Module 1 – Overview of IT Industry

Name: VAIDANGI GADHIYA

What is a Program?

A program is a collection of instructions written in a programming language that tells the computer how to perform a specific task.

In C language, a program consists of statements written in a proper structure, and it starts executing from the main() function. The computer follows these instructions step by step to produce a desired result.

a program is a set of commands that control the working of a computer.

LAB EXERCISE :

Write a simple "Hello World" program in two different programming languages and compare their structure and syntax.

C Language Program

```
#include <stdio.h>
int main()
{
   printf("Hello World");
   return 0;
}
```

C++ Language Program

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Hello World";
   return 0;
}
```

Comparison of Structure and Syntax

Basis	**C Language**	**C++ Language**
Header File	#include <stdio.h>	#include <iostream>
Output Statement	printf()	cout
Namespace	Not required	using namespace std;
Programming Type	Procedural	Object-Oriented + Procedural
Syntax Style	Function-based output	Stream-based output

Both programs produce the same output, but the syntax and structure are slightly different.
C uses printf() for output, while C++ uses cout.
C++ supports more advanced features compared to C.

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 tells the computer what task to perform.

In C language, a program starts from the main() function. When we run it, the compiler converts the code into machine language, and the computer executes the instructions step by step.

A program generally works by taking input, processing it, and giving output.

What is Programming?

Programming is the process of writing instructions in a programming language so that a computer can perform a specific task.

It involves understanding a problem, creating a solution, and writing code to implement that solution.In simple words, programming means giving step-by-step instructions to the computer to solve a problem.

THEORY EXERCISE

What are the key steps involved in the programming process?

The main steps involved in the programming process are:

1. Understanding the Problem – Clearly understand what needs to be solved.
2. Planning the Solution – Create an algorithm or flowchart.
3. Coding – Write the program in a programming language.
4. Compilation – Convert the code into machine language.
5. Testing – Run the program and check for errors.
6. Debugging – Fix the errors if any.
7. Maintenance – Update or improve the program when needed.

Types of Programming Languages

Programming languages are mainly divided into two types:
1. High-Level Languages

- Easy to read and write
- Close to human language
- Example: C, Java, Python

2. Low-Level Languages

- Close to machine language
- Difficult to understand
- Example: Assembly language

High-level languages are user-friendly, low-level languages provide more control over hardware.

THEORY EXERCISE

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

High-level and low-level programming languages are different based on how close they are to human language or machine language.

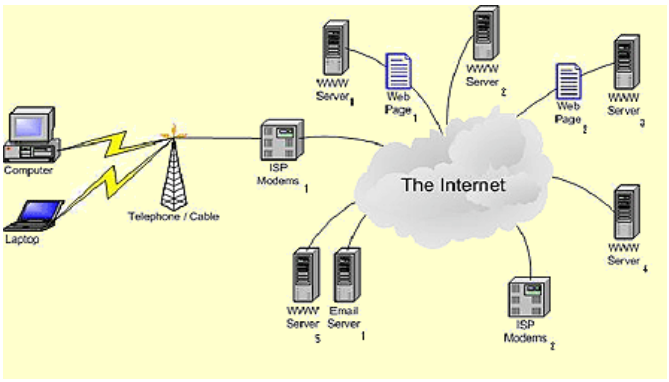
High-level languages : are easy to read, write, and understand. They use simple English-like words and are machine independent. Examples include C, Java, and Python.

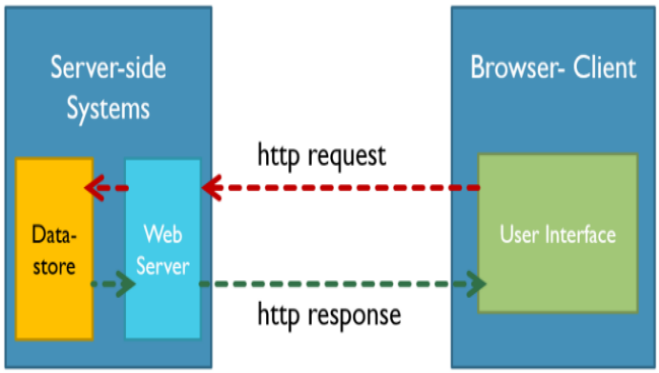
Low-level languages : are closer to machine language and are harder to understand. They are machine dependent and give more control over hardware. An example is Assembly language.

high-level languages are easier for programmers
low-level languages are closer to the computer's hardware.

World Wide Web & How Internet Works

The World Wide Web (WWW) is a collection of websites and web pages that we access using the internet through a web browser.
The Internet is a global network that connects computers and devices so they can communicate with each other.
How the Internet Works



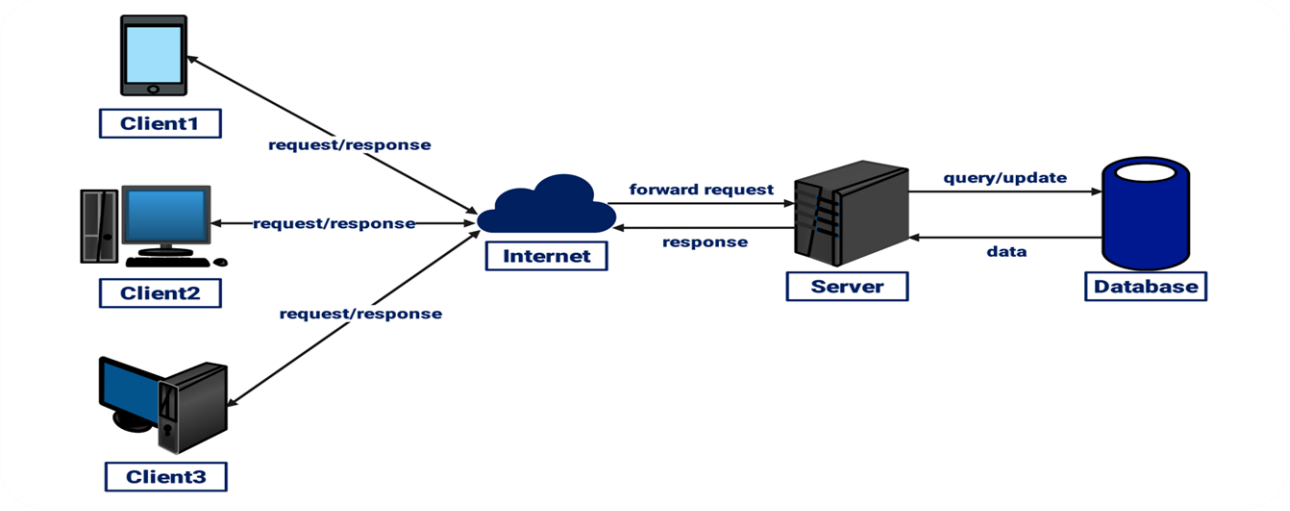


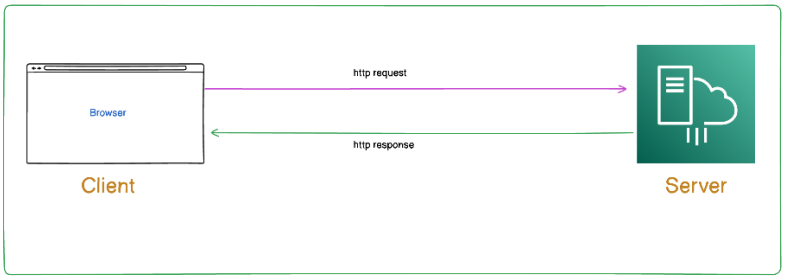
1. User enters a website address in the browser.
2. The browser sends a request to the server.
3. The server processes the request.
4. The server sends the response back.
5. The browser displays the webpage.

the internet connects devices, and the World Wide Web allows us to access websites.

LAB EXERCISE

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





Steps of Data Transmission:

1. The user enters a website address in the browser (Client).
2. The browser sends a request through the Internet Service Provider (ISP).
3. The request travels over the internet to the web server.
4. The server receives and processes the request.
5. The server sends a response back through the internet.
6. The browser receives the response and displays the webpage.

This complete process is called the request and response cycle and it happens within seconds.

THEORY EXERCISE

Describe the Roles of Client and Server in Web Communication

In web communication, the client and the server work together to exchange information.

The client is the user's device such as a computer, mobile phone, or web browser. It sends a request to the server to access a website, application, or data. After receiving the response from the server, it displays the information to the user.

The server is a powerful computer that stores websites, files, and databases. It receives the client's request, processes it, and sends back the required response.

In simple terms, the client requests information and the server provides it. This communication follows the request–response model.

Network Layers on Client and Server

Both client and server use networking layers to communicate over the internet. These layers ensure that data is transferred correctly.

1. Application Layer – Handles protocols like HTTP and HTTPS.
2. Transport Layer – Uses TCP or UDP to manage data transfer.
3. Internet Layer – Manages IP addresses and routing of data.
4. Network Access Layer – Handles physical transmission through cables or wireless networks.

These layers work together to send data from the client to the server and return the response successfully.

LAB EXERCISE

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

(Python)

Server Code

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"Hello Client")

server = HTTPServer(("localhost", 8000), Handler)
print("Server started on port 8000")
server.serve_forever()
```

