback.

5.4.4 Comprehensive Modelling Tools

1. **Enterprise Architect**

- **Features:** Comprehensive modelling, UML support, requirements management, and simulation.
- **Usage:** Used to create detailed models, manage requirements, and simulate system behavior.
- **Benefits:** Supports a wide range of modelling standards, integrates with other development tools, and provides robust analysis and simulation capabilities.

2. **IBM Rational DOORS**

- **Features:** Requirements management, traceability, collaboration, and integration with other IBM tools.
- **Usage:** Used to manage requirements, track changes, and ensure traceability throughout the project lifecycle.
- **Benefits:** Supports complex projects, provides robust traceability, and integrates with other tools in the IBM Rational suite.

5.5 Practices in Requirements Analysis and Modelling

To ensure effective requirements analysis and modelling, it is important to follow best practices that enhance the accuracy, clarity, and feasibility of the requirements.

5.5.1 Stakeholder Involvement

1. **Continuous Engagement**

- **Description:** Engage stakeholders throughout the requirements process to ensure their needs are accurately captured and addressed.
- **Benefits:** Ensures that the requirements reflect stakeholder needs, builds trust, and facilitates buy-in and support for the project.

2. **Stakeholder Analysis**

- **Description:** Identify and analyse stakeholders to understand their influence, interests, and needs.
- **Benefits:** Helps in prioritizing requirements, managing expectations, and ensuring that all relevant perspectives are considered.

5.5.2 Clear and Concise Documentation

1. **Standardized Templates**

- **Description:** Use standardized templates for documenting requirements to ensure consistency and clarity.
- **Benefits:** Facilitates understanding, reduces ambiguity, and ensures that all necessary information is captured.

2. **Use of Clear Language**

- **Description:** Write requirements in clear, concise, and unambiguous language.
- **Benefits:** Reduces misunderstandings, ensures that requirements are easily understood by all stakeholders, and improves communication.

5.5.3 Iterative Refinement

1. **Incremental Development**

- **Description:** Develop requirements iteratively, refining them based on feedback and changing needs.
- **Benefits:** Allows for continuous improvement, adapts to changing requirements, and reduces the risk of major rework.

2. **Feedback Loops**

- **Description:** Establish regular feedback loops with stakeholders to review and refine requirements.
- **Benefits:** Ensures that requirements remain aligned with stakeholder needs, identifies issues early, and enhances collaboration.

5.6 Entity Relationship Diagram

In a particular field of knowledge, interconnected objects of interest are described by an entity-relationship model, or ER model. Entity categories, which categorize the objects of interest, make up a basic ER model. It also describes the possible relationships between entities, which are examples of those entity types.

An ER model is typically created in software engineering to represent the information that a corporation needs to keep in mind in order to carry out business activities. Thus, the ER model turns into an abstract data model, wherein an information structure or data can be defined and implemented in a database, usually a relational database

Although there had been earlier iterations of the concept, entity-relationship modelling was created for databases and design by Peter Chen and presented in a 1976 paper. These days, it's frequently utilized to

instruct pupils on the fundamentals of database building. ER models can be used to define domain-specific ontologies and can display super and subtype entities related by generalization-specialization relationships.

Typically, an ER model is the outcome of a methodical analysis to identify and characterize the data produced and required by operations within a business domain. Rather than the processes themselves, it typically represents records of the entities and events that business processes oversee and control. The relationships and dependencies between entities are typically expressed graphically as boxes representing entities connected by lines representing relationships. Verbal expressions of it include the following: one building may contain zero or more apartments, but one apartment may only be found in a single building.

Uses of ER Diagrams

Database Design: Databases are designed using ER diagrams. They offer a graphic depiction of the relationships between various informational units, or entities. This aids in organizing the database's structure prior to construction.

Communication: ER diagrams serve as a common language for clients, developers, and designers who work together to create databases. They aid in everyone's understanding of the operation of the database and the structure of the data.

Database Maintenance: ER diagrams are still helpful for maintenance even after a database has been constructed. They guarantee the integrity of the database while assisting developers in understanding the current structure and making any necessary modifications or adjustments.

Documentation: ER diagrams act as the database system's documentation. They offer a thorough synopsis of the entities, properties, and relationships found in the database schema. This material is useful for teaching new team members and troubleshooting problems.

Finding Redundant and Inconsistent Data: ER diagrams assist in locating duplicate or inconsistent data in the database design by illustrating the relationships between entities. This guarantees data accuracy and permits optimization.

Components of ER Diagram

Entities

For which data is kept in the database, an entity is a representation of an actual item, idea, or thing. A person, location, object, occasion, or idea could be included.

