THEORY EXERCIS

1. What is a Program?

- *It is a set of Instructions*

2. Explain in your own words what a program is and how it functions.

- *A program is instructions for a computer to execute specific tasks. It contains code written in a programming language which may be interpreted, compiled or assembled into machine readable form and then executed.*

3. What is Programming?

- ***Programming** is the process of **writing instructions** that tell a computer **what to do** and **how to do it**.*
- *It is done using **programming languages** like Python, Java, C, etc.*

4. What are the key steps involved in the programming process?

- *The programming process typically involves six key steps:*

1. *understand the problem*
2. *plan the solution*
3. *writing the code*
4. *testing the program*
5. *documenting the program*
6. *and maintaining the program*

5. Types of Programming Languages

- *Procedural programming*
  - *C language*
- *Object oriented programming*
  - *C++ language*

- *Logical programming*
    - *Pro log language*
- *Functional programming*
    - *Python language*

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

- *High-level programming languages are designed to be human-readable and easier to learn and use, while low-level languages are closer to machine code and provide more direct control over hardware. This results in significant differences in abstraction, readability, efficiency, and portability.*

1. High-Level language

Abstraction:

Offer a high level of abstraction, meaning they hide many of the low-level details of the computer's hardware.

Readability:

Designed to be more readable and understandable by humans, often resembling natural language.

Ease of Use:

Relatively easy to learn and use, making them suitable for a wider range of programmers.

Portability:

Generally more portable, meaning they can be run on different types of computer systems without significant changes.

Development Time:

Can lead to faster development times as they require less coding and debugging.

Examples:

Python, Java, C#, JavaScript, PHP.

2. *Low-Level Languages:*

Abstraction: *Offer a low level of abstraction, meaning they provide more direct control over the computer's hardware.*

Readability: *Less readable and more difficult to understand for humans.*

Control: *Provide more direct control over hardware resources and memory, but require detailed knowledge of the computer's architecture.*

Efficiency: *Often more efficient in terms of memory and performance, but require more coding and debugging.*

Examples: *Assembly language, machine code.*

7. *World Wide Web & How Internet Works*.

- *The World Wide Web (WWW), often just called "the web," is an information system that allows users to access and share content on the internet through a userfriendly, interconnected network of web pages. It functions by using the Hypertext Transfer Protocol (HTTP) to transfer web pages and other resources between servers and clients (like your web browser). These web pages are linked together via hyperlinks, creating a vast, navigable network.*

8. Describe the roles of the client and server in web communication.

- *In web communication, the client and server work together using a request–response model. The client, such as a web browser or mobile app, sends a request to the server asking for certain information. The server receives this request and replies by sending back the requested data. Then, the client shows or uses this information so the user can see it or interact with it. This is how we are able to view websites and use online services.*

Client:

- ***Initiates Requests:***

The client, such as a web browser, sends requests to the server to access resources like web pages, images, or data.

- ***Interacts with the User:***

The client displays the requested information to the user and allows them to interact with the website or application.

- ***Parses and Renders:***

The client receives and interprets the data from the server, then displays it in a usable format for the user.

- ***Example:***

When you type a URL into your browser and press enter, the browser is the client sending a request to the server.

Server:

- ***Handles Requests:***

The server, often a web server or database, receives the client's requests.

- ***Processes Data:***

The server processes the request, retrieves the necessary data, and prepares the response.

- ***Sends Responses:***

The server sends the requested data or performs the requested action, responding to the client.

- ***Provides Services:***

The server manages resources like data storage, security, and overall application functionality.

- ***Example:***

The web server hosting a website receives a request from a client (your browser) to access a page, retrieves the page content, and sends it back to the client to be displayed.

Key Differences:

- ***Request/Response:***

The client initiates requests, and the server responds, following a defined protocol.

- ***Focus:***

The client focuses on user interaction and data presentation, while the server focuses on data processing and resource management.

- ***Location:***

Clients can be anywhere with network connectivity, while servers are typically hosted in dedicated data centres.

9. Network Layers on Client and Server.

Application Layer (Layer 7):

The client uses this layer to interact with applications like web browsers and email clients, initiating requests and receiving responses. The server also uses this layer to process requests from clients and provide the requested data.

