**Overview of IT Industry**

Module 1

1. What is a Program?

* A computer program is essentially a sequence of instructions written in a programming language that a computer can execute to perform specific tasks or solve problems.
* **Writing the code**: A programmer writes the instructions in a language like Python, Java, or C++. This is called **source code**.
* **Translation**: The source code is then converted into machine-readable instructions. This can be done by a **compiler** (which translates the entire program at once) or an **interpreter** (which translates and executes the code line by line).
* **Execution**: The computer's central processing unit (CPU) reads and carries out the machine code instructions, performing tasks like displaying text, playing a video, or calculating a spreadsheet.

1. What is Programming?

* Programming is the process of creating a set of instructions that tell a computer how to perform a specific task.
* **Planning & Requirement Analysis**
* Define the project's purpose, scope, resources, timelines, and risks.
* Gather inputs from customers, stakeholders, and business analysts to form a Software Requirements Specification (SRS).
  + **Defining Requirements / Analysis**
* Drill down into functional and non-functional needs.
* Review feasibility and desired outcomes.
* **Design & Architecture**
* Create blueprints that cover system structure, user interfaces, data flow, choice of technologies, and more.
* Document these in a Design Document Specification (DDS).
* **Implementation / Development**
* Convert design into actual code using programming tools like compilers, interpreters, and IDEs.
* Developers follow design guidelines and coding standards.
* **Testing & Integration**
* Validate the software through unit testing, integration testing, system testing, and user acceptance.
* Detect and resolve defects to ensure quality.
* **Deployment**
* Release the product to a live environment.
* Sometimes done in stages (e.g., beta pilots) for smoother adoption.
* **Maintenance & Support**
* Continually monitor, update, and refine software based on user feedback, bug reports, or evolving requirements.

1. Types of Programming Languages?

* There are 2 types of languages- i) Low level language

| **Aspect** | **High-Level Languages** | **Low-Level Languages** |
| --- | --- | --- |
| Abstraction | High — hides machine details | Low — explicit, hardware-focused |
| Ease of Use | Easy to code, read, and debug | Complex, requires hardware knowledge |
| Performance | Slower, less efficient | Fast and resource-efficient |
| Portability | Highly portable | Platform-specific |
| Hardware Control | Limited control | Full control over hardware and memory |
| Development Speed | Fast development and maintenance | Time-consuming, error-prone |
| Typical Use Cases | General apps, web, scripting, data analysis | System programming, embedded/real-time systems |

ii) High level language

1. World Wide Web & How Internet Works?

* The World Wide Web is a system of interconnected documents and resources (webpages, images, videos) accessible through the Internet. These documents are linked via hyperlinks and are located using Uniform Resource Locators (URLs).

Work:

* A user enters a URL into their browser (the client).
* The client sends an HTTP request to the server associated with that URL.
* The server receives the request, processes it, and generates a response.
* The server sends the HTTP response back to the client
* The client receives the response and displays the web page or other content to the user.

1. Explain the function of the TCP/IP model and its layers.

* The TCP/IP model, also known as the Internet Protocol Suite, is the foundation for how network communication occurs—especially over the internet.
* **Application Layer**
* Closest to the user, this layer interfaces with applications like web browsers, email clients, and more.
* It supports protocols like HTTP, FTP, SMTP, and DNS, enabling services like web browsing, file transfer, and email.
* It also handles data formatting, encryption, and session control—functions merged from OSI’s top three layers
* **Transport Layer**
* Ensures reliable host-to-host communication, handling data integrity, order, and flow control.
* Key protocols:
  + TCP: Connection-oriented, ensures error-free, in-order delivery.
  + UDP: Connectionless and faster, suitable for streaming, games, DNS queries.
  + **Internet (Network) Layer**
* Manages logical addressing and routing, directing data packets across networks.
* IP (Internet Protocol) provides unique addresses and routing logic; other protocols include ICMP, ARP, and IPv6.
* **Network Access (Link) Layer**
* Covers the physical and data link responsibilities—transmitting data over local networks via Wi-Fi, Ethernet, or other media.
* Handles framing, MAC addressing, and link-specific error detection and control.

1. Explain Client Server Communication

* Client Sends a Request:  
  The client initiates interaction by sending a structured request to the server, commonly using protocols like HTTP for web pages.
* Server Processes the Request:

The server receives the request, performs some action—such as querying a database, assembling content, or running logic—and prepares a response.

* Server Sends Back a Response:

A response is sent back to the client. In the case of HTTP, the response includes status information and requested content.

* Client Renders or Uses the Data:

The client processes the response—displaying a webpage, showing an email inbox, or loading images. This may involve client-side logic, rendering engines, or local caching.

