Module 1 – Overview of IT Industry

* **What is a Program?**

**THEORY EXERCISE: Explain in your own words what a program is and how it functions.**

A program is a set of instructions written in a programming language that a computer can execute to perform a specific task. A program tells the computer what to do and how to do it.

How a program functions:

Writing the code: A programmer writes instructions in a programming language

Translation: The code is either compiled or interpreted .

Execution: The computer’s processor follows these instructions one by one to perform the desired operations.

Output: Based on the instructions, the program produces results, like showing text on the screen, saving files, or controlling devices.

* What is Programming?

Programming is the process of writing instructions in a programming language that a computer can understand and execute. These instructions tell the computer how to perform specific tasks or solve problems.

THEORY EXERCISE: What are the key steps involved in the programming process?

## Key Steps Involved in the Programming Process:

**1. Problem Definition**

* Clearly understand and define the problem you are trying to solve.
* Identify input, output, and constraints.

**2. Planning the Solution**

* Decide how to approach the problem logically.
* Choose the right algorithms and data structures.
* Sometimes create flowcharts or pseudocode.

**3. Writing the Code**

* Convert the logic into actual code using a programming language (e.g., Python, Java, C++).
* Follow proper syntax and good coding practices.

**4. Compiling/Interpreting**

* Translate the source code into machine code (compiled) or execute directly (interpreted).
* Check for syntax errors.

**5. Testing and Debugging**

* Run the program with different inputs to test all conditions.
* Fix any bugs (errors) or logical mistakes found during testing.

**6. Documentation**

* Write comments in the code and create user manuals or documentation.
* Helps others (or future you) understand and maintain the program.

**7. Maintenance**

* Update the program when needed (e.g., bug fixes, new features).
* Ensure long-term usability and performance
* Types of Programming Languages.
* Low-Level programming Languages
* High-Level programming Languages
* Procedural Programming Languages
* Object-Oriented Programming Languages
* Scripting programming Languages
* Functional Programming Languages
* Logic Programming Languages

THEORY EXERCISE: What are the main differences between high-level and low-level programming languages?

| **Feature** | **High-Level Language** | **Low-Level Language** |
| --- | --- | --- |
| **Definition** | Easy-to-understand languages for humans | Hardware-near languages understood by computers |
| **Syntax** | Simple, human-readable (e.g., print("Hello")) | Complex, machine or symbolic (e.g., MOV A, B) |
|  |  |  |
| **Execution Speed** | Slower than low-level | Very fast, as it's close to machine instructions |
| **Memory Control** | Less control over memory | Full control over memory |
| **Portability** | Portable across different platforms | Not easily portable, hardware-specific |
| **Examples** | Python, Java, C++, JavaScript | Machine Code (0s & 1s), Assembly Language |
| **Use Case** | Web, apps, games, enterprise software | Embedded systems, OS, device drivers |
| **Compiler/Interpreter** | Required to convert into machine code | Machine code runs directly |

* World Wide Web & How Internet Works

## World Wide Web (WWW)

The World Wide Web (WWW) is a system of webpages and websites linked together and accessed via the Internet.

Uses web browsers (like Chrome) to view pages.

Each webpage has a URL (e.g., [www.google.com](http://www.google.com)).

## How the Internet Works

User types a URL in the browser.

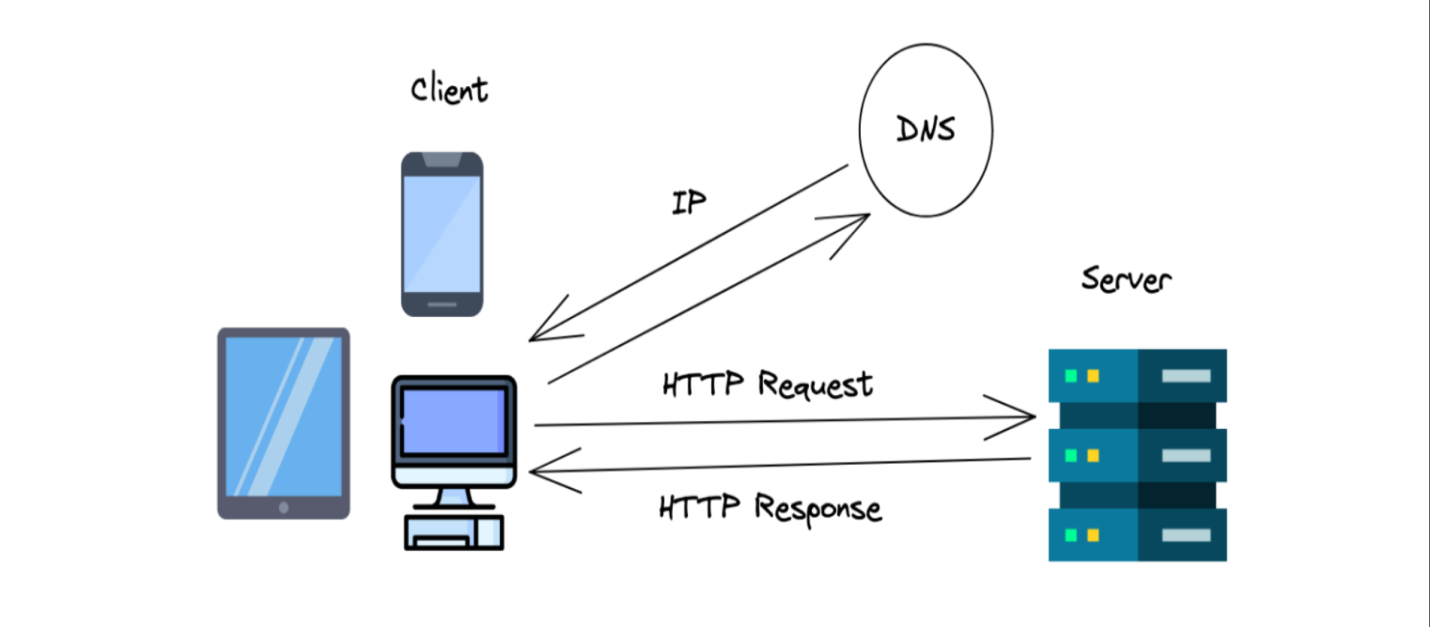
DNS (Domain Name System) converts it to an IP address.

A request is sent to the web server.

The server responds with the webpage.

The browser displays the page.

LAB EXERCISE: Research and create a diagram of how data is transmitted from a client to a server over the internet.



THEORY EXERCISE: Describe the roles of the client and server in web communication. Network Layers on Client and Server

### Roles of Client and Server in Web Communication

Client:

The client is typically a user’s device or application (like a web browser).

It initiates communication by sending a request to the server. For example, when you enter a URL in your browser, the client sends an HTTP request.

The client waits for the server's response.

It processes and displays the received data, like rendering a webpage.

Server:

The server is a powerful computer or program that waits for client requests.

When it receives a request, the server processes

The server sends back a response to the client .

Servers can handle requests from many clients simultaneously.

Network Layers on Client and Server:

Client-Side Layers:

Presentation Layer: This layer handles the user interface and presentation of information to the user.

Application Layer: This layer includes the client application software that interacts with the server.

Transport Layer: This layer ensures reliable data delivery between the client and server, often using protocols like TCP.

Network Layer: This layer manages the addressing and routing of data packets across the network.

Data Link Layer: This layer provides physical data transmission and error detection.

Physical Layer: This layer handles the physical transmission of data over the network.

Server-Side Layers:

Presentation Layer: This layer prepares the data for transmission to the client.

Application Layer: This layer includes the server application software that handles client requests.

Transport Layer: This layer ensures reliable data delivery between the server and client.

Network Layer: This layer manages the addressing and routing of data packets across the network.

Data Link Layer: This layer provides physical data transmission and error detection.

Physical Layer: This layer handles the physical transmission of data over the network.

LAB EXERCISE: Design a simple HTTP client-server communication in any language.

* Simple HTTP Server in C

// server.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char response[] =

"HTTP/1.1 200 OK\r\n"

"Content-Type: text/plain\r\n"

"Content-Length: 18\r\n"

"\r\n"

"Hello from server!";

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(8080);

bind(server\_fd, (struct sockaddr\*)&address, sizeof(address));

listen(server\_fd, 5);

printf("Server running on port 8080...\n");

while (1) {

new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen);

read(new\_socket, NULL, 0); // Just wait for client

write(new\_socket, response, strlen(response));

close(new\_socket);

}

return 0;

}

* **Simple HTTP Client in C**

// client.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

int main() {

int sock;

struct sockaddr\_in server;

char message[] = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n";

char buffer[1024] = {0};

sock = socket(AF\_INET, SOCK\_STREAM, 0);

server.sin\_family = AF\_INET;

server.sin\_port = htons(8080);

inet\_pton(AF\_INET, "127.0.0.1", &server.sin\_addr);

connect(sock, (struct sockaddr\*)&server, sizeof(server));

send(sock, message, strlen(message), 0);

read(sock, buffer, sizeof(buffer) - 1);

printf("Server response:\n%s\n", buffer);

close(sock);

return 0;

}

**THEORY EXERCISE: Explain the function of the TCP/IP model and its layers. Client and Servers**

## TCP/IP Model – Function and Layers

TCP/IP stands for Transmission Control Protocol / Internet Protocol. It is a foundational networking model that defines how data is transmitted over the internet and other networks.

