# Module 1 – Overview of IT Industry

## 1. What is a Program?

→ A **program** is a **set of instructions** written in a programming language like **C**, which tells the **computer what to do**.
It performs a specific task such as displaying text, doing calculations, or processing data.

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

→ **#include <stdio.h> // Header file for input/output functions**

```
int main() {        // Main function – starting point of the program
   printf("Hello, World!");  // Print output on the screen
   return 0;        // End of program
}
```

**Explanation:**

1. #include <stdio.h> → Includes the standard input/output library.
2. int main() → Defines the main function where program execution starts.
3. printf("Hello, World!"); → Displays the text **Hello, World!** on the screen.
4. return 0; → Ends the program successfully.

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

→ A **program** in C is a **collection of statements** that instructs the computer to perform a specific task.
When the program is compiled, it is converted into **machine code** that the computer's hardware can understand and execute.

The program generally follows this sequence:

1. **Input:** Accept data from the user.
2. **Processing:** Perform operations or logic.
3. **Output:** Display the result to the user using printf().

## 2.What is Programming?

→ **Programming** is the process of **creating a set of instructions** that tells a **computer how to perform a specific task**.
It involves using **programming languages** such as C, Python, or Java to write code that can solve problems or perform actions automatically.

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

- **Problem Definition** – Understand the task.
- **Algorithm Design** – Plan the steps to solve it.
- **Coding** – Write the program using a language.
- **Compilation/Execution** – Run and check for errors.
- **Testing/Debugging** – Fix mistakes and verify output.
- **Documentation** – Explain how the program works.
- **Maintenance** – Update or improve the program later.

## 3. Types of Programming Languages

- **High-Level Languages**
- **Low-Level Languages**

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

| Feature | High-Level Language | Low-Level Language |
|---|---|---|
| Meaning | Easy for humans to read and write | Close to machine language |
| Understanding | Uses words and symbols | Uses binary or assembly code |
| Portability | Can run on different systems | Machine-dependent |
| Execution Speed | Slower (needs translation) | Faster (directly executed) |
| Examples | C, Python, Java | Assembly, Machine Code |

## 4. World Wide Web & How Internet Works

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

→ Diagram: How Data is Transmitted from Client to Server

[Client Browser]

↓

[Internet Service Provider (ISP)]

↓

[DNS Server] → Converts website name to IP address

↓

[Web Server]

↓

[Response sent back to Client]

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

 Client**:**
The client (like a web browser) sends a request to access data or a webpage.

 Server**:**
The server receives the request, processes it, and sends the required webpage or data back to the client.

## 5. Network Layers on Client and Server

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

| Layer | Function | Examples |
|---|---|---|
| **1.** Application Layer | Provides network services to users | HTTP, FTP, DNS |
| **2.** Transport Layer | Handles end-to-end communication | TCP, UDP |
| **3.** Internet Layer | Routes data across networks | IP, ICMP |
| **4.** Network Access Layer | Manages hardware transmission | Ethernet, Wi-Fi |

## 6. Client and Servers

**THEORY EXERCISE:** Explain Client Server Communication

### Client-Server Communication

- **Client:** The device or program that requests services or resources from another computer.
- **Server:** The device or program that provides services or resources to clients.

**How it works:**

1. **Client sends request** → e.g., a web browser asks for a webpage.
2. **Server receives request** → processes it, e.g., retrieves the webpage data.
3. **Server sends response** → back to the client, e.g., the webpage content.
4. **Client receives response** → displays or uses the data.

**Example:**

- Using a web browser to access a website:
  - Browser = **Client**
  - Web server = **Server**
  - Communication uses **HTTP protocol** over **TCP/IP**.

**Key points:**

- Communication is usually **request-response**.
- Servers can handle **multiple clients** at the same time.
- It works using **network protocols** like TCP/IP.