1. How does broadband differ from fiber-optic internet?

| **Aspect** | **Broadband (Non-Fiber)** | **Fiber-Optic Internet** |
| --- | --- | --- |
| **Underlying Tech** | Uses copper (DSL, cable), wireless, or satellite methods. | Uses optical fibers transmitting light signals. |
| **Speeds** | Typically 25–100 Mbps; peak cable may reach up to ~1 Gbps. | Generally 100 Mbps to 1 Gbps+, with potential for much higher speeds. |
| **Upload vs Download** | Usually asymmetrical—slower uploads than downloads. | Often symmetrical (equal upload and download speeds). |
| **Reliability** | Prone to interference, weather issues, and signal loss over distance. | Highly resistant to interference; maintains stability over longer distances. |
| **Latency** | Relatively higher latency. | Very low latency, ideal for real-time applications. |
| **Consistency** | Speeds can drop during peak usage times. | Maintains consistent performance even under load. |

1. What are the differences between HTTP and HTTPS protocols?

| **Feature** | **HTTP** | **HTTPS** |
| --- | --- | --- |
| **Security** | No encryption—data visible | Encrypted via SSL/TLS |
| **Authentication** | None—no server verification | Validates server identity via certificates |
| **Integrity** | Unchecked—data can be altered | Verified—any tampering is detectable |
| **Default Port** | 80 | 443 |
| **URL Prefix** | http:// | https:// |
| **Performance** | Faster—no encryption overhead | Slight overhead from handshake; optimized with HTTP/2 |
| **Trust & SEO** | Browsers flag as insecure | Trusted, ranked higher, and displays padlock |
| **Protocol Layer** | Application layer, stateless | HTTP over encrypted SSL/TLS, Application layer |

1. What is the role of encryption in securing applications?

* Confidentiality:

Encryption transforms readable data (*plaintext*) into an unreadable format (*ciphertext*), ensuring only authorized users with the correct key can access it. This is essential for protecting sensitive information—like personal identifiable data, financial records, and private communications—whether it's stored on a device or transmitted across networks.

* Integrity & Authenticity:

Encrypted data includes mechanisms (e.g., digital signatures, checksums) that expose tampering attempts. If any modification occurs during transmission or storage, the data fails verification upon decryption—signaling a breach or corruption.

* Authentication & Trust Building

Public-key (asymmetric) cryptography enables authentication—verifying the identity of users or servers (e.g., via SSL/TLS certificates). Encryption establishes secure communication channels based on trust.

* Regulatory Compliance:

Many industries are legally required to protect customer data using encryption. Regulations like PCI-DSS, HIPAA, and GDPR mandate encryption to prevent breaches and avoid penalties.

1. What is the difference between system software and application software?

| **Aspect** | **System Software** | **Application Software** |
| --- | --- | --- |
| **Purpose** | Manages and integrates hardware | Performs specific user tasks |
| **Dependency** | Independent; essential for system operation | Dependent on system software |
| **Interaction Style** | Mostly background, indirect interaction | Direct, user-facing interaction |
| **Language Used** | Low-level languages (e.g., C, Assembly) | High-level languages (e.g., Java, Python) |
| **Startup Behavior** | Starts at system boot; runs persistently | Started/stopped by user as needed |
| **Examples** | OS, firmware, drivers, compilers | Word processors, web browsers, games, media editors |

1. What is the significance of modularity in software architecture?

* Maintainability & Simplicity:

Dividing a complex system into smaller, focused components makes it much easier to understand, fix, and evolve. Developers can update or debug a specific module without worrying about unintended effects on the more extensive system.

* Reusability & Consistency:

Modules built to be reusable can be leveraged across different projects or multiple parts of the same project. This reduces the need to rewrite similar functionality and ensures consistency.

* Scalability & Flexibility:

Modular structures allow systems to grow organically. New modules can be added— or swapped in—without disrupting existing functionality. This approach supports architectural flexibility and scalability.

* Parallel Development & Team Efficiency:

When modules are well-defined, different developers or teams can work on them independently. This enables parallel workflows and speeds up development cycles.

* Testing & Debugging:

Testing becomes more effective with modularity. Each module can be validated independently (unit testing), making bug detection and fixes faster and more precise.

1. Why are layers important in software architecture?

* Separation of Concerns:

Each layer focuses on a specific functionality—such as user interface, business logic, or data access—making systems easier to understand and manage. This structuring fosters clearer code and cleaner development boundaries.

* Modularity & Maintainability:

Layering enables you to update or replace one part of the system (e.g., swapping a database) without affecting others, improving both maintainability and long-term adaptability.