Client Code

```
import requests

response = requests.get("http://localhost:8000")
print("Response from Server:", response.text)
```

- The server runs on port 8000 and waits for requests.
- The client sends an HTTP GET request to the server.
- The server receives the request and sends back a response.
- The client prints the response on the screen.

THEORY EXERCISE

Explain the Function of the TCP/IP Model and Its Layers

The TCP/IP model is a communication model used for sending data over the internet. It defines how data is transmitted from one device to another.

It has four main layers:

1. **Application Layer** –
   This layer interacts with the user. It includes protocols like HTTP, FTP, and SMTP. It prepares data for communication.

2. **Transport Layer** –
   This layer ensures reliable data transfer.
   TCP provides reliable communication, while UDP provides faster but less reliable communication.
3. **Internet Layer** –
   This layer is responsible for logical addressing and routing. It uses IP addresses to send data to the correct destination.
4. **Network Access Layer** –
   This layer handles the physical transmission of data through cables, Wi-Fi, or other hardware.

In simple words, the TCP/IP model divides communication into layers so data can travel properly from sender to receiver.

Client and Servers

A client is a device or program that requests services or data.
A server is a device or program that provides services or data.
In web communication:

- The client sends a request (for example, opening a website).
- The server processes the request.
- The server sends back a response.

This request-response process forms the basic structure of internet communication.

THEORY EXERCISE

Explain Client–Server Communication

Client–server communication is a process where one computer (client) sends a request and another computer (server) responds to that request.
The client asks for some service or data, and the server processes the request and sends back the result.
Example:
When we open a website, the browser (client) sends a request to the web server. The server then sends the webpage back to the browser.

Types of Internet Connections

There are different types of internet connections:

1. Broadband – Common and affordable, used in homes and offices.
2. Fiber Optic – Very fast and reliable, uses light signals.
3. DSL – Uses telephone lines for internet connection.
4. Mobile Data (4G/5G) – Internet through mobile networks.
5. Satellite – Used in remote areas where other connections are not available.

Each type differs in speed, cost, and reliability.