Presentation Layer (Layer 6):

This layer handles data translation, encryption/decryption, and compression to ensure data is compatible between client and server.

Session Layer (Layer 5):

This layer manages the communication sessions between client and server, establishing connections and handling login/logout processes.

Transport Layer (Layer 4):

The client uses this layer to send data to the server, while the server uses it to receive and deliver data back.

Network Layer (Layer 3):

This layer handles routing data between the client and server, ensuring packets reach the correct destination using IP addresses.

Data Link Layer (Layer 2):

This layer transmits data within a local network between the client and the initial server or switch.

Physical Layer (Layer 1):

This layer transmits the physical bits and bytes over the network cable or wireless signal.

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

- ➢ *The TCP/IP model is a foundational framework for internet communication, providing a standardized set of rules (protocols) for devices to exchange data. It's a four-layer model*

(Application, Transport, Internet, and Network Access) where each layer performs specific tasks, enabling reliable and efficient communication across networks.

Layer Functions:

- ***Application Layer:***

This layer provides the interface for user applications to access network services. It includes protocols like HTTP, SMTP, and FTP, which handle data formatting, encoding, and presentation.

- ***Transport Layer:***

The transport layer is responsible for reliable data delivery between applications on different hosts. It uses protocols like TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) to ensure data is delivered in the correct order and without loss, or to handle flow control.

- ***Internet Layer:***

The internet layer manages addressing and routing of data packets across networks. It uses the IP (Internet Protocol) to assign addresses and determine the best path for data packets to travel.

- ***Network Access Layer:***

This layer handles the physical transmission of data across the network. It uses protocols like Ethernet and Wi-Fi to manage how data is transmitted over the physical medium.

11. Client and Servers

➢ *In a client-server model, a client is a device or program that requests information or services from a server, which is a dedicated computer or software that provides those services. This architecture is fundamental for how data is exchanged over a network, like the internet.*

Elaboration:

- ***Clients:***

Clients are the end-user devices or programs that initiate communication and request resources or services from the server. Examples include web browsers, email clients, and various applications.

- ***Servers:***

Servers are powerful computers or software that manage and provide services to multiple clients. They receive requests from clients, process them, and send back responses. Examples include web servers, database servers, and email servers.

12. Explain Client Server Communication

- *Client–server communication is a model used in computer networks where two main types of programs (or devices) interact:*

- ***Client:*** *Requests services or resources.*
- ***Server:*** *Provides those services or resources.*

How it Works :

1. ***Client Sends Request***
The client (e.g., your web browser) creates a request message and sends it over the network to the server.

2. ***Server Receives Request***
The server (e.g., a web server) listens for incoming requests, receives the client's message, and processes it.

3. ***Server Sends Response***
After processing, the server sends back a response message to the client (e.g., HTML content of a webpage).

4. ***Client Receives Response***
The client gets the response and uses it—like displaying the webpage on your screen.

13. Types of Internet Connections

1. DSL (Digital Subscriber Line)

- *Uses* ***telephone lines*** *(the copper wires that bring phone service to your home).*
- *You can* ***use the phone and internet at the same time****.*
- *Speed is usually* ***medium*** *(about 1–100 Mbps).*
- *Often used in homes and small offices.*

2. Cable Internet

- *Uses the* ***same cables that bring cable TV*** *to your house.*
- *Usually* ***faster than DSL*** *(can be 10 Mbps up to 1 Gbps).*
- *Many people in the same area share the same cable, so speed may drop when everyone is online at the same time.*

3. Fiber Optic Internet

- *Uses **thin glass or plastic fibers** to send data as light.*
- *Very **high speed and reliable** — often 100 Mbps to several Gbps.*
- *Great for watching HD videos, gaming, working from home, and businesses.*

4. Satellite Internet

- *Uses **satellites in space** to connect your home to the internet.*
- *Works in **remote or rural areas** where cables are not available.*
- *Speed can be good (around 25–100 Mbps) but there is usually **high latency** — which means web pages and videos may load slower.*

