# 

# What is a Program?

1. **[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 computer language that tells a computer what tasks to perform. It works like a recipe — giving the computer step-by-step directions to follow. When the program runs, the computer reads each instruction in order, processes any input, performs calculations or actions, and then gives the output. Programs can be simple, like showing a message, or complex, like running a video game or managing hospital records.

1. **[Lab exercise] : Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.**

* **C :**
* #include<stdio.h>

Void main()

{

printf(“Hello World”);

getch();

}

* **PYTHON :**
* Print(“Hello World”)
* **Python** is concise, user-friendly, and great for quick scripts.
* **C** is more verbose, requiring explicit structure and compilation, reflecting its low-level control and performance orientation.

# What is Programming?

* **Programming** is the process of designing, writing, testing, and maintaining a set of precise instructions (code) that a computer can execute to perform tasks or solve problems.

1. **[THEORY EXERCISE]: What are the key steps involved in the programming process?**

* The programming process, also known as the software development life cycle, generally **involves these key steps**: problem analysis, design, coding, testing and debugging, and documentation. Some also include implementation, deployment, and maintenance.
* **Here's a more detailed breakdown:**
* **Problem Analysis:** Clearly define the problem you're trying to solve. This involves understanding the requirements, inputs, outputs, and constraints**.**
* **Design:** Plan the solution, which includes designing the algorithm (a step-by-step solution) and potentially creating flowcharts or code to visualize the logic.
* **Coding:** Translate the design into a programming language, writing the actual code**.**
* **Testing and Debugging:** Run the code with various inputs to find and fix errors (bugs).
* **Implementation**: Put the code into a usable form, like creating an executable file.
* **Deployment:** Make the software available to users.
* **Maintenance:** This ongoing step involves fixing bugs, adding new features, and adapting the software to changing needs.
* **Documentation:** Create documentation (user manuals, code comments) to explain how the program works for future reference and maintenance

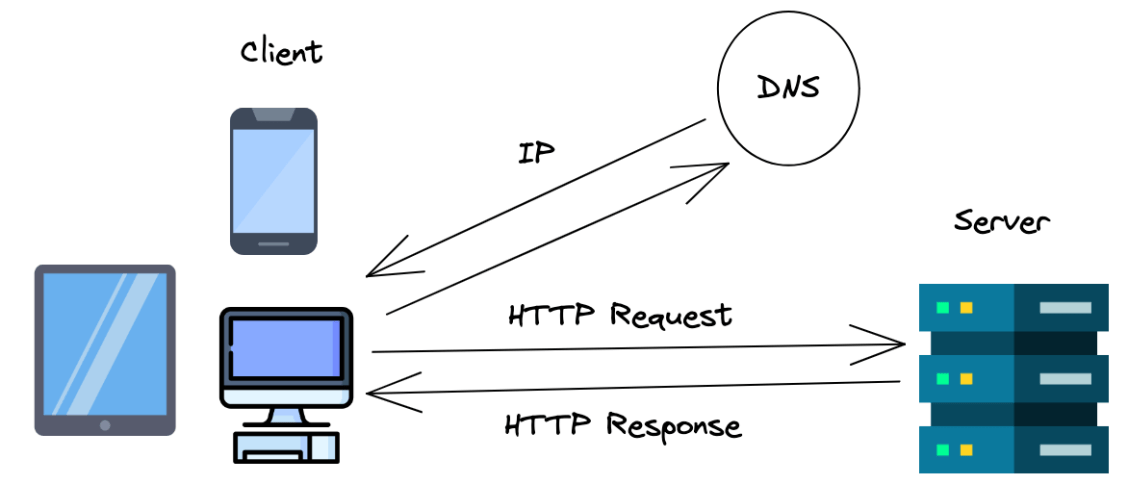
# Types of Programming Languages:

1. **[THEORY EXERCISE]: What are the main differences between high-level and low-level Programming languages?**

|  |  |  |
| --- | --- | --- |
| Parameter | High-Level Language | Low-Level Language |
| Abstraction | High – human-friendly | Low – close to machine |
| Control | Indirect (via compiler/runtime) | Direct hardware manipulation |
| Performance | Moderate, abstraction overhead | Fast, efficient |
| Portability | High – architecture-independent | Low – platform-specific |
| Development speed | Fast, easier maintenance | Slow, complex debugging |
| Memory efficiency | Lower – more memory usage | High – minimal overhead |

# World Wide Web & How Internet Works

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



* The diagram shows how multiple clients (like a laptop, phone, or desktop) connect to a server through the internet to send requests and receive responses, representing a basic client-server communication model.