### 🔷 The TCP/IP Model has 4 layers:

|  |  |
| --- | --- |
| Layer Name | Function |
| 1. Application | This layer interacts directly with user applications. It handles protocols like HTTP, FTP, SMTP, etc., enabling communication between software and the network. |
| 2. Transport | Responsible for reliable data transfer between devices. It manages error checking and data flow control. |
| 3. Internet | Manages logical addressing and routing of data packets across networks using IP (Internet Protocol). It decides the best path for data to reach its destination. |
| 4. Network Access | Handles the physical transmission of data over the network hardware such as Ethernet, Wi-Fi, and other link layer protocols |

## Client and Server in Network Communication

### 🔹 Client:

* A client is a device or program that requests services or resources from another device.
* Examples: Web browsers (Chrome, Firefox), email clients.

### 🔹 Server:

* A server is a device or program that provides services or resources to clients.
* Examples: Web servers (Apache, Nginx), mail servers.

### 🔁 How they communicate:

1. The client sends a request to the server (e.g., to fetch a webpage).
2. The server processes the request.
3. The server sends back a response (e.g., the webpage data).
4. The client displays or uses the response.

THEORY EXERCISE: Explain Client Server Communication

**Client-server communication** is a model used in computer networks where two devices – a **client** and a **server** – communicate with each other to share data, services, or resources.

**🔹 Client:**

* The client is the **requester**.
* It is usually a user's device (like a web browser, mobile app, or desktop software).
* The client **sends requests** to the server to access data or services.

**🔹 Server:**

* The server is the **provider**.
* It is a powerful computer or system that **listens for client requests** and **responds** with the required information or services.

**🔁 How Communication Happens:**

1. The **client** initiates communication by sending a **request** (e.g., asking for a webpage).
2. The **server** receives the request, **processes** it, and prepares a **response**.
3. The **server** sends the response back to the **client**.
4. The **client** receives the data and displays or uses it.

* **Types of Internet Connections**

**1. Dial-Up Connection**

* **Uses** telephone lines.
* Very **slow** (~56 Kbps).
* Can’t use internet and phone at the same time.
* **Obsolete** now, used in early internet days.

**2. DSL (Digital Subscriber Line)**

* Also uses telephone lines but **faster** than dial-up.
* Can use phone and internet **simultaneously**.
* Speed ranges from **256 Kbps to 100 Mbps**.

**3. Cable Broadband**

* Uses **TV cable lines** to provide high-speed internet.
* Speed is usually **faster than DSL**.
* Shared bandwidth – can slow down if many users in the area are online.

**4. Fiber Optic Internet**

* Uses **fiber-optic cables** for **very high-speed** internet.
* Speed can go up to **1 Gbps or more**.
* Best for **streaming, gaming, and large downloads**.
* Expensive but very reliable.

**5. Wireless (Wi-Fi)**

* Uses **radio signals** instead of wires.
* Needs a router connected to a broadband line.
* Used in **homes, offices, cafes**, etc.

**6. Mobile Internet (3G/4G/5G)**

* Uses **cellular networks**.
* Can connect via **mobile phones or dongles**.
* Speed depends on network type (3G, 4G, or 5G).
* 5G is the **fastest** and latest.
* **7. Satellite Internet**
* Used in **remote or rural areas**.
* Internet is beamed via **satellites**.
* Slower than fiber or cable; more **delay (latency)**.
* Expensive, but useful where other options are unavailable.

**8. Hotspot**

* Internet shared from a **mobile device or hotspot device**.
* Uses mobile data.
* Portable, but depends on data plan and mobile signal.

LAB EXERCISE: Research different types of internet connections (e.g., broadband, fiber, satellite)and list their pros and cons.

|  |  |  |  |
| --- | --- | --- | --- |
| Internet Type | Description | Pros | Cons |
| Broadband (DSL) | Uses telephone lines to deliver internet | - Widely available - Can use phone + internet | - Slower than cable/fiber - Speed drops with distance |
| Cable Broadband | Uses coaxial TV cables | - Faster than DSL - Suitable for streaming | - Shared bandwidth - Speed may drop during peak hours |
| Fiber Optic | Uses light signals over fiber cables | - Extremely high speed - Low latency | - Expensive - Limited availability in rural areas |
| Satellite | Connects via orbiting satellites | - Available in remote areas | - High latency - Weather affects performance |
| Mobile Data (4G/5G) | Internet via cellular networks | - Portable - Fast with 4G/5G - No wiring needed | - Depends on network coverage - Limited data plans |
| Dial-Up | Old method using phone line | - Very cheap (almost free) | - Very slow - Cannot use phone & internet together |
| Hotspot/Tethering | Internet sharing from a mobile device | - Convenient for travel | - Drains phone battery - Limited data and slower |
| Wi-Fi (Wireless LAN) | Local wireless access from a broadband line | - Easy access within home/office - No cables needed | - Signal weakens with distance or walls |

**THEORY EXERCISE: How does broadband differ from fiber-optic internet?**

**1. Definition**

* **Broadband (DSL/Cable):**  
  Broadband is a general term for high-speed internet that includes DSL (Digital Subscriber Line) and cable connections. It usually uses **telephone lines or coaxial cables**.
* **Fiber-Optic Internet:**  
  Fiber-optic internet uses **thin strands of glass or plastic fibers** to transmit data as **light signals**, allowing for much faster speeds.

**🔹 2. Speed**

* **Broadband:**  
  Speed ranges from **1 Mbps to 100 Mbps** (depending on DSL or cable).
* **Fiber-Optic:**  
  Speed can go up to **1 Gbps (1000 Mbps)** or more – **much faster** than regular broadband.

**🔹 3. Reliability**

* **Broadband:**  
  less reliable than fiber-optic.ssssss
* **Fiber-Optic:**  
  Highly reliable with **stable and consistent speeds**, even during peak usage.

**🔹 4. Latency**

* **Broadband:**  
  Higher latency, which can affect gaming and video calls.
* **Fiber-Optic:**  
  Lower latency, ideal for **real-time communication** and streaming.

**🔹 5. Availability**

* **Broadband:**  
  Widely available, especially in **urban and semi-urban** areas.
* **Fiber-Optic:**  
  **Limited availability**, mostly in **cities and developed areas**.

**🔹 6. Cost**

* **Broadband:**  
  Usually **cheaper** and more affordable for general users.
* **Fiber-Optic:**  
  Slightly **more expensive**, but offers **better value** for heavy users.
* **Protocols:**

**Protocol** in computer networking refers to a set of rules or standards that define how data is formatted, transmitted, and received across a network.

**THEORY EXERCISE: What are the differences between HTTP and HTTPS protocols?**

| **Feature** | **HTTP** | **HTTPS** |
| --- | --- | --- |
| **Full Form** | HyperText Transfer Protocol | HyperText Transfer Protocol Secure |
| **Security** | Not secure – data is sent as plain text | Secure – data is encrypted using SSL/TLS |
| **Port Used** | Port 80 | Port 443 |
| **Data Encryption** | No encryption | Yes, data is encrypted |
| **SSL/TLS Certificate** | Not required | Required |
| **URL Prefix** | http:// | https:// |
| **Browser Indicator** | May show “Not Secure” | Shows a padlock 🔒 in the address bar |
| **Used For** | Non-sensitive websites | Secure websites like banking, login pages, online payments |

**LAB EXERCISE: Identify and explain three common application security vulnerabilities. Suggestpossible solutions.**

#### 1. SQL Injection

* Explanation:  
  SQL Injection occurs when user input is directly included in SQL queries without proper validation or sanitization. This allows attackers to manipulate queries and access, modify, or delete database data.
* Example:

SELECT \* FROM users WHERE username = '$username';

If an attacker inputs ' OR '1'='1, the query becomes:

SELECT \* FROM users WHERE username = '' OR '1'='1';

This always returns true and may bypass authentication.

* Solution:
  + Use prepared statements or parameterized queries.
  + Sanitize and validate all user inputs.
  + Use ORM frameworks that handle query building safely.

#### 2. Cross-Site Scripting (XSS)

* Explanation:  
  XSS allows attackers to inject malicious scripts into webpages viewed by other users. This can lead to session hijacking, data theft, or redirecting users to malicious websites.
* Types:
  + Stored XSS
  + Reflected XSS
  + DOM-based XSS
* Solution:
  + Escape and encode user inputs before displaying them in HTML.
  + Use frameworks that automatically handle escaping (e.g., React, Angular).
  + Implement Content Security Policy (CSP) headers.