5. Wireless Internet

- *No wires or cables needed.*
- *Types:*
    - ***Wi-Fi:** Connects your devices to the internet inside your home.*
    - ***Mobile Data (3G, 4G, 5G):** Lets you use internet on your phone or through portable hotspots.*
    - ***Fixed Wireless:** Connects your home to the internet using radio signals from a local tower.*
- *Useful for phones, laptops, tablets — and places where cables are hard to install.*

6. BPL (Broadband over Power Lines)

- *Uses **electricity power lines** to deliver internet.*
- *The idea is you can get internet through the same wires that bring electricity.*
- *Not common because of technical challenges and interference.*

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

***Broadband** is a **general term** that means **high-speed, always-on internet**.*

- *It **includes many types**: DSL, cable, satellite, wireless, and fiber.*
- *Broadband describes the **speed and always-on nature**, not the specific technology.*

***Fiber-optic internet** is a **specific type of broadband**:*

- *Uses **thin glass or plastic fibers** to send data as light signals.*
- *Offers **much higher speeds**, very low latency, and is more reliable.*
- *Is considered the **fastest and most advanced broadband technology**.*

15. Protocols

- *A **protocol** is like a **set of rules** that computers follow to communicate over the internet. Just like people speak in a common language to understand each other, computers use protocols.*

1. HTTP / HTTPS

- ***Full form:** HyperText Transfer Protocol / HyperText Transfer Protocol Secure*
- ***Use:** Used by browsers to load websites.*
- ***Example:** When you open https://www.google.com, your browser uses HTTPS to get the webpage safely.*
- ***HTTPS** adds encryption to keep data private and secure.*

2. TCP

- ***Full form:** Transmission Control Protocol*
- ***Use:** Makes sure data is sent **reliably and in the correct order**.*
- *Think of it like sending letters with tracking — the sender checks if the receiver got them.*

3. UDP

- ***Full form:** User Datagram Protocol*
- ***Use:** Sends data **quickly without checking for errors**.*
- *Useful for things like video calls, live games, or streaming — faster, but may lose some data.*

4. FTP

- ***Full form:** File Transfer Protocol*
- ***Use:** Transfers files from one computer/server to another.*
- *Example: Uploading files to a website.*

16. What are the differences between HTTP and HTTPS protocols? Application Security

- *HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure) are both protocols used to transfer data over the web, but they have key differences related to security. Here's a breakdown*
- ***Security:***

  HTTP: *Does not provide any encryption for data transferred between the client (e.g., web browser) and the server. This means that data sent via HTTP is vulnerable to eavesdropping, tampering, and man-in-the-middle attacks.*

  HTTPS: *Uses encryption (via SSL/TLS) to secure the communication between the client and server. This ensures that the data is encrypted before being transmitted, making it much harder for third parties to intercept or alter the data.*
- ***Port Numbers:***
  - ***HTTP:*** *Uses port* ***80*** *by default.*
  - ***HTTPS:*** *Uses port* ***443*** *by default.*
- ***SSL/TLS Encryption:***
  - ***HTTP:*** *No encryption is applied.*
  - ***HTTPS:*** *Uses* ***SSL (Secure Sockets Layer)*** *or* ***TLS (Transport Layer Security)*** *protocols to encrypt the connection, ensuring that the data is secure and private.*
- ***URL Prefix:***
  - ***HTTP:*** *URLs begin with http:// (e.g., http://www.example.com).*
  - ***HTTPS:*** *URLs begin with https:// (e.g., https://www.example.com).*
- ***Data Integrity:***
  - ***HTTP:*** *Since there is no encryption or data integrity check, data sent via HTTP can be modified or corrupted during transmission.*

HTTPS: *Ensures data integrity, meaning that the data cannot be altered during transfer without being detected. If data is tampered with, the communication will fail.*

- ***Authentication:***

- ***HTTP:*** *There is no mechanism for verifying the identity of the server. This means it is possible for users to connect to fraudulent or malicious websites.*
- ***HTTPS:*** *Provides server authentication through SSL/TLS certificates. The server must have a valid certificate signed by a trusted certificate authority (CA). This confirms that the website is legitimate and not an impostor.*