LAB EXERCISE:

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

Internet Connection	Pros	Cons
Broadband (Cable/DSL)	Affordable, widely available, good for daily use	Speed can be lower than fiber, affected by network traffic
Fiber Optic	Very fast, reliable, low latency	Expensive, limited availability in some areas
DSL	Uses existing phone lines, easy to install	Slower than fiber, speed decreases with distance
Mobile Data (4G/5G)	Works anywhere with network coverage, portable	Can be costly, speed depends on signal quality
Satellite	Works in remote areas, no cables needed	High latency, slower speed, more expensive

THEORY EXERCISE

How Broadband Differs from Fiber-Optic Internet ?

Broadband: usually uses copper cables (like DSL or cable) to deliver internet. It provides moderate speed and is widely available.

Fiber-Optic Internet: uses light signals through glass or fiber cables. It is much faster, more reliable, and less affected by distance.

Fiber is faster and better for heavy internet use

broadband is more common and affordable.

Protocols

Protocols are rules that allow computers to communicate over a network. They define how data is sent, received, and interpreted.

Common protocols:

- HTTP – Used for websites (web pages).
- HTTPS – Secure version of HTTP with encryption.
- FTP – Used to transfer files between computers.
- TCP/IP – Core protocol for internet communication, ensures data is sent correctly.

LAB EXERCISE

Simulate HTTP and FTP Requests Using Command Line

You can use the **curl** command to test HTTP and FTP requests.

HTTP Request (Get a Web Page)

```
curl http://example.com
```

- Sends a request to a website.
- Shows the webpage content in the terminal.

FTP Request (Download a File)

```
curl -u username:password ftp://ftp.example.com/file.txt -O
```

- Connects to an FTP server.
- Downloads the file file.txt.

HTTP: For web pages.

FTP: For files.
curl: Command line tool to request data from servers.

THEORY EXERCISE:

What are the differences between HTTP and HTTPS protocols?

HTTP (HyperText Transfer Protocol) and HTTPS (HyperText Transfer Protocol Secure) are protocols used for communication between a client and a server.

Feature	**HTTP**	**HTTPS**
Full Form	HyperText Transfer Protocol	HyperText Transfer Protocol Secure
Security	Not secure – data is sent in plain text, so hackers can read it	Secure – data is encrypted using SSL/TLS, making it unreadable to attackers
Port Number	80	443
Encryption	No encryption	Uses SSL/TLS encryption for security
Use Case	General websites where security is not critical	Websites that handle sensitive data like banking, shopping, and login pages
SEO & Trust	Less trusted by browsers; may show "Not Secure"	More trusted; browsers show a padlock icon, boosts credibility
Data Integrity	Data can be intercepted or modified	Ensures data is not tampered with during transfer

Application Security

Application security is the process of protecting software applications from threats and attacks to keep data and users safe.

Common Vulnerabilities:

1. SQL Injection – Hackers send malicious database queries to access or modify data.
2. Cross-Site Scripting (XSS) – Malicious scripts are injected into websites to attack users.
3. Weak Passwords – Easy passwords can be guessed, allowing unauthorized access.
4. Insecure Data Storage – Storing sensitive data without encryption can be risky.
5. Improper Session Management – Hackers can steal session tokens to impersonate users.

LAB EXERCISE:

Identify and explain three common application security vulnerabilities. Suggest possible solutions.

SQL Injection

- What it is: Hackers insert malicious SQL queries into input fields to access or manipulate the database.
- Solution: Always validate and sanitize user inputs. Use prepared statements or parameterized queries.

Cross-Site Scripting (XSS)

- What it is: Malicious scripts are injected into a website to attack users' browsers.
- Solution: Escape or sanitize user inputs, use Content Security Policy (CSP), and avoid directly displaying user input.

Weak Passwords

- What it is: Users use easy or common passwords, making accounts vulnerable to hacking.
- Solution: Enforce strong password rules, implement multi-factor authentication, and store passwords using hashing.

THEORY EXERCISE

What is the role of encryption in securing applications?

Encryption is the process of converting data into a secret code so that only authorized users can read it.
Role in Application Security:

1. Protects Data: Sensitive data like passwords, credit card info, and personal details are safe from hackers.
2. Secure Communication: Ensures data sent over the internet (like between client and server) cannot be intercepted or read by attackers.
3. Maintains Privacy: Only users with the correct key can access the information.
4. Prevents Tampering: Encryption helps ensure that data is not changed or corrupted during transmission.

Encryption keeps applications and user data safe, private, and secure from unauthorized access.

Software Applications and Its Types

What is Software :
Software is a set of instructions or programs that tell a computer how to perform specific tasks. Without software, the computer hardware is useless.

Types of Software

1. System Software – Helps the computer run and manage hardware.
    - Examples: Windows, Linux, Antivirus
2. Application Software – Helps users perform specific tasks.
    - Examples: Microsoft Word, Google Chrome, VLC Player

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

Application	Type of Software	Reason
Google Chrome	Application Software	Used for browsing the internet.
Microsoft Word	Application Software	Used for writing documents.
Windows 10/11	System Software	The operating system that runs the computer.

Application	Type of Software	Reason
VLC Media Player	Application Software	Used for playing audio/video files.
Antivirus (e.g., Avast, Windows Defender)	System Software	Helps manage and protect the system.

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

Feature	System Software	Application Software
Definition	Software that manages and controls the computer hardware and provides a platform for running application software.	Software designed to perform specific tasks for the user.
Purpose	To make the computer system functional.	To help users perform specific tasks like writing, browsing, or gaming.
Examples	Operating systems (Windows, Linux), device drivers, utilities	Microsoft Word, Google Chrome, VLC Media Player
Dependency	Runs independently but allows applications to run.	Depends on system software to function.
User Interaction	Indirect, mostly runs in the background.	Direct, user interacts frequently.

