Module 1 – Overview of IT Industry – Theory Exercises

1. What is a Program?

A program is a collection of instructions written in a programming language that tells a computer how to perform a specific task. These instructions define the logic and steps needed to process inputs and produce outputs.
For example, a calculator program takes numbers from the user, applies mathematical operations (addition, subtraction, etc.), and displays the result.
Without programs, a computer is just a piece of hardware — programs bring it to life and make it useful.

2. What is Programming?

Programming is the process of designing and building a set of instructions that a computer can execute. It involves problem-solving, creating algorithms, and writing code using programming languages like Python, Java, or C.
The purpose of programming is to translate human ideas into a form that a machine can understand and act upon. For example, creating a mobile app or a website both require programming.

3. Key Steps in the Programming Process

The programming process is generally divided into the following stages:

1. Problem Definition – Clearly understanding the problem to be solved.
2. Planning & Design – Choosing the right approach, writing pseudocode, or creating flowcharts.
3. Coding – Writing the actual program using a programming language.
4. Testing & Debugging – Running the program to find and fix errors.
5. Deployment – Making the program available for users.
6. Maintenance – Updating the program to fix issues or add new features over time.
   Following these steps ensures that the final program is reliable and meets user requirements.

4. Differences Between High-Level and Low-Level Languages

Point	High-Level Languages	Low-Level Languages
Definition	Closer to human language, easy to understand and write.	Closer to machine language, harder to read and write.
Readability	Easy for humans to read and debug.	Difficult for humans to understand.
Execution Speed	Slower due to translation process.	Very fast and efficient.
Portability	Can run on multiple platforms with minimal changes.	Platform-dependent.
Examples	Python, Java, C++.	Assembly, Machine code.

5. Roles of the Client and Server in Web Communication

- Client – The device or application (like a browser) that requests services or data from the server. It displays information to the user and sends user actions (clicks, forms) to the server.
- Server – A powerful computer or program that processes client requests, retrieves or processes data, and sends a response back.
  Example: When you visit Google, your browser (client) requests a search page, and Google's server responds with the search results.

6. Function of the TCP/IP Model and Its Layers

The TCP/IP model describes how data moves across a network. Layers:

1. Application Layer – Interfaces for communication (e.g., HTTP for websites, FTP for file transfer).
2. Transport Layer – Ensures reliable delivery of data (TCP) or fast, connectionless transfer (UDP).
3. Internet Layer – Routes packets across networks using IP addresses.

4. Network Access Layer – Handles physical network connections (Ethernet, Wi-Fi). This model ensures that devices on different networks can communicate effectively.

7. Client-Server Communication

In client-server communication, the client initiates a request, and the server processes it and sends back a response.
Example: When you open YouTube, your device (client) requests video data, and YouTube's server sends it back for playback. This interaction happens using defined network protocols like HTTP.

8. Broadband vs Fiber-Optic Internet

Point	**Broadband Internet**	**Fiber-Optic Internet**
1. **Technology**	Uses copper cables (DSL or coaxial).	Uses thin strands of glass or plastic to transmit data as light.
2. **Speed**	Slower compared to fiber, usually up to 100 Mbps.	Extremely fast, can reach 1 Gbps or more.
3. **Reliability**	More prone to interference and signal loss over distance.	Highly reliable with minimal signal loss.
4. **Cost**	Generally cheaper and widely available.	More expensive and limited to certain areas.

9. HTTP vs HTTPS

• Point	• HTTP	• HTTPS
• Security	• Data is sent without encryption.	• Data is encrypted using SSL/TLS.
• Privacy	• Vulnerable to eavesdropping.	• Protects against interception and tampering.

• Trust	• No padlock icon in browser address bar.	• Shows padlock icon for secure connection.

10. Role of Encryption in Securing Applications

Encryption transforms readable data into an unreadable format using algorithms and keys. Its main roles are:

- Protecting sensitive information like passwords and credit card details.
- Ensuring that even if data is intercepted, it cannot be understood without the decryption key.
- Maintaining trust between users and applications.

11. System Software vs Application Software

Point	System Software	Application Software
Purpose	Manages hardware and provides platform for applications.	Helps users perform specific tasks.
Dependency	Can run independently.	Requires system software to function.
Examples	Operating systems, device drivers.	MS Word, browsers, games.

