Unit 4
Software Requirement Analysis & Specification:
(Marks: 11)

Software Requirement Analysis and Specification is a critical phase in the software development lifecycle, ensuring that the final product aligns with stakeholders' needs and expectations. This process encompasses several key activities:

1. Requirement Engineering

Requirement Engineering is a systematic process used to define, document, and manage software requirements. It ensures that software meets user needs and business objectives. The process involves:

- **Requirement Elicitation**: Gathering requirements from stakeholders.
- **Requirement Analysis**: Refining and validating requirements.
- **Requirement Specification**: Documenting requirements in a structured format.
- **Requirement Validation**: Ensuring requirements are correct and complete.
- **Requirement Management**: Handling changes in requirements throughout the development lifecycle.

Importance of Requirement Engineering:

- Reduces software failures by addressing user needs early.
- Prevents costly changes later in development.
- Improves communication between stakeholders and developers.

---

2. Requirement Elicitation

Requirement Elicitation is the process of collecting information from stakeholders to define software requirements. It ensures that the final product aligns with user needs.

2.1. Interviews

Interviews involve direct communication with stakeholders to gather detailed requirements. They can be:

- **Structured Interviews**: Predefined questions with fixed responses.
- **Unstructured Interviews**: Open-ended discussions without a fixed format.
- **Semi-structured Interviews**: A mix of both, allowing flexibility.

Advantages:

- Provides deep insights into user needs.
- Helps clarify ambiguous requirements.

Disadvantages:

- Time-consuming and may lead to biased results if not conducted properly.

2.2. Brainstorming Series

Brainstorming is a collaborative technique where multiple stakeholders discuss ideas to identify software requirements.

Steps in Brainstorming:

1. Define the problem statement.
2. Encourage stakeholders to propose ideas.
3. Analyze and prioritize ideas.

4. Document relevant requirements.

Advantages:

- Generates creative solutions.
- Encourages stakeholder involvement.

Disadvantages:

- Can be unstructured and lead to irrelevant discussions.
- Requires a skilled moderator.

2.3. Use Case Approach

A use case defines how users interact with the system to achieve a goal. It includes:

- **Actors**: Users or systems that interact with the software.
- **Use Cases**: Specific tasks performed by the system.
- **Scenarios**: Step-by-step execution of a use case.

Example of a Use Case: For a banking system, a use case could be *"Withdraw Money from ATM."*

1. User inserts card.
2. System validates card and PIN.
3. User selects the withdrawal amount.
4. System processes the transaction.
5. User collects cash.

Advantages:

- Clearly defines system behavior.
- Helps developers understand functional requirements.

Disadvantages:

- Requires continuous refinement as requirements evolve.

---

3. Requirement Analysis

Requirement Analysis involves understanding, refining, and structuring the gathered requirements. It ensures that requirements are consistent, complete, and feasible.

3.1. Data Flow Diagram (DFD)

A **Data Flow Diagram (DFD)** visually represents how data moves through a system.

Components of a DFD:

- **External Entities**: Sources or destinations of data (e.g., Users, Systems).
- **Processes**: Actions performed on data (e.g., Login Validation).
- **Data Stores**: Where data is stored (e.g., Database).
- **Data Flows**: Movement of data between components.

Levels of DFD:

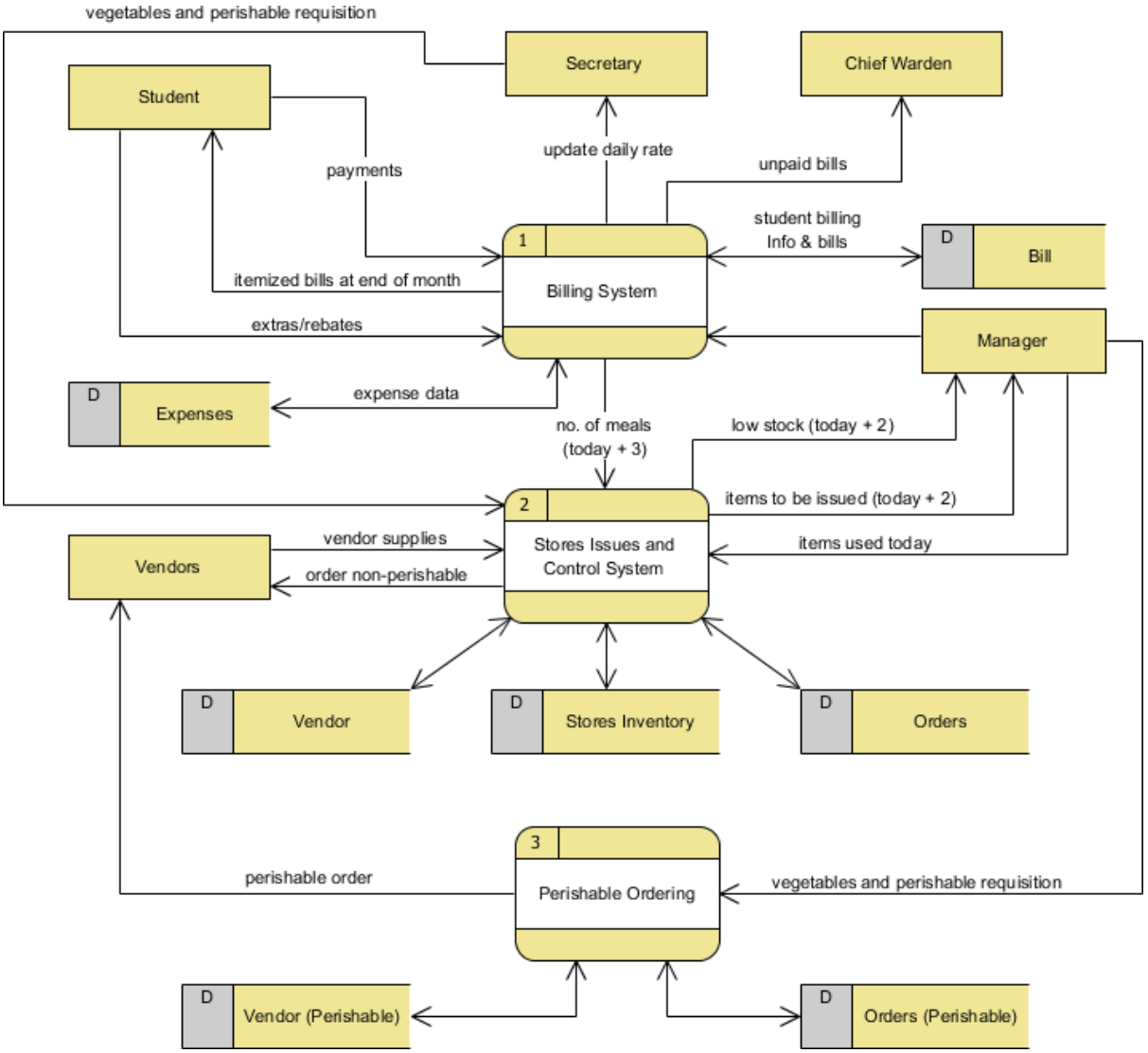
- **Level 0 (Context Diagram)**: High-level overview of the system.
- **Level 1**: Detailed breakdown of major processes.
- **Level 2+**: Further decomposition of processes.

Advantages:

- Provides a clear visual representation.
- Helps identify inefficiencies in data flow.

Disadvantages:

- Can become complex for large systems.



3.2. Data Dictionary

A **Data Dictionary** is a structured repository of all data elements used in the system.

Contents of a Data Dictionary:

- Data Name (e.g., `user_id`)
- Data Type (e.g., Integer, String)
- Description (Purpose of data)
- Possible Values (Valid range or format)
- Source (Where it comes from)

Advantages:

- Ensures consistency in data usage.
- Improves communication among developers and database designers.

Disadvantages:

- Requires continuous updates as requirements change.

Data Dictionary

Data Dictionary outlining a Database on Driver Details in NSW

Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

3.3. Entity-Relationship Diagram (ERD)

An **Entity-Relationship Diagram (ERD)** visually represents the data model of a system.

Components of an ERD:

- **Entities**: Objects that store data (e.g., Students, Courses).
- **Attributes**: Properties of entities (e.g., Student Name, Course ID).
- **Relationships**: Connections between entities (e.g., *Student Enrolls in Course*).

Types of Relationships:

- **One-to-One (1:1)**: One entity relates to one entity.

