DEPARTMENT: COMPUTER ENGINEERING
SPECIALTY: SOFTWARE ENGINEERING
OPTION: BACKEND WEB DEVELOPMENT
COURSE TITLE: DATA REQUIREMENT ANALYSIS
LECTURER: Engr. BESONG MICHAEL E.

TABLE OF CONTENTS

Lesson 4: Writing SRS

1. Description
2. Components of a SRS
3. Importance of SRS
4. CASE STUDY

Lesson 5: Conduct a data requirement analysis for a employee attendance system. NB: Deliverables are the database and the SRS

Lesson 1: Software Requirement Analysis.

Definition of SRA

Software Requirement Analysis (SRA) is a crucial phase in the software development lifecycle where the needs and expectations of stakeholders are gathered, analyzed, and documented to ensure a successful project outcome.

Steps in Software Requirement Analysis

1. **Requirement Gathering:**
   - Collect input from stakeholders, such as clients, users, and management, to understand their needs.
   - Techniques: Interviews, surveys, focus groups, observation, and brainstorming.
2. **Requirement Classification:**
   - Categorize requirements into:
     - **Functional Requirements:** What the system should do (e.g., features and functionalities).
     - **Non-Functional Requirements:** How the system should perform (e.g., performance, security, scalability).
     - **Business Requirements:** High-level objectives of the system.
     - **User Requirements:** Expectations and needs from the user's perspective.
3. **Requirement Analysis:**
   - Validate and prioritize requirements based on feasibility, importance, and technical constraints.
   - Address conflicts or ambiguities among stakeholder requirements.
   - Use models like use case diagrams, data flow diagrams (DFDs), or entity-relationship diagrams (ERDs) for clarity.
4. **Requirement Documentation:**
   - Clearly document all requirements in a Software Requirements Specification (SRS) document.
   - The SRS should include functional and non-functional requirements, system constraints, assumptions, and acceptance criteria.
5. **Requirement Validation:**
   - Review requirements with stakeholders to ensure accuracy and completeness.
   - Use prototypes or mockups for better understanding and feedback.

Methods used by SA for requirement gathering

System Analysts (SAs) use various methods to gather requirements effectively during the Software Requirement Analysis (SRA) phase. Here are some commonly used techniques:

1. **Interviews**
- One-on-one or group discussions with stakeholders to understand their needs, expectations, and goals.
- Useful for exploring detailed or complex requirements.

2. **Questionnaires and Surveys**
- Distributing structured forms or questionnaires to stakeholders to collect input.
- Suitable for gathering feedback from a large audience.

3. **Workshops**
- Collaborative sessions with multiple stakeholders to brainstorm, discuss, and prioritize requirements.
- Encourages interaction and consensus-building.

4. **Observation**
- Watching how users currently interact with systems or perform tasks in their environment.
- Helpful for identifying implicit requirements or inefficiencies.

5. **Document Analysis**
- Reviewing existing documentation (e.g., user manuals, business processes, and contracts) to extract relevant information.
- Useful for understanding legacy systems and existing workflows.

6. **Prototyping**
- Developing a mockup or prototype to visualize and validate requirements with stakeholders.
- Helps identify missing or unclear requirements.

7. **Focus Groups**
- Engaging a diverse group of users or stakeholders to discuss needs and expectations collectively.
- Facilitates a variety of perspectives on the system.

8**. Brainstorming**

- Conducting open-ended sessions to generate new ideas and requirements from stakeholders.
- Encourages creativity and inclusivity.

9. **Use Case Analysis**

- Creating use case diagrams or narratives to illustrate how users will interact with the system.
- Ensures functional requirements are well understood.

10. **JAD (Joint Application Development) Sessions**

- Structured meetings involving developers, analysts, and stakeholders to define requirements collaboratively.
- Speeds up the requirement-gathering process.

11. **Interface Analysis**

- Examining interactions between the new system and existing systems, interfaces, or devices.
- Ensures compatibility and seamless integration.

Each method has its strengths and is chosen based on the complexity of the project, the stakeholders involved, and the available resources.

7. Benefits of Requirement Analysis

- **Clear Understanding:** Reduces misunderstandings among stakeholders and developers.
- **Risk Mitigation:** Helps identify potential challenges early on.
- **Scope Definition:** Ensures alignment on what will and won't be delivered.
- **Efficient Development:** Provides a solid foundation for design and development.

8. Challenges in Requirement Analysis

- **Ambiguity:** Vague or unclear requirements.
- **Changing Requirements:** Stakeholder needs may evolve during the project.
- **Communication Gaps:** Misalignment between technical teams and non-technical stakeholders.

9. Case Study

3.9.1. Case Study: Online Education Platform

An online education platform was developed to offer graduate-level courses for working professionals. The goal was to enable students to enroll in courses, access study materials, and interact with instructors online.