➢ ***SEO Impact:***

- ***HTTP:*** *Websites using HTTP may not rank as highly in search engines like Google compared to HTTPS sites.*
- ***HTTPS:*** *Google and other search engines give a ranking boost to HTTPSsecured websites, as they prioritize user security.*

➢ ***Performance:***

HTTP: *Generally faster because there's no encryption or decryption involved.*

- ***HTTPS:*** *Historically, HTTPS was thought to be slower due to the overhead of encryption, but with modern optimizations (like HTTP/2 and QUIC), the performance difference has become negligible.*

➢ ***Trust Indicators:***

- ***HTTP:*** *Browsers often display a warning or "Not Secure" message when you visit an HTTP site, especially when entering sensitive information.*
- ***HTTPS:*** *Browsers show a padlock symbol and typically indicate that the site is secure, giving users more confidence when interacting with the website.*

In Summary:

- ***HTTP*** *is the basic, non-secure version of the protocol.*
- ***HTTPS*** *provides secure, encrypted communication, ensuring confidentiality, integrity, and authentication.*

For any modern website handling sensitive data or aiming for better security and SEO, ***HTTPS*** *is the recommended choice.*

17. Application Security

• Application security refers to security precautions used at the application level to prevent the theft or hijacking of data or code within the application.

- *It includes security concerns made during application development and design, as well as methods and procedures for protecting applications once they've been deployed.*

- *All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.*

- *Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.*

- *It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance...*

- *All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.*

- *Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.*

It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance.

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

- ***What is encryption?***
  Encryption turns normal data (like text, passwords, or files) into secret code. Only someone with the right "key" can turn it back into readable data.

Why is encryption used in apps?

1. *To **protect sensitive data** (like user passwords, bank details, personal info)*
2. *To **keep data private** so no one else can read it, even if it's stolen*
3. *To **protect data while it's moving** over the internet so hackers can't see it*
4. *To **check data integrity**, making sure data hasn't been changed by anyone*

19. Software Applications and Its Types

● *Application software*

● *System software*

● *Driver software*

- *Middleware*
- *Programming software*

1.Application Software*: The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application. An application can be self-contained, or it can be a group of programs that run the application for the user. Examples of Modern Applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms*

Example: *Microsoft Office, Paint, Powerpoint etc..*

2.system Software: *These software programs are designed to run a computer's application programs and hardware.*

- System software coordinates the activities and functions of the hardware and software.

- It controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in.

- The OS is the best example of system software; it manages all the other computer programs.

- Other examples of system software include the firmware, computer language translators and system utilities..

Example: *Notepad ,Calculator etc..*

3.Driver Software: *Also known as device drivers, this software is often considered a type of system software.*

- Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks

- Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks

- Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers

Example: *Audio Driver,Video Driver etc..*

4.Middleware: *The term middleware describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word.*

- It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.

***Example**: database middleware,application server middleware*

***5.Programming Software:** Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs.*

- Examples of programming software include assemblers, compilers, debuggers and interpreters.

***Examples :** Turbo c,Eclipse,Sublime etc.*

20. What is the difference between system software and application software? Software Architecture.

System Software:

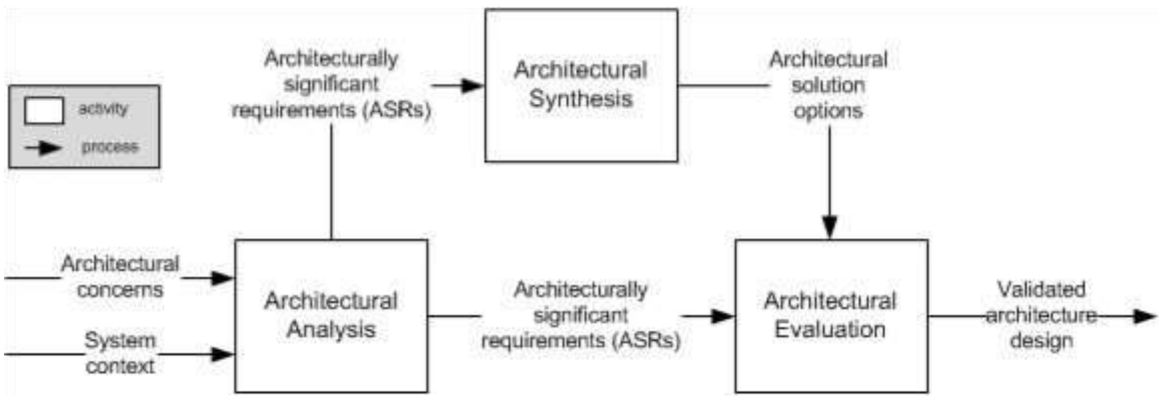
- *Runs the computer and manages hardware*
- *Works mostly in the background*
- *Needed for the computer to function*
- *Examples: Windows, Linux, MacOS*