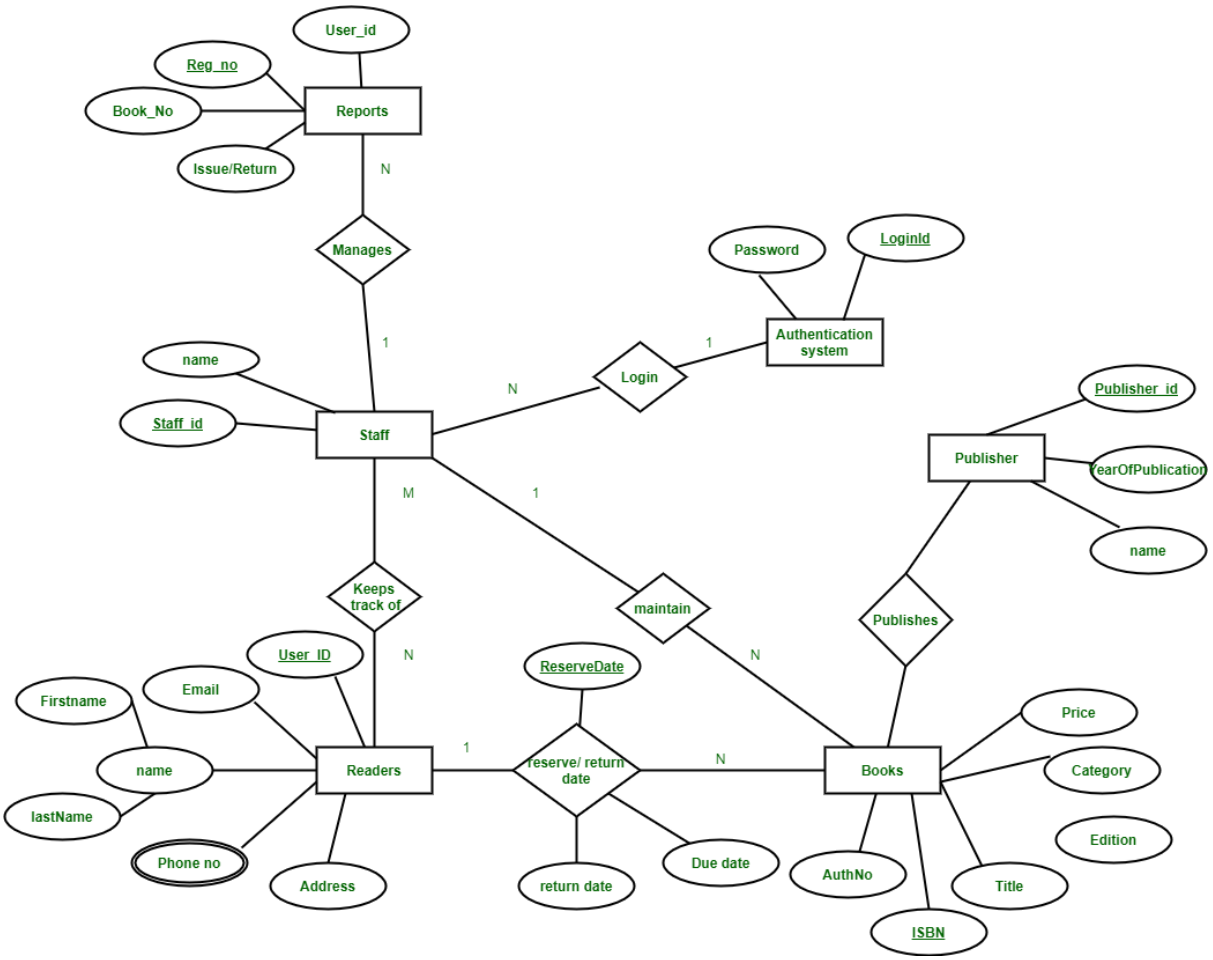
- **One-to-Many (1:M)**: One entity relates to multiple entities.
- **Many-to-Many (M:N)**: Multiple entities relate to multiple entities.

Advantages:

- Helps in database design.
- Improves understanding of system structure.

Disadvantages:

- Can become complex for large databases.



---

4. Requirement Documentation

Requirement Documentation involves writing a **Software Requirements Specification (SRS)** document that serves as a reference for developers, testers, and stakeholders.

4.1. Nature of SRS

The **Software Requirements Specification (SRS)** defines:

- The system's functional and non-functional requirements.
- Constraints, assumptions, and dependencies.
- Performance and security requirements.

An SRS ensures that:

- All stakeholders have a common understanding.
- Developers build the right product.
- Testers validate the system correctly.

4.2. Characteristics of a Good SRS

A high-quality SRS should have the following characteristics:

1. **Correctness**: Accurately represents stakeholder needs.
2. **Unambiguity**: Avoids vague or unclear statements.
3. **Completeness**: Covers all functional and non-functional requirements.
4. **Consistency**: Does not contradict itself.
5. **Verifiability**: Requirements should be testable.
6. **Modifiability**: Easy to update as requirements evolve.
7. **Traceability**: Requirements should be traceable to their origin.

4.3. Organization of SRS

A well-structured **SRS document** typically follows IEEE 830 standards and includes:

1. **Introduction**
    - Purpose of the system.
    - Scope and objectives.

    - Stakeholders.
2. **Overall Description**
    - User needs.
    - Product functions.
    - System constraints.
3. **Specific Requirements**
    - Functional requirements.
    - Non-functional requirements (performance, security).
    - Interface requirements.
4. **Appendices and References**
    - Additional details.
    - External references.

Advantages of a Well-Organized SRS:

- Provides a clear roadmap for development.
- Reduces misunderstandings between teams.
- Helps in project estimation and planning.