Steps in Requirement Analysis:

1. **Requirement Gathering:**
   - **Business Requirements:** Increase student enrollment from 2,000 to 5,000 within two years and reduce operational costs by moving administration online.
   - **User Requirements:** Students needed features like course browsing, enrollment, and access to study materials. Instructors required tools for managing courses and interacting with students.
2. **Requirement Classification:**
   - **Functional Requirements:** User login system, course catalog, payment processing, and discussion should support 500 concurrent users and ensure a response time of under 2 seconds.
3. **Requirement Validation:**
   - Prototypes were created to validate user requirements and gather feedback.
   - Stakeholders reviewed the Software Requirements Specification (SRS) document to ensure alignment.
4. **Challenges:**
   - Managing frequent changes in requirements from stakeholders.
   - Ensuring compatibility with existing university systems.
5. **Outcome:**
   - The platform successfully launched, meeting the business goals and user expectations.

3.9.2. Hotel reservation system

Requirement analysis for a hotel reservation system involves identifying the needs and expectations of stakeholders to design a functional and efficient system. Here's a breakdown of the process:

1. Functional Requirements

- **User Management:** Allow users to register, log in, and manage their profiles.
- **Room Booking:** Enable users to search for available rooms, select dates, and make reservations.
- **Payment Processing:** Support secure online payments and generate invoices.

- **Cancellation and Refunds:** Provide options for booking cancellations and automated refund processing.
- **Admin Panel:** Allow administrators to manage room availability, pricing, and user accounts.
- **Notifications:** Send booking confirmations, reminders, and updates via email or SMS.

2. Non-Functional Requirements

- **Performance:** The system should handle at least 500 concurrent users with minimal latency.
- **Scalability:** Support future expansion, such as adding more hotels or features.
- **Security:** Ensure data protection through encryption and secure authentication methods.
- **Usability:** Provide an intuitive and user-friendly interface for both customers and administrators.
- **Availability:** Ensure 99.9% uptime to avoid disruptions in service.

3. Stakeholder Analysis

- **Customers:** Need a seamless booking experience with clear information on room availability and pricing.
- **Hotel Staff:** Require tools to manage bookings, check-ins, and check-outs efficiently.
- **Administrators:** Need control over system settings, reports, and user management.

4. Tools and Technologies

- **Frontend:** HTML, CSS, JavaScript (React, Angular, or Vue.js).
- **Backend:** Python (Django/Flask), PHP (Laravel), or Node.js (Express).
- **Database:** MySQL, PostgreSQL, or MongoDB for storing user and booking data.
- **APIs:** Integration with payment gateways (e.g., Stripe, PayPal) and notification services (e.g., Twilio).

5. Challenges

- **Data Consistency:** Ensuring accurate room availability across multiple users.
- **Integration:** Seamlessly connecting with third-party services like payment gateways.
- **Scalability:** Designing the system to handle peak booking periods.

6. Documentation

- Create a **Software Requirements Specification (SRS)** document detailing all functional and non-functional requirements, use cases, and system constraints.

Lesson 2: Database Basics

2.1. Definition of terms

- **A Database** is a collection of organized data that can be easily accessed, managed, and retrieved.
- **A DBMS** (Database Management System**)** is a Software used to interact with databases (e.g., MySQL, PostgreSQL, MongoDB).
- **Data redundancy** refers to the duplication of the same data across multiple locations or tables in a database or system.
- **Data coherence** refers to the consistency and logical integrity of data within a database
- A **flat file database** is a simple and straightforward way to store data in a single file, often in a text or spreadsheet format. Unlike relational databases, flat file databases do not use structured tables, relationships, or indexing

2.2. Types of Databases

- **Relational Databases:** Organize data in tables with rows and columns (e.g., MySQL, PostgreSQL, Oracle DB).
- **NoSQL Databases:** Handle unstructured or semi-structured data, often used for large-scale applications (e.g., MongoDB, Cassandra, Redis).
- **In-Memory Databases:** Store data in RAM for faster access (e.g., Redis, Memcached).

2.3. Database Components

- **Tables:** Structured format to store data in rows and columns (used in relational databases).
- **Schemas:** Define the structure, layout, and organization of data in a database.
- **Indexes:** Speed up data retrieval by creating shortcuts.
- **Primary Key:** A unique identifier for each row in a table.
- **Foreign Key:** A field in one table that links to the primary key of another table to establish relationships.

2.4. CRUD Operations

- **Create:** Adding new data to the database.
- **Read:** Retrieving data from the database.
- **Update:** Modifying existing data.
- **Delete:** Removing data from the database.