## 7. Types of Internet Connections

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

### LAB EXERCISE: Types of Internet Connections

1. **DSL** – Internet via telephone lines.
   - **Pros:** Cheap, widely available.
   - **Cons:** Slower, speed decreases with distance.
2. **Cable** – Internet via TV cable lines.
   - **Pros:** Faster than DSL, reliable.
   - **Cons:** Shared bandwidth can slow it down.
3. **Fiber-Optic** – Internet via light signals in fiber cables.
   - **Pros:** Very fast, low latency, reliable.
   - **Cons:** Expensive, limited availability.
4. **Satellite** – Internet via satellites.
   - **Pros:** Works anywhere, no cables needed.
   - **Cons:** High latency, affected by weather.
5. **Mobile 4G/5G** – Internet via mobile networks.
   - **Pros:** Portable, easy setup.
   - **Cons:** Limited coverage, may be unstable.

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

- **Broadband:**

  - Uses copper/cable lines.
  - Moderate speed, higher latency, widely available.

- **Fiber-Optic:**

  - Uses fiber cables (light signals).
  - Very high speed, very low latency, highly reliable, limited availability.

## 8. Protocols

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

You can **simulate using command-line tools** like curl or wget:

**HTTP Request Example:**

curl http://example.com

FTP Request Example:

curl ftp://ftp.example.com --user username: password

- These commands **request data from a server** using HTTP or FTP.
- The server responds with **webpage content (HTTP)** or **files (FTP)**.

**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 (encrypted with SSL/TLS) |
| **Port** | 80 | 443 |
| **Data Safety** | Data can be intercepted | Data is encrypted, safe from attackers |
| **Usage** | Normal websites | Online banking, login pages, sensitive data |

## 9. Application Security

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

| Vulnerability | Explanation | Possible Solution |
|---|---|---|
| SQL Injection | Attacker injects malicious SQL code into input fields to access or modify database | Use prepared statements, parameterized queries, input validation |
| Cross-Site Scripting (XSS) | Attacker injects malicious scripts into web pages viewed by other users | Validate and sanitize user input, use Content Security Policy (CSP) |
| Broken Authentication | Weak password policies or session management allows attackers to take over accounts | Implement strong passwords, multi-factor authentication, secure session handling |

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

- **Encryption** converts data into an unreadable format using algorithms and keys.
- **Role in application security:**

1. Protects sensitive data (passwords, credit card info) from unauthorized access.
2. Ensures **data integrity**, preventing tampering.
3. Secures **communication** between client and server (e.g., HTTPS).

## 10. Software Applications and Its Types

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

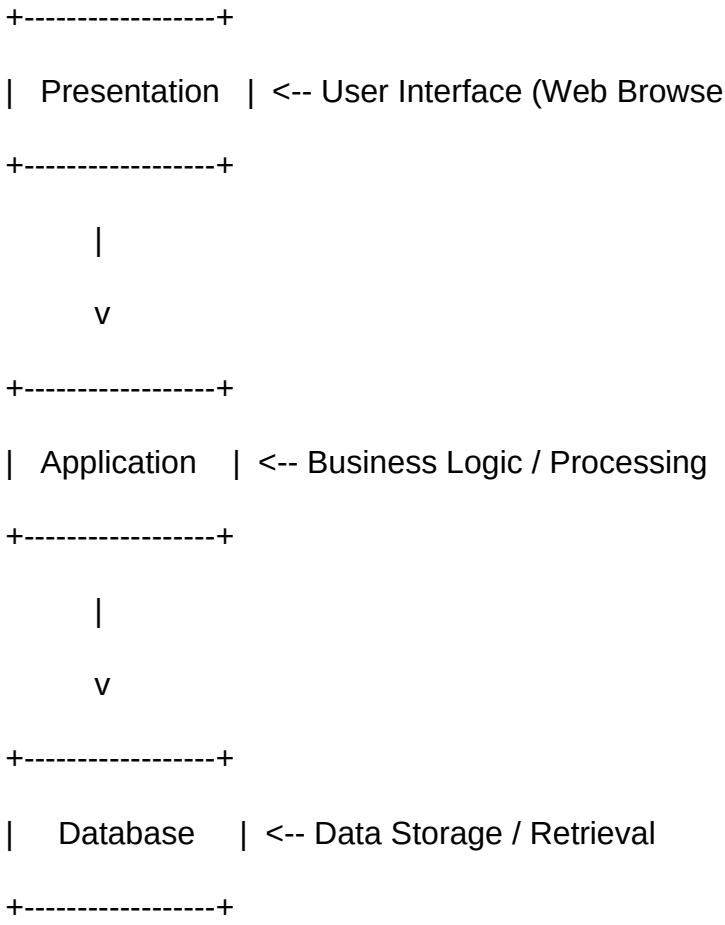
| Application | Type of Software |
|---|---|
| **Windows OS** | System Software |
| **Microsoft Word** | Application Software |
| **Google Chrome** | Application Software |
| **Antivirus (e.g., Avast)** | System Software |
| **VLC Media Player** | Application Software |