1. **[THEORY EXERCISE]: Describe the roles of the client and server in web communication.**

* **Client :**
* The **client** is typically a **web browser** (like Chrome, Firefox, etc.) or any application that sends requests to access resources on the internet Sends **HTTP/HTTPS requests** to the server for web pages, images, videos, or data.
* **Server :**
* The **server** is a **powerful computer or system** that hosts websites, data, or applications. It **waits for requests** from clients

# Network Layers on Client and Server.

1. **[LAB EXERCISE]: Design a simple HTTP client-server communication in any language.**

**Client:**

* • Sends HTTP request to the server.
* • Example: A browser or app sending a GET or POST request.
* **Server:**
* • Receives the request, processes it.
* • Sends back an HTTP response with data or a web page.
* **Protocol:**
* • HTTP (Hypertext Transfer Protocol) – standard web communication protocol.

1. **[THEORY EXERCISE]: Explain the function of the TCP/IP model and its layers.**

* The TCP/IP model (Transmission Control Protocol/Internet Protocol) is a set of communication protocols used to connect computers over the internet.
* It defines how data is transmitted from one device to another and ensures reliable communication between them.
* **Layers**:

1. Network Access Layer (Link Layer):

This layer is responsible for the physical transmission of data over the network. It defines how data is sent over hardware like cables or wireless.

2. Internet Layer:

The internet layer handles the routing of data packets between devices on different networks. It ensures that each packet reaches its correct destination using IP addresses.

3. Transport Layer:

This layer provides reliable or fast communication between two devices. It manages data flow, error checking, and retransmission.

4. Application Layer:

The top layer interacts directly with end-user applications. It provides network services like file transfer, email, and web browsing.

# Client and Servers:

1. **[THEORY EXERCISE]: Explain Client Server Communication.**

* It’s a **way for two devices to talk to each other**: one acts as the **client** (usually the user’s device), and the other acts as the **server** (a powerful machine that provides services or resources).

**🧍 Client:**

* The **client** is typically a **web browser**, mobile app, or desktop application.
* It **sends requests** to the server asking for information or services.
* Example: When you open a website, your browser is the client.

**🖥️ Server:**

* The **server** is a computer that **listens for incoming requests** and sends back the appropriate responses.
* It might return a **webpage**, store data in a **database**, or run a program in response.
* Example: When you access Google, their server receives your request and sends back search results.

# Types of Internet Connections

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

**1. Dial-Up Internet:**

: Pros:

• Very low cost

* + Available almost everywhere with a landline

Cons:

• Extremely slow

• Ties up the phone line

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

Pros:

• Available in many areas

• Doesn't interfere with phone calls

Cons:

• Speed decreases with distance from the provider

• Slower than cable or fiber

**3. Fiber Optic Internet**

Pros:

• Ultra-fast speeds

• Highly reliable and low latency

Cons:

• Expensive installation

• Limited availability, especially in rural areas

**4. Satellite Internet**

Pros:

• Available in remote and rural locations

• No need for physical cables

Cons:

• Weather can affect signal

• Data caps and higher cost

**5. broadband**

Pros:

• High-speed internet

• Supports multiple devices

Cons:

• Can be expensive

• Some plans have data limits

1. **[THEORY EXERCISE]: How does broadband differ from fiber-optic internet?**

* Broadband is a general term for high-speed internet that includes various types like DSL, cable, satellite, and fiber, while fiber-optic internet is a specific type of broadband that uses light signals through glass fibers to deliver much faster, more reliable, and lower-latency internet compared to other broadband types.

# protocols

1. **[LAB EXERCISE]: Simulate HTTP and FTP requests using command line tools (e.g., curl).**

* To simulate HTTP and FTP requests using command line tools, we use commands like curl. HTTP is used to request web pages or data from a server, while FTP is used to upload or download files. These tools help us understand how data is transferred between a client and a server over the internet.

1. **[THEORY EXERCISE]: What are the differences between HTTP and HTTPS protocols?**

* HTTP and HTTPS are both used to transfer data between a user's web browser and a web server. The main difference is that HTTP is a protocol used to transfer data over the web, but it is not secure. HTTPS is the secure version of HTTP that uses encryption (SSL/TLS) to protect data, making it safer for sensitive information like passwords and payments.

# Application Security

1. **[LAB EXERCISE]: Identify and explain three common application security vulnerabilities.Suggest possible solutions.**

* Three common application security vulnerabilities are SQL Injection, Cross-Site Scripting (XSS), and Broken Authentication.

1. SQL Injection

An attacker inserts harmful SQL code into a form input (like login fields) to access or manipulate the database.

Solution: Use prepared statements and validate input.

2. Cross-Site Scripting (XSS)

An attacker injects Malicious scripts are injected into web pages to attack users.

Solution: Sanitize input and escape output.

3. Broken Authentication

Poorly designed login systems allow attackers to gain unauthorized access to user accounts.

Solution: Use strong passwords, MFA, and secure sessions.