2.5 Data Models

- **Relational Model:** Represents data in tables and uses SQL for queries.
- **Document Model:** Stores data as JSON-like documents (commonly used in NoSQL databases like MongoDB).
- **Key-Value Model:** Data stored as key-value pairs (e.g., Redis).
- **Graph Model:** Represents data as nodes and relationships (e.g., Neo4j).

2.6 Transactions

A transaction is a sequence of operations performed as a single unit. Transactions respect certain properties known as **ACID Properties.** These properties ensure reliable transactions in databases:

- **Atomicity:** Transactions are all-or-nothing.
- **Consistency:** Data remains valid after a transaction.
- **Isolation:** Transactions don't interfere with each other.
- **Durability:** Changes persist even after failures.

2.7 Database Normalization

- **Purpose:** Reduce redundancy and improve data integrity.
- **Forms:** Ranges from 1NF (First Normal Form) to 5NF, each addressing specific types of redundancy.

2.8 Database Security

- **Authentication:** Ensures only authorized users access the database.
- **Encryption:** Protects data at rest and during transmission.
- **Backup and Recovery:** Safeguards data against accidental loss or corruption.

2.9. Query Languages

- **SQL (Structured Query Language):** Used for relational databases to perform CRUD operations (e.g., SELECT, INSERT, UPDATE, DELETE).
- **NoSQL Query Languages:** Database-specific queries for NoSQL systems (e.g., MongoDB's query syntax).

2.10. Database Design Principles

- **Scalability:** Ensure the database can handle increased data or traffic.
- **Optimization:** Improve query performance using indexing and efficient schema design.
- **Data Integrity:** Maintain accuracy and reliability in the database.

Lesson 3: Relational database modelling

3.1. Definition

It is the process of designing a structured representation of data that defines how data is stored, organized, and related in a **relational database**. This model uses tables (also called relations) to represent entities and their relationships.

3.2. Concepts in Relational Database Modeling

1. **Entities:**
   - Represents a real-world object or concept (e.g., Customer, Order, Product).
   - Each entity is mapped to a table.
2. **Attributes:**
   - Represents the properties or characteristics of an entity (e.g., a Customer entity might have attributes like ID, Name, and Email).
   - Attributes correspond to columns in a table.
3. **Primary Key:**
   - A unique identifier for each row in a table (e.g., CustomerID for the Customer table).
   - Ensures that each record is distinct.
4. **Foreign Key:**
   - A field in one table that references the primary key of another table.
   - Used to establish relationships between tables.
5. **Relationships:**
   - **One-to-One (1:1):** A single record in one table is related to a single record in another table.
   - **One-to-Many (1:N):** A single record in one table is related to multiple records in another table.
   - **Many-to-Many (M:N):** Multiple records in one table are related to multiple records in another table (usually implemented using a junction table).

3.3. Steps in Relational Database Modeling

1. **Identify Entities:**
   - Determine the main objects/concepts in the system (e.g., Customer, Order, Product).
2. **Define Attributes:**
   - Specify the relevant properties for each entity.
3. **Establish Relationships:**
   - Map how entities interact with each other using relationships (e.g., a Customer places an Order).
4. **Design Primary and Foreign Keys:**
   - Assign unique identifiers and create references for linked tables.
5. **Create an Entity-Relationship (ER) Diagram:**
   - Visualize entities, attributes, and their relationships using tools like ER diagrams.

ERD

CDM

CASE STUDY

Lesson 4: Writing SRS

4.1. Description

A **Software Requirements Specification (SRS)** is a comprehensive document that outlines the functional and non-functional requirements of a system or application. It serves as a blueprint for developers, stakeholders, and project managers, ensuring everyone has a clear understanding of what the system is supposed to achieve. Here's how to write one effectively:

4.2. Components of a SRS

1. Title Page

- Include the project title, date, version number, and the authors of the document.

2. Table of Contents

- Provide a clear and organized list of sections and subsections for easy navigation.

3. Introduction

- **Purpose:** State the objectives and scope of the software.
- **Document Overview:** Summarize the structure of the SRS document.
- **Stakeholders:** Identify the audience for the SRS, such as developers, project managers, and end-users.

4. Overall Description

- **System Context:** Provide a high-level overview of the system and its environment.
- **Business Objectives:** Describe the goals the software aims to achieve.
- **Constraints:** Mention any limitations, such as deadlines, technology stack, or budget.
- **Assumptions and Dependencies:** List assumptions about system usage or dependencies on other systems/components.

5. Functional Requirements

- **Features:** Break down the system's major functionalities, such as user authentication, data entry, or reporting.
- **Use Cases:** Describe how users will interact with the system using use case diagrams or narratives.
- Example:
    - *Login System: Users must log in with valid credentials to access the dashboard.*

6. Non-Functional Requirements

- **Performance:** Define response times, maximum concurrent users, etc.
- **Scalability:** Indicate how the system will handle growth in data or users.
- **Security:** Describe measures like encryption, authentication, or data protection.
- **Usability:** Highlight design guidelines for an intuitive user experience.