12. Significance of Modularity in Software Architecture

Modularity means dividing software into smaller, manageable components (modules). Benefits include:

- Easier debugging and maintenance.
- Reusability of modules in other projects.
- Parallel development by multiple teams.

- Improved readability and organization.

13. Importance of Layers in Software Architecture

Layers organize a system into different functional parts, such as:

- Presentation Layer – User interface.
- Business Logic Layer – Application logic and rules.
- Data Access Layer – Communication with the database.
  This separation increases maintainability, scalability, and flexibility.

14. Importance of a Development Environment

A development environment is a setup that includes tools like code editors, compilers, libraries, and debugging utilities.
It allows developers to build and test applications in a controlled space before deploying to production, reducing errors and downtime.

15. Source Code vs Machine Code

Point	Source Code	Machine Code
Format	Written in human-readable language.	Written in binary (0s and 1s).
Understandable By	Programmers.	Computer CPU.
Conversion	Needs compilation or interpretation.	Directly executed by CPU.
Example	printf("Hello");	10110000 01100001

16. Importance of Version Control

Version control tracks changes to files and allows developers to collaborate without overwriting each other's work. Benefits:

- Undoing mistakes by reverting to previous versions.
- Working on separate features using branches.

- Keeping a complete history of changes.

17. Benefits of Using GitHub for Students

- Free hosting for projects.
- Collaboration with classmates.
- Building a public portfolio for future employers.
- Exposure to industry-standard tools and workflows.

18. Open-source vs Proprietary Software

Point	Open-Source Software	Proprietary Software
Accessibility	Source code is freely available.	Source code is closed and restricted.
Cost	Usually free.	Usually paid.
Modification	Can be modified by anyone.	Cannot be modified without permission.
Examples	Linux, GIMP.	Windows, MS Office.

19. How Git Improves Collaboration

Git is a distributed version control system that allows multiple people to work on the same project **at the same time** without overwriting each other's changes. It makes teamwork more organized, efficient, and reliable.

20. Role of Application Software in Businesses

Application software refers to programs designed to help users perform specific tasks, such as word processing, accounting, customer management, or data analysis. In a business context, it's a core tool for **efficiency, productivity, and decision-making**..

21. Main Stages of the Software Development Process (SDLC)

1. Requirement Analysis – Understanding user needs.
2. Design – Planning system architecture.
3. Implementation – Writing code.
4. Testing – Finding and fixing errors.
5. Deployment – Delivering to users.
6. Maintenance – Updating and improving the system.

22. Importance of Requirement Analysis

Requirement Analysis is the **first and most critical stage** of the Software Development Life Cycle (SDLC). It's the process of gathering, understanding, and documenting what the stakeholders expect from the software. A strong requirement analysis ensures the project is built **correctly and efficiently**.

Why Requirement Analysis is Important

1. Prevents Misunderstandings

- Ensures developers and clients have the same understanding of the project scope.
- Example: Avoids situations where the client wanted a "chat feature" but the team built only a "contact form."

---

2. Saves Time and Cost

- Catching unclear or incorrect requirements early prevents costly rework later.
- Example: Changing a core feature during coding is much harder than clarifying it before development starts.

---

3. Defines Clear Goals

- Sets measurable objectives for the project.
- Example: “Process 1,000 transactions per minute” is a clear, testable goal.

---

4. Guides the Design & Development

- The requirement document acts as a blueprint for system design and coding.
- Example: UI wireframes and database structures are based on confirmed requirements.

---

5. Improves Quality

- Well-defined requirements lead to software that meets user needs and expectations.
- Example: Including accessibility features because it was identified during requirements gathering.

---

6. Reduces Scope Creep

- Prevents uncontrolled changes to the project by defining what’s in and out of scope.
- Example: If the client later requests an unrelated feature, the team can refer back to the signed-off requirements.

23. Role of Software Analysis

Software Analysis is the process of **examining, understanding, and defining** the requirements and constraints of a software project before actual development begins. It acts as the **foundation** for building software that meets user needs, works efficiently, and stays within budget.

24. Key Elements of System Design

1. Architecture Design

- Defines the **overall structure** of the system.
- Includes **client-server models**, layered architecture, or microservices.
- Example: Choosing a 3-tier architecture (presentation, logic, data layers) for a banking app.