1. **[THEORY EXERCISE]: What is the role of encryption in securing applications?**

* **Protects data in transit**: Secures data sent over networks (e.g., HTTPS).
* **Protects data at rest**: Keeps stored data safe (e.g., databases, files).
* **Prevents data breaches**: Even if hackers access the data, they can't read it without the key.
* **Ensures privacy and trust**: Builds user confidence in using the application.

# Software Applications and Its Types

1. **[LAB EXERCISE]: Identify and classify 5 applications you use daily as either system software Or application software**.

I use the following 5 applications daily:

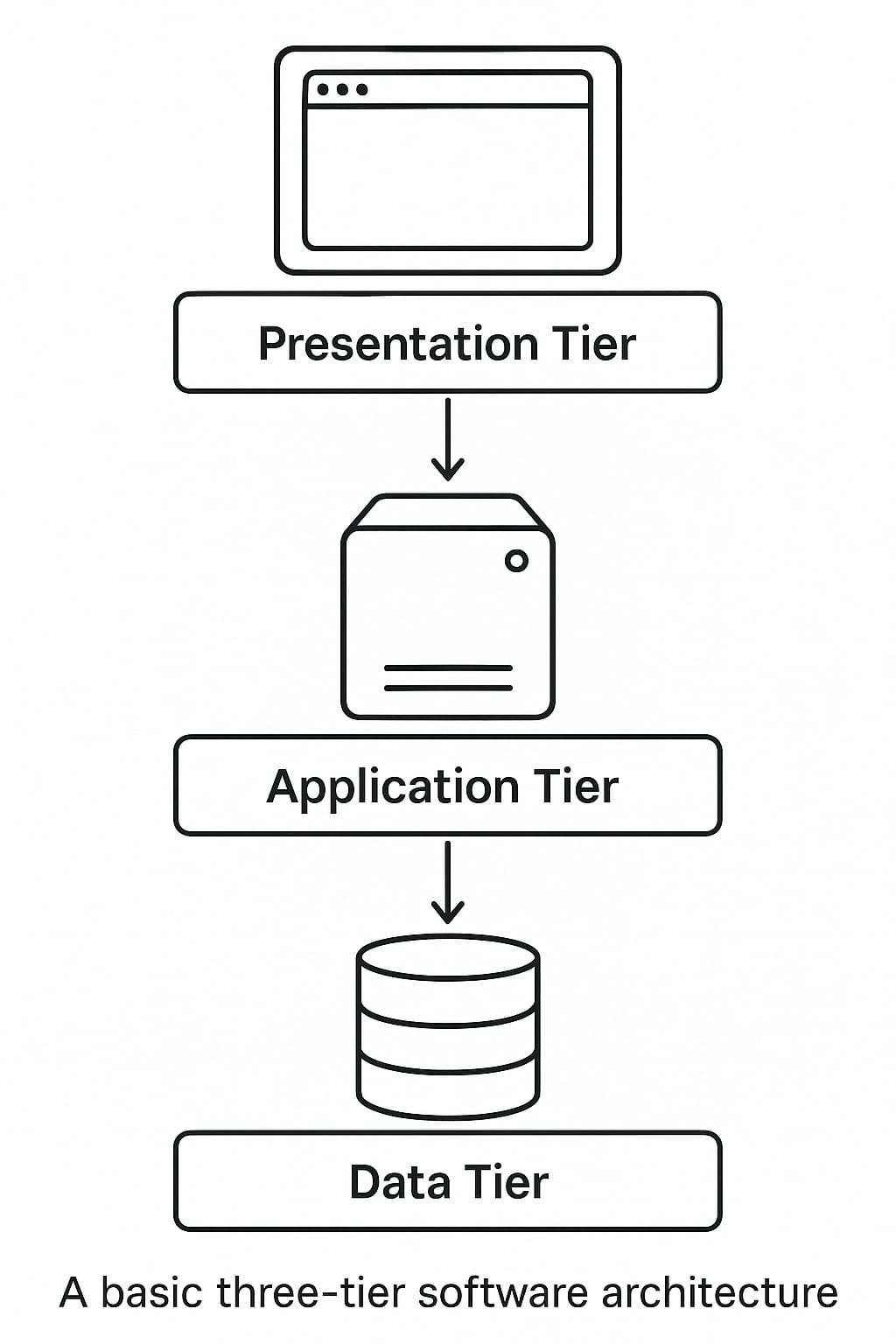
1. **Google Chrome** – *Application Software* (for browsing the internet)
2. **Microsoft Word** – *Application Software* (for writing documents)
3. **WhatsApp** – *Application Software* (for messaging and calling)
4. **Windows OS** – *System Software* (runs the computer and manages hardware)
5. **Antivirus Software** – *System Software* (protects the system from threats)

These are classified based on whether they help users do tasks (application) or help the system run (system).

1. **[THEORY EXERCISE]: What is the difference between system software and application software?**
2. System software runs and manages the computer’s hardware and basic functions (like Windows or Linux), while application software is used by users to perform specific tasks like writing documents, browsing the web, or chatting (like MS Word or WhatsApp).

