Module 1 – Overview of IT Industry

What is a Program?

- A **program** is a set of instructions written in a programming language that tells a computer what to do.

LAB EXERCISE: Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

1. Hello World in C Language

```
#include <stdio.h>

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

2. Hello World in Python Language

```
print("Hello, World!")
```

Comparison of Structure and Syntax:

Feature	C Language	Python Language
Complexity	More complex	Very simple
Main function required	Yes (int main())	No
Library required	Yes (#include <stdio.h>)	No
Output statement	printf("Hello, World!");	print("Hello, World!")
Semicolon required	Yes	No
Code length	Longer	Shorter
Readability	Less readable for beginners	More readable

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

- A **program** is a set of step-by-step instructions written in a programming language that tells a computer how to perform a specific task.

How a Program Functions:

A program works in the following steps:

1. **Input**
   The program receives data from the user or another source.
   Example: Entering numbers in a calculator.
2. **Processing**
   The computer follows the instructions in the program to process the input data.
   Example: Adding two numbers.
3. **Output**
   The program displays the result to the user.
   Example: Showing the sum on the screen.
4. **Execution by Computer**
   The program is converted into machine language so the computer can understand and execute it.

What is Programming?

- **Programming** is the process of creating a set of instructions that tells a computer how to perform a specific task using a programming language.

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

The programming process consists of several important steps that help create a working and correct program.

1. Problem Definition

First, understand the problem clearly.
You must know:

- What needs to be done
- What input is required
- What output is expected

Example: Create a program to add two numbers.

2. Planning the Solution (Algorithm / Flowchart)

Plan how to solve the problem step-by-step using:

- Algorithm (step-by-step instructions)
- Flowchart (diagram representation)

Example Algorithm:

1. Start
2. Enter two numbers
3. Add the numbers
4. Display the result
5. Stop

3. Writing the Code

Convert the planned solution into a programming language such as C, Python, or Dart.

Example:

```
a = 5
b = 3
print(a + b)
```

4. Compiling or Interpreting

The code is translated into machine language so the computer can understand and execute it.

- Compiler → Used in C, C++
- Interpreter → Used in Python, Dart

5. Testing and Debugging

Run the program to check for errors and fix them.

- Testing → Checking if program works correctly
- Debugging → Finding and fixing errors

6. Execution

Run the final program to get the correct output.

7. Maintenance

Update or improve the program when needed.

Example: Adding new features or fixing future errors.

Types of Programming Languages

Programming languages are of two main types:

1. Low-Level Language

- Hard to understand
- Close to computer language
- Uses 0 and 1 or simple codes

Examples:

- Machine Language
- Assembly Language

2. High-Level Language

- Easy to understand
- Uses English words
- Used to make apps, websites, and software

Examples:

- C
- C++
- Java
- Python
- Dart

Example:

```
printf("Hello World");
```

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

Feature	High-Level Language	Low-Level Language
Ease of use	Easy to read and write	Difficult to read and write
Language style	Uses English-like words	Uses binary or simple codes
Understanding	Easy for humans	Easy for computers
Development speed	Faster to develop programs	Slower to develop programs
Hardware dependency	Not hardware dependent	Hardware dependent
Examples	C, C++, Java, Python, Dart	Machine Language, Assembly Language

World Wide Web & How Internet Works

- The **World Wide Web (WWW)** is a system that allows people to access websites and information using the Internet.
- It contains **websites, web pages, images, videos, and documents**
- You use a **web browser** (like Chrome or Safari) to open websites
- Each website has a unique address called a **URL** (example: www.google.com)
- The World Wide Web was invented by **Tim Berners-Lee** in **1989** while working at CERN.
- **Example:**
  When you open YouTube in your browser, you are using the World Wide Web.

How the Internet Works

- The **Internet** is a global network that connects millions of computers together.

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.

- In web communication, the **client** and **server** work together to send and receive information over the Internet.