Application Software:

- *Helps users do specific tasks*
- *Used directly by people*
- *Optional; installed as needed*
- *Examples: MS Word, web browsers, games, media players*

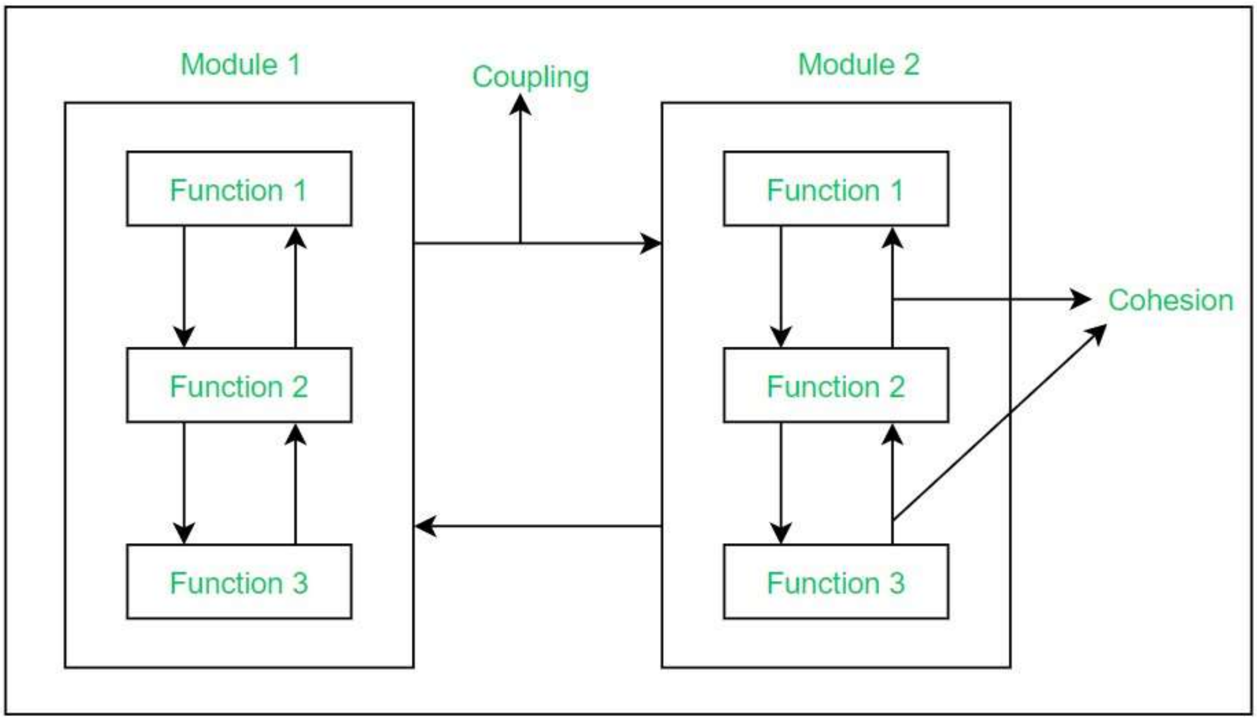
21. Software Architecture.