#### 3. Cross-Site Request Forgery (CSRF)

* Explanation:  
  CSRF tricks an authenticated user into submitting an unwanted request to a web application, like changing a password or making a purchase, without their knowledge.
* Solution:
  + Use CSRF tokens in forms and verify them on the server.
  + Ensure same-site cookies are set.
  + Prompt for re-authentication for sensitive actions.

**THEORY EXERCISE: What is the role of encryption in securing applications? Software Applications and Its Types.**

* **What is the Role of Encryption in Securing Applications?**

Encryption is the process of converting readable data into an unreadable format to protect it from unauthorized access. It ensures that even if data is intercepted or stolen, it cannot be read without the correct key.

**Role of Encryption:**

* **Confidentiality:** Keeps sensitive data private and accessible only to authorized users.
* **Data Integrity:** Helps detect if data has been tampered with.
* **Secure Communication:** Encrypts data transmitted over networks (e.g., HTTPS, SSL/TLS).
* **Password Security:** Passwords are stored as encrypted or hashed values to prevent misuse.

**Examples:**

* Encrypting user passwords in a database.
* Using HTTPS for secure data transmission during login or online transactions.
* **Software Applications and Its Types**

A software application is a program or group of programs designed to perform specific tasks for users, such as writing documents, managing finances, or editing images.

**Types of Software Applications:**

| **Type** | **Description** | **Examples** |
| --- | --- | --- |
| **1. Desktop Applications** | Installed and run on a personal computer | MS Word, VLC Player |
| **2. Web Applications** | Run through a web browser | Gmail, Facebook |
| **3. Mobile Applications** | Designed for smartphones and tablets | WhatsApp, Instagram |
| **4. Enterprise Applications** | Used by organizations for business operations | SAP, Salesforce |
| **5. System Software** | Manages computer hardware and system resources | Windows OS, Antivirus software |

**LAB EXERCISE: Identify and classify 5 applications you use daily as either system software or application software.**

## Practical Identification Steps

### 🔹 1. Check How the Software Interacts With the System

| **Criteria** | **Application Software** | **System Software** |
| --- | --- | --- |
| Purpose | Performs specific tasks for users | system resources and hardware |
| Installed by | User | System/OS or Admin |
| User Interface | Usually has clear UI | May have no or limited UI |
| Examples | MS Word, Chrome, VLC | Windows OS, Drivers, BIOS |

### 🔹 2. Observe in Task Manager (Windows Example)

Steps:

1. Press Ctrl + Shift + Esc to open Task Manager.
2. Go to the "Processes" tab.
3. Try opening applications like Chrome or Word — see how they appear as separate programs (Application Software).
4. Observe background services like System, Windows Explorer, or Antivirus Service — these are usually System Software.

### 🔹 3. Check Installation Path

* System Software: Usually found in C:\Windows\System32 or installed automatically with OS.
* Application Software: Found in C:\Program Files or installed by users manually.

### 🔹 4. Using Command Line (Optional Advanced Way)

* On Windows:  
  Open CMD and type:

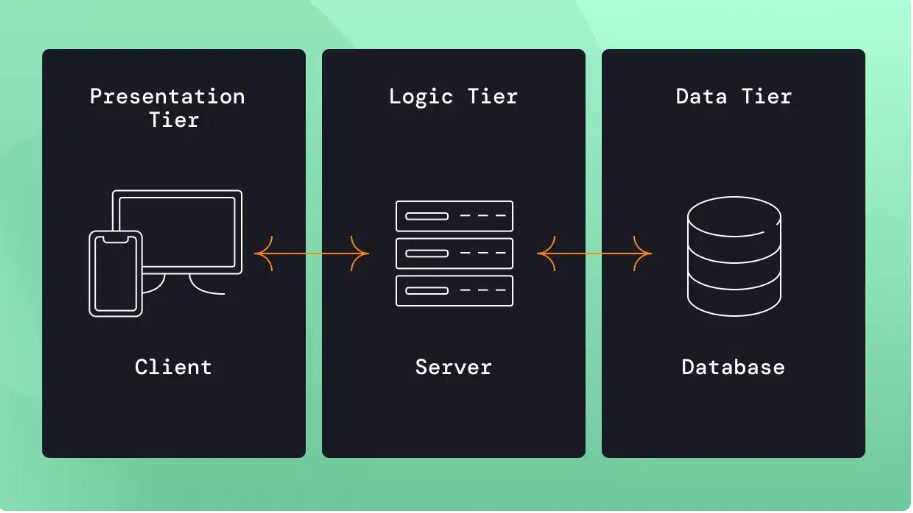
tasklist

You’ll see a list of all running applications and processes. Based on names and paths, you can distinguish them.

**THEORY EXERCISE: What is the difference between system software and application software?**

| **Aspect** | **System Software** | **Application Software** |
| --- | --- | --- |
| **Definition** | Software designed to operate and control the computer hardware and provide a platform for running application software. | Software designed to help the user perform specific tasks or applications. |
| **Purpose** | Manages system resources and enables the functioning of hardware. | Solves a particular user problem or performs a specific function. |
| **Examples** | Operating systems (Windows, Linux), device drivers. | Word processors (MS Word), web browsers (Chrome), media players. |
| **User Interaction** | Works in the background; not directly used by end users most of the time. | Directly used by the end users. |
| **Dependency** | Application software requires system software to run. | Depends on system software to function. |
| **Installation** | Usually comes pre-installed with the computer. | Installed by the user based on need. |
| **Functionality** | General purpose (e.g., resource management, hardware control). | Task-specific (e.g., writing, browsing, gaming). |

**LAB EXERCISE: Design a basic three-tier software architecture diagram for a web application.**

****

**THEORY EXERCISE: What is the significance of modularity in software architecture?**

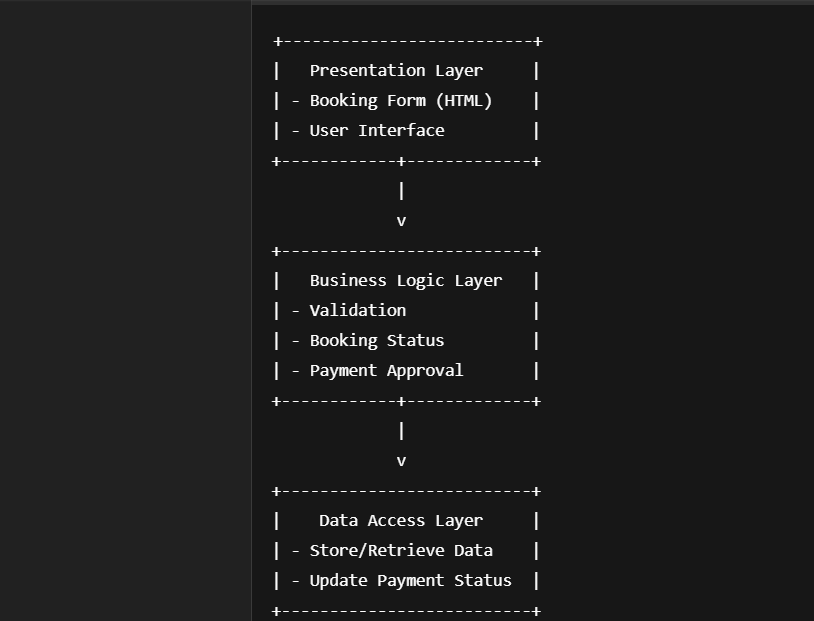
**What is Modularity?**

**Modularity** in software architecture means dividing a large software system into smaller, manageable, and independent parts called **modules**.  
Each module is responsible for a **specific functionality** and interacts with other modules through well-defined interfaces.

**🧩 Significance & Benefits of Modularity:**

|  |  |
| --- | --- |
| * **Reusability** | Once a module is developed, it can be reused in other projects or parts of the same system without rewriting the code. |
| * **Maintainability** | If a bug or issue occurs, it's easier to find and fix it within the specific module without affecting the entire application. |
| * **Flexibility** | Changes can be made to one module without disturbing the rest of the system, making updates safer and faster. |
| * **Better Team Collaboration** | Different teams or developers can work on different modules at the same time, increasing productivity. |
| * **Improved Testing & Debugging** | Each module can be tested independently (unit testing), making it easier to find and fix issues. |
| * **Scalability** | As user demands grow, new modules can be added to the system without redesigning the whole application. |
| * **Code Organization** | Keeps the code clean, structured, and logically separated by function. |

**LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and dataaccess layers of a given software system(practical folder booking\_form.html )**



**THEORY EXERCISE: Why are layers important in software architecture?**

Layers in software architecture are important because they divide the system into different parts, where each part has its own responsibility. This makes the software easy to build, understand, test, and maintain.