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

| Feature | System Software | Application Software |
|---|---|---|
| **Purpose** | Manages hardware and system resources | Performs specific tasks for users |
| **Examples** | Operating system, antivirus, drivers | Word processors, browsers, media players |
| **Requirement** | Necessary for running the computer | Optional, used according to user needs |
| **Interaction** | Runs in the background, interacts with hardware | Runs on top of system software |

## 11. Software Architecture

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




- **Presentation Tier:** Interface for users to interact.
- **Application Tier:** Processes data and applies business rules.
- **Database Tier:** Stores and retrieves data.

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

 Modularity means dividing software into independent, manageable modules.

 Significance**:**

1. **Easier maintenance:** Fix or update one module without affecting others.
2. **Reusability:** Modules can be reused in other projects.
3. **Simplifies development:** Teams can work on separate modules simultaneously.
4. **Improves readability:** Clear structure makes code easier to understand.

## 12. 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 Software System:** Online Shopping Website

| Layer | Functionality Example |
|---|---|
| Presentation Layer | Displays products, shopping cart, checkout forms to the user; handles user input. |
| Business Logic Layer | Calculates total price, applies discounts, checks inventory, processes orders. |
| Data Access Layer | Connects to the database to retrieve product info, store orders, update stock levels. |

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

 **Separation of Concerns:** Each layer handles its own responsibility.

 **Maintainability:** Changes in one layer (e.g., database) do not affect others (e.g., UI).

 **Reusability:** Layers can be reused in different projects or systems.

 **Scalability:** Easy to expand or replace a layer without redesigning the whole system.

 **Improved Testing:** Layers can be tested independently.

## 13. Software Environments

**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**
   - Used by developers to write and test code.
   - Example tools: IDEs (VS Code, Eclipse), compilers, local databases.
2. **Testing Environment**
   - Used to test applications before release.
   - Detects bugs and ensures quality.
   - Example: QA server with test data.
3. **Production Environment**
   - Live environment where the software is used by end-users.
   - Must be stable and secure.
   - Example: Live website or application server.

**Setting up a basic environment in a virtual machine:**

- Install OS (e.g., Ubuntu).
- Install necessary software: IDE, database, web server.
- Configure network, storage, and user permissions.
- Test by running a sample application.

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

- Provides a **safe space** for developers to write and test code without affecting live systems.
- Enables **debugging and experimentation**.
- Supports **team collaboration** using version control tools.
- Helps in **detecting errors early**, saving time and cost.
- Ensures **smooth transition** to testing and production environments.