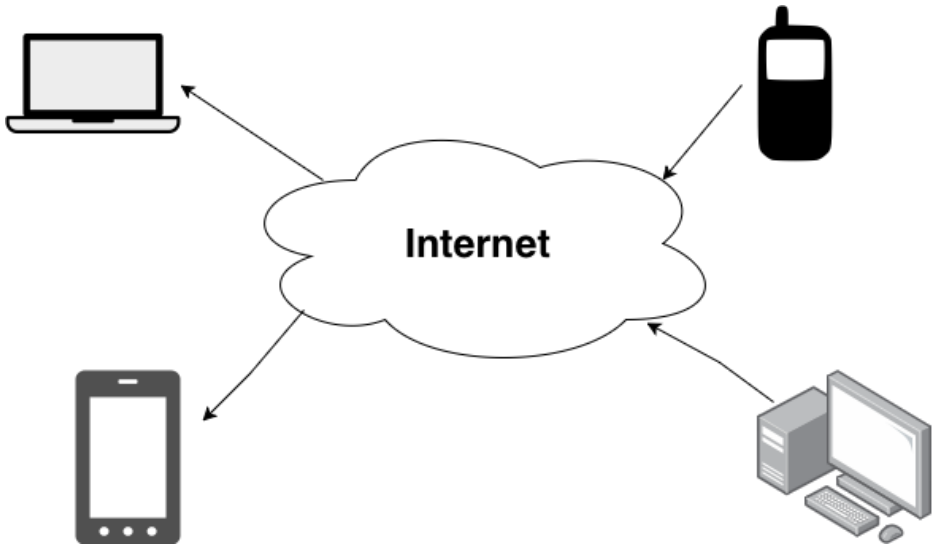
Role of the Client

- The **client** is the device or application that requests information from the server.

Role of the Server

- The **server** is a powerful computer that stores websites, data, and applications.

Network Layers on Client and Server



THEORY EXERCISE: Explain Client Server Communication

Client-server communication is the process in which a client requests information and a server responds by sending the requested data over the Internet.

- The **client** is the user's device (computer, phone, browser).
- The **server** is a powerful computer that stores data and provides services.

How Client-Server Communication Works (Step-by-Step):

Step 1: Client Sends Request

The client sends a request to the server for information.
Example: Opening a website.

Step 2: Server Receives Request

The server receives the request and understands what the client needs.

Step 3: Server Processes Request

The server processes the request by finding the required data.

Step 4: Server Sends Response

The server sends the requested data back to the client.

Step 5: Client Displays Result

The client receives the data and shows it to the user.

Types of Internet Connections

1. Dial-Up Connection

- Uses telephone line
- Very slow speed
- Old method

Example: Landline phone connection

2. Broadband Connection

- Fast and commonly used
- Uses telephone line, cable, or fiber

Examples:

- DSL
- Cable
- Fiber

3. Wi-Fi Connection

- Wireless Internet connection
- Uses router
- Common in homes, schools, and offices

Example: Home Wi-Fi

4. Mobile Data Connection

- Internet through mobile network
- Used on smartphones

Examples:

- 3G
- 4G
- 5G

5. Fiber Optic Connection

- Very high speed
- Uses fiber optic cables
- Fastest and most reliable

6. Satellite Connection

- Uses satellites
- Used in remote areas
- Slower than fiber

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

1. Broadband (DSL/Cable)

Description:
Uses telephone lines (DSL) or cable TV lines to provide Internet.

Pros:

- Faster than dial-up
- Always connected
- Widely available

- Affordable

Cons:

- Speed may vary
- Slower than fiber
- Can be affected by network traffic

2. Fiber Optic

Description:
Uses fiber optic cables to transmit data using light.

Pros:

- Very high speed
- Most reliable connection
- Best for streaming and gaming
- Low signal loss

Cons:

- Expensive
- Not available everywhere
- Installation cost is high

3. Satellite

Description:
Uses satellites in space to provide Internet.

Pros:

- Available in remote areas
- No need for cables
- Wide coverage
- **Cons:**
- Slower than fiber and broadband
- High latency (delay)
- Expensive
- Affected by weather

4. Mobile Data (3G / 4G / 5G)

Description:
Uses mobile network towers to provide Internet.

Pros:

- Wireless and portable
- Easy to use
- Available almost everywhere

Cons:

- Limited data plans
- Speed depends on signal strength
- Can be expensive

5. Wi-Fi

Description:
Wireless connection provided by a router connected to broadband or fiber.

Pros:

- Wireless connection
- Convenient and easy
- Supports multiple devices

Cons:

- Limited range
- Speed decreases with distance
- Can be affected by interference

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

Feature	Broadband	Fiber-Optic
Technology	Uses copper cables (telephone or cable lines)	Uses fiber optic cables (light signals)
Speed	Fast	Very fast (faster than broadband)
Reliability	Less reliable than fiber	More reliable
Signal quality	Signal can weaken over distance	Signal stays strong over long distance
Cost	Usually cheaper	More expensive
Availability	Widely available	Limited availability in some areas

Protocols

- A **protocol** is a set of rules that allows computers to communicate with each other over a network or the Internet.