7. System Models

- **Data Flow Diagrams (DFDs):** Represent the flow of information within the system.
- **Entity-Relationship Diagrams (ERDs):** Define the relationships between data entities.
- **Architectural Design:** Provide a high-level description of the system architecture (e.g., client-server, microservices).

8. Constraints

- Outline any technical, legal, or environmental constraints, such as platform limitations or compliance requirements.

9. Acceptance Criteria

- Define how the system's functionality will be tested to ensure it meets requirements.
- Example: *The system must support up to 1,000 concurrent users without exceeding a response time of 2 seconds.*

10. Glossary

- Include definitions of technical terms and acronyms for clarity.

11. Appendices

- Add any supplemental information, such as references, diagrams, or additional notes.

Writing Tips

- Use clear and concise language.
- Collaborate with stakeholders to ensure the requirements are accurate.
- Prioritize requirements to focus on critical functionalities.

4.3. Importance of SRS

A **Software Requirements Specification (SRS)** is incredibly important for the success of a software development project. Here's why it plays a critical role:

1. Clear Communication

- Acts as a bridge between stakeholders (clients, developers, and managers) by ensuring everyone has a shared understanding of the software's objectives and requirements.

2. Prevents Misunderstandings

- Minimizes ambiguity by clearly defining functional and non-functional requirements.
- Reduces the risk of building a system that doesn't meet user expectations.

3. Foundation for Development

- Serves as a roadmap for developers, guiding them in designing, coding, and testing the system.

4. Ensures Scope Control

- Prevents scope creep by setting boundaries for what the software will and won't include.
- Helps track changes to requirements during the project's lifecycle.

5. Aids in Testing

- Provides a basis for writing test cases and ensures that every functionality is verified against specified requirements.

6. Saves Time and Cost

- By identifying potential issues early in the requirement analysis phase, the SRS helps avoid costly changes later in the development process.

7. Supports Project Management

- Facilitates task planning, resource allocation, and timeline estimation.
- Allows project managers to monitor progress and ensure alignment with requirements.

8. Facilitates Maintenance

- Provides comprehensive documentation that future developers can refer to when updating or fixing the system.

CASE STUDY

Software Requirements Specification (SRS) for Hotel Reservation System

1. Introduction

- **Purpose:** The purpose of this SRS is to define the functional, non-functional, and system requirements for the development of a Hotel Reservation System. The system will allow users to book rooms, manage reservations, and process payments efficiently.
- **Scope:** The Hotel Reservation System will cater to hotel guests, administrators, and staff by providing functionalities such as room bookings, payment processing, and availability management. The system will improve operational efficiency and customer satisfaction.
- **Definitions, Acronyms, and Abbreviations:**
  - SRS: Software Requirements Specification
  - CRUD: Create, Read, Update, Delete
  - GUI: Graphical User Interface
- **References:**
  - User feedback documents
  - Design standards for hotel management systems

2. Overall Description

- **System Context:** The system will act as an intermediary between customers and hotel staff by automating tasks such as room reservations, billing, and room availability updates.
- **System Objectives:**
  - Reduce manual errors in bookings.
  - Provide real-time room availability updates.
  - Streamline the payment and billing process.
- **Assumptions and Dependencies:**
  - The system will require internet connectivity for real-time updates.
  - Integration with third-party payment gateways will be necessary.

3. Functional Requirements

- The system shall allow customers to search for rooms based on check-in and check-out dates.
- The system shall enable users to create, modify, and cancel bookings.
- The system shall process payments through secure payment gateways.

- The admin panel shall allow staff to manage room availability and pricing.

4. Non-Functional Requirements

- **Performance:** The system must handle 500 concurrent users with a response time of under 2 seconds.
- **Usability:** The GUI must be intuitive and provide clear navigation for customers.
- **Security:** All payment data must be encrypted using SSL/TLS protocols.
- **Scalability:** The system must support adding more hotels to the platform in the future.

5. System Models

- **Use Case Diagram:**
  - Actors: Customers, Hotel Staff, Admins
  - Use Cases: Search rooms, Book rooms, Make payments, Manage bookings
- **Entity-Relationship Diagram (ERD):**
  - Tables: Customers, Rooms, Reservations, Payments

6. Acceptance Criteria

- The system shall successfully process at least 95% of bookings without errors.
- Users shall be able to complete a booking within 5 minutes on average.

7. Glossary

- **Reservation:** The act of booking a hotel room for specific dates.
- **Payment Gateway:** A service used to process online transactions.

8. Appendices

- Sample wireframes for the booking interface
- References to design patterns used