## 14. Source Code

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

**Write source code** in a programming language (e.g., hello.c for C):

- **Save the file** on your computer.
- **Create a GitHub repository.**

 **Upload the file** using either:

- **GitHub website:** Click “Add file → Upload files”.
- **Git CLI:**

git init

git add hello.c

git commit -m "First source code"

git branch -M main

git remote add origin <repository-URL>

git push -u origin main

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

| Feature | Source Code | Machine Code |
|---|---|---|
| **Definition** | Human-readable instructions written in programming languages (C, Python, Java) | Binary instructions (0s and 1s) executed by the computer |
| **Readability** | Readable and understandable by humans | Not readable by humans |
| **Purpose** | To create programs | To run programs on hardware |
| **Conversion** | Needs **compiler or interpreter** to convert to machine code | Directly executed by the CPU |

## 15. GitHub and Introductions

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

**Steps to create a GitHub repository and push code changes:**

1. **Create a repository on GitHub**
   - Go to GitHub → New Repository → Give name → Create.
2. **Clone the repository locally**

**git clone <repository-URL>**

**cd <repository-folder>**
3. Add or modify files

echo "Hello World" > file.txt

git add file.txt

git commit -m "Add first file"

4. Push changes to GitHub

git push origin main

**Workflow:**

1. Edit files → 2. Stage changes (git add) → 3. Commit (git commit) → 4. Push (git push)

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

- Tracks **all changes** made to code over time.
- Allows **multiple developers** to work on the same project simultaneously.
- Enables **rollback** to previous versions if errors occur.
- Keeps the project **organized and safe**.
- Supports **collaboration and teamwork** efficiently.

## 16. Student Account in GitHub

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

- **Sign up for a GitHub student account**
  - Go to GitHub Education → Apply for student benefits → Verify your student email.
- **Create or join a repository**
  - Create a new repository for your project or collaborate on a classmate's repo.
- **Collaborate on a small project**
  - Add files, make commits, and push changes.
  - Use branches to work separately and then merge changes.
- **Track progress**
  - Use issues, pull requests, and commit history to coordinate with your classmate.

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

- **Free student benefits:** Access to private repositories, developer tools, and GitHub Student Pack.
- **Learn version control:** Gain practical skills in Git and collaboration.

- **Collaboration:** Work with classmates on projects efficiently.
- **Portfolio building:** Showcase projects to teachers or future employers.
- **Project tracking:** Keep track of all changes and progress in your code.

## 17. 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.

| Software Used | Type of Software |
|---|---|
| **Windows OS** | System Software |
| **Microsoft Word** | Application Software |
| **Google Chrome** | Application Software |
| **VLC Media Player** | Application Software |
| **Antivirus (Avast)** | Utility Software |
| **WinRAR / 7-Zip** | Utility Software |

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

| Feature | Open-Source Software | Proprietary Software |
|---|---|---|
| **Source Code** | Publicly available, can be modified | Closed source, cannot be modified |
| **Cost** | Usually free | Paid license required |
| **Customization** | Highly customizable | Limited customization |
| **Support** | Community support | Official vendor support |
| **Examples** | Linux, Firefox, LibreOffice | Windows OS, Microsoft Office |

## 18. GIT and GITHUB Training

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

**Steps to practice basic GIT commands:**

1. **Clone a repository**

**git clone <repository-URL>**

2. Create a new branch

**git checkout -b new-feature**

3. Make changes and commit
   **git add .**
   **git commit -m "Add new feature"**
4. Merge branch back to main
   **git checkout main**
   **git merge new-feature**
5. Push changes to remote repository
   **git push origin main**

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

- Tracks **all changes** made by team members.
- Enables **multiple developers** to work on separate branches simultaneously.
- Allows **merging changes** without losing work.
- Supports **rollback** to previous versions if errors occur.
- Keeps project **organized, transparent, and efficient** for teamwork.

## 19. Application Software

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

- **Word Processing:** Microsoft Word – creates and edits documents.
- **Spreadsheet:** Excel – organizes data, calculations.
- **Presentation:** PowerPoint – visual presentations.
- **Database:** MySQL – stores and manages data.
- **Communication:** Zoom/Slack – team collaboration.

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

- Performs specific business tasks efficiently.
- Automates work, reduces errors, saves time.
- Supports decision-making, communication, and collaboration.