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

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	Secure
Encryption	No encryption	Uses encryption (SSL/TLS)
Data Protection	Data can be intercepted	Data is protected and safe
URL	Starts with http://	Starts with https://
Use	Normal websites	Secure websites (banking, login, payments)
Port Number	80	443

Application Security

- **Application Security** is the process of protecting software applications from threats, attacks, and unauthorized access.

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

1. SQL Injection

Explanation:

SQL Injection happens when a hacker enters harmful SQL code into input fields (like login forms) to access or modify the database.

Example:
Entering special code instead of a username to bypass login.

Risks:

- Unauthorized access
- Data theft
- Database damage

Solution:

- Use **input validation**
- Use **prepared statements (parameterized queries)**
- Avoid directly using user input in database queries

2. Weak Password Authentication

Explanation:

Using weak passwords makes it easy for hackers to guess and access accounts.

Example:
Password like: 123456 or password

Risks:

- Unauthorized account access
- Data theft
- Account misuse

Solution:

- Use **strong passwords**
- Enable **Two-Factor Authentication (2FA)**
- Set password rules (minimum length, special characters)

3. Cross-Site Scripting (XSS)

Explanation:

XSS happens when hackers insert malicious scripts into websites that run in other users' browsers.

Example:
Malicious script entered in a comment section.

Risks:

- Stealing user data
- Session hijacking
- Redirecting users to fake websites

Solution:

- Validate user input
- Escape special characters
- Use secure coding practices

THEORY EXERCISE: What is the role of encryption in securing applications?

1. Protects Data

Encryption keeps important data like passwords and personal information safe.

2. Prevents Hacking

Hackers cannot read encrypted data.

3. Secures Internet Communication

Encryption protects data sent between user and server (HTTPS).

4. Keeps User Privacy Safe

Only authorized users can read the data.

Example:

Without encryption:
Password = 123456 (easy to read)

With encryption:
Password = A$7kP9#L (secret code)

Software Applications and Its Types

- A **software application** is a program used to do a specific task on a computer or mobile.

Types of Software Applications

1. General Purpose Software
Used for common work.

Examples:

- MS Word
- MS Excel
- Browser

2. Special Purpose Software

Used for specific work.

Examples:

- Banking software
- Billing software

3. Web Applications

Used on Internet in browser.

Examples:

- Gmail
- Google Docs

4. Mobile Applications

Used on mobile phones.

Examples:

- WhatsApp
- YouTube

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

Here are 5 commonly used applications and their classification:

Application	Type	Explanation
Windows	System Software	It controls the computer and manages hardware and software
Google Chrome	Application Software	Used to browse websites on the Internet
WhatsApp	Application Software	Used for sending and receiving messages
Microsoft Word	Application Software	Used for typing and editing documents
Android	System Software	It manages mobile device functions

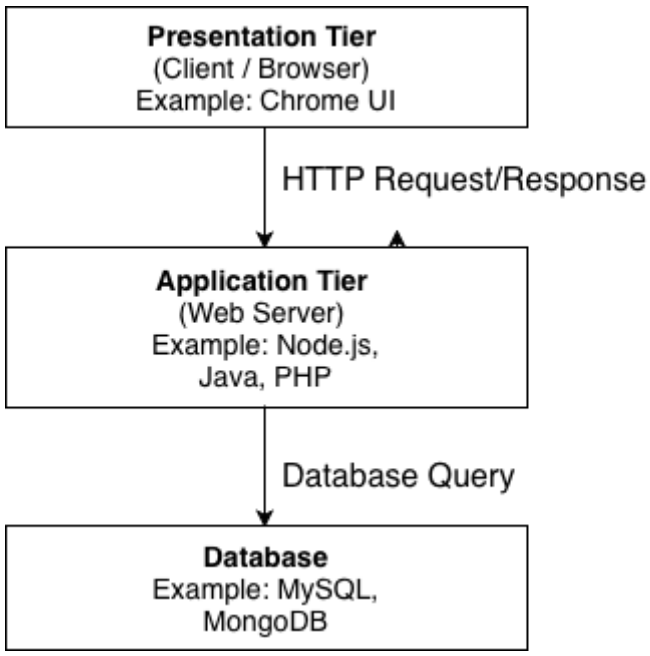
THEORY EXERCISE: What is the difference between system software and application software?

Feature	System Software	Application Software
Purpose	Controls and manages computer hardware	Helps users perform specific tasks
Role	Runs the computer system	Runs on top of system software
User Interaction	Works in background	User interacts directly
Necessity	Required for computer to work	Not required to run the computer
Examples	Windows, Linux, Android	Google Chrome, Microsoft Word, WhatsApp

Software Architecture

- **Software architecture** is the structure or design of a software system. It shows how different parts of the software are organized and how they work together.