* Testability:

With clear boundaries between layers, developers can test them individually—using stubs or mocks—yielding more reliable and efficient testing strategies.

* Reusability:

Well-defined layers—like a business logic or data access layer—can be reused across multiple applications or modules, reducing duplication and simplifying development

* Scalability:

Individual layers can be scaled independently. For instance, you might deploy multiple UI servers to handle traffic spikes while backend services remain unchanged

* Improved Security:

Security—such as access control or input validation—can be enforced at the appropriate layer, building a defense-in-depth architecture.

1. Explain the importance of a development environment in software production.

* **Isolation and Safety**

A development environment acts as a **sandbox**, allowing developers to make changes, introduce new features, or fix bugs without any risk of breaking the live, "production" version of the application that end-users are using. This isolation is critical for preventing downtime and service disruptions. It allows for mistakes and learning without negative consequences for the business or its users.

* **Standardization and Consistency**

A well-configured development environment ensures that all developers on a team are working with the exact same versions of software, libraries, databases, and other dependencies. This standardization is vital for collaboration, as it eliminates the "it works on my machine" problem. By maintaining consistency, teams can easily share code and work together, knowing that the code will behave predictably across all their environments.

* **Efficiency and Productivity**

Modern development environments are equipped with an array of tools that boost productivity. These tools often come integrated into a single application, known as an **Integrated Development Environment (IDE)**, which typically includes:

* A **code editor** with features like syntax highlighting and auto-completion.
* A **debugger** for stepping through code to find and fix errors.
* A **compiler** or **interpreter** to transform the code into an executable program.
* **Version control** integration to manage code changes and collaboration.

1. What is the difference between source code and machine code?

* Source code: **Human-readable** code written in high-level languages (like Python, Java, C++) or assembly language. It’s designed for developers to understand, maintain, and modify.
* Machine code: **Binary instructions** (0s and 1s) that the CPU executes directly; it's the only code the hardware truly understands

| **Aspect** | **Source Code** | **Machine Code** |
| --- | --- | --- |
| **Readability** | High—designed for human comprehension | Very low—binary and opaque |
| **Purpose** | Defines application logic and behavior | Directly executed by CPU |
| **Modification** | Easy to edit and debug | Hard to reverse-engineer or change |
| **Production Path** | Translated via compiler/assembler into machine code | End result ready to run on hardware |
| **Analogy** | Recipe (understandable, modifiable) | Baked cake (final product—hard to "unbake") |

1. What is the role of application software in businesses?

* **Automating Repetitive Tasks**

Many business processes involve routine, repetitive tasks that are time-consuming and prone to human error. Application software automates these tasks, freeing up employees to focus on more strategic and creative work.

* **Examples:** Accounting software (like QuickBooks or Xero) automates billing and payroll, while marketing automation software handles tasks like sending email blasts and social media posting.
* **Enhancing Communication and Collaboration**

Business application software provides a unified platform for internal and external communication, ensuring that teams can work together efficiently, regardless of their location.

* **Examples:** Collaboration platforms (like Slack or Microsoft Teams) provide instant messaging and file sharing, while project management tools (like Trello or Asana) help teams track progress, assign tasks, and manage resources.
* **Improving Customer Experience**

In a competitive marketplace, providing a seamless and satisfying customer experience is crucial. Application software helps businesses manage customer interactions and provide better service.

* **Examples:** CRM software tracks customer interactions and helps sales teams manage pipelines, while e-commerce platforms (like Shopify) provide the infrastructure for online sales and customer transactions.
* **Optimizing Business Processes**

From supply chain to finance, different departments within a business have unique needs. Specialized application software is designed to optimize these specific workflows, leading to increased efficiency and reduced costs.

* **Examples:** Supply Chain Management (SCM) software helps businesses track inventory and optimize logistics, while Human Resources (HR) software simplifies tasks like recruitment, performance evaluation, and payroll management.

1. What are the main stages of the software development process?

| **Phase** | **Purpose** |
| --- | --- |
| **Planning / Analysis** | Define objectives, scope, feasibility, and user requirements |
| **Design** | Translate requirements into architecture, UI, data models, and technical specifications |
| **Development (Coding)** | Build the actual software through coding and integration |
| **Testing** | Identify and fix defects; verify that software meets specified requirements |
| **Deployment & Implementation** | Release the software into production and onboard users |
| **Maintenance** | Monitor usage, address issues, update features, and adapt to evolving needs |

1. Why is the requirement analysis phase critical in software development?

* The **requirements analysis** phase is the most critical stage of software development because it serves as the **blueprint for the entire project**.