Characteristics

- **Attributes:** Entities have attributes that describe their qualities or properties.
- **Instances:** Individual occurrences of each entity in the real world are represented by their instances, also known as occurrences.
- **Identity:** A given entity's instances are all distinguishable from one another and individually recognizable.
- **Relationships:** Entities can build connections within the database schema by connecting to other entities through relationships.

Types of Entities

- **Strong Entities:** Strong entities possess characteristics that allow each instance to be uniquely identified. Independent of other entities, they are capable of existing.

- **Weak Entities:** Since they lack a primary key characteristic of their own, weak entities are dependent on related strong entities, such as owner entities, in order to survive. Based on their connection to the owner entity, they can be distinguished.

Types of Entity Keys

- **Simple key:** A simple key is one where the primary key is one attribute.
- **Composite key:** The primary key is a result of combining two or more qualities. When every instance cannot be uniquely identified by a single characteristic, it is utilized.
- **Surrogate key:** The primary key is an artificially manufactured identification, like an auto-incremented number. When a natural key is deemed unsuitable or when issues arise regarding the stability or dimensions of the natural key, it is frequently added.

Attributes

The features or qualities of entities are called attributes. Each entity's specifics are described. "Customer ID," "Name," and "Email Address," for instance, might be characteristics of a "Customer" entity.

Characteristics

- **Name:** Every characteristic within an entity is identified by its name.
- **Data Type:** Attributes have a data type that identifies the kind of information they can hold, such as text, numbers, dates, etc.
- **Constraints:** Attributes may be subject to restrictions that set guidelines or requirements for the values they can have, such as uniqueness or necessity.
- **Domain:** The set of all potential values that an attribute can take is known as its domain.
- **Nullability:** For some instances of the entity, attributes may accept null values, i.e., no value may be assigned.

Characteristics

- **Simple Attribute:** An attribute that is simple has only one atomic value stored in it. Say, "Name and Age."
- **Composite Attribute:** A composite attribute is one that has smaller components, each of which represents a different attribute, and may be separated into smaller portions.
- Example: A street, city, and zip code address is an example.
- **Single-valued Attribute:** An attribute with only one value stored for each instance of the entity is called a single-valued attribute.
- **Multi-valued Attribute:** Attribute with numerous values that can be stored for each instance of the entity is called a multi-valued attribute. Example: phone numbers

Relationships

Relationships indicate the connections or connections between entities in the database. They depict relationships or exchanges among entities. Buying, working for, and owning are a few examples.

Characteristics

- **Name:** A name designating the type of connection between two things identifies each relationship.

- **Cardinality:** The quantity of one entity's instances that can be connected to another entity's instances in a relationship is known as histology. It indicates how frequently linked entities must occur, both minimum and maximum.
- **Degree:** An association's degree represents the total number of entities involved. Relationships can be binary (involving two entities), ternary (involving three entities), or unary.
- **Attributes:** Relationships can contain attributes that explain additional details or characteristics about the connection between entities.

Types of Relationships

- **One-to-one (1:1):** Every instance of one type of entity is linked to precisely one type of another.
- **One-to-many (1: N):** Every instance of one entity is linked to several instances of another; nevertheless, every instance of the other entity is solely connected to one instance of the initial entity.
- **Many-to-One:** Exactly one instance of one entity is linked to several instances of another in a many-to-one (N:1) relationship.
- **Many-to-Many:** Numerous instances of one entity are linked to numerous instances of another, and vice versa, in a many-to-many (M: N) architecture.

Cardinality

Cardinality describes the link between the quantity of instances of one entity and the quantity of instances of another entity. It indicates how many instances of one thing can be connected to instances of another entity, both at minimum and maximum.