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?

Modularity means dividing software into small parts called **modules**.

Why Modularity is Important:

1. Easy to Develop

Software can be built in small parts.

2. Easy to Fix Errors

If there is an error, only that module needs to be fixed.

3. Easy to Understand

Small parts are easier to understand.

4. Easy to Test

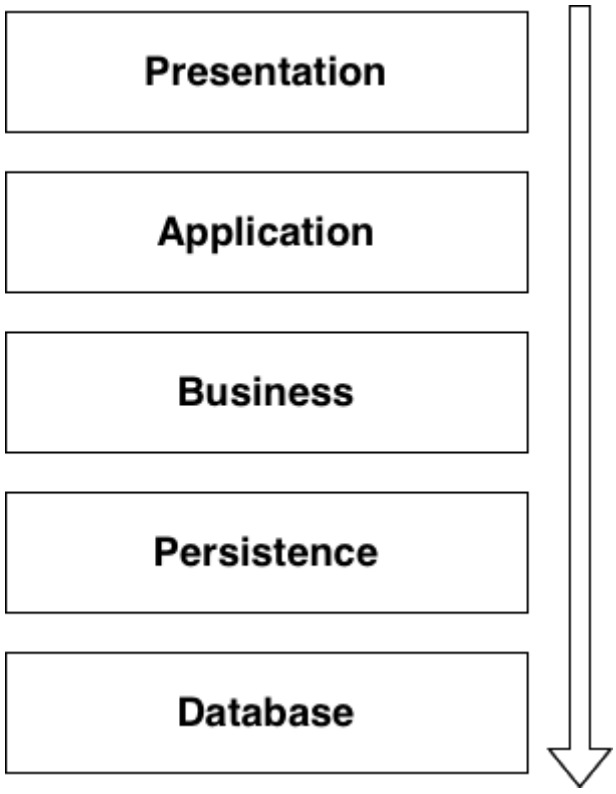
Each module can be tested separately.

Example:

In an app:

- Login module → login
- Payment module → payment
- Profile module → profile

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.

Example: Online Shopping App

An online shopping app allows users to see products and buy them. It has three layers.

1. Presentation Layer (User Interface)

Function:
This layer interacts with the user.

Examples:

- Login screen
- Product page
- Buy button

Role:

- Shows information to user
- Takes input from user

2. Business Logic Layer

Function:
This layer processes the user request.

Examples:

- Checks login details
- Calculates total price
- Processes orders

Role:

- Takes request from presentation layer
- Processes it

3. Data Access Layer (Database Layer)

Function:
This layer stores and retrieves data.

Examples:

- Stores user details
- Stores product details
- Stores order details

Role:

- Saves data
- Gets data from database

THEORY EXERCISE: Why are layers important in software architecture?

1. Maintainability and Isolation

When code is layered, changes in one area shouldn't break everything else. If you decide to switch your database from MySQL to MongoDB, you only need to update the **Data Layer**. Your **Presentation Layer** (the UI) doesn't even need to know a change happened.

2. Parallel Development

Layers allow teams to work on different parts of the application simultaneously without stepping on each other's toes.

- **Front-end devs** can mock the API and build the UI.
- **Back-end devs** can focus on complex business logic.
- **DBAs** can optimize the storage schema.

3. Reusability

A well-defined **Business Layer** can be reused by multiple "heads." For example, the same logic that calculates a shipping discount can be used by your web app, your mobile app, and your internal admin tool. You write the heavy lifting once and call it from different presentation layers.

4. Testability

It is much easier to test a small, isolated piece of logic than a "Big Ball of Mud." You can run unit tests on the **Service Layer** by "mocking" the database, allowing you to verify business rules without actually writing data to a disk.

Software Environments

- Software Environments are different setups used to develop, test, and run software.

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

1. Types of Software Environments

Development Environment

- Used for writing and testing code.
- Developers create programs here.

Testing Environment

- Used to test the software.
- Errors are checked and fixed.

Production Environment

- Used by real users.
- Final version of software runs here.

2. Basic Environment Setup in a Virtual Machine

Step 1: Install Virtual Machine Software

- Install VirtualBox or any virtual machine software.

Step 2: Create a Virtual Machine

- Create a new virtual machine.
- Install an operating system (Windows or Linux).

Step 3: Set Up Development Tools

- Install a text editor or programming software.
- Install required programs.