- *Software architecture is the high-level structure of a software system, encompassing its components, interactions, and the principles governing their design and evolution. It defines how a system is organized, how its parts relate to each other, and how they communicate. Essentially, it's the blueprint for building and maintaining a software system.*


activity
process
Architecturally significant requirements (ASRs)
Architectural Synthesis
Architectural solution options
Architectural concerns
System context
Architectural Analysis
Architecturally significant requirements (ASRs)
Architectural Evaluation
Validated architecture design

22. *What is the significance of modularity in software architecture?*

- *Modularity is crucial in software architecture because it allows for better organization, maintainability, and reusability of code, leading to more efficient development, testing, and scalability. By breaking down a system into smaller, independent modules, developers can work on different parts simultaneously, isolate issues, and reuse components across multiple projects*



23. Layers in Software Architecture

1. Presentation layer

2. Application layer

3. Business layer

4. Persistence layer

5. Database layer

1. Presentation layer:

- *The presentation layer, also called the UI layer, handles the interactions that users have with the software. It's the most visible layer and defines the application's overall look and presentation to the end-users. This is the tier that's most accessible, which anyone can use from their client device, like a desktop, laptop, mobile phone or tablet.*

2. Application layer:

- *The application layer handles the main programs of the architecture. It includes the code definitions and most basic functions of the developed application. This is the layer that programmers spend most of their time in when working on the software. You can use this layer to implement specific coordination logic that doesn't align exactly with either the presentation or business layer.*

3. Business layer:

- *The business layer, also called the domain layer, is where the application's business logic operates. Business logic is a collection of rules that tell the system how to run an application, based on the organization's guidelines. This layer essentially determines the behavior of the entire application. After one action finishes, it tells the application what to do next.*

4. Persistence layer:

- *The persistence layer, also called the data access layer, acts as a protective layer. It contains the code that's necessary to access the database layer. This layer also holds the set of codes that allow you to manipulate various aspects of the database, such as connection details and SQL statements.*

5. Database layer:

- *The database layer is where the system stores all the data. It's the lowest tier in the software architecture and houses not only data but indexes and tables as well. Search, insert, update and delete operations occur here frequently. Application data can store in a file server or* *database server, revealing crucial data but keeping data storage and retrieval procedures hidden.*

24. Why are layers important in software architecture?

- *Layers are crucial in software architecture because they promote modularity, separation of concerns, and ease of maintenance, making it easier to develop, maintain, and extend complex systems. By organizing code into distinct layers with defined responsibilities, developers can focus on specific parts of the application, enhancing productivity and simplifying the overall architecture.*

why layers are important:

1. *Modularity and Separation of Concerns:*

- *Layered architecture encourages the separation of different functionalities into distinct layers.*
- *Each layer has a specific responsibility, making the system easier to understand and maintain.*
- *This modular design allows for independent development and testing of each layer, reducing overall complexity.*

2. *Maintainability and Scalability:*

- *Changes within one layer typically do not impact other layers, simplifying maintenance and upgrades.*

Individual layers can be scaled independently to meet specific performance or load requirements, enhancing scalability.

3. *Reusability and Collaboration:*

- *Components within a layer can often be reused across different parts of the application or even in different projects.*
- *The clear structure facilitates collaboration among developers, as they can focus on specific layers without interfering with others.*

4. *Enhanced Testability and Debugging:*

- *Each layer can be tested independently, simplifying the testing process.*
- *Mock objects or stubs can be used to simulate the behaviour of adjacent layers, making it easier to isolate and debug issues.*

5. *Flexibility and Adaptability:*

- *Different technologies can be used in different layers, allowing for the use of the best tools and practices for each part of the system.*

The modular structure makes it easier to adapt the system to evolving requirements and new technologies.

25. Software Environments

- *A software environment is a* ***setup*** *where software is created, tested, and run.*
- *It includes:*
  - ***Operating system*** *→ like Windows, Linux, MacOS*
  - ***Programming languages*** *→ like Python, Java, C++*
  - ***Libraries & frameworks*** *→ extra tools that help build apps faster (like React, Django)*
  - ***Databases*** *→ where data is stored (like MySQL, MongoDB)*
  - ***Development tools*** *→ like code editors (VS Code), compilers, and testing tools*

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

- *A development environment is the* ***workspace or setup*** *where developers write, test, and improve software.*

- Code editors or IDEs (e.g., VS Code, IntelliJ)
- Compilers or interpreters
- Debugging tools
- Libraries and frameworks
- Databases
- Version control tools (e.g., Git)

1. Safe testing space

- *Developers can try out new features and changes without affecting the live (production) system.*
- *Bugs and mistakes can be fixed early*