Entity Relationship Diagram (ERD) Symbols and Notations

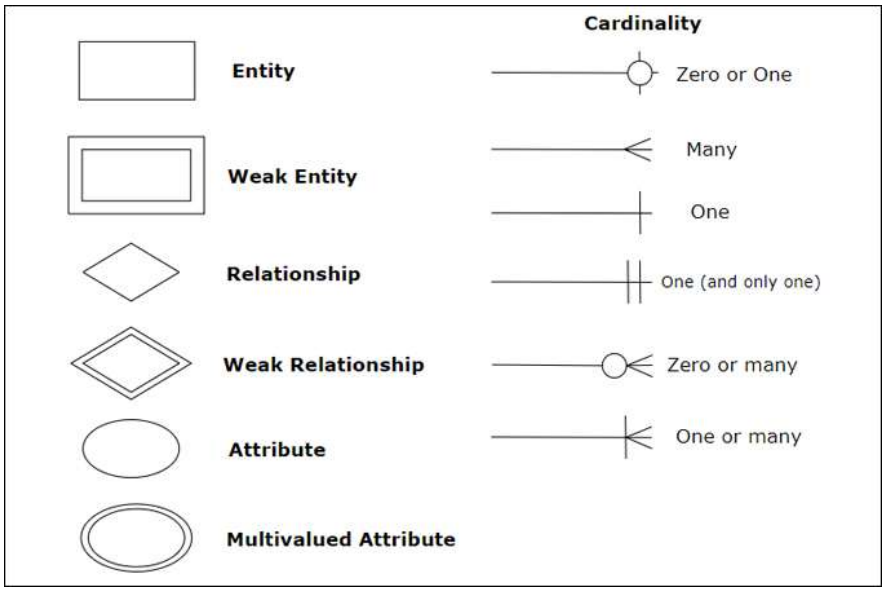

Image Source: GitMind

5.7 Data Flow Diagram

A data flow diagram (DFD) shows how information moves through a system or process. It displays data inputs, outputs, storage locations, and the paths between each location using well-defined symbols like rectangles, circles, and arrows together with brief text labels. Data flow diagrams (DFDs) can be as basic as hand-drawn process overviews or as complex as multi-level DFDs that gradually delve deeper into the data

handling process. They can be applied to model a new system or examine an existing one. Like all the best charts and diagrams, a DFD works for both technical and nontechnical audiences, from developers to CEOs, and it can frequently "say" things visually that would be difficult to explain in words. That's why DFDs are still so well-liked even after all these times. Although they are still useful for data flow software and systems, they are becoming less useful for database-oriented, interactive, or real-time software visualization.

A DFD is not a flowchart, it should be noted. The designer must indicate the key changes in the data flow path from the input to the output when creating the DFD. The ability to structure DFDs hierarchically facilitates the progressive division and analysis of huge systems.

It provides an overview of

- What data is system processes.
- What transformation are performed.
- What data are stored.
- What results are produced etc.

There are various ways to represent a data flow diagram. Among the modelling tools used in structured analysis is the Data Flow Diagram (DFD). Data flow diagrams are widely used because they make the key data and actions in software-system processes easier to see.

Characteristics of Data Flow Diagram (DFD)

Below are some characteristics of Data Flow Diagram (DFD):

- **Graphical Representation:** To depict data flow within a system, Data Flow Diagrams (DFDs) include a variety of symbols and notations. This makes the intricate model simpler.
- **Problem Analysis:** Data Flow Diagrams (DFDs) can be used efficiently during analysis and are highly helpful in comprehending a system. Data Flow Diagrams (DFDs) are a very generic tool that are not just used for software requirements formulation problem analysis.
- **Abstraction:** Data flow diagrams, or DFDs, offer an abstraction for complex models by hiding extraneous implementation details and displaying just the data and process flow inside an information system.
- **Hierarchy:** System structure: A Data Flow Diagram (DFD) shows the system structure. While lower-level diagrams, such as 1-level DFD and beyond, show a thorough data flow of a single process, high-level diagrams, or 0-level diagrams, give an overview of the complete system.
- **Data Flow:** Visualizing the data flow between external entities, processes, and data stores is the main goal of a data flow diagram (DFD). The symbol for data flow is an arrow.
- **Ease of Understanding:** Technical and non-technical stakeholders may both grasp Data Flow Diagrams (DFD) with ease.
- **Modularity:** The Data Flow Diagram (DFD) is a useful tool for achieving modularity since it may divide a large system into smaller modules or processes. This makes system analysis and design simple.

Levels of Data Flow Diagram (DFD)

Multilevel Data Flow Diagrams (DFDs) can be developed because DFDs use hierarchy to ensure transparency. The following are the Data Flow Diagram (DFD) levels:

0-Level DFD

Another name for it is a context diagram. The perspective is intended to be abstract, portraying the system as a unified process with a connection to outside entities. With incoming and outgoing arrows designating input and output data, the system is represented as a single bubble.

1-Level DFD

By dissecting the primary processes found in the level 0 DFD into smaller processes, this level offers a more thorough understanding of the system. On the level 1 DFD, each subprocess is represented as a distinct process. Each sub-process's related data stores and flows are also displayed. The context diagram in 1-level DFD is broken down into several bubbles or processes. At this level, we deconstruct the high-level 0-level DFD process into smaller processes and emphasize the primary features of the system.

2-level DFD

By dissecting the sub-processes found in the level 1 DFD into even more sub-processes, this level offers an even more thorough understanding of the system. On the level 2 DFD, each subprocess is represented as a distinct process. Each sub-process's related data stores and flows are also displayed.