Step 4: Test the Environment

- Run a simple program to check if the environment works.

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

- A development environment is a setup where programmers write, test, and run software.

Importance:

- Helps programmers write code easily
- Makes testing software easier
- Helps find and fix errors
- Saves time in development
- Improves software quality

Source Code

- Source Code is the set of instructions written by a programmer using a programming language.

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

Step 1: Write Source Code

Create a simple program. Example:

```
print("Hello World")
```

Save the file as hello.py

Step 2: Create a Repository

- Login to GitHub
- Click New Repository
- Enter repository name
- Click Create Repository

Step 3: Upload Source Code

Use the following commands:

```
git add hello.py
git commit -m "First program"
git push origin main
```

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

Source Code:

- Written by programmers
- Uses programming languages (like C, Java, Python)
- Easy for humans to understand

Machine Code:

- Understood by the computer
- Written in binary (0 and 1)
- Not easy for humans to understand

Github and Introductions

GitHub is a website used to store and share projects online. It helps users manage and track changes in their code.

Features of GitHub:

- Stores projects online
- Allows sharing of code
- Helps track changes
- Supports teamwork

Uses:

- Save projects safely
- Work with others
- Learn programming

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

Step 1: Create a Repository

- Login to GitHub.
- Click New Repository.
- Enter repository name.
- Click Create Repository.

Step 2: Add Files

- Create or add a file in the project folder.

Step 3: Commit Changes
Use the following commands:

```
git add .
git commit -m "First commit"
```

Step 4: Push Changes
Upload files to GitHub:

```
git push origin main
```

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

- Version control is important because it helps manage changes in software projects.

Importance of Version Control:

- Keeps track of changes in code
- Allows multiple developers to work together
- Helps restore previous versions
- Reduces mistakes
- Keeps project organized

Student Account in Github

- A Student Account in GitHub allows students to store and share their projects online.

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

Step 1: Create a GitHub Account

- Go to the GitHub website.
- Click Sign Up.
- Enter username, email, and password.
- Create your account.

Step 2: Create a Repository

- Login to GitHub.
- Click New Repository.

- Enter repository name.
- Click Create Repository.

Step 3: Collaborate with Classmate

- Open the repository.
- Go to Settings → Collaborators.
- Add your classmate's username.

Step 4: Work on Project

- Both students can add files and make changes.
- Save changes using commit.

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

GitHub helps students manage and share their projects easily.

Benefits:

- Stores projects safely online
- Helps track changes in code
- Makes sharing projects easy
- Helps students work in teams
- Improves programming skills

Types of Software

There are three main types of software:

1**. System Software**
System software controls and manages the computer.
Example: Operating System

2. **Application Software**
Application software helps users perform specific tasks.
Example: Word processor, spreadsheet

3. **Utility Software**
Utility software helps maintain and protect the computer.
Example: Antivirus, Disk cleanup

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

1. System Software

- Operating System (Windows or Linux)

2. Application Software

- MS Word – Used for writing documents
- MS Excel – Used for calculations
- Web Browser – Used for browsing the internet

3. Utility Software

- Antivirus – Protects the computer from viruses
- Disk Cleanup – Removes unwanted files

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

Open-Source Software:

- Source code is available to everyone
- Users can modify the software
- Usually free to use

Proprietary Software:

- Source code is not available
- Users cannot modify the software
- Usually paid software

GIT and GITHUB Training

- Git is a tool used to manage and track changes in code.

- GitHub is a website used to store and share code online.

Basic Git Commands:

- Clone Repository
  Copies a project to your computer.
  git clone repository-link

- Add Files
  Adds files to Git.
  git add .
- Commit Changes
  Saves changes.
  git commit -m "message"
- Push Changes
  Uploads code to GitHub.
  git push
- Pull Changes
  Downloads updates from GitHub.
  git pull

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

Step 1: Clone a Repository

- Open Git.
- Use the command:

git clone repository-link

This copies the repository to your computer.

Step 2: Create a Branch

- Go to the project folder.

cd project-folder

- Create a new branch:

git branch new-branch

- Switch to the branch:

git checkout new-branch

Step 3: Make Changes

- Edit a file and save it.

- Add changes:

git add .

- Save changes:

git commit -m "Updated file"

Step 4: Merge Branch

- Go back to main branch:

git checkout main

- Merge the branch:

git merge new-branch

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

- Git helps developers work together on the same project easily.

