Day 2

Software Development
&
Project Planning

Workshop By:

Dr. Vishal Bharvesh

Ishanvi
Info-Soft Pvt. Ltd.

Introduction to Design & Development

- Design means planning how system will work
- Development means actual coding and implementation
- It connects idea with real software

Importance of Design Phase

1. **Gives clear system structure**
   Provides a well-defined blueprint of system components, modules, and interactions, making development organized, understandable, and easier for the entire team.

2. **Reduces errors during coding**
   Clear design minimizes confusion, helps developers understand logic properly, and reduces chances of mistakes, bugs, and rework during coding phase.

3. **Saves time and cost**
   Proper planning avoids unnecessary changes, reduces development delays, minimizes resource wastage, and ensures project completion within estimated time and budget.

4. **Improves efficiency of system**
   Well-designed systems perform better, use resources efficiently, respond faster, and provide better user experience with optimized performance and scalability.

High-Level Design (HLD)

1. **Defines overall system architecture**
   Describes the complete structure of the system, including layers, components, and technologies, providing a foundation for development and system organization.

2. **Shows modules and components**
   Identifies different modules and system components, their roles, and responsibilities, helping developers understand how the system is divided and managed effectively.

3. **Explains data flow**
   Illustrates how data moves between different modules, processes, and layers, ensuring proper communication, integration, and smooth functioning of the entire system.

4. **Gives big picture of system**
   Provides a high-level overview of the entire system, helping stakeholders and developers understand overall functionality, structure, and purpose before detailed development begins.

Low-Level Design (LLD)

1. **Detailed design of each module**
   Provides in-depth design of individual modules, including functions, inputs, outputs, and internal workings, ensuring clarity before actual coding begins properly.

2. **Includes algorithms and logic**
   Defines step-by-step algorithms and logical flow for each function, helping developers understand how tasks should be processed within the system correctly.

3. **Defines database structure**
   Specifies tables, fields, relationships, keys, and constraints, ensuring proper data organization, storage, and retrieval for efficient system performance and consistency.

4. **Helps developers in coding**
   Acts as a clear guide for developers, reducing confusion, improving coding speed, and ensuring implementation matches the planned system design accurately.

System Architecture

1. **Defines how system components interact**

   Explains how different parts of the system communicate, exchange data, and work together to ensure smooth functionality and proper coordination.

2. **Types: Monolithic, Client-Server, 3-Tier**

   Describes different architecture models, showing how systems can be structured based on complexity, scalability, and distribution of responsibilities across components.

3. **Important for scalability and performance**

   Proper architecture ensures system can handle growth, improve speed, manage load efficiently, and deliver better performance under increasing user demand.

3-Tier Architecture

1. **Presentation Layer (UI)**
   Handles user interface, displaying information and collecting input from users, ensuring interaction between user and system is simple and effective.

2. **Application Layer (Logic)**
   Processes user input, applies business rules, performs calculations, and controls overall system functionality, acting as the core working layer of application.

3. **Database Layer (Data)**
   Stores, retrieves, and manages data efficiently, ensuring data integrity, security, and consistency for smooth functioning of the entire system operations.

4. **Separates concerns for better design**
   Divides system into independent layers, improving maintainability, scalability, flexibility, and making it easier to update or modify specific parts independently.

Database Design

1. **Define tables and fields:**
   Define tables to store structured data and fields to represent attributes, ensuring clarity, normalization, and data retrieval operations in systems.

2. **Use primary and foreign keys:**
   Use primary keys to uniquely identify records and foreign keys to link related tables, enforcing referential integrity across database structures.

3. **Maintain relationships:**
   Maintain relationships between tables using defined constraints and joins, enabling accurate data association, efficient querying, and logical organization of information.

4. **Ensure data consistency:**
   Ensure data consistency by applying constraints, validations, and transactions, preventing anomalies, duplication, and conflicts while preserving accuracy and reliability overall.

ER Diagram

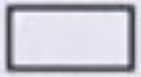
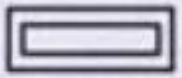
1. **Represents database visually:**
   Represents the database visually using diagrams, making complex structures easier to interpret, analyze, and communicate effectively among developers and stakeholders.

2. **Shows entities and attributes:**
   Shows entities as objects and attributes as their properties, helping to clearly define data elements and their characteristics within the system.

3. **Defines relationships:**
   Defines relationships between entities, indicating how data elements are connected, including one-to-one, one-to-many, and many-to-many associations clearly.

4. **Helps in clear understanding:**
   Helps in clear understanding of database design, reducing confusion, improving communication, and supporting efficient development, maintenance, and future system scalability efforts.

ER Diagram

Symbols	Figures	Represents
	Rectangle	Entities in ER Model
	Ellipse	Attributes in ER Model
	Diamond	Relationships among Entities
	Line	Attributes to Entities and Entity Sets with Other Relationship Types
	Double Ellipse	Multi-Valued Attributes
	Double Rectangle	Weak Entity

UI/UX Design

1. **Focus on user-friendly interface:**
   A user-friendly interface helps users understand features quickly, interact smoothly, reduce confusion, and complete tasks efficiently without extra guidance easily.

2. **Simple navigation and layout:**
   Simple navigation and layout allow users to move easily, find information faster, and perform actions without unnecessary steps or complexity.

