SEPM
IU2341231551
6CSE-F

Assignment - 2

1] Explain software Requirement Analysis & specification in detail

→ Software Requirement Analysis & Specification is a crucial phase in the SDLC where the functional and non-functional requirements of the system are gathered, analyzed, and documented.

1. Software Requirment analysis

→ Software Requirement analysis is the process of collecting, understanding, and refining user needs into well-defined software requirements. It ensures that the final software meets the expectations of users and stakeholders.

- Types of Requirements

1. Functional Requirements: Define the core functionalities of the software.
2. Non-Functional Requirements: Define quality attributes like performance, security, and usability.
3. Domain Requirements: Specific needs related to the industry or organization.

2. Software Requirement specification (SRS)

⇒ SRS is a formal document that details the software requirements in a structured manner. It acts as a contract between developers and stakeholders ensuring clarity and mutual understanding

• characteristics of good SRS

⇒ Correct :- Accurately defines all system requirements
⇒ Unambiguous :- Clearly written with no vague terms
⇒ Complete :- Covers all functional and non-functional requirements.
⇒ Consistent :- No contradictions between different requirements
⇒ Verifiable :- Each requirement can be tested.
⇒ Modifiable :- Easy to update when requirements change.

⇒ SRS provides a reference for software developers and testers. It helps in estimating costs and project timelines

⇒ SRS Acts a foundation for system design and architecture. Ensures all stakeholders expectations are documented and met.

2] Explain Requirement Analysis and Requirement engineering tasks

1. Requirement Analysis

→ Requirement Analysis is the process of gathering, understanding, analyzing, and refining the needs of users and stakeholders to define the functionalities and constraints of a software system.

→ It ensures that the final product meets the expectations and business objectives.

2. Requirement Engineering tasks

→ Requirements Engineering is a systematic approach to eliciting, documenting, analyzing, verifying and managing software requirements throughout the software development life cycle.

1. Requirement Elicitation

→ Gathering requirements from stakeholders using methods such as:

- Interviews, surveys, Questionnaires
- Prototyping
- Brainstorming
- Observation.

2. Requirement specification

→ Documenting the gathered requirements in an organized manner in the SRS document

→ SRS includes functional requirements, non-functional requirements and constraints and assumptions.

3. Requirement Analysis & Negotiations

→ Analyzing requirements for completeness, correctness, consistency, and feasibility.

→ Resolving conflicts between different stakeholders

4. Requirement validation

→ Ensuring that the requirements accurately describe the system to be developed.

→ Method used for validation :-
Reviews and Inspections, Prototyping, Test case generation

5) Requirement Management

→ Handling changes in requirements during software development. Maintaing a version control system for req. changes

3] Define design concepts: abstraction, modularity, information hiding, functional independent cohesion and coupling.

i Abstraction

→ Abstraction is the process of hiding complex implementation details and exposing only the essential features of a system.

ii Modularity

→ Modularity refers to dividing a software system into smaller, manageable, and independent units called modules.

iii Information hiding

→ Information hiding is the principle of restricting access to certain parts of a module and exposing only what is necessary..

iv Functional Independence

→ Functional independence ensures that each module performs a single function with minimal dependency on other modules.

v Cohesion

→ Cohesion refers to the degree to which the elements of a module work together to achieve a single purpose.

vi coupling

→ Coupling is the degree of dependency between different modules in a software system.

4. Explain types of testing in brief

→ Software testing is the process of evaluating a software application to ensure it meets the specified requirements and is free from defects.

i Functional Testing

→ Functional testing verifies that the software functions correctly according to the given requirements.

- Unit testing: Tests individual components or modules.
- Integration Testing: Tests the interaction between different modules.
- System Testing: Tests the complete application as whole

ii Non-Functional testing

→ Non-functional testing evaluates the performance, security, usability, and reliability of the application.

- Performance testing: checks how the system performs under different conditions.

- Load testing :- Measures system behavior under expected user load
- Stress testing :- Tests the system under extreme conditions to identify its breaking point.
- Security testing :- Ensures data protection and prevents unauthorized access.

iii) White - Box

⇒ White-box testing is a code-based testing is code-based testing method where the internal structure, logic, and code flow are tested.