---

2. Data Design

- Deals with **how data is stored, organized, and accessed**.
- Includes database schemas, data flow diagrams (DFDs), and ER diagrams.
- Example: Designing a relational database for customer and transaction records.

---

3. Interface Design

- Specifies how different parts of the system **communicate** with each other and with the user.
- Includes **UI/UX design**, API endpoints, and integration methods.

- Example: A clean dashboard interface for an e-commerce admin panel.

---

4. Component Design

- Breaks the system into **modules** or components.
- Defines the functionality and interactions of each module.
- Example: Separate modules for login, product catalog, and payment gateway.

---

5. Security Design

- Plans how to **protect data and system operations** from threats.
- Includes encryption, authentication, and authorization mechanisms.
- Example: Role-based access control for employees.

---

6. Performance Design

- Ensures the system meets **speed, scalability, and reliability** requirements.
- Example: Caching frequently used data to improve load times.

---

7. Integration Design

- Defines how the system will work with **other systems or external services**.
- Example: Integration with a third-party payment processor like PayPal.

25. Importance of Software Testing

Software Testing is the process of **evaluating a software application** to ensure it is free from defects, meets the specified requirements, and delivers a high-quality user experience. It plays a crucial role in the **Software Development Life Cycle (SDLC)** to avoid failures after release

26. Types of Software Maintenance

Main 4 Types (Primary Classification – IEEE Standard)

These are the most widely accepted categories:

1. **Corrective Maintenance**
    - **Purpose:** Fix bugs, defects, or faults found after deployment.
    - **Example:** Resolving a crash in a mobile app caused by invalid input.
2. **Adaptive Maintenance**
    - **Purpose:** Modify the software to work in a changing environment (OS upgrades, new hardware, legal rules).
    - **Example:** Updating an accounting system for new tax laws.

3. **Perfective Maintenance**
    - **Purpose:** Improve or enhance the software based on user needs and performance feedback.
    - **Example:** Adding a search filter to improve usability.
4. **Preventive Maintenance**
    - **Purpose:** Make changes to prevent potential future issues and ensure long-term stability.
    - **Example:** Refactoring code to reduce complexity and avoid technical debt.

Extended Categories (Some Models Include These Separately)

These are sometimes considered subtypes or extensions:

5. **Emergency Maintenance**
    - **Purpose:** Urgently fix critical issues that cause major system downtime or security breaches.
    - **Example:** Patching a vulnerability that's actively being exploited.
6. **Enhancement Maintenance** *(often grouped under Perfective)*
    - **Purpose:** Add new features or expand system capabilities.
    - **Example:** Integrating a new payment method into an e-commerce site.
7. **Routine Maintenance** *(often considered Preventive)*
    - **Purpose:** Regular updates, backups, and optimizations to keep the system healthy.
    - **Example:** Database optimization every quarter.

27. Web Application

Point	Web Applications	Desktop Applications
Location	Runs inside a web browser.	Installed locally on a device.
Point	Web Applications	Desktop Applications

Internet Requirement	Needs internet to function.	Can work offline.
Updates	Updated on server, instantly available to all users.	Requires manual updates on each device.
Accessibility	Accessible from any device with a browser.	Accessible only on the installed device.

28. Advantages of Web Applications Over Desktop Applications

1. Accessibility from Anywhere

- **Advantage:** Can be accessed from any device with a browser and internet connection.
- **Example:** Google Docs can be used on a laptop, tablet, or phone without installation.

---

2. No Installation Required

- **Advantage:** Users don't need to download or install the app locally.
- **Example:** You can use Gmail instantly through a browser without setup.

---

3. Platform Independence

- **Advantage:** Works across different operating systems (Windows, macOS, Linux, Android).
- **Example:** Trello works the same way on both Windows and Mac.

---

4. Easy Updates and Maintenance

- **Advantage:** Updates are applied on the server, so users always get the latest version automatically.
- **Example:** Netflix adds new features without users having to download updates.

---

5. Lower Hardware Requirements

- **Advantage:** Processing is often done on the server, reducing strain on the user's device.
- **Example:** Cloud-based video editors like Canva work smoothly on basic laptops.