2. Faster and better development

- *Tools like auto-complete, error highlighting, and debuggers help write code quickly and catch errors sooner.*

3. Team collaboration

- *When all team members use the same environment, code works the same way on everyone's computer.*

- *Makes it easier to share and merge code.*

4. Helps with quality

- *Testing in a controlled environment ensures the software works correctly before releasing it to users.*
- *Reduces the chances of errors in the final product.*

5. Keeps work organized

- *With version control, developers can track changes, roll back to earlier versions, and manage different parts of a project easily.*

6. Supports different stages

- *Often, there are separate environments:*
    - ***Development environment** → for coding and initial testing*
    - ***Testing (QA) environment** → for more thorough testing*
    - ***Production environment** → where the final software is used by real users*

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

1. Consistency Across Teams

- *A standardized development environment ensures that all developers work with the same tools, libraries, and configurations.*
- *This reduces the "it works on my machine" problem and improves team collaboration.*

2. Efficient Development Workflow

- *Integrated tools like IDEs, debuggers, version control, and compilers streamline the software development process.*
- *Automation tools (e.g., linters, formatters, build systems) improve productivity and code quality.*

3. Rapid Testing and Debugging

- *Development environments often include features for real-time code execution, unit testing, and debugging.*
- *This accelerates the feedback loop, allowing developers to catch and fix issues early.*

4. Environment Isolation

- *Using virtual environments or containerization (e.g., Docker) allows developers to isolate project dependencies.*
- *This prevents conflicts between projects and ensures that software behaves the same way in development, testing, and production.*

5. Safe Experimentation

- *Developers can test new features or changes in a local or staging environment without affecting the live product.*
- *This reduces risk and encourages innovation.*

6. Reproducibility

- *A well-configured development environment ensures that the code can be reliably built and run, even by new team members or in different setups.*
- *Tools like docker-compose or configuration management (e.g., Ansible, Vagrant) help recreate environments easily.*

28. Source Code

➢ ***Source code** is the human-readable set of instructions written in a programming language that defines how a software program operates. It's the original form of a software program before it is compiled or interpreted into a format the computer can execute.*

1. ***Foundation of Software***
   - *Source code is the blueprint for all software—applications, websites, operating systems, etc.*
2. ***Modifiability***
   - *Developers can edit the source code to fix bugs, add features, or improve performance.*
3. ***Collaboration & Version Control***
   - *Tools like Git allow multiple developers to work on the same source code, manage changes, and collaborate effectively.*
4. ***Transparency** (especially in open-source projects)*
   - *Anyone can review, learn from, or contribute to the software.*
5. ***Reusability***
   - *Well-written source code can be modular, allowing reuse in other projects or applications.*

27. *What is the difference between source code and machine code?*

➢ *The main difference between **source code** and **machine code** lies in their form and purpose in the software development process:*

1. Source Code:

- ***Definition:** Source code is the human-readable set of instructions written in a high-level programming language (like Python, Java, C++, etc.).*

- ***Purpose:*** *It defines the logic, functions, and behaviour of a program. It's written by developers to create software.*
- ***Human-Readable:*** *The code is understandable to humans but cannot be executed directly by a computer. It needs to be translated into machine code.*

Example: print("Hello word")

2. Machine Code:

Definition: *Machine code is a low-level programming language that consists of binary instructions (0s and 1s) directly understood by the computer's CPU.*

- ***Purpose:*** *It's the actual code that a computer's processor executes. It's hardware-specific and represents the most basic instructions for the computer.*
- ***Not Human-Readable:*** *Machine code is typically not readable by humans and is specific to the type of processor architecture (like x86, ARM, etc.).*
- ***Example:*** *A sequence of binary values like:*
  Actual machine code looks like:
  *Written in **binary** (e.g., 11001010 01100001)*

28. Github and Introductions

***GitHub** is a popular **web-based platform** used for **version control**, **code hosting**, and **collaboration** on software development projects. It is built on top of **Git**, which is a version control system that helps developers track and manage changes in their code over time.*

GitHub allows multiple people to work on the same project at the same time without overwriting each other's work. It is widely used by individuals, teams, and open-source communities around the world.