| **Benefit** | **Why It Matters** |
| --- | --- |
| Clear Objectives | Aligns team on goals, reduces misunderstandings |
| Cost & Time Efficiency | Minimizes rework and delays |
| Better Communication | Ever-transparent expectations and roles |
| Effective Planning | Accurate estimates and risk anticipation |
| Quality Assurance | Captures real needs—both functional and performance-based |
| Traceability & Flexibility | Easier to adapt and verify changes |

1. What is the role of software analysis in the development process?

* **Requirements Elicitation**
* The primary role is to actively **gather** information from all stakeholders—clients, end-users, domain experts, and project managers—to determine the goals of the software.
* **Activities:** Conducting interviews, workshops, surveys, and reviewing existing documentation to capture initial ideas and high-level business goals.
* **Analysis and Modeling**

The gathered, often vague or conflicting, information is analyzed and refined into a coherent set of requirements. This involves using modeling techniques to visualize the proposed system.

* **Activities:**
  + **Functional Requirements:** Defining specific actions the system must perform (e.g., "The user must be able to log in using an email and password").
  + **Non-Functional Requirements (Constraints):** Defining quality attributes (e.g., performance, security, scalability, usability).
  + **Modeling:** Creating diagrams (like use-case diagrams, sequence diagrams, or data flow diagrams) to illustrate the system's structure and behavior.
  + **Requirements Specification**
    - The analyst produces a formal document, typically the **Software Requirements Specification (SRS)**, which is the official agreement on what the software will do. This document is written in a clear, verifiable, and consistent language.
* **Result:** The SRS serves as the **blueprint** for the design and coding teams and is the basis for all testing and quality assurance activities.
* **Scope Management:**

By defining the boundaries of the project early, the analyst helps **prevent scope creep** (the uncontrolled addition of features). Any proposed changes to the software's functionality must be evaluated against the initial analyzed requirements.

In essence, software analysis ensures that development efforts are focused on **solving the correct problem** for the intended users, thereby maximizing the chances of project success and stakeholder satisfaction.

1. What are the key elements of system design?

* Requirements and Constraints:

Before any design work begins, it's crucial to understand the functional requirements (what the system must do) and non-functional requirements (qualities like performance, security, and reliability). System design must also account for constraints such as budget, timeline, and available technology.

* Architecture:

This is the high-level structure that defines how the system's components are organized and how they interact. Common architectural styles include monolithic, microservices, and client-server, each with its own trade-offs.

* Databases and Storage:

A system's design must specify how data will be stored, managed, and retrieved. This involves choosing the right type of database (e.g., SQL or NoSQL), considering how data will be structured, and planning for data scaling techniques like sharding or replication.

* APIs and Communication:

System components need to communicate with each other. Application Programming Interfaces (APIs) define the rules for this interaction, while communication protocols (like HTTP or RPC) govern how data is exchanged.

1. Why is software testing important?

* Software testing is crucial because it ensures the quality, reliability, and security of a software product before it's released to the public.
* **Cost Savings**: Finding and fixing bugs after a product has been released is significantly more expensive than addressing them during the development phase. Testing early and often helps a team catch issues when they are easier and cheaper to fix, preventing costly recalls or emergency patches.
  + **Risk Mitigation**: Testing helps to identify risks related to security, performance, and functionality. For example, security testing can uncover vulnerabilities that could lead to data breaches, while performance testing can ensure the software won't crash under heavy user traffic. By addressing these issues proactively, testing protects a company's reputation and its users' data.
* **Quality and Reliability**: A properly tested application is more likely to function as expected and meet the requirements outlined in the design phase. This leads to a more stable and reliable product, which is essential for building user trust and satisfaction.
* **Customer Satisfaction**: Users expect a seamless and bug-free experience. A poorly tested application that crashes, is slow, or has glaring errors can quickly lead to user frustration and negative reviews. By ensuring the software is high-quality, testing directly contributes to a better user experience and helps maintain customer loyalty.
* **Security**: In today's interconnected world, software security is paramount. Testing is essential for finding and patching security weaknesses that could be exploited by malicious actors. It protects both the user's data and the business's assets.

1. What types of software maintenance are there?

* **Corrective Maintenance**
* Involves fixing bugs and defects reported by users or detected during operation.
* It’s reactive—meant to restore proper functionality.
  + **Adaptive Maintenance**
* Ensures the software continues functioning in a changing external environment—such as new operating systems, hardware updates, third-party integrations, or regulatory shifts.
* **Perfective Maintenance**
* Enhances existing capabilities to improve performance, usability, maintainability, or to add new features based on user feedback or evolving needs.
* **Preventive Maintenance**
* Proactively addresses potential issues before they cause failures—through code cleanup, refactoring, documentation improvements, and performance optimizations—to reduce future risk.