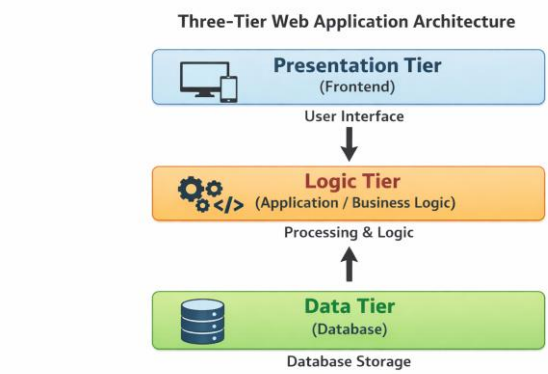
Software Architecture

Definition

Software architecture is the high-level structure of a software system, showing how different components interact to achieve functionality. It defines layers, modules, and communication between them.

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

Three-tier architecture is a software design pattern that separates a web application into three distinct layers: Presentation Tier, Logic Tier, and Data Tier. Each layer has a specific responsibility, which makes the application modular, scalable, and easier to maintain.



1. Presentation Tier (Frontend)
   - This is the user interface where users interact with the application.
   - Responsible for displaying information and capturing user input.
   - Examples: HTML, CSS, JavaScript, React, Angular.
   - Communicates with the Logic Tier to process requests.

2. Logic Tier (Application/Business Logic)
   - Handles the processing of data and business rules.
   - Acts as a bridge between the Presentation Tier and the Data Tier.
   - Examples: Node.js, Java, Python, PHP.
   - Ensures correct processing, calculations, and application behavior.

3. Data Tier (Database)
   - Responsible for storing, retrieving, and managing data.
   - Provides data to the Logic Tier when requested and stores updated information.
   - Examples: MySQL, MongoDB, PostgreSQL.

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

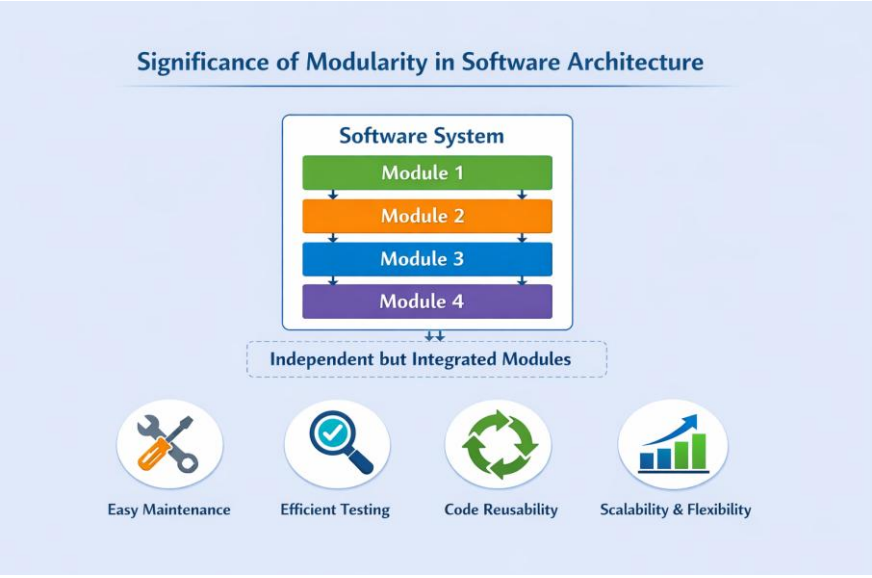
Significance of Modularity in Software Architecture:

Modularity is the practice of dividing software into smaller, independent modules, each performing a specific function. It is significant because it:

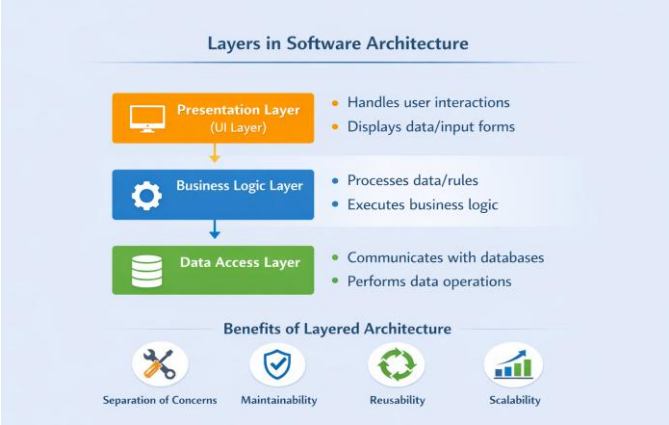
- Makes software easier to maintain and understand
- Allows efficient testing of individual modules
- Supports code reusability
- Enables scalability and flexibility for future enhancements
- Allows parallel development by multiple teams

Diagram:

Each module works independently but integrates to form the complete software system, highlighting the key benefits listed above.



Layers in Software Architecture



LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

Overview:
The Library Management System allows users to search for books, borrow them, and return them. It is divided into three layers: Presentation, Business Logic, and Data Access.

1. Presentation Layer (UI)

- Shows the list of books to the user.
- Takes input like "Enter book ID to borrow."
- Displays messages like "Book borrowed" or "Not available."

2. Business Logic Layer (Core Rules)

- Checks if the book is available.
- Handles borrowing rules, like updating availability.
- Sends messages back to the UI.

3. Data Access Layer (Database / Storage)

- Stores book records (book ID, title, availability).
- Updates book status when borrowed or returned.
- Retrieves book details for the business logic layer.