→ It requires programming knowledge and helps in optimizing performance, but it is time-consuming

iv) Black-Box testing

⇒ Black-box testing evaluates software functionality without looking at the internal code, focusing on user inputs and expected outputs.

→ It is useful for detecting missing requirements missing requirements but cannot identify internal code errors.

5. Explain software design concepts and design principles

i Abstraction :- Abstraction is the process of hiding implementation details and exposing only necessary functionality

ii Modularity :- The principle of dividing software into independent, self-contained modules that can be developed and tested seperately

iii Information hiding prevents direct access to internal components of a module, exposing only required interfaces.

iv Coupling :- The degree of interdependency between modules. Low coupling is preferred for maintainability.

v Cohesion . The degree to which elements within a module work together. High cohesion is desirable.

- Software Design Principle

i SOLID principle

⇒ Single Responsibility principle (SRP) :- A class should have only one reason to change

- O - Open/closed Principle (OCP)

→ Software entities should be open for extension but closed for modification.

- L - Liskov substitution principle (LSP)

→ A subclass should be replaceable with its parent class without affecting functionality

- I - Interface segregation principle (ISP)

→ clients should not be forced to depend on interfaces they do not use

- D - Dependency Inversion principle (DIP)

→ High-level modules should not depend on low level modules but rather on abstractions.

ii DRY (Don't Repeat Yourself)

→ Avoid code duplication by using functions, classes, or reusable modules.

iii YAGNI (You Ain't Gonna Need it)

→ Avoid implementing unnecessary features that are not required in the current scope.

6 What is Architectural Design and component level design? Explain in detail.

i Architectural Design

⇒ Architectural design is the high-level structure of a software system defining how components interact with each other. It serves as a blueprint for software development, focusing on the overall framework, modules, and their relationships.

1. Defines system structure :- Identifies the main components and their interaction in the system.

2. Specifies communication mechanisms :- Determines how modules communicate.

3. Ensures Scalability and Performance :- Helps in designing a system that handles growth efficiently

4. Improves Maintainability and Reusability :- promotes modular development for easy updates and code reuse.

For Ex. A banking system layers into UI Layer, Business Logic layer, DB layer.

ii Component-level Design

→ Component-level design focuses on designing individual software components within the architecture. A component is a self-contained module that provides specific functionality and can be reused.

1. Defines Functionality of Each Module:
   Specifies what each component does and how it interacts with others.

2. Encapsulation and Modularity:-
   Each component should be independent exposing only necessary data.

3. Data Flow between components:-
   Defines input, output and communication method.

4. Reusability:-
   Components should be designed for reuse in different parts of the application.

- For Ex, In library management system different components are:-
  - User Authentication component
  - Book management system
  - Transaction component.

7 Explain function oriented design and DFD with example of Library management system.

i Function - oriented Design

→ It is a top-down approach that focuses on designing the software based on functions (Process) rather than objects.

● → It decomposes a system into smaller functional modules, making it easier to manage and develop.