# Software Architecture

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



* **Presentation Tier** – The user interface (e.g., web browser) where users interact with the application.
* **Application Tier** – The logic layer that processes user input, makes decisions, and handles operations.
* **Data Tier** – The database layer where all application data is stored and managed.

1. **[THEORY EXERCISE]: What is the significance of modularity in software architecture?**

* Modularity in software architecture means breaking a system into smaller, independent parts (modules). It’s important because it makes the software easier to understand, test, maintain, and update without affecting the whole system.

# Layers in Software Architecture

1. **[LAB EXERCISE]: Create a case study on the functionality of the presentation, business logic, and Data access layers of a given software system.**

📚 **Case Study Example: Hospital Management System**

**1. Presentation Layer (User Side)**

This is what the user sees: pages to book appointments, view doctor schedules, or access reports.

**Example:** A patient logs into the hospital portal to book an appointment with a doctor.

**2. Business Logic Layer (Processing Side)**

This checks the rules: Is the doctor available? Is the patient registered?

**Example:** The system checks if the doctor is free at the selected time and if the patient’s ID is valid.

**3. Data Access Layer (Database Side)**

This layer interacts with the database: stores appointments, retrieves patient records.

**Example:** The system saves the appointment details to the database and updates the doctor’s schedule.

1. **[THEORY EXERCISE]: Why are layers important in software architecture?**

* Layers are important in software architecture because they **separate different concerns of the system**, making the software easier to develop, maintain, and scale. Each layer handles a specific responsibility—such as user interaction, business rules, or data storage—so changes in one layer don’t heavily affect others. This clear organization improves code quality, teamwork, and flexibility.

# Software Environments

1. **[LAB EXERCISE]: Explore different types of software environments (development, testing, production).Set up a basic environment in a virtual machine.**

* **Different Types of Software Environments**

1. **Development Environment**
   * This is where programmers write and create the software.
   * It has tools like code editors and debuggers.
   * It’s flexible and changes often.
2. **Testing Environment**
   * After writing the software, it is moved here to check if everything works well.
   * Bugs are found and fixed in this environment.
   * It tries to copy the real-world setup but isn’t live.
3. **Production Environment**
   * This is the real, live environment where the software is used by customers.
   * It is stable and reliable.
   * Changes here are made carefully to avoid breaking things.

* **How to Set Up a Basic Environment in a Virtual Machine (VM)**

1. **Choose a Virtual Machine software:**  
   Examples: VirtualBox, VMware, or Hyper-V.
2. **Download an Operating System (OS):**  
   For example, get an ISO file of Ubuntu (Linux) or Windows.
3. **Create a New Virtual Machine:**
   * Open the VM software.
   * Create a new VM and assign memory (RAM), disk space, and CPU cores.
   * Attach the OS ISO file to the VM.
4. **Start the Virtual Machine:**
   * Boot up the VM with the OS installer.
   * Follow the steps to install the OS inside the VM.
5. **Set Up Your Environment:**
   * Install software like code editors (e.g., VS Code), programming languages, or tools needed.
   * This can be your **development environment** inside the VM.
6. **Use Snapshots:**
   * Save the VM state so you can return to it if needed (useful for testing).
7. **[THEORY EXERCISE]: Explain the importance of a development environment in software production.**

* A development environment is important in software production because it provides a safe space for programmers to write, test, and debug code using helpful tools. It helps find and fix errors early, supports teamwork, and speeds up development, all without affecting the live (production) system.

# Source Code

1. **[LAB EXERCISE]: Write and upload your first source code file to GitHub.**
2. **THEORY EXERCISE: What is the difference between source code and machine code?**

* **Source Code**
* Written by programmers in human-readable programming languages like Python, C++, or Java.
* Example: print("Hello, world!")
* Easy to read, write, and edit.
* **Machine Code**
* The binary code (1s and 0s) that a computer’s processor understands and executes.
* Not readable by humans.
* Generated by translating (compiling or interpreting) source code

# GitHub and Introductions

1. **[LAB EXERCISE]: Create a GitHub repository and document how to commit and push code changes.\**

✅ 1. Create a Repository on GitHub

1. Go to <https://github.com> and log in.
2. Click the + icon → New repository.
3. Enter a repository name (e.g., my-project), choose Public or Private.
4. *(Optional)* Check "Initialize with a README".
5. Click Create repository.

**EXAMPLE :**

git status

git add .

git commit -m "Initial project files"

git push origin main

1. **[THEORY EXERCISE]: Why is version control important in software development?**

* Version control is important in software development because it helps track changes, manage code history, and allows multiple developers to work together without losing or overwriting each other’s work. It ensures better collaboration, easier debugging, and safer project management.