How the layers work together:

1. User sees books → Presentation Layer.
2. User chooses a book → Business Logic Layer checks rules.

3. Book data updated → Data Access Layer saves info.
4. Result shown → Presentation Layer displays message.

Diagram :

[User] → [Presentation Layer] → [Business Logic Layer] → [Data Access Layer] → [Database]

←---------- Response ----------←

THEORY EXERCISE: Why are layers important in software architecture?

- Layers separate concerns: UI, business rules, and data storage are handled separately.
- Makes the system easy to maintain: You can change the UI without touching the database.
- Reusability: Business logic can be reused in other applications.
- Testability: Each layer can be tested individually.
- Scalability: Layers can grow independently (e.g., more database servers or more web pages).

Layers make software organized, flexible, maintainable, and easy to test.

Software Environments

A software environment is a setup that provides the tools, platforms, and systems needed to develop, run, and maintain software. It defines the conditions under which software operates.

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

Types of Software Environments

1. Development Environment
    - Where software is created and tested by developers.
    - Includes: Code editors, compilers, IDEs, libraries, and test databases.
    - Example: Writing a C program in Code::Blocks or Visual Studio.
2. Testing / QA Environment
    - Where software is tested for bugs, errors, and performance before release.
    - Simulates real conditions without affecting actual users.
    - Example: Checking a web app on a test server before going live.
3. Staging / Pre-Production Environment
    - Final testing environment that closely mimics the real production environment.
    - Ensures software behaves as expected when deployed.
    - Example: A website deployed on a staging server to check all functionalities.
4. Production Environment
    - Where the software is used by end users.
    - Must be stable, secure, and fully functional.
    - Example: A live banking app or e-commerce website.

Setting Up a Basic Environment in a Virtual Machine

Step 1: Install Virtual Machine

- Download VirtualBox or VMware.
- Create a new VM (choose OS like Ubuntu Linux or Windows).

Step 2: Set Up Development Environment

- Install a code editor/IDE: VS Code, Code:Blocks, or Sublime Text.
- Install necessary compilers/interpreters (C, Python, Java).
- Test with a simple program like “Hello World.”

Step 3: Set Up Testing Environment

- Use a separate folder or VM snapshot for testing.
- Run programs and check for errors or bugs.

Step 4: Simulate Production Environment

- Run programs in a separate folder/VM as if it were live.
- Verify everything works correctly.

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

Importance of a Development Environment in Software Production

A development environment is where software is created, coded, and initially tested. It provides the tools and setup needed for programmers to write and debug code efficiently.

Key Points:

1. Safe Space for Development: Developers can write and test code without affecting live systems.
2. Debugging and Error Detection: Helps identify bugs and fix them before deployment.
3. Consistency: Provides a standard setup (IDEs, compilers, libraries) for all developers on a project.
4. Efficiency: Tools like code editors, compilers, and version control make coding faster and organized.
5. Testing Before Production: Small-scale testing ensures that errors are caught early, reducing risks in production.

Source Code

LAB EXERCISE: Write and upload your first source code file to Github.

1. Write Source Code:
    - Open an IDE or text editor (VS Code, Code:Blocks, etc.).
    - Example in C:

```
#include <stdio.h>

int main() {
   printf("Hello, World!\n");
   return 0;
}
```

2. Save the File:
    - Save as hello.c or appropriate name.
3. Upload to GitHub:
    - Create a GitHub repository.
    - Use git init in your project folder.
    - Add files: git add hello.c
    - Commit changes: git commit -m "First source code"
    - Push to GitHub: git push origin main

Result: Your source code is now available on GitHub.

THEORY EXERCISE: Difference Between Source Code and Machine Code ?

Feature	Source Code	Machine Code
Definition	Human-readable instructions written in a programming language (C, Java, Python).	Computer-readable binary code (0s and 1s) executed by the CPU.
Readability	Readable by programmers.	Not readable by humans.
Purpose	To define what the program should do.	To perform the program's instructions on hardware.
Conversion	Must be **compiled or interpreted** to machine code before execution.	Directly executed by the computer.

- Source code = Instructions for humans to write programs.
- Machine code = Instructions the computer can execute.

Github and Introductions

GitHub: A platform to store code, track changes, and collaborate with others safely.

LAB EXERCISE: Create a Github repository and document how to commit and push code changes.

Create a GitHub repository and document how to commit and push code changes.

Steps:

Create a Repository on GitHub:

```
Go to GitHub → New Repository → Give a name → Create.
```

Set up Git Locally:

```
git init                                    # Initialize Git in your project folder
git remote add origin <repository-URL>      # Link local folder to GitHub
```

Add and Commit Code:

```
git add .               # Stage all files
git commit -m "Initial commit"              # Commit changes with a message
Push Code to GitHub:git push origin main    # Push changes to GitHub repository
```

Your code is now versioned and uploaded to GitHub.

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

- Version control keeps track of all changes made to code over time.
- Helps collaboration between multiple developers.
- Allows rollback to previous versions if something breaks.
- Makes it easier to track bugs and maintain software history.
- Essential for organized, safe, and professional software development.

Version control ensures safety, collaboration, and efficient management of code changes.

Student Account in Github

LAB EXERCISE: Create a student account on Github and collaborate on a small project with a classmate.

Objective: Create a student account and collaborate on a small project.

Steps:

1. Create Student Account:
   Go to GitHub → Sign Up → Use your student email.
2. Set Up Repository:
   Click New Repository → Give a project name → Create.
3. Collaborate with Classmate:
   Add your classmate as a collaborator in repository settings.
   Both can push, pull, and edit files in the project.
4. Commit and Push Changes:
   git add .
   git commit -m "First commit"
   git push origin main

A collaborative project where multiple students can contribute code safely.