1 Modular Approach :- Breaks down the system into hierarchical functional modules.

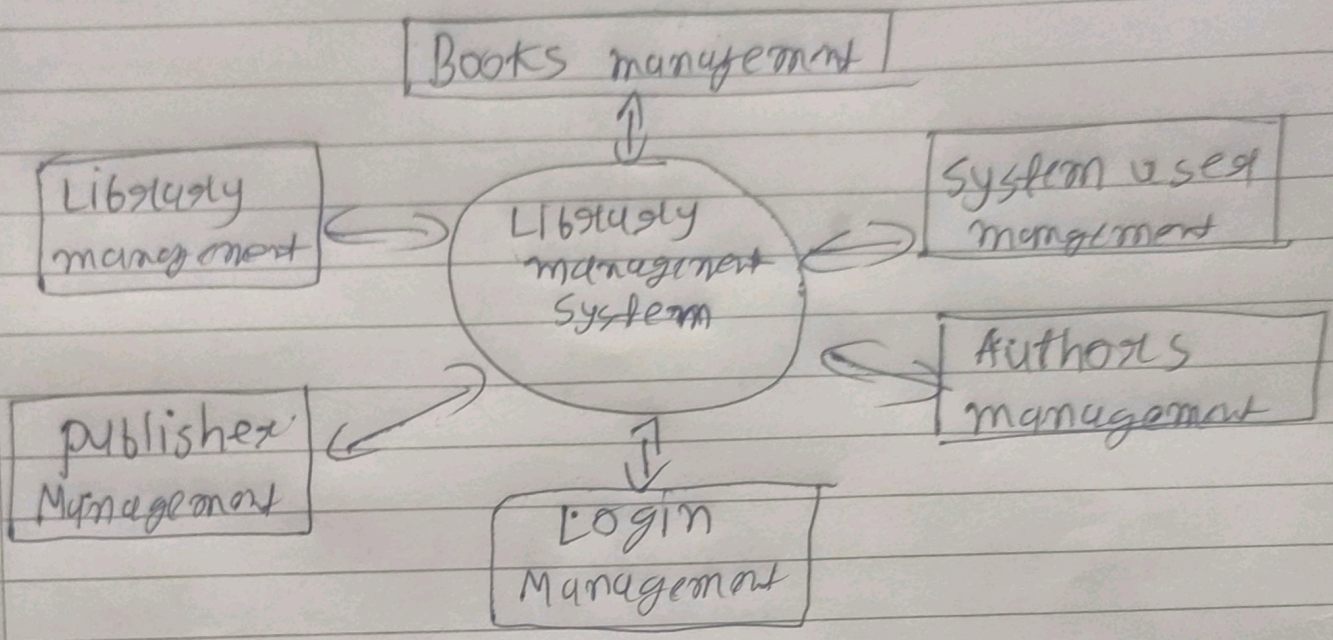
2. DFD :- Uses Data flow to represent data movement between modules.

● 3. Process - oriented :- Each function performs a specific task and interacts with data

ii Data Flow Diagram (DFD)

→ A DFD is a graphical representation of how data flows through a system, depicting processes, data stores, external entities, and data flow connections.

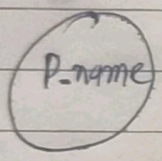
- Level zero dfd for Library management



8 What is Data flow diagram and leveling of DFD Draw Level 0, level 1 and level 2 for LMS

→ A DFD is a graphical representation of how data flows through a system. It shows the processes, external entities, data stores, and data movement within a system, helping in understanding how information is processed and transferred.

- Process

→ 

It is denoted by circle. Represents a function that transforms incoming data into outgoing data.

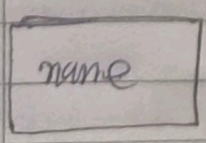
- Data Flow

→ It is denoted Arrow. Shows the movement of data between processes, data stores, or external entities.

- Data store

—— It is denoted within parallel lines. Represents
—— a place where data is stored

- Entity

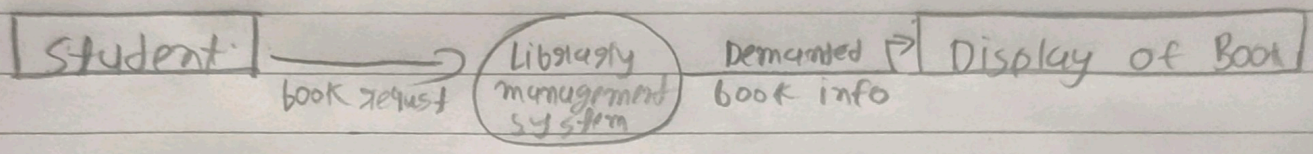


It is denoted by Rectangle. Represents sources or destinations of data outside the system.