- **Ways Git improves collaboration:**
  - Multiple developers can work on the same project
  - Changes can be tracked and saved
  - Developers can share code easily
  - Errors can be corrected by restoring previous versions
  - Work can be combined without conflicts

Application Software

- Application Software is software used to perform specific tasks for users.

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

Types of Application Software:

- **Word Processing Software**
  Used to write documents.
  Makes writing faster.
- **Spreadsheet Software**
  Used to store numbers and data.
  Makes calculations easy.
- **Presentation Software**
  Used to make presentations.
  Helps explain ideas clearly.
- **Database Software**
  Used to store information.
  Keeps records organized.

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

- Application software helps businesses perform daily tasks efficiently.

Roles of application software in businesses:

- Helps manage business data
- Makes work faster and easier
- Keeps records organized
- Helps in communication
- Improves productivity

Examples:

- Accounting software
- Billing software
- Inventory software

Software Development Process

- **Requirement Analysis** – Understanding what the software should do.
- **Design** – Planning how the software will work.
- **Implementation** – Writing the program code.
- **Testing** – Checking for errors.
- **Maintenance** – Updating and fixing the software.

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

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

The main stages of the software development process are:

- **Requirement Analysis**
  Understanding what the software should do.
- **System Design**
  Planning how the software will be built.
- **Implementation**
  Writing the program code.
- **Testing**
  Checking if the software works correctly.
- **Maintenance**
  Fixing errors and updating the software.

Software Requirement

- Software Requirement is a description of what a software system should do and what features it must have.

LAB EXERCISE: Write a requirement specification for a simple library management system.

Requirements:

- Users can search for books.
- Users can borrow books.
- Users can return books.
- Librarian can add books.
- Librarian can remove books.
- The system stores book and user information.

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

- The requirement analysis phase is important because it helps developers understand what the software should do before development begins.

Reasons it is critical:

- Defines user needs clearly
- Helps in proper system planning
- Reduces mistakes in development

- Saves time and cost
- Improves software quality

Software Analysis

- Software Analysis is the process of understanding and studying the requirements of a software system before designing and developing it.

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

- **User Registration and Login**
- Users can create an account
- Users can log in to the system
- **Product Search**
- Users can search for products
- Users can view product details
- **Shopping Cart**
- Users can add products to the cart
- Users can remove products from the cart
- **Order Placement**
- Users can place orders
- Users can enter delivery details
- **Payment**
- Users can make online payments
- System confirms payment
- **Order Tracking**
- Users can check order status

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

- Software analysis is the step where developers study and understand what the system needs to do before building it.

Role of Software Analysis:

- Identifies user requirements
- Defines system functions
- Helps in planning the system
- Reduces errors in development

- Saves time and cost

System Design

- System Design is the process of planning how a system will work by defining its structure, components, and data.

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

1. User Interface (Frontend)

- Mobile app or website
- Customers can view restaurants, order food, and make payments.

2. Application Server (Backend)

- Processes user requests
- Manages orders, users, and restaurants.

3. **Database**
Stores data such as:

- User information
- Restaurant details
- Menu items
- Orders

4. Payment System

- Handles online payments
- Confirms payment status.

5. Delivery System

- Assigns delivery persons
- Tracks delivery status.

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

Key Elements of System Design:

- **Requirements**
  Understanding what the system should do and what users need.

- **Architecture**
  The basic structure of the system (how parts are organized).

- **Components**
  Different parts or modules that make up the system.

- **Data Design**
  How data is stored and managed (like databases).

- **Interface Design**
  How users interact with the system (screens, inputs, outputs).

- **Security**
  Protecting the system and its data.

- **Testing**
  Checking that the system works correctly.

Software Testing

- Software Testing is the process of checking a software application to make sure it works correctly and meets the required standards.

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

Test Cases for Calculator Program

Test Case No.	Operation	Input	Expected Output	Notes
1	Addition	5 + 3	8	Normal addition
2	Addition	-2 + 7	5	Handling negative numbers
3	Subtraction	10 - 4	6	Normal subtraction
4	Subtraction	3 - 7	-4	Result is negative
5	Multiplication	6 × 3	18	Normal multiplication
6	Multiplication	-2 × 5	-10	Handling negative numbers
7	Division	12 ÷ 3	4	Normal division
8	Division	5 ÷ 0	Error / Cannot divide by zero	Handle division by zero
9	Addition	0 + 0	0	Edge case with zeros
10	Multiplication	0 × 8	0	Multiplication with zero

THEORY EXERCISE: Why is software testing important?

- **Finds Errors:** Detects bugs and mistakes before users face them.