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

1. Version Control: Track all changes and avoid losing work.
2. Collaboration: Work with classmates on group projects easily.
3. Portfolio: Showcase projects publicly to potential employers.
4. Learning Git: Gain practical skills for professional software development.
5. Backup: Code is safely stored online.

GitHub helps students learn, collaborate, and showcase their work efficiently.

Types of Software

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

Category	Software Examples
System Software	Windows OS, Linux, macOS
Application Software	MS Word, VS Code, Chrome, Spotify
Utility Software	Antivirus (Avast), WinRAR, Disk Cleanup

THEORY EXERCISE: Differences Between Open-Source and Proprietary Software

Feature	Open-Source Software	Proprietary Software
Source Code	Publicly available for anyone to view/edit	Not available; source code is hidden
Cost	Usually free	Usually paid
Customization	Can be modified and improved by users	Cannot be modified without permission
Examples	Linux, Firefox, VLC Media Player	Windows, MS Office, Adobe Photoshop

GIT and GITHUB Training

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

Objective: Practice cloning, branching, and merging repositories using Git.
Steps:

1. Clone a Repository
   git clone <repository-URL>
   Copies an existing repository to your local system.
2. Create a Branch
   git checkout -b feature-branch
   Makes a separate branch to work on a new feature.
3. Make Changes and Commit
   git add .
   git commit -m "Added new feature"
4. Merge Branch to Main
   git checkout main
   git merge feature-branch
5. Push Changes to GitHub
   git push origin main

You have successfully cloned, branched, edited, merged, and pushed a repository.

THEORY EXERCISE: How does GIT improve collaboration in a software development team?

1. Version Control: Tracks all changes made by team members.
2. Branching: Developers can work on features independently without affecting the main code.
3. Merging: Combines changes from multiple developers safely.
4. History & Rollback: Easy to see who made changes and revert if needed.
5. Conflict Management: Detects conflicts and allows safe resolution.

Git allows multiple developers to work together efficiently, keeps code organized, and reduces errors during collaboration.

Application Software

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

Introduction: Application software helps users do specific tasks and improves productivity.

Types & Benefits:

- Word Processor (Word, Docs): Create documents quickly.
- Spreadsheet (Excel, Sheets): Calculate and analyze data easily.
- Presentation (PowerPoint, Slides): Make slides for meetings.
- Database (Access, MySQL): Store and manage data efficiently.
- Communication (Teams, Zoom): Chat, call, and meet online

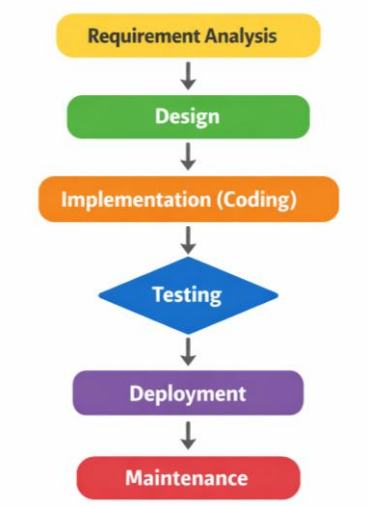
THEORY EXERCISE: What is the role of application software in businesses?

Role of Application Software in Businesses:

- Automates tasks: Helps with billing, payroll, and inventory.
- Improves productivity: Employees can work faster and more efficiently.
- Manages data: Stores, organizes, and analyzes information for better decisions.
- Enhances communication: Supports email, chats, and video meetings.
- Supports business activities: Helps in marketing, designing, and customer management.

Software Development Process

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



Software Development Process (SDLC):

- Follows stages: Requirement → Design → Coding → Testing → Deployment → Maintenance.
- Ensures software meets user needs, works correctly, and is easy to maintain.

Conclusion: SDLC makes software development organized, efficient, and reliable.

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

Main Stages of Software Development Process:

1. Requirement Analysis: Understand what the software should do.
2. Design: Plan how the software will work.
3. Implementation (Coding): Write the program.
4. Testing: Find and fix errors.
5. Deployment: Release the software to users.
6. Maintenance: Update and improve the software after release.

These stages help develop software systematically and efficiently.

Software Requirement

LAB EXERCISE:

Write a requirement specification for a simple library management system.

1. Introduction:

The Library Management System (LMS) is software to manage library operations like issuing books, tracking inventory, and maintaining member records.

2. Functional Requirements:

- Add, update, and delete books.
- Register and manage library members.
- Issue and return books.
- Track overdue books and fines.
- Search books by title, author, or category.

3. Non-Functional Requirements:

- User-friendly interface.
- Secure access for admins and members.
- Fast response time for searches.
- Data backup and recovery.

4. System Requirements:

- Runs on Windows or Linux.
- Database to store books and member details.
- Internet access optional for online features.

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

Importance of Requirement Analysis

- Requirement analysis identifies what the user really needs.
- Prevents errors and misunderstandings later in development.
- Helps plan the design and resources efficiently.
- Ensures the final software meets user expectations.