→ DFD is designed in multiple levels starting from a high-level overview and progressively breaking it down into detailed components

1 Level 0 DFD

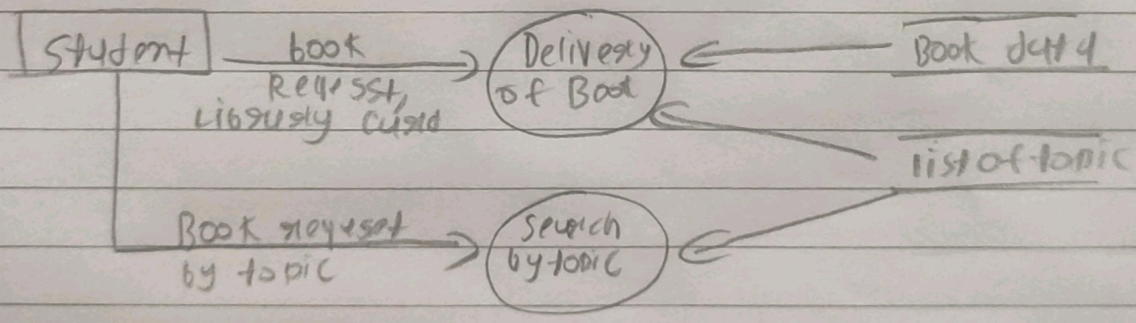
→ This is the highest level of DFD. Represents the entire system as a single process.



2 Level 1 DFD

→ Breaks down the single process from level 0 into main subprocesses.

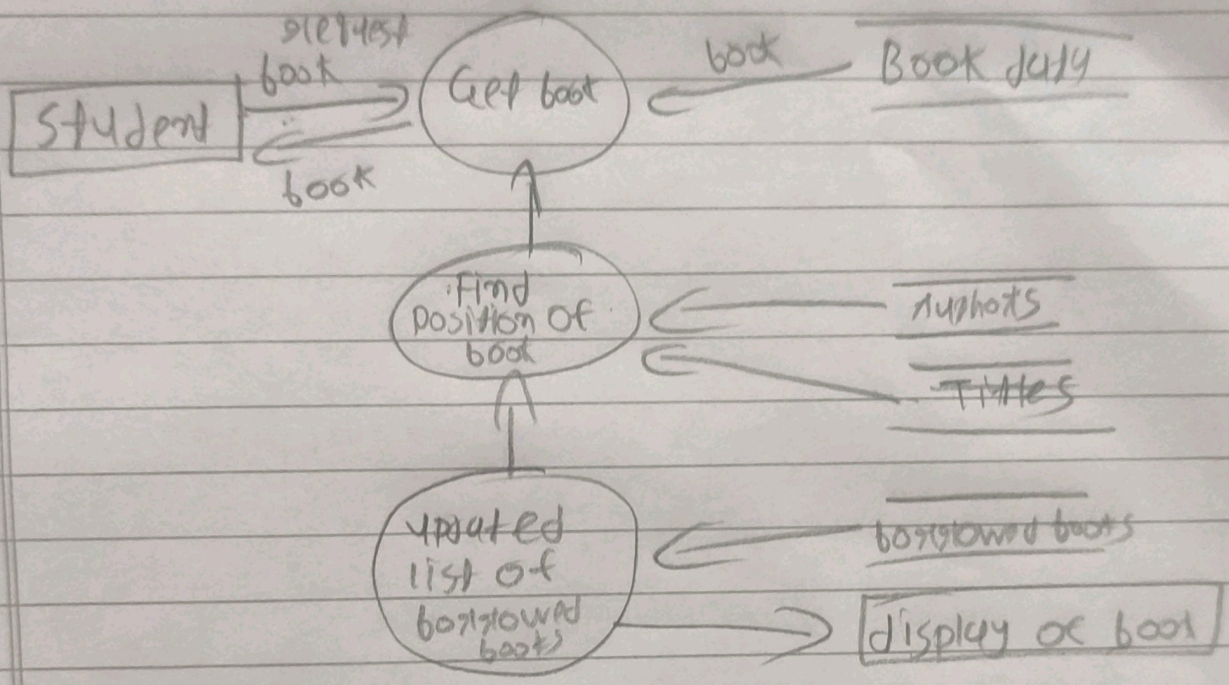
→ Shows how data moves between processes.



3 Level 2 DFD

→ Further breaks down the subprocesses from level 1 into more detailed steps.

→ Each process in level 1 is expanded to show specific actions within that proces


request book
Student
book
Get book
book
Book data
Find position of book
Authors
Titles
updated list of borrowed books
borrowed books
display of book