- **Ensures Quality:** Makes sure the software works as intended and meets requirements.
- **Improves Reliability**: Reduces crashes, errors, and unexpected behaviour.
- **Enhances User Experience:** Ensures the app is smooth, easy, and enjoyable to use.
- **Saves Time and Cost:** Fixing problems early is cheaper than after release.

Maintenance

- Maintenance is the process of keeping a software application working properly after it has been released. It includes fixing problems, updating the software, and improving performance.

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

WhatsApp Maintenance

What happened:
WhatsApp stopped working for many users worldwide. Messages couldn't be sent or received.

Maintenance done:

- Corrective Maintenance: Fixed the server and network problems.
- Preventive Maintenance: Updated systems to avoid the same problem in the future.

Result:

- WhatsApp started working again.
- Users could send messages safely without problems.

Lesson:
Apps need maintenance to fix problems and keep working smoothly.

THEORY EXERCISE: What types of software maintenance are there?

Types of Software Maintenance

- **Corrective Maintenance**
  - Fixes errors or bugs found in the software after it is released.
  - Example: Fixing a login error in an app.
- **Adaptive Maintenance**
  - Updates the software to work with new environments, hardware, or operating systems.
  - Example: Making an app compatible with the latest version of Android.

- **Perfective Maintenance**
  - Improves or enhances the software's performance, features, or usability.
  - Example: Adding a new search feature or improving app speed.
- **Preventive Maintenance**
  - Prevents future problems by updating code, optimizing performance, or improving security.
  - Example: Refactoring code to avoid crashes later.

Development

- Development is the process of building and creating an application so it can work on devices or browsers. It includes:
- Planning: Deciding what the app will do and how it will work.
- Designing: Creating the app's look (UI) and user experience (UX).
- Coding/Programming: Writing the software that makes the app function.
- Testing: Checking for errors and making sure the app works correctly.
- Deployment: Releasing the app for users to download or access online.

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

Feature	**Web Application**	**Desktop Application**
Installation	No installation needed; runs in a browser	Must be installed on a specific device
Accessibility	Can be accessed from any device with internet	Works only on the device it's installed on
Updates	Updated automatically on the server	Users must manually update software
Platform	Cross-platform (Windows, macOS, Linux, mobile)	Usually platform-specific
Storage & Resources	Most processing and storage on server	Uses local device storage and resources
Collaboration	Easy for multiple users to work together online	Collaboration is limited unless synced manually

Web Application

- A web application is a software program that runs inside a web browser over the internet, instead of being installed on a computer or mobile device.

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

Advantages of Web Applications over Desktop Applications

- **Accessible Anywhere:** Can be used on any device with an internet browser, no installation needed.
- **Easy Updates:** Updates happen on the server, so users always have the latest version.
- **Cross-Platform:** Works on Windows, macOS, Linux, and mobile devices.
- **Lower Cost:** No need to install or maintain software on every device.
- **Collaboration:** Multiple users can access and work on the app at the same time.
- **Less Storage Needed:** Most processing happens on the server, saving device storage.

Designing

- Designing in app development is the process of planning how the app will look and work before building it. It includes:
- UI Design: Choosing colors, fonts, buttons, and layout so the app looks nice.
- UX Design: Making sure the app is easy and smooth to use.
- Flow Design: Deciding how screens connect and how users move through the app.

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

1. User Interface (UI) Design

- Focuses on the visual elements of the app: buttons, icons, colors, typography, layout, and overall aesthetics.
- Ensures the app is visually appealing and consistent.
- Helps users navigate easily through the app without confusion.

2. User Experience (UX) Design

- Focuses on the overall experience of the user while using the app.
- Ensures the app is intuitive, efficient, and enjoyable.
- Involves understanding user behaviour, needs, and feedback to optimize workflows.

3. Importance in Development

- Increases user engagement – users are more likely to use apps that are easy and pleasant to use.
- Reduces errors and frustration – a good UX prevents confusion and makes tasks straightforward.
- Improves retention and reputation – apps with excellent UI/UX get higher ratings and more downloads.
- Supports business goals – a well-designed interface can drive sales, subscriptions, or other conversions.

Mobile Application

- A mobile application (or mobile app) is a software program designed to run on smartphones, tablets, or other mobile devices. Unlike traditional desktop software, mobile apps are optimized for **touch interfaces**, smaller screens, and mobile operating systems such as **iOS** or **Android**.