5.8 Summary

In this chapter on Requirements Analysis and Modelling, the fundamental processes and methodologies essential for the successful development of systems are explored. Beginning with Requirements Analysis, the chapter delves into the critical phase of understanding stakeholder needs and constraints. Techniques such as requirements elicitation, documentation, and validation are discussed in detail, emphasizing the importance of accurately capturing and verifying requirements to ensure alignment with project objectives. By employing these techniques effectively, stakeholders can gain a comprehensive understanding of the system's scope and functionalities, laying the groundwork for the subsequent stages of development.

Moving forward, the chapter transitions to Requirements Modelling, where various modelling techniques are employed to represent system requirements systematically and visually. From data modelling to behavioural modelling, each technique offers unique insights into different aspects of system functionality, facilitating clear communication and informed decision-making among project stakeholders. Moreover, the chapter underscores the significance of employing appropriate tools and best practices throughout the requirements analysis and modelling process. By leveraging tools for requirements management, diagramming, and prototyping, developers can streamline the requirements process, enhance collaboration, and mitigate potential risks.

In conclusion, the chapter on Requirements Analysis and Modelling serves as a comprehensive guide for navigating the complexities of system development. By emphasizing the importance of accurately capturing, validating, and modelling requirements, the chapter equips readers with the knowledge and tools necessary to develop robust and user-centric systems. Through a combination of practical techniques, best practices, and innovative tools, stakeholders can effectively translate stakeholder needs into tangible system requirements, laying the foundation for successful system development and deployment.

5.9 Questions & Answers

1. What is the primary purpose of Requirements Analysis in system development, and what are some key techniques employed in this phase?

Answer: Requirements Analysis is crucial for understanding stakeholder needs and constraints, ensuring that the final system meets user expectations and business objectives. Some key techniques employed in this phase include requirements elicitation through methods such as interviews, surveys, and observations;

requirements documentation to capture and organize gathered information; and requirements validation to ensure that the documented requirements are accurate, feasible, and aligned with stakeholder needs.

2. How do Requirements Modelling techniques contribute to the development process, and what are some commonly used modelling techniques?

Answer: Requirements Modelling techniques play a vital role in representing system requirements systematically and visually, facilitating clear communication and decision-making among stakeholders. Commonly used modelling techniques include data modelling, which represents the structure and relationships of data within the system; process modelling, which illustrates the workflows and processes within the system; behavioural modelling, which depicts how the system behaves in response to various inputs; and user interface modeling, which designs the visual and interactive aspects of the system.

3. Why is stakeholder involvement considered a best practice in Requirements Analysis and Modelling, and how does it contribute to project success?

Answer: Stakeholder involvement is crucial because it ensures that the developed system aligns with stakeholder needs, expectations, and objectives. By actively engaging stakeholders throughout the requirements analysis and modelling process, developers can gain valuable insights into user preferences, organizational goals, and constraints, leading to more accurate and relevant system requirements. Additionally, stakeholder involvement fosters collaboration, buy-in, and support for the project, ultimately enhancing the likelihood of project success.

4. What role do tools play in Requirements Analysis and Modelling, and what are some commonly used tools in this context?

Answer: Tools play a significant role in streamlining the requirements analysis and modelling process, enhancing efficiency, accuracy, and collaboration. Commonly used tools include requirements management tools like JIRA and Confluence, diagramming and modelling tools like Lucidchart and Microsoft Visio, prototyping tools like Balsamiq and Axure RP, and comprehensive modelling tools like Enterprise Architect and IBM Rational DOORS. These tools provide functionalities for gathering, documenting, visualizing, and validating requirements, empowering developers to effectively manage and communicate system requirements throughout the development lifecycle.

5. How do iterative refinement and continuous feedback contribute to the success of Requirements Analysis and Modelling?

Answer: Iterative refinement and continuous feedback are essential for ensuring that requirements accurately reflect stakeholder needs and are feasible for implementation. By adopting an iterative approach to requirements analysis and modelling, developers can refine and adjust requirements based on ongoing feedback from stakeholders, changing project requirements, and evolving business needs. This iterative process enables developers to uncover potential issues early, address them proactively, and adapt to changes efficiently, ultimately leading to the development of more robust and user-centric systems.

5.10 References

- Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.
- Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Education.
- Kotonya, G., & Sommerville, I. (1998). Requirements Engineering: Processes and Techniques. John Wiley & Sons.
- Wiegers, K. E. (2003). Software Requirements (2nd ed.). Microsoft Press.

- Gottesdiener, E., & Gorman, M. (2012). Discover to Deliver: Agile Product Planning and Analysis. EBG Consulting, Inc.

***