---

6. Easier Collaboration

- **Advantage:** Multiple users can work on the same data in real time.
- **Example:** Multiple people editing the same spreadsheet in Google Sheets.

---

7. Cost-Effective for Deployment

- **Advantage:** Developers don't need to package and distribute installers for different platforms.
- **Example:** A single web app version works for all users.

29. Role of UI/UX Design in Application Development

Point	Native Mobile Apps	Hybrid Mobile Apps
Development	Built for a specific OS using platformspecific languages.	Built using web technologies for multiple OS.
Performance	High performance, optimized for platform.	Slightly lower due to compatibility layers.
Maintenance	Requires separate code for each OS.	One codebase works for multiple platforms.
Examples	Android apps in Kotlin/Java, iOS apps in Swift.	Apps built with Flutter, React Native.

30. Native vs Hybrid Mobile Apps

Aspect	Native Apps	Hybrid Apps
Definition	Apps developed specifically for a single platform (iOS, Android, etc.) using platform-specific languages.	Apps built using web technologies (HTML, CSS, JavaScript) wrapped in a native container to run on multiple platforms.
Languages Used	- **iOS**: Swift, Objective-C - **Android**: Java, Kotlin	HTML, CSS, JavaScript (with frameworks like Ionic, React Native, Flutter, Cordova)
Platform Compatibility	One platform only (need separate codebases for iOS and Android).	Cross-platform (one codebase runs on multiple platforms).
Performance	High performance due to direct access to device APIs and hardware.	Slightly lower performance due to extra layer between code and device APIs.
User Experience (UX/UI)	Best UX — follows platform-specific design guidelines.	UX may not be as smooth, but can be close to native with the right framework.
Access to Device Features	Full access to all device features (camera, GPS, sensors, etc.).	Limited access; may require plugins for advanced features.
Development Time & Cost	Longer time and higher cost	Faster and cheaper (single

• **Aspect**	• **Native Apps**	• **Hybrid Apps**
	(separate development for each platform).	codebase for multiple platforms).
• **Maintenance**	• More effort — updates must be made for each platform separately.	• Easier — update once for all platforms.
• **Offline Capability**	• Strong offline support.	• Can have offline support, but may be less efficient.
• **Examples**	• WhatsApp, Instagram, Google Maps	• Instagram (older version), Uber, Gmail (partially hybrid)

- .

31. Significance of DFDs in System Analysis

Point	Desktop Applications	Web Applications
Installation	Requires installation on the system.	No installation required.
Platform Dependency	Usually works only on specific OS.	Works on multiple platforms via browser.
Performance	Often faster and can use full system resources.	May be slower, limited by browser capabilities.
Accessibility	Accessible only on installed device.	Accessible anywhere with internet.

32. Pros and Cons of Desktop Applications

Pros	Cons
Works offline without internet.	Requires installation on each device.
Often faster and more responsive than web apps.	Platform-dependent, may not work on all OS.
Can utilize full system resources (CPU, GPU, storage).	Updating requires manual effort on each device.
Generally offers better performance for heavy tasks (e.g., video editing, gaming).	Not accessible remotely unless additional setup is done.

33. How Flowcharts Help in Programming and System Design

. Visualizing the Process

- Flowcharts present the logic of a program or system **step-by-step in a graphical form**.
- Makes it easier to **understand the flow of control** without reading complex code.

Example: A decision point in a login system ("Is password correct?") is easy to follow in a flowchart.

---

2. Improving Communication

- Acts as a **common language** between developers, testers, designers, and non-technical stakeholders.
- Reduces misunderstandings by providing a **clear visual representation** of the process.

---

3. Simplifying Complex Logic

- Breaks down complicated processes into **smaller, understandable steps**.
- Makes it easier to **identify dependencies** and **sequence of operations**.

---

4. Aiding Debugging and Problem Solving

- Helps programmers **trace logic errors** and find where a process fails.
- Makes it easy to **simulate different scenarios** before actual coding.

---

5. Supporting Documentation

- Serves as a **permanent reference** for future maintenance or upgrades.
- Useful in **training new team members**.

---

6. Enhancing System Design

- Assists in **designing system architecture** before implementation.
- Helps in **comparing alternative solutions** and selecting the most efficient one.