## 20. Software Development Process

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

**Requirement Analysis → Design → Implementation (Coding) → Testing → Deployment → Maintenance**

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

- **Requirement Analysis** – Understand what the software should do.
- **Design** – Plan the structure and interface of the software.
- **Implementation (Coding)** – Write the program using a programming language.
- **Testing** – Verify the software works correctly.
- **Deployment** – Deliver the software to users.
- **Maintenance** – Update, fix bugs, and improve the software.

## 21. Software Requirement

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

**Functional Requirements:**

- Add, update, and delete books.
- Register and manage library members.
- Issue and return books.
- Search for books by title, author, or category.
- Generate reports on issued/returned books.

**Non-Functional Requirements:**

- User-friendly interface.
- Data security for member information.
- Fast response time for searches.
- Support multiple users simultaneously.

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

- Ensures **software meets user needs**.
- Helps **identify problems early** before coding.
- Reduces **costs and errors** by clarifying requirements.

 Provides a **clear blueprint** for design and development.

## 22. Software Analysis

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

**Key Functions:**

- **User Registration/Login:** Users can create accounts and log in securely.
- **Product Browsing/Search:** View and search products by category, price, or brand.
- **Shopping Cart:** Add, remove, or update products in the cart.
- **Checkout & Payment:** Process payments and generate invoices.
- **Order Tracking:** Track the status of orders and delivery.
- **Admin Functions:** Add/update/delete products, manage users, view reports.

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

 **Identifies user needs:** Determines what the system should do.

 **Defines system requirements:** Functional and non-functional specifications.

 **Reduces errors:** Detects potential issues before coding begins.

 **Guides design and development:** Provides a clear roadmap for implementation.

## 23. System Design

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

 **Architecture Design:** Structure of system layers and components.

 **Data Design:** Database schemas and data storage planning.

 **Interface Design:** How users and other systems interact with the software.

 **Component Design:** Individual modules, functions, and their responsibilities.

 **Security Design:** Ensuring data safety, access control, and privacy.

 **Scalability & Performance:** Planning for growth and fast response times.

## 24. Software Testing

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

- **Ensures correctness:** Verifies that the software works as intended.
- **Detects errors early:** Finds bugs before deployment.
- **Improves quality:** Enhances reliability, performance, and user experience.
- **Reduces cost:** Fixing issues during testing is cheaper than after release.
- **Builds confidence:** Users and developers trust the software.

## 25. Maintenance

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

**Case Example:** Facebook Platform Updates

- **Issue:** Users reported bugs in the news feed and privacy settings.
- **Action:** Developers released patches to fix bugs, updated the interface, and improved security features.
- **Outcome:** The platform became more stable, secure, and user-friendly.

**Explanation:**

- This is an example of **critical maintenance** to fix urgent problems affecting users.

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

- **Corrective Maintenance:** Fixing bugs or errors in the software after release.
- **Adaptive Maintenance:** Updating software to work with new hardware, OS, or platforms.
- **Perfective Maintenance:** Improving performance, usability, or adding new features.
- **Preventive Maintenance:** Modifying software to prevent future problems or failures.

## 26. Development

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

| Feature | Web Application | Desktop Application |
|---|---|---|
| **Access** | Runs via web browser, online | Installed on a specific computer |
| **Installation** | No installation needed | Requires installation |
| **Platform Dependency** | Platform-independent (Windows, Mac, Linux) | Usually platform-specific |

| | | |
|---|---|---|
| **Updates** | Updated on server automatically | User must manually update |
| **Connectivity** | Requires internet connection | Can work offline |
| **Performance** | Depends on browser & internet | Usually faster, full hardware access |
| **Examples** | Gmail, Google Docs, Facebook | Microsoft Word, Photoshop |

## 27. Web Application

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

Advantages of Web Applications over Desktop Applications

1. **Accessibility:** Can be used from any device with an internet connection and browser.
2. **No Installation Needed:** Users don't need to install or update software manually.
3. **Automatic Updates:** Updates are applied on the server, instantly available to all users.
4. **Platform Independence:** Works across different operating systems (Windows, Mac, Linux).
5. **Lower Maintenance Costs:** Centralized management reduces IT support and maintenance efforts.
6. **Collaboration:** Easier for multiple users to access and work simultaneously.