# Student Account in GitHub

1. **[LAB EXERCISE]: Create a student account on GitHub and collaborate on a small project with a classmate.**

* Create GitHub Student Account & Collaborate
* Sign up at github.com using your student email.
* Apply for Student Pack: education.github.com/pack
* Create a repo: Click "+" → "New repository"
* Go to Settings → Collaborators, add your classmate’s GitHub username.
* They accept the invite to collaborate.

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

* The benefits of using GitHub for students include:
* Easy collaboration on projects with classmates.
* Tracking changes in code through version control.
* Building a portfolio to showcase their work to employers.
* Learning industry tools like Git, which are widely used in software development.
* Access to free resources and tools through the GitHub Student Pack.
* Backing up code safely online to prevent data loss.

# Types of Software

1. **LAB EXERCISE: Create a list of software you use regularly and classify them into the**

**Following categories: system, application, and utility software.**

Sure! Here’s an example list of software classified into **system**, **application**, and **utility software**:

**System Software**

* **Windows 11** (Operating System)

**Application Software**

* **Microsoft Word** (Word processor)
* **Google Chrome** (Web browser)
* **Adobe Photoshop** (Image editor)

**Utility Software**

* **Disk Cleanup** (Tool to free up space)

 **CCleaner** – System Cleanup

 **Backup Software** – Data Backup

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

* Open-source software is freely available with its source code, allowing users to view, modify, and share it, while proprietary software is owned by a company, has restricted access to its source code, and usually requires payment or a license to use.

# GIT and GITHUB Training

1. **[LAB EXERCISE]: Follow a GIT tutorial to practice cloning, branching, and merging repositories.**

|  |  |
| --- | --- |
| Task | Command Example |
| Clone | **git clone URL** |
| branch | **git checkout -b feature-name** |
| Merge | **git checkout main → git merge branch** |

1. **[THEORY EXERCISE]: How does GIT improve collaboration in a software development team?**

* Git improves collaboration in a software development team by allowing multiple developers to work on the same project at the same time without overwriting each other's code. It tracks all changes, supports branching for individual tasks, and enables easy merging of updates**.**

# Application Software

1. **LAB EXERCISE: Write a report on the various types of application software and how they Improve productivity.**

• Types of Application Software and Their Productivity Benefits

1. **Word Processing Software**

o Example: Microsoft Word, Google Docs

o Use: Writing documents, letters, reports

o Productivity: Speeds up writing with tools like auto-correct, templates, and collaboration.

**2. Spreadsheet Software**

o Example: Microsoft Excel, Google Sheets

o Use: Data analysis, financial planning, reports

o Productivity: Automates calculations and data organization; useful in business and accounting.

3. **Presentation Software**

o Example: Microsoft PowerPoint, Prezi

o Use: Creating visual slides for meetings or lectures

o Productivity: Enhances communication of ideas clearly and quickly.

4. **Database Management Software**

o Example: Oracle, MS Access, MySQL

o Use: Storing and managing large datasets

o Productivity: Ensures fast data access and organization for decision-making.

5. **Multimedia Software**

o Example: VLC Media Player, Adobe Premiere Pro

o Use: Creating and editing audio, video, and images

o Productivity: Supports creative tasks and professional media production.

6. **Communication Software**

o Example: Zoom, Microsoft Teams, Slack

o Use: Messaging, video conferencing, file sharing

o Productivity: Improves team collaboration and reduces time spent on meetings.

7. **Web Browsers**

o Example: Chrome, Firefox

o Use: Internet access for research and online tools

o Productivity: Quick access to information, tools, and cloud services.

1. **[THEORY EXERCISE]: What is the role of application software in businesses?**

* The role of application software in businesses is to help perform specific tasks efficiently, such as managing documents, handling finances, communicating with clients, analyzing data, and automating daily operations—ultimately improving productivity, accuracy, and decision-making across the organization.

# Software Development Process

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

**1. Requirement Gathering**

**2. Planning**

**3. Design**

**4. Development**

**5. Testing**

**6. Deployment**

**7. Maintenance**

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

* **Requirement Analysis** – Understanding what the software needs to do by gathering requirements from users or clients.
* **Design** – Planning the software’s structure, architecture, and interface before coding begins.
* **Implementation (Coding)** – Writing the actual code based on the design using programming languages.
* **Testing** – Checking the software for bugs, errors, and performance issues to ensure it works as intended.

* **Deployment** – Releasing the software to users or moving it to a live environment.
* **Maintenance** – Updating, fixing, and improving the software after it is in use.

# Software Requirement

1. **[LAB EXERCISE]: Write a requirement specification for a simple library management system.**

=> Library Management System – Requirement Specification

1. Purpose:

To automate book management, issue, return, and member handling in a library.