Software Analysis

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

- User login/registration
- Browse products & add to cart
- Payment and order tracking
- Reviews & ratings
- Admin manages products and orders

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

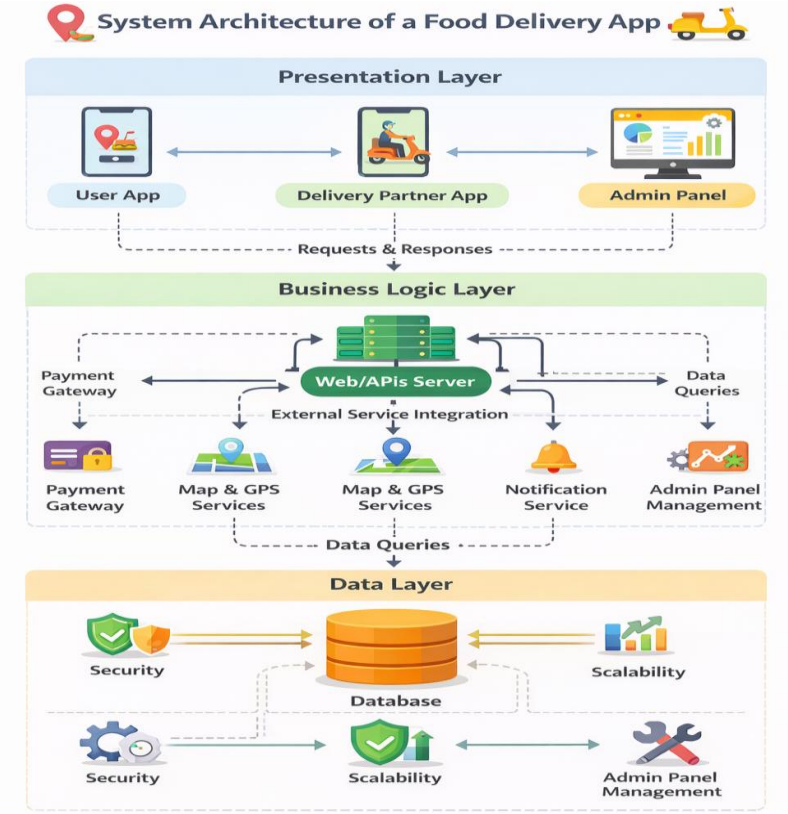
- Understands user needs and system requirements
- Identifies necessary functions
- Reduces errors and guides development

Ensures software meets user needs efficiently.

System Design

LAB EXERCISE:

Design a basic system architecture for a food delivery app.



System design defines how a software system is structured and how its components interact. It includes the system architecture, data storage, user interfaces, and design of individual modules, while ensuring security and performance. Good system design makes the software efficient, organized, user-friendly, and easy to maintain.

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

Key Elements of System Design:

- Architecture: Structure and organization of the system.
- Data Design: How data is stored, managed, and accessed.
- Interface Design: How users interact with the system.
- Component Design: Design of individual modules or features.
- Security & Performance: Ensuring safe, reliable, and efficient operation.

These elements ensure the software is well-organized, functional, and easy to maintain.

Software Testing

LAB EXERCISE: Develop test cases for a simple calculator program.

```
#include <stdio.h>
```

```c
int main() {
   int num1, num2;
   char op;

   printf("Enter first number: ");
   scanf("%d", &num1);

   printf("Enter operator (+, -, *, /): ");
   scanf(" %c", &op);

   printf("Enter second number: ");
   scanf("%d", &num2);

   if(op == '+') {
      printf("Result: %d\n", num1 + num2);
   } else if(op == '-') {
      printf("Result: %d\n", num1 - num2);
   } else if(op == '*') {
      printf("Result: %d\n", num1 * num2);
   } else if(op == '/') {
      if(num2 != 0)
         printf("Result: %d\n", num1 / num2);
      else
         printf("Error: Division by zero!\n");
   } else {
      printf("Invalid operator!\n");
   }

   return 0;
}
```

Simple Test Cases :

Input	Output	Description
5 + 3	8	Addition
10 - 4	6	Subtraction
6 * 7	42	Multiplication
20 / 5	4	Division
10 / 0	Error	Division by zero
5 + -3	2	Addition with negative

THEORY EXERCISE: Why is software testing important?

- Ensures the software works correctly and meets requirements.
- Detects errors and bugs before release.
- Improves software quality, reliability, and performance.
- Saves time and cost by preventing future issues.

Maintenance

LAB EXERCISE: Document a real-world case where a software application required critical maintenance.

Banking Mobile App

- Problem: Users reported that transactions were not updating in real-time, causing errors in account balances.
- Action: Developers released an urgent patch to fix database syncing issues and update security protocols.
- Result: App functionality was restored, errors were fixed, and users could safely perform transactions.

THEORY EXERCISE: What types of software maintenance are there?

Corrective Maintenance: Fixing bugs and errors found after release.
Adaptive Maintenance: Updating software to work with new hardware, OS, or technology.
Perfective Maintenance: Improving performance, adding features, or enhancing usability.
Preventive Maintenance: Making changes to prevent future problems or failures.

Development

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

Key Differences Between Web and Desktop Applications:

Feature	Web Application	Desktop Application
Access	Runs in a web browser, needs internet	Installed on a computer, can work offline
Platform	Platform-independent (Windows, Mac, Linux)	Usually platform-specific
Installation	No installation needed	Must be installed on each computer
Updates	Updated on the server, automatically available	User must install updates manually
Performance	Depends on internet speed and browser	Usually faster, uses local system resources
Examples	Gmail, Google Docs, Amazon	MS Word, Photoshop, VLC Player

Web Application

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

Advantages of Web Applications over Desktop Applications:

1. Accessible Anywhere: Can be used from any device with a browser and internet.
2. No Installation Needed: Users don't need to install or update software manually.
3. Automatic Updates: Updates are applied on the server and available to all users instantly.
4. Platform Independent: Works on Windows, Mac, Linux, or mobile devices.

5. Easier Collaboration: Multiple users can access and work on the same application simultaneously.

Designing

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

Role of UI/UX Design in Application Development:

- UI (User Interface) Design: Focuses on the look and layout of the application, making it visually appealing and easy to navigate.
- UX (User Experience) Design: Ensures the app is intuitive, efficient, and provides a smooth experience for users.
- Importance: Good UI/UX improves user satisfaction, increases engagement, reduces errors, and makes the application more successful.