3. **Consistent design improves usability:**
   Consistent design improves usability by maintaining uniform colors, fonts, and layouts, helping users learn faster, feel comfortable, and interact confidently.

Development Phase

1. **Convert design into code:**
   Convert system design into working code by implementing planned features, structures, and logic using appropriate programming practices and standards carefully.

2. **Use programming languages and tools:**
   Use suitable programming languages and development tools to build, test, and deploy the system efficiently, ensuring reliability, performance, and maintainability.

3. **Follow modular approach:**
   Follow modular approach by dividing system into smaller parts, making development easier, improving readability, testing, debugging, and future enhancements effectively.

Technology Stack

1. **Frontend: HTML, CSS, JS:**
   Frontend technologies like HTML CSS and JavaScript are used to design user interfaces create layouts and ensure interactive user experiences.

2. **Backend: PHP, Python, Java:**
   Backend technologies such as PHP Python and Java handle server side logic processing requests managing data and ensuring functionality securely.

3. **Database: MySQL:**
   Database like MySQL stores organizes and retrieves data efficiently supports queries maintains integrity and enables secure data management within applications.

4. **Choose based on project needs:**
   Technology selection should depend on project requirements scalability budget performance and team skills ensuring the suitable and efficient solution overall.

Coding Standards

1. **Use proper naming conventions:**
   Use meaningful and consistent naming conventions for variables functions and classes to improve clarity understanding and maintainability of the code.

2. **Write clean and readable code:**
   Write simple clean and well structured code using proper formatting and indentation so that it is easy to read understand.

3. **Add comments for understanding:**
   Add appropriate comments in code to explain logic functionality and complex parts so that others can easily understand and maintain it.

Modular Development

1. **Divide system into modules:**
   Break the system into smaller modules so each part handles a specific task making development simple organized and manageable.

2. **Develop independently:**
   Each module can be developed and tested separately allowing multiple developers to work simultaneously and reducing overall development time effectively.

3. **Easy debugging and maintenance:**
   Modular structure helps quickly identify errors fix issues easily and update specific parts without affecting the entire system functionality.

Version Control

1. **Track changes in code:**
   Helps record every modification in code over time making it easy to review history, revert mistakes, and maintain project integrity.

2. **Use tools like Git:**
   Version control tools like Git help manage code efficiently, track changes, create branches, and collaborate smoothly among multiple developers.

3. **Supports teamwork:**
   Allows multiple team members to work on same project simultaneously, manage conflicts, share updates, and maintain coordination throughout development process effectively.

Integration

1. **Combine all modules:**
   All developed modules are integrated together to form a complete system ensuring each part works correctly with others.

2. **Ensure proper communication:**
   Check that modules exchange data correctly using defined interfaces so there are no errors or mismatches during interaction between components.

3. **Check system flow:**
   Verify the overall workflow of the system to ensure smooth execution of processes without interruptions, delays, or logical errors.

Testing Overview

1. **Unit Testing, Integration Testing, System Testing:**
   Different testing levels verify individual modules, combined components, and complete system to identify errors and ensure proper functionality.

2. **Ensures system works correctly:**
   Testing confirms that the entire system performs as expected, meets requirements, and delivers accurate results without failures or unexpected behavior.

Debugging

1. **Find and fix errors:**
   Testing helps identify bugs, issues, and defects in system so developers can correct them before final deployment and user usage.

2. **Improve system quality:**
   Fixing errors and refining performance ensures the system becomes more reliable, efficient, stable, and delivers better user experience consistently.

Documentation

1. **Prepare technical documents:**
   Create detailed documentation explaining system design, architecture, code structure, and workflows to support development, understanding, and future updates effectively.

2. **Create user manuals:**
   Develop simple guides for users explaining system usage, features, navigation steps, and troubleshooting to ensure smooth and independent system operation.

3. **Helps future maintenance:**
   Proper documentation helps developers understand system easily, fix issues quickly, and make updates or enhancements without confusion or loss of information.

Common Mistakes

1. **Poor design planning:**
   Lack of proper planning leads to unclear structure, confusion during development, increased errors, and difficulty in scaling or maintaining system later.

2. **No documentation:**
   Without documentation, understanding system becomes difficult, causing problems in maintenance, updates, and knowledge transfer between team members over time.

3. **Ignoring testing:**
   Skipping testing leads to undetected bugs, system failures, poor performance, and unreliable software that may not meet user expectations or requirements.

4. **Hardcoding values:**
   Using fixed values in code reduces flexibility, makes updates difficult, increases errors, and affects scalability and maintainability of the overall system.

Best Practices

1. **Plan before coding:**
   Proper planning helps define clear requirements, structure, and workflow, reducing confusion, saving time, and ensuring smoother development process from the beginning.

2. **Keep design simple:**
   Simple design makes system easy to understand, develop, and maintain, reducing complexity, minimizing errors, and improving overall efficiency and performance.

3. **Test regularly:**
   Regular testing helps detect issues early, improves system reliability, ensures continuous improvement, and prevents major problems during later stages of development.

4. **Maintain clean code:**
   Writing clean code improves readability, reduces errors, simplifies debugging, supports teamwork, and makes future updates or maintenance easier and more efficient.

End of Day 2

Thanks you