2. Users:

• Admin/Librarian: Manage books and members

• Students/Members: Search and view books

3. Main Features:

• Login system for admin

• Add/edit/delete books

• Register/edit/remove members

• Issue and return books

• Search books by title/author

• Generate reports

4. System Requirements:

• Simple user interface

• Fast access and search

• Role-based access for security

5. Assumptions:

• Unique ID for each book

• Max 3 books issued per user

• 15-day return period

1. **THEORY EXERCISE: Why is the requirement analysis phase critical in software development?**

* The requirement analysis phase is critical because it helps identify what the software must do, ensures developers and clients have a shared understanding, and reduces errors and costly changes later in the project.

# Software Analysis

1. **[LAB EXERCISE:] Perform a functional analysis for an online shopping system.**

**• Functional Analysis – Online Shopping System**

**1. User Roles:**

**• Customer/User**

**• Admin**

**•** (Optional: Vendor/Seller if multi-vendor platform**)**

**2. Functional Requirements:**

**Customer Functions:**

**• User Registration/Login –** Create and access user account

**• Browse Products –** View items by category, brand, etc.

**• Search Products –** Search by name, price, rating, etc.

**• Add to Cart –** Add selected items to shopping cart

**• Place Order –** Checkout and complete purchase

**• Payment –** Online payment via card, UPI, or COD

**• Order History –** View past orders and status

**• Product Review/Rating –** Leave feedback

**Admin Functions:**

**• Manage Users –** Add/edit/remove user accounts

**• Manage Products –** Add/edit/delete product listings

**• Manage Categories –** Create or update product categories

**• View Orders –** See customer orders and statuses

**• Process Refunds/Returns –** Handle customer complaints

**• Report Generation –** Sales, revenue, and customer data

**3. System Functional Modules:**

**•** User Module

• Product Catalog Module

• Shopping Cart Module

• Order Management Module

• Payment Gateway Module

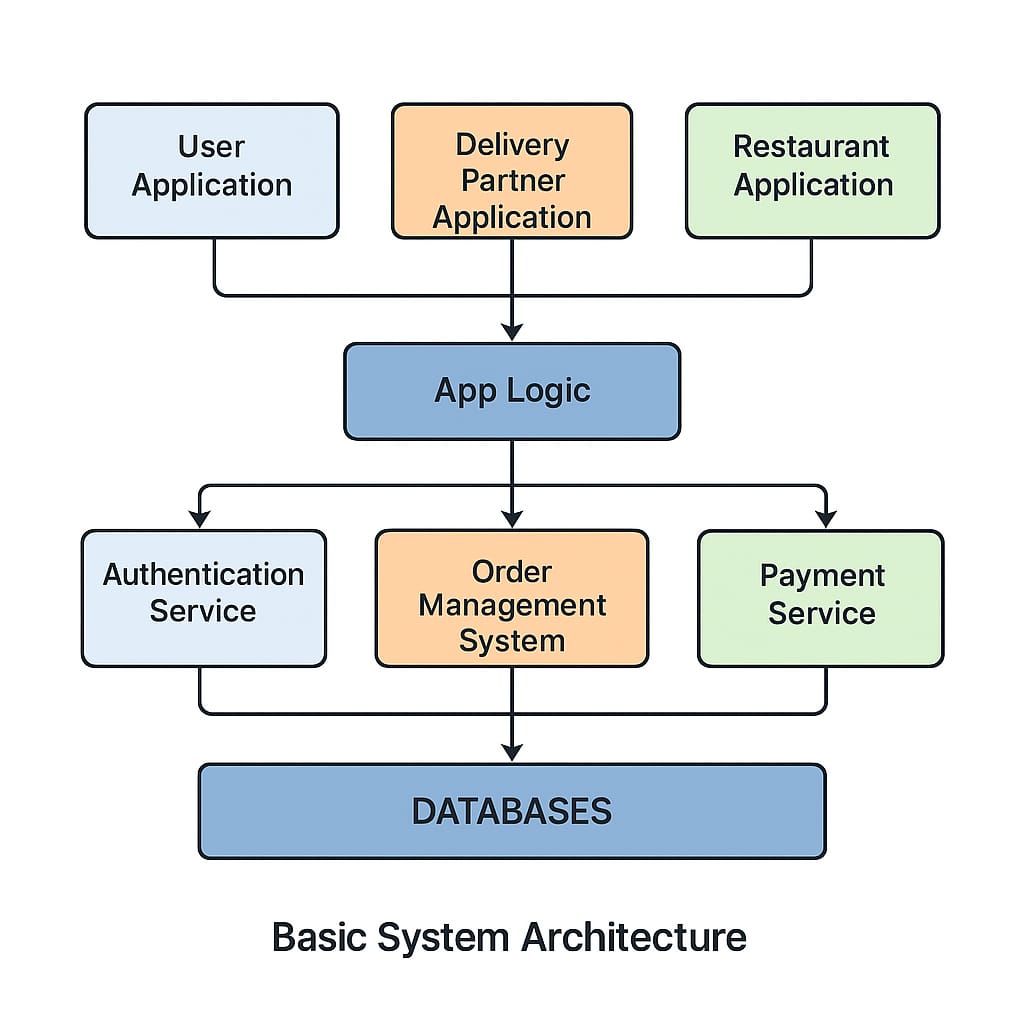
• Admin Dashboard Module

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

* The role of software analysis in the development process is to understand and define the needs, problems, and goals of the software, ensuring that the final product meets user requirements and functions correctly within its intended environment.