1. What are the key differences between web and desktop applications?

| **Aspect** | **Web Applications** | **Desktop Applications** |
| --- | --- | --- |
| **Installation** | No installation—runs directly in the browser via a URL | Requires installation on each device |
| **Connectivity** | Needs internet access; limited or no offline use | Often works fully offline |
| **Performance** | Slower due to reliance on browser/server response | Faster and more responsive using local hardware |
| **Updates** | Automatically updated server-side—users always get the latest version | Manual updates needed—or automated per machine policies |
| **Accessibility & Portability** | Accessible from any browser on any platform | Limited to the installed device and OS |
| **Collaboration** | Excellent for real-time collaboration & sharing via the cloud | Usually lacks built-in collaborative features |

1. What are the advantages of using web applications over desktop applications?

| **Feature** | **Web Application** | **Desktop Application** |
| --- | --- | --- |
| Installation | None required | Manual installation per device |
| Updates | Automatic & centralized | Manual or per-device auto-updates |
| Platform Compatibility | Runs on any browser/device | OS-specific; limited cross-platform |
| Accessibility | Anywhere via internet | Restricted to installed devices |
| Resource Use | Low on client resources | Heavier usage of local CPU, RAM, storage |
| Development Cost | Lower; single codebase | Higher; multi-platform development needed |
| Collaboration | Real-time, cloud-based sync | Typically isolated per device |
| Scalability | Easily scalable via cloud | Limited by local resources |
| Integration | Easy API integrations with ecosystem | Often siloed or complex to link systems |
| Security & Maintenance | Managed centrally | Requires per-device security and patching |

1. What role does UI/UX design play in application development?

* UI and UX are not just design checkboxes—they’re the user’s lens into your app and the glue that holds the experience together. Investing in sound UI/UX design can lead to:
* Increased user retention, engagement, and loyalty
* Boosted conversion rates and revenue potential
* Stronger brand identity and trust
* Enhanced usability, speed, and accessibility
* Reduced costs through early error detection and design systems.

1. What are the differences between native and hybrid mobile apps?

| **Aspect** | **Native Apps** | **Hybrid Apps** |
| --- | --- | --- |
| Performance | Excellent, platform-optimized | Moderate; depends on web container |
| User Experience (UX) | Polished and platform-consistent | Fair; may feel generic |
| Device Feature Access | Full access | Limited; via plugins |
| Development Cost | Higher (dual codebases) | Lower (single codebase) |
| Time-to-Market | Longer | Faster |
| Maintenance | Complex (separate updates) | Easier (unified codebase) |
| Offline Capability | Strong, robust offline support | Limited functionality |
| Security & Stability | Higher, native-level assurance | Dependent on framework quality |

1. What is the significance of DFDs in system analysis?

* Data Flow Diagrams (DFDs) are a significant tool in system analysis because they provide a simple, visual representation of how data moves through a system. They serve as a blueprint, helping to **document, analyze, and communicate** the processes and data exchanges, making complex systems easier to understand.

1. What are the pros and cons of desktop applications compared to webapplications?

| **Aspect** | **Web Applications** | **Desktop Applications** |
| --- | --- | --- |
| **Accessibility** | Global via browser; no installation | Local device only |
| **Installation** | Not required; instant access | Required on each device |
| **Updates** | Automatic via server | Manual, per device |
| **Performance** | Slower, dependent on network & browser | Fast, optimized for local hardware |
| **Offline Use** | Limited (unless built for PWA) | Fully functional offline |
| **Security & Privacy** | Relies on cloud and browser security | More control; depends on local security |
| **Features & UX** | Constrained by browser limits | Rich, hardware-integrated features |
| **Collaboration** | Strong; real-time sync | Requires external sync tools |
| **Cost Structure** | Ongoing subscriptions common | Often one-time payments or licenses |
| **Cross-Platform** | High—one codebase | Low—requires separate versions |
| **Resource Use** | Higher energy, CPU, memory (web overhead) | Generally more efficient locally |

1. How do flowcharts help in programming and system design?

Flowcharts are not just diagrams—they’re strategic tools. They sharpen logical clarity, enhance collaboration, and fortify the design lifecycle. While they demand effort upfront, their payoff in reduced bugs, smoother communication, and maintainable architecture often justifies the investment.

**Benefits of Using Flowcharts:**

* Visualizing Logic & Algorithms
* Streamlined Problem Analysis & Design
* Improved Coding & Debugging
* Enhanced Communication & Documentation.