Let’s look at the importance in more detail:

**1. Separation of Concerns**

* Each layer does a specific job.
* Example: The UI layer only shows the design to users, the business logic layer checks rules, and the data layer saves or gets data from the database.
* This helps to avoid mixing code, so everything stays clean and simple.

**2. Easy Maintenance and Updates**

* If there’s a bug or you want to make a change, you can update one layer without affecting the others.
* Example: If you want to change how data is stored, you only update the data layer, not the UI or logic layer.

**3. Reusability**

* You can reuse code in the logic layer or data layer for other parts of the software.
* Example: The same login check code can be used for both admin and customer sections.

**4. Better Testing**

* You can test each layer separately.
* Example: You can test only the logic layer without using the UI.

**5. Scalability**

* If your software grows bigger, layers help in adding more features without breaking the system.
* You can add new layers or expand existing ones easily.

**6. Team Collaboration**

* Developers can work on different layers at the same time.
* Example: One developer works on UI, another on logic, and another on the database layer. This saves time.

**THEORY EXERCISE: Explain the importance of a development environment in software production**

**Importance of a Development Environment in Software Production**

A **development environment** is a setup that includes tools, software, and hardware where developers write, test, and debug their code before it is deployed to production.

**Key reasons why a development environment is important:**

1. **Safe Space for Experimentation**  
   Developers can write and test new features without affecting the live application or end users. This reduces the risk of bugs or crashes in production.
2. **Early Detection of Errors**  
   By testing code in the development environment, developers can find and fix bugs early, which saves time and cost compared to fixing issues after deployment.
3. **Consistency and Reproducibility**  
   Having a dedicated environment ensures everyone on the team works with the same setup (same software versions, libraries, configurations). This consistency reduces "it works on my machine" problems.
4. **Better Collaboration**  
   Teams can share code and configurations easily in a controlled environment, facilitating collaboration and integration of work from multiple developers.
5. **Testing and Debugging Tools**  
   Development environments often have debugging tools, simulators, and other utilities that help developers understand and fix problems efficiently.
6. **Faster Development Cycle**  
   Because it’s isolated from production, developers can quickly try new ideas, test changes, and iterate without waiting for approvals or risking live system stability.
7. **Security**  
   Testing sensitive features or data handling in a development environment reduces the chance of exposing vulnerabilities in the live system.

**THEORY EXERCISE: What is the difference between source code and machine code?**

### What is Source Code?

Source code is a set of instructions written by a programmer using a high-level programming language like:

* C, C++
* Java
* Python
* JavaScript
* PHP

This code is human-readable, meaning it is easy for developers to read, write, and understand. It contains the logic and flow of the program, using words and symbols that make sense to humans.

#### 🔸 Key Features of Source Code:

* Written in text form
* Can be edited easily
* Requires a compiler or interpreter to run

### 🔹 What is Machine Code?

Machine code (also called binary code) is the lowest-level language that a computer understands. It is made up of binary digits (0s and 1s) and directly executed by the CPU (Central Processing Unit).

#### 🔸 Key Features of Machine Code:

* Not human-readable
* Written in binary (0s and 1s)
* Specific to the hardware architecture
* Created by compiling or interpreting source code
* Looks like:  
  10110000 01100001 10110010 01100010

**Main Differences Between Source Code and Machine Code**

| **Aspect** | **Source Code** | **Machine Code** |
| --- | --- | --- |
| Written by | Programmer | Compiler/Interpreter |
| Readability | Human-readable (easy to understand) | Not human-readable (binary only) |
| Format | Text-based (uses words and syntax) | Binary format (0s and 1s) |
| Language Level | High-level | Low-level |
| Needs Conversion | Yes (needs to be compiled/interpreted) | No (already in executable form) |
| Portability | Portable across systems (if language supported) | Specific to system hardware |
| Example | print("Hello") | 01001000 11000011... |
| Execution | Cannot be executed directly by CPU | Executed directly by CPU |

* **Github and Introductions**

**What is GitHub?**

GitHub is a web-based platform that allows developers to host, manage, and share their code using the Git version control system. It is widely used for collaborative software development, where multiple developers can work on the same project simultaneously.

**Why Use GitHub?**

* To manage code versions efficiently
* To collaborate with team members on the same project
* To keep track of who made what changes
* To work on open-source projects or contribute to others
* To host documentation and manage issues or tasks.

**➤ Benefits of GitHub:**

* Cloud-based storage for code
* Easy integration with other development tools
* Public and private repositories
* Community support and collaboration
* Secure and trackable project history

**THEORY EXERCISE: Why is version control important in software development?**

**Answer:**

Version control is a system that helps manage and track changes made to source code over time. It plays a vital role in software development, especially when projects involve multiple developers and frequent updates.

Below are the main reasons why version control is important:

**🔹 1. Tracks Changes Over Time**

Version control systems (like Git) keep a detailed history of all changes made to a project. Developers can go back to previous versions if something breaks or needs to be reviewed.

**🔹 2. Enables Collaboration**

Multiple developers can work on the same project simultaneously without interfering with each other’s work. Each developer can work on separate branches and later merge their changes safely.

**🔹 3. Improves Code Quality**

With version control, changes are clearly recorded along with the developer’s name and commit message. This supports code reviews and helps maintain clean, organized code.

**🔹 4. Acts as a Backup System**

Repositories stored on platforms like GitHub or GitLab serve as backups of the project. If files are lost locally, they can easily be retrieved from the remote repository.

**🔹 5. Supports Experimentation**

Developers can create branches to test new features or fixes without affecting the main codebase. If the experiment fails, they can simply discard the branch.

**THEORY EXERCISE: What are the benefits of using Github for students?**

**1. Version Control**

* GitHub uses Git to track changes in code and documents.
* Students can revert to previous versions, recover lost work, and understand the evolution of their projects.

**2. Collaboration**

* GitHub makes it easy for students to work in teams.
* Multiple users can contribute to the same project without overwriting each other’s work.
* Pull requests and comments help manage and review contributions.

**3. Portfolio Building**

* Public repositories can showcase students' skills to potential employers or universities.
* Projects on GitHub can act as a professional portfolio.

**4. Larning Git and Industry Tools**

* Git is widely used in the tech industry.
* Students gain hands-on experience with tools and workflows (e.g., branching, merging, resolving conflicts).

**5. Access to Open Source Projects**

* Students can explore and contribute to real-world open-source projects.
* This builds skills, confidence, and professional networks.

**6. Free Resources**

* GitHub Student Developer Pack offers free access to premium tools, cloud services, and learning resources.

**7. Project Management**

* GitHub offers tools like Issues, Projects (Kanban boards), and Wikis for managing tasks and documentation.

**8. Learning and Feedback**

* Instructors and peers can review and provide feedback on code.
* Students can learn better coding practices and debugging techniques.
* **Types of Software**

**1. System Software**

* This software acts as a bridge between the computer hardware and the user.
* It helps the computer run and manage hardware resources.
* Examples:
  + Operating Systems (Windows, Linux, macOS)
  + Utility programs (Disk cleanup, Antivirus)
  + Device drivers

**2. Application Software**

* Software designed to perform specific tasks for users.
* Examples:
  + Word processors (Microsoft Word)
  + Web browsers (Google Chrome)
  + Multimedia players
  + Spreadsheet software (Excel)

**3. Development Software**

* Tools used by programmers to write, test, and debug code.
* Examples:
  + Compilers
  + Integrated Development Environments (IDEs) like Visual Studio, Eclipse
  + Debuggers

**4. Multimedia Software**

* Software used to edit or play photos, videos, and audio.
* Examples:
  + Adobe Photoshop
  + VLC Media Player

**5. Utility Software**

* Helps maintain, optimize, and protect the computer system.
* Examples:
  + Antivirus programs
  + File managers
  + Disk defragmenters

**LAB EXERCISE: Create a list of software you use regularly and classify them into the following categories: system, application, and utility software**.

|  |  |  |
| --- | --- | --- |
| Software Name | Category | Reason |
| Microsoft Word | Application Software | Used for creating and editing documents |
| Windows 10 | System Software | Operating system that runs your computer |
| Google Chrome | Application Software | Web browser to access the internet |
| File Explorer | System Software | Manages files and folders on your computer |
| Notepad++ | Application Software | Text editor used for writing and editing text or code |

**THEORY EXERCISE: What are the differences between open-source and proprietary software?**

|  |  |  |
| --- | --- | --- |
| Aspect | Open-Source Software | Proprietary Software |
| Source Code | The source code is freely available to everyone. | The source code is kept secret and not shared. |
| Usage | Usually free to use, modify, and distribute. | Requires purchasing a license or subscription. |
| Modification | Users can modify, customize, and improve the software. | Users cannot modify or change the software. |
| Distribution | Can be freely copied and shared. | Redistribution is prohibited or restricted. |
| Support | Community-based support, forums, and volunteers. | Official support from the company, often paid. |
| Examples | Linux, Firefox, LibreOffice | Microsoft Windows, Adobe Photoshop, Microsoft Office |

### Application Software

### LAB EXERCISE: Report on Various Types of Application Software and How They Improve Productivity

### Types of Application Software and Their Role in Productivity

#### 1. Word Processing Software

* Examples: Microsoft Word, Google Docs
* Purpose: Used to create, edit, and format text documents.
* Productivity Benefits: Saves time, enables quick editing, provides professional formatting tools, and supports collaboration.