# System Design

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



**1.** **Applications (Top Layer):**

• User Application – For customers to place orders.

• Delivery Partner Application – For delivery agents to view and deliver orders.

• Restaurant Application – For restaurants to receive and manage orders.

**2. App Logic (Middle Layer):**

• The main control unit of the system.

• Connects all the applications to the backend services.

**3. Services (Supporting Systems):**

• Authentication Service – Manages user login and security.

• Order Management System – Handles order processing, updates, and tracking.

• Payment Service – Processes online payments safely.

1. **Databases (Bottom Layer):**

• Stores all data like user profiles, menu items, orders, payments, and history.

1. **[THEORY EXERCISE]: What are the key elements of system design?**

**Architecture Design** – Defines overall system structure and component interaction.

**Data Design** – Organizes how data is stored, processed, and accessed.

**Interface Design** – Focuses on user and system interaction.

**Component Design** – Breaks system into smaller, manageable modules.

**Security and Performance Design** – Ensures safety, speed, and scalability of the system.

# Software Testing

1. **[LAB EXERCISE]: Develop test cases for a simple calculator program.**

**1. Functional Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Input | Expected Output |
| TC001 | Addition of two positive numbers | 5 + 3 | 8 |
| TC002 | Addition of positive and negative number | 7 + (-2) | 5 |
| TC003 | Subtraction of two numbers | 10 - 4 | 6 |
| TC004 | Multiplication of two numbers | 6 \* 3 | 18 |
| TC005 | Division of two numbers | 12 / 4 | 3 |
| TC006 | Division resulting in a float | 7 / 2 | 3.5 |
| TC007 | Multiplying by zero | 15 \* 0 | 0 |
| TC08 | Division by zero | 5 / 0 | Error or "Cannot divide by zero" |

**2. Boundary Test Cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| TestCase ID | Description | Input | Expected Output |
| TC011 | Maximum integer addition | 2147483647 + 1 | Overflow or Error |
| TC012 | Minimum integer subtraction | -2147483648 - 1 | Underflow or Error |
| TC013 | Very small float multiplication | 0.0000001 \* 0.0000001 | 1e-14 |
| TC014 | Large float division | 1e10 / 2 | 5e9 |

**3. Invalid Input Test Cases:**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Input | Expected Output |
| TC015 | Non-numeric input | "a" + 5 | Error: Invalid input |
| TC016 | Empty input | "" | Error: Input required |
| TC017 | Symbol instead of number | $# + 10 | Error: Invalid input |
| TC018 | Division by string | 10 / "two" | Error: Invalid input |
| TC019 | Multiple operators | 2 ++ 3 | Error: Syntax error |

1. **Chained Operations (if supported):**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Input | Expected Output |
| TC020 | Chain addition and multiplication | 2 + 3 \* 4 | 14 (if operator precedence) |
| TC021 | Chain with parentheses | (2 + 3) \* 4 | 20 |
| TC022 | Mix of operations | 10 - 2 + 5 | 13 |

1. **[THEORY EXERCISE]: Why is software testing important?**

* Software testing is important because it helps find and fix bugs or errors before the software is released. It ensures the software works correctly, meets user requirements, and is reliable and secure. Testing improves the quality of the software, prevents costly problems later, and makes sure users have a good experience.

# Maintenance

1. **[LAB EXERCISE]: Document a real-world case where a software application required.**

* **Critical maintenance.**

**Case Study: Facebook Outage – October 2021**

**Background:**  
On October 4, 2021, Facebook, WhatsApp, and Instagram experienced a global outage lasting over 6 hours. The disruption was caused by a router misconfiguration during routine maintenance.

**Issue:**  
The misconfiguration disrupted network traffic, disconnecting data centers . Even Facebook’s internal tools failed, significantly delaying recovery efforts.

**Impact:**

* Worldwide service outage
* Over $60 million in lost ad revenue
* Facebook’s stock dropped by 5%

**Action Taken:**

* Engineers manually accessed data centers to regain control
* Systems were gradually restarted
* Recovery tools and protocols were updated for future resilience

**Lessons Learned:**

* Backup access methods are critical
* Automated rollback mechanisms for configuration changes improve safety
* Regular disaster recovery drills are essential

1. **[THEORY EXERCISE]: What types of software maintenance are there?**

**Corrective Maintenance**

Fixes bugs or errors found after the software is released.