Mobile Application

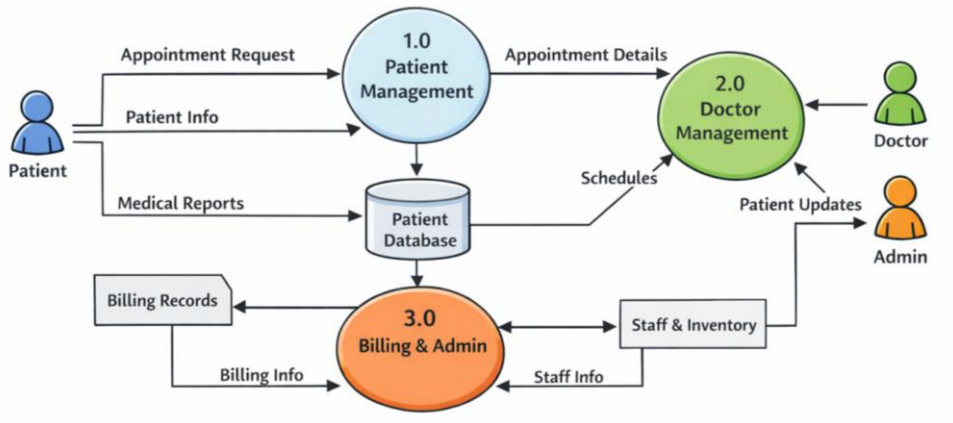
THEORY EXERCISE: What are the differences between native and hybrid mobile apps?

Feature	Native App	Hybrid App
Platform	Built for a specific OS (iOS or Android)	Works on multiple platforms using a single codebase
Performance	Fast and smooth	Slightly slower than native apps
Access to Device Features	Full access to camera, GPS, sensors	Limited access, may need plugins
Development	Separate code for each platform	Single code for all platforms
Examples	WhatsApp, Instagram	Instagram Lite, Uber (early versions)

DFD (Data Flow Diagram)

A diagrammatic representation that shows how data flows through a system, illustrating the processes, data stores, and interactions with external entities.

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



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

Significance of DFDs in System Analysis:

- DFDs visually show how data flows through a system.

- Help identify processes, data stores, and interactions between users and the system.
- Make it easier to understand, analyze, and design the system.
- Help detect missing or redundant processes before development.

Desktop Application

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

GUI : A GUI library is a toolkit that lets you make windows, buttons, and other visual elements in a program, so users can interact with it easily instead of using text only.

```
from tkinter import *

root = Tk()
root.title("Calculator")

e = Entry(root, width=16, font=('Arial', 20))
e.grid(row=0, column=0, columnspan=4)

def press(key):
    if key == "=":
        try:
            result = str(eval(e.get()))
            e.delete(0, END)
            e.insert(0, result)
        except:
            e.delete(0, END)
            e.insert(0, "Error")
    elif key == "C":
        e.delete(0, END)
    else:
        e.insert(END, key)
buttons = [
    ['7','8','9','/'],
    ['4','5','6','*'],
    ['1','2','3','-'],
    ['C','0','=','+']
]
for i in range(4):
    for j in range(4):
        Button(root, text=buttons[i][j], width=5, height=2, font=('Arial', 18),
               command=lambda key=buttons[i][j]: press(key)).grid(row=i+1, column=j)

root.mainloop()
```

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

Desktop Applications: Installed on a computer, run locally.

Web Applications: Run in a web browser, over the internet.

Pros of Desktop Applications

- Faster performance (runs locally).
- Works offline (no internet needed).
- Can use full hardware power.
- Can have rich, custom interfaces.

Cons of Desktop Applications

- Must be installed on each device.
- Updates need manual installation.
- Usually platform-specific (Windows, Mac, Linux).
- Limited remote access.

Pros of Web Applications

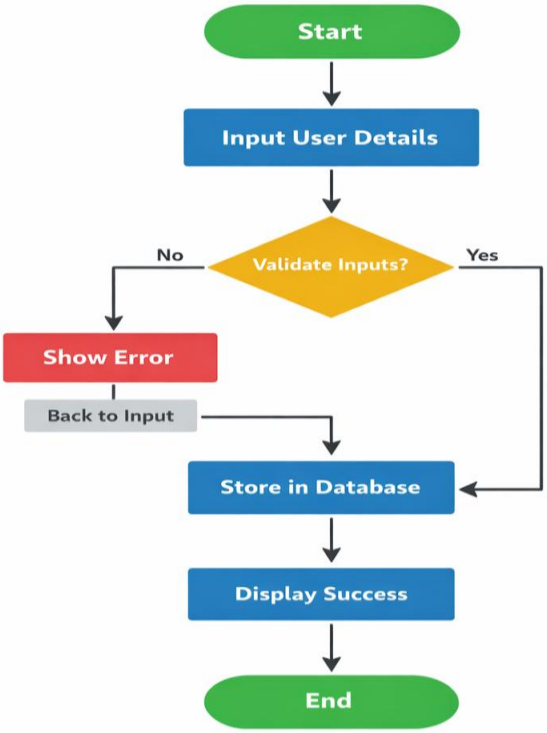
- Accessible from any device with a browser.
- No installation needed.
- Updates are automatic for all users.
- Can be used anywhere with internet.

Cons of Web Applications

- Needs internet connection.
- Slower than desktop apps.
- Limited access to hardware.
- Can have security risks online.

32. 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?

Visualize Logic: Shows program steps and decisions clearly.
Simplifies Planning: Helps programmers design before writing code.
Reduces Errors: Makes it easier to spot mistakes in logic.
Easy to Understand: Both programmers and non-programmers can follow it.
Improves Communication: Helps explain system design to team members.