## 28. Designing

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

Role of UI/UX Design in Application Development

1. **Enhances Usability:** Makes the application easy to navigate and understand.
2. **Improves User Satisfaction:** A visually appealing and intuitive interface keeps users engaged.
3. **Increases Productivity:** Well-designed workflows help users complete tasks efficiently.
4. **Reduces Errors:** Clear design and feedback minimize mistakes by users.
5. **Supports Branding:** Consistent UI/UX reinforces the application's identity and trustworthiness.

## 29. Mobile Application

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

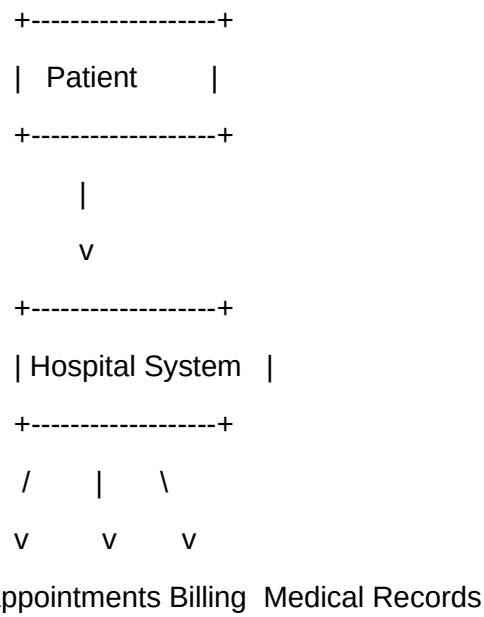
| Feature | Native App | Hybrid App |
|---|---|---|
| **Development** | Built for a specific platform (iOS or Android) using platform-specific languages | Built using web technologies (HTML, CSS, JavaScript) and wrapped in a native container |
| **Performance** | High performance, faster execution | Slightly slower, depends on web view performance |

| | | |
|---|---|---|
| **Access to Device Features** | Full access to device hardware and APIs | Limited access; some features may require plugins |
| **User Experience** | Smooth, platform-specific UI/UX | May feel less native, consistent across platforms |
| **Cost & Time** | Higher cost, separate apps for each platform | Lower cost, single codebase for multiple platforms |
| **Examples** | WhatsApp (iOS/Android) | Instagram (originally hybrid features), Ionic/Cordova apps |

## 30. DFD (Data Flow Diagram)

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

Level 0 (Context Diagram):




**Level 1 (Simplified):**

- **Patient:** Requests appointments, provides info.
- **Reception:** Manages patient registration and scheduling.
- **Doctors:** Access patient medical records.
- **Billing:** Generates invoices, processes payments.
- **Medical Records:** Stores patient history and treatment details.

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

- **Visual Representation:** Shows how data flows through the system.
- **Clarifies Processes:** Helps understand inputs, outputs, and data stores.

- **Identifies Inefficiencies:** Reveals redundant or missing processes.
- **Communication Tool:** Easy for developers and stakeholders to understand system flow.
- **Foundation for Design:** Guides database and system design.

## 31. Desktop Application

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

| Feature | Desktop Application | Web Application |
|---|---|---|
| **Performance** | Usually faster, full hardware access | Depends on browser and internet |
| **Offline Access** | Works offline | Requires internet connection |
| **Installation** | Needs to be installed on each device | No installation needed |
| **Updates** | Manual updates | Automatic updates via server |
| **Platform Dependency** | Usually platform-specific | Platform-independent (cross-browser) |

## 32. Flow Chart

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

- **Visual Representation:** Shows steps and decision points clearly.
- **Simplifies Logic:** Helps understand complex processes before coding.
- **Reduces Errors:** Identifies missing steps or logic flaws early.
- **Communication Tool:** Easy for developers and stakeholders to understand system flow.
- **Guides Development:** Provides a roadmap for coding and testing.