**Adaptive Maintenance**

Updates software to work with new environments (e.g., new OS or hardware).

**Perfective Maintenance**

Improves performance or adds new features based on user feedback.

**Preventive Maintenance**

Makes changes to prevent future problems and improve long-term stability.

# Development

1. **[THEORY EXERCISE]: What are the key differences between web and desktop applications?**

|  |  |  |
| --- | --- | --- |
| Feature | Web Applications | Desktop Applications |
| Installation | No installation; runs in web browsers | Requires installation on the computer |
| Internet | Usually needs internet connection | Can work without internet |
| Platform | Works on any device with a browser | Designed for specific operating systems |
| Updates | Updated automatically by the provider | Users must manually install updates |
| Performance | Depends on internet speed and browser | Generally faster, uses local system power |
| Accessibility | Accessible from anywhere with internet | Accessible only on the installed device |

# 27. Web Application

1. **THEORY EXERCISE: What are the advantages of using web applications over desktop applications?**

**No Installation Required**

* Web apps run directly in a browser—no need to install software on your device.

**Access from Anywhere**

* You can use web apps on any device with internet access, anytime, anywhere.

**Automatic Updates**

* Updates are done on the server side, so users always use the latest version without downloading anything.

**Cross-Platform Compatibility**

* Works on multiple operating systems (Windows, macOS, Linux) as long as a browser is available.

**Easier Collaboration**

* Web apps often support real-time sharing and editing (e.g., Google Docs), making teamwork easier.

# 28. Designing

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

* UI/UX design plays a key role in application development by making the application easy to use, visually appealing, and user-friendly, which enhances user satisfaction, reduces errors, and improves overall usability and engagement.

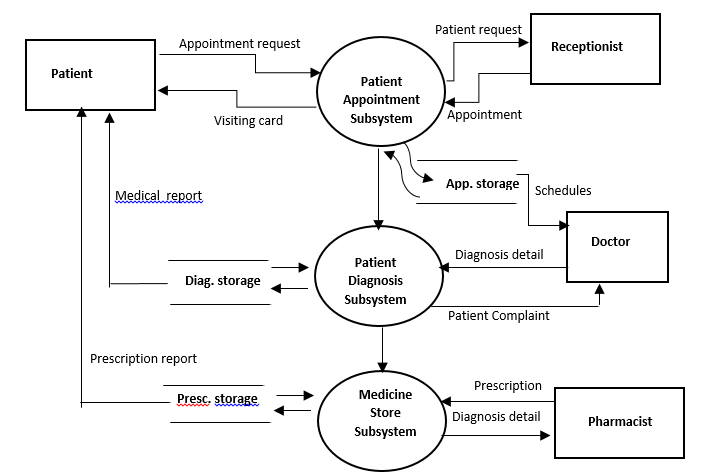
# 29. Mobile Application

1. **[THEORY EXERCISE]: What are the differences between native and hybrid mobile apps?**

|  |  |  |
| --- | --- | --- |
| Feature | Native Apps | Hybrid Apps |
| Platform | One platform (iOS/Android) | Multiple platforms (cross-platform) |
| Performance | Faster and smoother | Slightly slower |
| Development | Separate code for each platform | Single codebase for all |
| Device Access | Full access | Limited or plugin-based |
| Cost & Time | Higher | Lower |

# 30. DFD (Data Flow Diagram)

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



1. **[THEORY EXERCISE]:** **What is the significance of DFDs in system analysis?**

* Data Flow Diagrams (DFDs) are important in system analysis because they visually show how data moves through a system. They help analysts understand the processes, data inputs, outputs, and storage involved. DFDs make it easier to identify system requirements, spot inefficiencies, and communicate with both technical and non-technical stakeholders. This leads to better system design, fewer errors, and smoother development.

# 31. Desktop Application

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

#include <stdio.h>

int main() {

double num1, num2, result;

char op;

printf("Simple Calculator\n");

printf("Enter expression (e.g., 3 + 4): ");

scanf("%lf %c %lf", &num1, &op, &num2);

switch (op) {

case '+':

result = num1 + num2;

printf("Result: %.2lf\n", result);

break;

case '-':

result = num1 - num2;

printf("Result: %.2lf\n", result);

break;

case '\*':

result = num1 \* num2;

printf("Result: %.2lf\n", result);

break;

case '/':

if (num2 == 0) {

printf("Error: Division by zero\n");

} else {

result = num1 / num2;

printf("Result: %.2lf\n", result);

}

break;

default:

printf("Error: Invalid operator\n");

}

return 0;

}

# 32. Flow Chart

1. **[THEORY EXERCISE]: How do flowcharts help in programming and system design?**

* Flowcharts help in programming and system design by visually representing the flow of logic or processes.

They make it easier to understand, debug, and communicate system behavior. Flowcharts also help in planning the structure before actual coding, reducing errors and improving efficiency.

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