#### 2. Spreadsheet Software

* Examples: Microsoft Excel, Google Sheets
* Purpose: Used for calculations, data analysis, and chart creation.
* Productivity Benefits: Automates complex calculations, organizes data efficiently, and presents data visually for quick understanding.

#### 3. Presentation Software

* Examples: Microsoft PowerPoint, Google Slides
* Purpose: Used to create visual slideshows for communication or teaching.
* Productivity Benefits: Helps communicate ideas clearly and effectively, especially in meetings or classrooms.

#### 4. Database Software

* Examples: Microsoft Access, MySQL
* Purpose: Used to store, manage, and retrieve large amounts of structured data.
* Productivity Benefits: Makes data access faster and more reliable, helps in managing business records, and improves decision-making.

#### 5. Communication Software

* Examples: Zoom, Microsoft Teams, Gmail
* Purpose: Enables people to communicate through messages, video calls, or emails.
* Productivity Benefits: Speeds up communication, supports remote work, and helps teams collaborate in real time.

#### 6. Graphic Design Software

* Examples: Adobe Photoshop, Canva
* Purpose: Used for creating and editing images, designs, and marketing materials.
* Productivity Benefits: Enhances creative tasks, produces high-quality visual content quickly, and supports brand promotion.

THEORY EXERCISE: What is the role of application software in businesses?

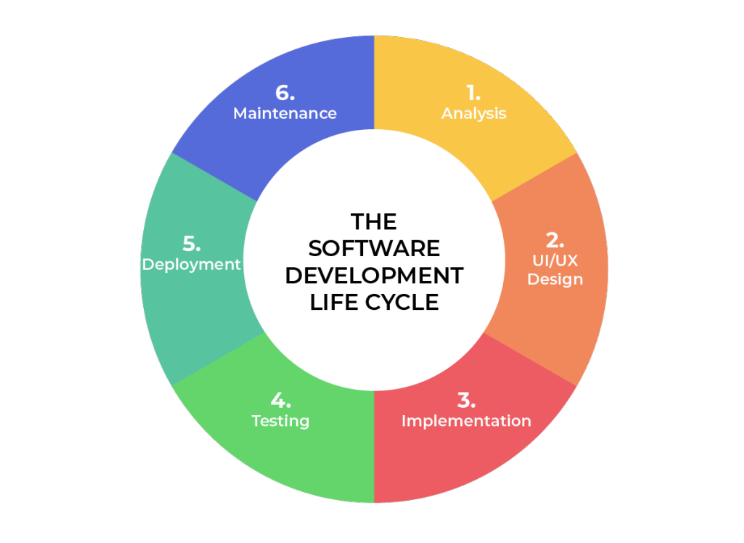
**Role of Application Software in Businesses**

Application software plays a vital role in modern businesses by helping them perform day-to-day tasks efficiently and effectively. Here are the key roles of application software in business operations:

1. **Automation of Tasks**:  
   Application software automates repetitive and time-consuming tasks such as payroll processing, inventory management, billing, and customer data entry. This reduces manual effort and increases productivity.
2. **Improved Communication**:  
   Software like email clients, video conferencing tools, and instant messaging platforms helps businesses maintain internal and external communication with employees, clients, and partners.
3. **Data Management**:  
   Applications such as databases and customer relationship management (CRM) systems help in storing, organizing, retrieving, and analyzing business data, leading to better decision-making.
4. **Financial Management**:  
   Accounting software like Tally, QuickBooks, or Zoho Books enables businesses to manage their financial records, generate invoices, monitor cash flow, and ensure compliance with tax regulations.
5. **Customer Service**:  
   Businesses use helpdesk software, chatbots, and CRM applications to enhance customer experience, respond to queries quickly, and maintain service quality.
6. **Project Management**:  
   Tools like Trello, Asana, or Microsoft Project allow teams to plan, track, and collaborate on projects, ensuring timely completion and efficient resource use.
7. **Marketing and Sales**:  
   Marketing software helps in email marketing, social media management, SEO, and ad campaigns. Sales tools help manage leads, track customer interactions, and close deals faster.
8. **Security and Compliance**:  
   Security applications protect business data from unauthorized access, malware, and cyber threats. Compliance tools help ensure that businesses follow legal and regulatory requirements.
9. **Customization and Integration**:  
   Businesses can choose or develop application software tailored to their specific needs and integrate it with other systems for seamless operations.

* **Software Development Process**

LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDLC)



**THEORY EXERCISE: What are the main stages of the software development process? Software Requirement.**

The software development process typically follows a structured approach known as the Software Development Life Cycle (SDLC). Here are the main stages, starting from Software Requirement:

**1. Requirement Analysis (Software Requirement)**

* **Goal**: Understand what the users need from the software.
* **Activities**:
  + Gather requirements from stakeholders (clients, users).
  + Define functional (what the system should do) and non-functional (performance, security) requirements.
  + Create a Software Requirements Specification (SRS) document.

**2. System Design**

* **Goal**: Plan how the software will be built.
* **Activities**:
  + Design system architecture, database structure, and user interface.
  + Choose technologies (e.g., programming languages, frameworks).
  + Create diagrams (e.g., flowcharts, UML) for better visualization.

**3. Implementation (Coding)**

* **Goal**: Develop the actual software based on the design.
* **Activities**:
  + Developers write code according to design specifications.
  + Follow coding standards and practices.
  + Version control systems (like Git) are often used.

**4. Testing**

* **Goal**: Ensure the software works as expected.
* **Activities**:
  + Perform different types of testing (unit, integration, system, user acceptance).
  + Identify and fix bugs.
  + Ensure the software meets the original requirements.

**5. Deployment**

* **Goal**: Release the software to users.
* **Activities**:
  + Move the software from development to production environment.
  + Provide installation/setup instructions.
  + Sometimes released in phases (e.g., beta release).

**6. Maintenance and Support**

* **Goal**: Keep the software running smoothly after release.
* **Activities**:
  + Fix bugs found after deployment.
  + Update software to meet new requirements or improve performance.
  + Provide technical support.
* **Software Requirement**

### . Introduction

#### Purpose

The purpose of this document is to outline the functional and non-functional requirements of a simple Library Management System that allows librarians to manage books and users, and enables users to borrow and return books.

#### Scope

This system will manage:

* Book inventory (add, update, delete books)
* Member registration
* Book issue and return
* Fine calculation for late returns

### 2. Functional Requirements

#### User Roles

* Admin (Librarian)
* Library Member (User)

#### Admin Functions

* Add, update, or delete book details
* Register or remove library members
* View issued/returned books
* Track overdue books and fines

#### Member Functions

* View available books
* Search books by title, author, or category
* Borrow and return books
* View personal borrowing history

### 3. Non-Functional Requirements

#### Performance Requirements

* The system should support at least 10 concurrent users.
* Book search results should be returned within 2 seconds.

#### Security Requirements

* Only admins can modify book or member records.
* Users must log in to access their accounts.

#### Usability Requirements

* The system should have a user-friendly interface.
* It should be accessible on desktop and tablet browsers.

### 4. System Requirements

#### Hardware

* Server with minimum 4GB RAM, 100GB storage

#### Software

* Backend: PHP / Python / Java
* Database: MySQL / PostgreSQL
* Frontend: HTML, CSS, JavaScript

### 5. Assumptions and Constraints

* Each user can borrow a maximum of 3 books at a time.
* Books can be issued for a maximum of 14 days.
* Fine is ₹5 per day after the due date.

THEORY EXERCISE: Why is the requirement analysis phase critical in software development? Software Analysis

**1. Foundation for Success:**

* If requirements are unclear or incorrect, the entire software development process can go in the wrong direction.
* This phase helps build a strong foundation by understanding and documenting the requirements clearly.

**2. Understanding User and Stakeholder Needs:**

* It is essential to know and clarify what the users and clients want (both functional and non-functional requirements).
* Requirement analysis helps to capture exactly what the users expect from the system.

**3. Guides Other Phases:**

* System design, coding, and testing phases all depend on the requirements gathered here.
* If requirements are incomplete or incorrect, subsequent phases will also be flawed.

**4. Saves Time and Cost:**

* Detecting and fixing errors in requirements early saves significant time and money.
* Changes in later phases like development or testing are usually more expensive.

**5. Reduces Risks:**

* Potential project risks can be identified early during this phase.
* It helps to avoid delays, cost overruns, or scope creep.