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

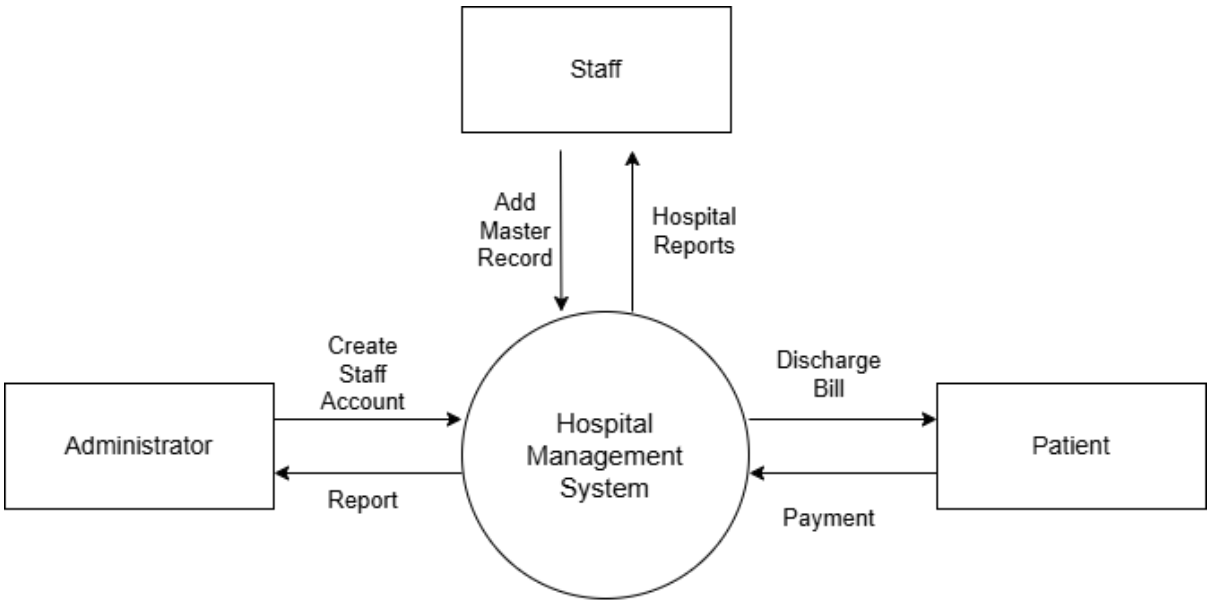
Feature	Native App	Hybrid App
Definition	Built for a specific platform (Android or iOS) using platform-specific languages	Built using web technologies (HTML, CSS, JavaScript) and run inside a wrapper for multiple platforms
Performance	Very fast, smooth, uses device hardware efficiently	Slower than native apps, may lag with heavy tasks
Platform Dependency	Works only on its platform (Android or iOS)	Works on multiple platforms with the same code
Development Cost	Higher (need separate code for each platform)	Lower (single code for all platforms)
Access to Device Features	Full access (camera, GPS, sensors)	Limited access; some features may require plugins

Updates	Installed via app stores	Can be updated more easily like a website

DFD (Data Flow Diagram)

- A Data Flow Diagram (DFD) is a visual representation of how data moves in a system.
- It shows processes, data stores, and data flow between users and the system.

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



Context Level DFD for Hospital Management System

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

- **DFD (Data Flow Diagram)** is a diagram that shows how data moves in a system.

🔹 **Importance of DFDs**

- **Shows Data Flow Clearly**
  It shows how information moves from one process to another.
- **Easy to Understand**
  Simple diagrams help users and developers understand the system.

- **Helps in System Planning**
  It helps analysts design and improve the system.
- **Identifies Problems**
  Helps find missing data or unnecessary processes.
- **Better Communication**
  Makes it easier to explain the system to others.

Desktop Application

- A desktop application is a software program that is installed and runs on a computer or laptop.

🔹 **Examples**

- Microsoft Word
- VLC Media Player
- Adobe Photoshop

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

Pros and Cons of Desktop Applications (Compared to Web Applications):

Advantages of Desktop Applications

- Works without internet
- Usually faster
- Can use computer hardware (printer, files, etc.)
- Good for heavy software (games, editing)

Disadvantages of Desktop Applications

- Must be installed
- Needs manual updates
- Takes up storage space
- May not work on all operating systems

Flow Chart

- A flowchart is a picture that shows the steps of a process from start to end.

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?

- **Easy to Understand**
  A flowchart shows how a program works step by step. It makes the logic clear.
- **Helps in Planning**
  Before writing code, you can draw a flowchart to plan what the program will do.
- **Finds Mistakes Early**
  It helps you see errors in logic before you start coding.
- **Better Communication**
  Other people can understand the program easily by looking at the flowchart.
- **Good for Documentation**
  It keeps a clear record of how the system works.