**LAB EXERCISE: Perform a functional analysis for an online shopping system**

## Functional Analysis for Online Shopping System

### 1. User Types (Actors):

* Customer (Buyer)
* Admin
* Seller (Optional, if system allows multiple sellers)
* Payment Gateway

### 2. Main Functionalities

#### Customer Functions:

* User Registration/Login:  
  Customers can create accounts and log in securely.
* Browse Products:  
  Customers can search and browse products by categories, brands, price, etc.
* Product Details:  
  View detailed information about a product including price, images, description, reviews.
* Add to Cart:  
  Customers can add products to their shopping cart.
* Update Cart:  
  Change quantity or remove products from cart.
* Checkout:  
  Proceed to buy the selected products by providing shipping and payment details.
* Payment Processing:  
  Make payment via integrated payment gateways (credit card, debit card, digital wallets).
* Order Tracking:  
  Customers can track the status of their orders (processing, shipped, delivered).
* Order History:  
  View past orders and download invoices.
* Product Reviews & Ratings:  
  Customers can leave feedback on purchased products.

#### Admin Functions:

* Product Management:  
  Add, update, or delete products.
* User Management:  
  Manage customer and seller accounts.
* Order Management:  
  View all orders, update order status (confirmed, shipped, delivered, canceled).
* Inventory Management:  
  Track stock levels and update inventory.
* Reports:  
  Generate sales reports, customer activity, product performance.

#### Seller Functions (If applicable):

* Product Listing:  
  Sellers can add or update their product listings.
* Order Management:  
  View orders for their products, update status.
* Sales Reports:  
  Track their sales performance.

### 3. Non-Functional Requirements (Brief):

* Performance: Fast response time for browsing and checkout.
* Security: Secure user authentication, data protection, secure payment.
* Usability: User-friendly interface on desktop and mobile.
* Scalability: Support for increasing users and products.

THEORY EXERCISE: What is the role of software analysis in the development process?

## Role of Software Analysis in the Development Process

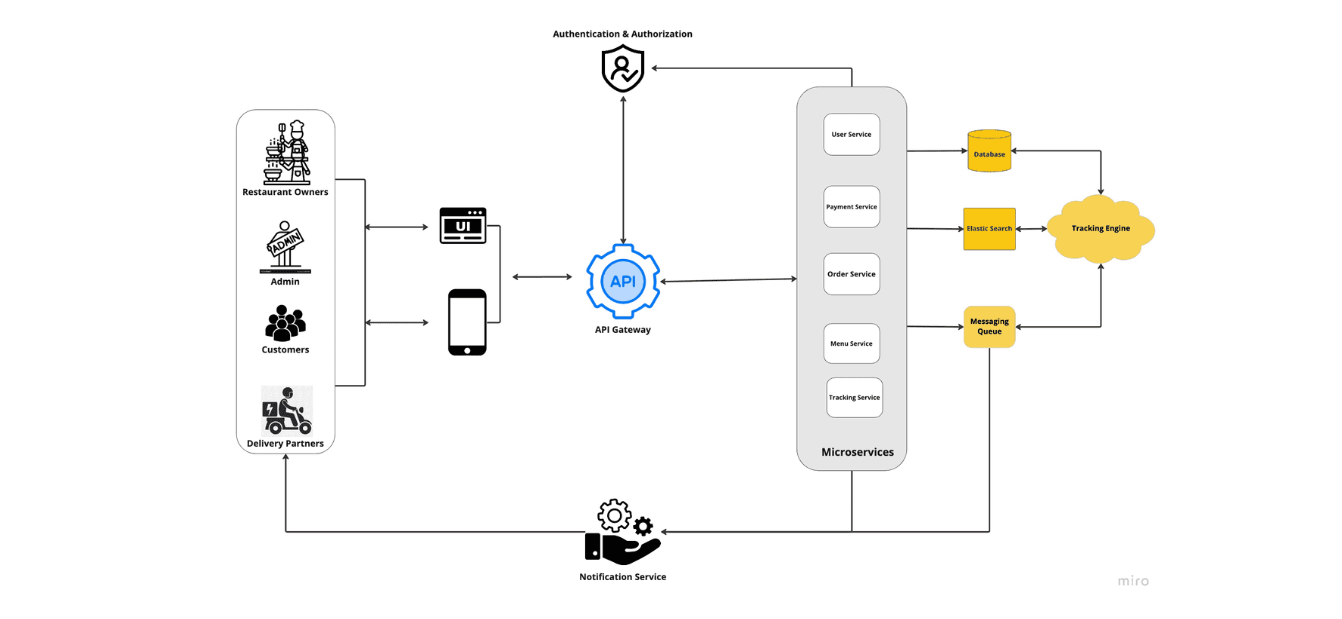
Software analysis is a crucial phase in the software development lifecycle. Its primary role is to understand and specify what the software needs to do to meet user and business requirements.

### Key Roles of Software Analysis:

1. Understanding Requirements:
   * It helps gather detailed functional and non-functional requirements from stakeholders, users, and clients.
   * Ensures that the development team clearly understands what the software should accomplish.
2. Defining System Scope:
   * Helps set boundaries for what the system will and will not do, avoiding scope creep.
   * Establishes clear objectives and deliverables.
3. Bridging Communication:
   * Acts as a communication bridge between users/business and developers.
   * Translates user needs into technical specifications.
4. Identifying Constraints and Risks:
   * Recognizes any technical, business, or operational constraints early in the project.
   * Helps identify potential risks and challenges.
5. Foundation for Design and Development:
   * The output of software analysis (requirement specifications) guides the system design, coding, and testing phases.
   * Ensures all teams work towards common goals.
6. Improving Quality and Reducing Cost:
   * Early analysis helps catch misunderstandings and errors, reducing costly changes later.
   * Leads to better software quality and user satisfaction.

* System Design

LAB EXERCISE: Design a basic system architecture for a food delivery app.



THEORY EXERCISE: What are the key elements of system design?

System design is the process of defining the architecture, components, modules, interfaces, and data for a system. It plays a vital role in converting requirements into a working system.

**🔑 Key Elements:**

**1. Architecture Design:**

* Defines the **structure** of the entire system.
* Decides between **monolithic** or **microservices** architecture.
* Involves selection of technologies and frameworks.

**2. Module Design:**

* Breaks the system into smaller **logical modules** (e.g., User Module, Payment Module).
* Helps in simplifying development and maintenance.

**3. Data Design:**

* Focuses on the **data structures**, **databases**, and **data flow** between components.
* Ensures data consistency, normalization, and integrity.

**4. Interface Design:**

* Defines how users interact with the system (UI/UX).
* Also includes how different modules and external systems interact (APIs, interfaces).

**5. Component Design:**

* Each module is further broken into **components** or **classes** with defined roles.
* Ensures **separation of concerns** and reusability.

**6. Security Design:**

* Plans for **authentication**, **authorization**, **data encryption**, and **secure access**.
* Protects the system against potential threats.

**7. Scalability & Performance Planning:**

* Ensures system can handle **increased load** (horizontal or vertical scaling).
* Optimizes for **speed, responsiveness**, and **availability**.

**8. Error Handling & Fault Tolerance:**

* Plans how the system will behave in case of **failures** or **errors**.
* Includes **logging**, **retries**, and **backup systems**.
* **Software Testing**

**THEORY EXERCISE: Why is software testing important?**

Software Testing is important because it helps ensure that the software product is **reliable**, **functional**, and **free of bugs or errors** before it is delivered to users. It is a critical step in the software development lifecycle (SDLC).

**🔹 Key Reasons Why Software Testing is Important:**

1. **Ensures Quality**  
   Testing verifies that the software meets the expected requirements and performs as intended.
2. **Detects Bugs Early**  
   It helps identify and fix errors or defects early, which reduces the cost and effort of correction.
3. **Improves User Experience**  
   A well-tested application performs smoothly and earns user trust, satisfaction, and confidence.
4. **Ensures Security**  
   Testing helps find vulnerabilities and security loopholes in the software, protecting data and users.
5. **Reduces Failure Risk**  
   It minimizes the risk of failure after deployment by ensuring stability and reliability.
6. **Validates Performance**  
   Testing checks if the software performs efficiently under different conditions (speed, load, stress).
7. **Supports Decision Making**  
   Clear test results help stakeholders make informed decisions about release or improvement.

* **Maintenance**

**LAB EXERCISE: Document a real-world case where a software application required criticalmaintenance.**

**Real-world case of software application requiring critical maintenance**

**Case: The WannaCry Ransomware Attack on NHS (National Health Service), UK - 2017**

In May 2017, the NHS in the UK was severely impacted by the WannaCry ransomware attack. The ransomware exploited a critical vulnerability in Microsoft Windows operating systems that had not been patched in time.

**Why critical maintenance was required:**  
The NHS software systems, which managed patient records, appointments, and medical devices, suddenly became inaccessible. This caused widespread disruption in hospitals, leading to canceled appointments and delayed treatments.

**Critical maintenance actions taken:**

* Emergency security patches were applied to vulnerable systems immediately.
* Systems were isolated from the network to prevent further spread.
* IT teams worked round the clock to restore data from backups and remove the malware.
* Additional monitoring and security protocols were implemented to prevent recurrence.

**Outcome:**  
This case highlights the importance of timely critical maintenance — especially applying security updates and patches — to prevent catastrophic failures in real-world software applications.

**THEORY EXERCISE: What types of software maintenance are there?**

Software maintenance typically falls into **four main types**:

1. **Corrective Maintenance**
   * Fixing bugs and errors discovered after the software is released.
   * Addresses issues that cause the software to malfunction or behave unexpectedly.
2. **Adaptive Maintenance**
   * Modifying the software to keep it compatible with changing environments, such as new operating systems, hardware, or third-party software updates.
   * Ensures the software continues to function properly in a changing technical landscape.
3. **Perfective Maintenance**
   * Improving or enhancing software performance, usability, or maintainability based on user feedback or new requirements.
   * Includes adding new features or optimizing existing ones to make the software better.
4. **Preventive Maintenance**
   * Making changes to prevent future problems, such as refactoring code to reduce complexity or improve reliability.
   * Aims to reduce the risk of future failures and improve long-term software stability.

* **Development**

**THEORY EXERCISE: What are the key differences between web and desktop applications?**

| **Aspect** | **Web Applications** | **Desktop Applications** |
| --- | --- | --- |
| **Access** | Run in a web browser over the internet or intranet. | Installed and run directly on the user’s computer. |
| **Installation** | No installation needed; accessed via URL. | Requires installation on each device. |
| **Platform Dependency** | Usually platform-independent (works on any OS with a compatible browser). | Often platform-dependent (Windows, macOS, Linux). |
| **Updates** | Updates happen on the server side; users get latest version automatically. | Users need to manually install updates or patches. |
| **Internet Requirement** | Usually requires internet connection to function (some may support offline mode). | Can often run without internet connection. |
| **Performance** | May be slower due to network latency and browser limitations. | Generally faster, can utilize system resources fully. |
| **Security** | Security depends on web protocols and server protection; vulnerable to web-specific attacks. | Security depends on OS security; less exposed to web attacks. |
| **User Interface** | Limited by browser capabilities and standards. | More control over UI design and capabilities. |
| **Data Storage** | Data usually stored on servers or cloud. | Data stored locally on the user’s device. |

* **Web Application**

 **Accessibility:**  
Web applications can be accessed from any device with an internet connection and a web browser, regardless of the operating system or location.

 **No Installation Required:**  
Users don’t need to download or install anything; they just open the web app in their browser.

 **Automatic Updates:**  
Web applications are updated on the server side, so users always have access to the latest version without needing to manually update.

 **Cross-Platform Compatibility:**  
They work across multiple platforms like Windows, macOS, Linux, Android, iOS, etc., without needing separate versions.

 **Lower Maintenance Cost:**  
Since the software runs on the server, maintaining and fixing issues is easier and less costly.

 **Centralized Data Storage:**  
Data is stored centrally on servers, which simplifies backup, security, and sharing.

 **Easier Collaboration:**  
Multiple users can access and work on the same data in real-time through web applications.

* **Designing**

**THEORY EXERCISE: What role does UI/UX design play in application development?**

**Role of UI/UX Design in Application Development**

1. **Enhances User Satisfaction:**  
   Good UI (User Interface) and UX (User Experience) design make applications easy and enjoyable to use, increasing user satisfaction.
2. **Improves Usability:**  
   UX focuses on how users interact with the app, ensuring that tasks can be completed efficiently and intuitively.
3. **Boosts Engagement:**  
   A well-designed interface attracts users and keeps them engaged longer, reducing bounce rates.
4. **Increases Accessibility:**  
   UI/UX design considers accessibility, making the application usable by people with different abilities.
5. **Supports Brand Identity:**  
   Consistent and attractive design reinforces brand recognition and trust.
6. **Reduces Development Costs:**  
   Investing in UI/UX early helps identify problems before development, reducing costly redesigns later.
7. **Competitive Advantage:**  
   Applications with superior UI/UX stand out in the market and gain more users.

* **Mobile Application**

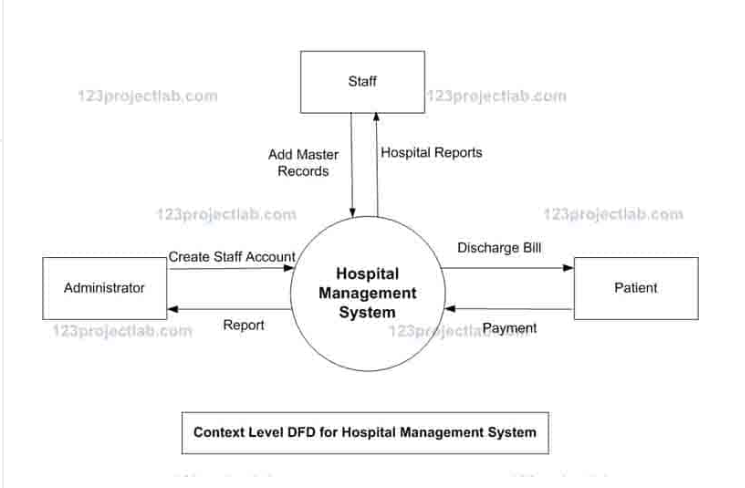
**THEORY EXERCISE: What are the differences between native and hybrid mobile apps?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Native Mobile Apps** | **Hybrid Mobile Apps** |
| **Development Language** | Built using platform-specific languages (e.g., Java/Kotlin for Android, Swift/Objective-C for iOS) | Built using web technologies (HTML, CSS, JavaScript) wrapped inside a native container |
| **Performance** | Faster and smoother because they interact directly with device hardware | Slightly slower as they run inside a web view within the app |
| **Access to Device Features** | Full access to device features like camera, GPS, sensors | Limited access, depending on plugins or third-party libraries |
| **Development Time & Cost** | Separate development needed for each platform, which increases time and cost | Single codebase for multiple platforms, reducing time and cost |
| **Maintenance** | Updates and maintenance need to be done separately for each platform | Easier maintenance with a single codebase |
| **User Experience (UX)** | Provides native look and feel tailored to each platform | May not be as smooth as native apps but customizable |
| **Distribution** | Distributed via Google Play Store and Apple App Store | Also distributed vi |

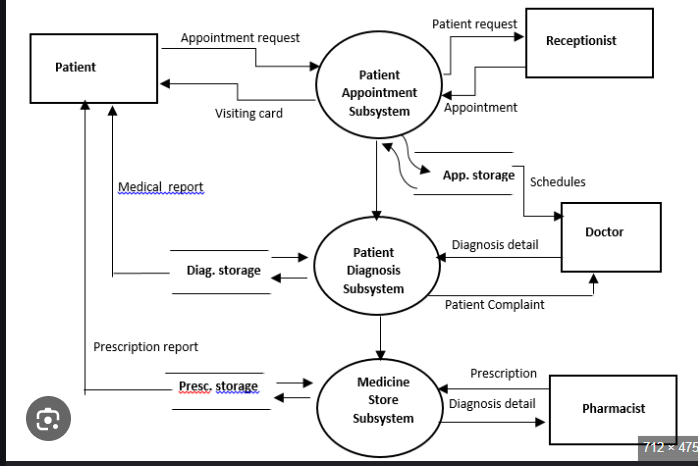
* **DFD (Data Flow Diagram)**

**LAB EXERCISE: Create a DFD for a hospital management system.**

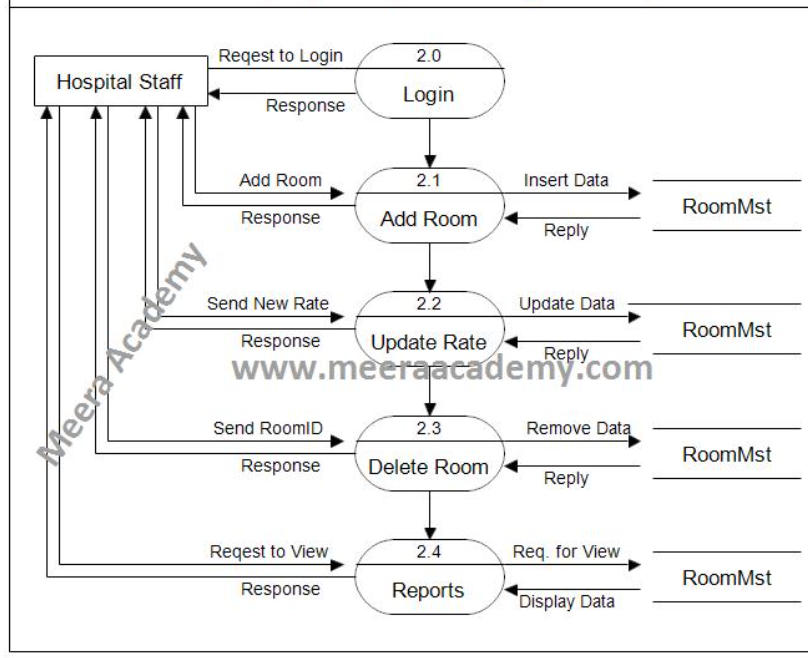
**Context level:**

****

**First level:**

****

Second level:



**THEORY EXERCISE: What is the significance of DFDs in system analysis?**

1. **Clear Understanding of the System:**  
   DFD helps both developers and stakeholders understand how data flows and how different processes are connected within the system.
2. **Simplifies Complex Systems:**  
   Complex systems can be broken down into smaller, manageable parts (processes), making analysis and design easier.
3. **Improves Communication:**  
   Acts as a visual communication tool between developers, analysts, and non-technical stakeholders.
4. **Identifies Data Sources and Destinations:**  
   Clearly shows where data comes from (sources) and where it goes (destinations).
5. **Helps in Identifying Inefficiencies:**  
   By visualizing processes, one can detect redundant steps, data bottlenecks, or unnecessary data flows.
6. **Foundation for System Design:**  
   DFDs serve as the starting point for designing the database structure, interfaces, and system architecture.
7. **Supports Documentation:**  
   Useful in preparing system documentation and technical reports for future reference or audits.

* **Desktop Application**

**LAB EXERCISE: Build a simple desktop calculator application using a GUI library.**

**#include <windows.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <tchar.h>**

**#define ID\_EDIT1 1**

**#define ID\_EDIT2 2**

**#define ID\_BUTTON 3**

**#define ID\_RESULT 4**

**LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);**

**int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int nCmdShow) {**

**WNDCLASSW wc = {0};**

**wc.lpszClassName = L"CalculatorApp";**

**wc.hInstance = hInst;**

**wc.hbrBackground = GetSysColorBrush(COLOR\_3DFACE);**

**wc.lpfnWndProc = WindowProcedure;**

**wc.hCursor = LoadCursor(NULL, IDC\_ARROW);**

**RegisterClassW(&wc);**

**CreateWindowW(wc.lpszClassName, L"Simple Calculator",**

**WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,**

**100, 100, 300, 200, NULL, NULL, hInst, NULL);**

**MSG msg = {0};**

**while (GetMessage(&msg, NULL, 0, 0)) {**

**TranslateMessage(&msg);**

**DispatchMessage(&msg);**

**}**

**return 0;**

**}**

**HWND hEdit1, hEdit2, hResult;**

**LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp) {**

**switch(msg) {**

**case WM\_CREATE:**

**hEdit1 = CreateWindowW(L"Edit", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER,**

**30, 20, 100, 25, hwnd, (HMENU) ID\_EDIT1, NULL, NULL);**

**hEdit2 = CreateWindowW(L"Edit", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER,**

**150, 20, 100, 25, hwnd, (HMENU) ID\_EDIT2, NULL, NULL)**

**CreateWindowW(L"Button", L"Add", WS\_VISIBLE | WS\_CHILD,**

**90, 60, 100, 25, hwnd, (HMENU) ID\_BUTTON, NULL, NULL);**

**hResult = CreateWindowW(L"Static", L"Result: ", WS\_VISIBLE | WS\_CHILD,**

**90, 100, 200, 25, hwnd, (HMENU) ID\_RESULT, NULL, NULL);**

**break;**

**case WM\_COMMAND:**

**if (LOWORD(wp) == ID\_BUTTON) {**

**wchar\_t text1[10], text2[10];**

**GetWindowTextW(hEdit1, text1, 10);**

**GetWindowTextW(hEdit2, text2, 10);**

**int num1 = \_wtoi(text1);**

**int num2 = \_wtoi(text2);**

**int sum = num1 + num2;**

**wchar\_t result[50];**

**wsprintfW(result, L"Result: %d", sum);**

**SetWindowTextW(hResult, result);**

**}**

**break;**

**case WM\_DESTROY:**

**PostQuitMessage(0);**

**break;**

**}**

**return DefWindowProcW(hwnd, msg, wp, lp);**

**}**

**THEORY EXERCISE: What are the pros and cons of desktop applications compared to webapplications?**

**Pros of Desktop Applications:**

1. **Performance:**
   * Faster and more responsive since they run locally on the user's machine.
2. **Offline Access:**
   * Can work without an internet connection.
3. **Better Hardware Access:**
   * Easier access to system resources (like file system, printer, USB, etc.).
4. **Security:**
   * Data is stored locally, reducing some risks of data breaches from online attacks.
5. **Custom Integration:**
   * Easier to integrate with local software and hardware.

**❌ Cons of Desktop Applications:**

1. **Platform Dependency:**
   * Often built for specific OS (Windows, macOS, Linux), not cross-platform by default.
2. **Installation Required:**
   * Needs to be installed manually, which can be time-consuming.
3. **Difficult to Update:**
   * Users must download and install updates manually.
4. **Limited Remote Access:**
   * Cannot be accessed from anywhere like web applications.

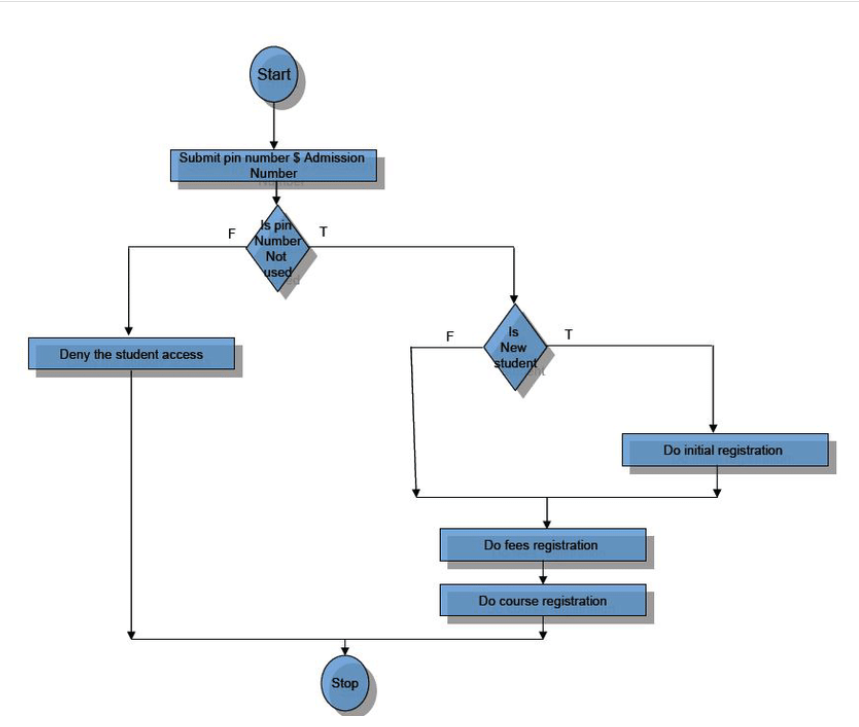
**✅ Pros of Web Applications:**

1. **Accessible Anywhere:**
   * Runs in browser, accessible from any device with internet.
2. **Cross-Platform:**
   * Works across Windows, Mac, Linux, mobile, etc.
3. **No Installation Needed:**
   * Just open the browser and use.
4. **Easy to Update:**
   * Updates are applied on the server side; users always use the latest version.

**❌ Cons of Web Applications:**

1. **Internet Dependency:**
   * Requires internet connection to work.
2. **Slower Performance:**
   * Can be slower compared to local apps, especially with large data.
3. **Security Concerns:**
   * More exposed to online threats like hacking and phishing.
4. **Limited System Access:**
   * Cannot directly access user files or hardware easily (for security reasons).

* **Flow Chart**

**LAB EXERCISE: Draw a flowchart representing the logic of a basic online registration system.**

**THEORY EXERCISE: How do flowcharts help in programming and system design?**

#### 1. Better Understanding of Logic:

* Flowcharts help developers visualize the flow of the program logic step-by-step.
* Makes it easier to understand conditions, loops, and processes.

#### 2. Effective Planning Tool:

* Helps in designing systems or applications before writing actual code.
* Serves as a blueprint to avoid logical errors.

#### 3. Easy Debugging and Maintenance:

* With a clear flowchart, it's easier to trace errors in the logic.
* Helps in future updates or changes in the program.

#### 4. Improves Communication:

* Flowcharts are useful when explaining system logic to team members, clients, or non-programmers.

#### 5. Saves Development Time:

* Since the process is already planned visually, coding becomes faster and more accurate.

#### 6. Documentation Purpose:

* Flowcharts serve as good documentation for system design and program logic.