- ***Repositories:***

GitHub stores code in repositories, which are essentially online folders where projects are managed.

- ***Collaboration:***

GitHub enables developers to collaborate on projects by sharing code, tracking changes, and reviewing each other's work.

- ***Profile READMEs:***

Users can create profile READMEs to showcase their skills, interests, and projects to the GitHub community.

- ***Open Source:***

Many projects on GitHub are open-source, meaning anyone can contribute to the code and collaborate on projects.

- ***GitHub Actions:***

GitHub Actions allow for automating tasks like building, testing, and deploying code, making it easier to manage workflows.

- ***Introduction to GitHub:***

GitHub Docs offers resources for beginners to learn about Git and GitHub.

29. Why is version control important in software development?

➢ *Version control is crucial in software development because it allows developers to track changes, collaborate effectively, manage different versions of code, and revert to previous states if needed. It also enables experimentation without disrupting the main project.*

Importance of Version Control in Software Development:

1. ***Collaboration***
Version control systems (VCS) allow many developers to work on the same project at the same time without overwriting each other's changes. Features like branching and merging help manage this smoothly.

2. ***Tracking Changes***
VCS keeps a record of all changes made to the code, including who made the change, when, and why. This helps in finding and fixing bugs and understanding how the project has evolved.

3. ***Rollback and Recovery***
If a mistake happens, developers can easily go back to an earlier version of the code to undo the error, making recovery safe and quick.

4. ***Branching and Merging***
Developers can create separate branches to work on new features or experiments without disturbing the main code. Later, these changes can be merged back safely.

5. ***Backup and Disaster Recovery***
VCS automatically saves copies of the code, so even if there is a problem like hardware failure, the project can be restored without loss.

*6. **Code Experimentation***
Developers can try out new ideas in isolated branches without risking the stability of the main project.
*7. **Continuous Integration***
Version control works well with automated testing and deployment tools, helping to keep the software reliable and up-to-date.
*8. **Traceability***
VCS provides a clear history of all changes, making it easy to track where bugs came from and who made specific changes.
*9. **Error Reduction***
By managing changes carefully, version control helps prevent mistakes and accidental overwriting of code.
*10. **Improved Communication***
VCS makes it easier for team members to share their work and communicate about changes.
*11. **Project Management***
It also offers features like tracking issues, managing tasks, and reviewing code to help organize the project better.

30. Student Account in Github

https://github.com/hardik191/hardik191

31. What are the benefits of using Github for students?

- *GitHub is a powerful platform that offers many advantages for students learning software development and working on projects. Here are some key benefits:*

1. Version Control and Collaboration

GitHub helps students keep track of changes in their code and collaborate easily with classmates on group projects. Multiple people can work on the same project without conflicts.

2. Learning Real-World Tools

Using GitHub exposes students to tools and workflows commonly used in the software industry, giving them practical skills that are valuable for future jobs.

3. Portfolio Building

Students can showcase their projects publicly on GitHub. This portfolio can be shared with teachers, potential employers, or collaborators to demonstrate their skills and experience.

4. Access to Open-Source Projects

GitHub hosts millions of open-source projects. Students can explore, learn from, and even contribute to these projects, which enhances their learning and coding skills.

5. Backup and Security

GitHub stores students' code safely in the cloud, preventing loss of work due to hardware failure or accidental deletion.

6. Free Hosting for Projects

Students can host their websites or applications on GitHub Pages for free, making it easy to publish and share their work online.

7. Community and Support

GitHub provides access to a global community of developers where students can ask questions, get feedback, and collaborate on coding challenges.

8. Integration with Other Tools

GitHub integrates with many programming tools and learning platforms, helping students automate testing, deployment, and project management.

32. What are the benefits of using Github for students?

- *GitHub is a website where students can **store their code**, **share it with others**, and **work together** on projects. It helps in learning and improving programming skills.*

Benefits of Using GitHub for Students :

*1. **Free Tools***
*GitHub gives many **free tools** for students to learn and build projects.*

*2. **Work with Friends***
*Students can **work in teams** and share their code easily.*

*3. **Learn New Skills***
*Students learn how to use **Git**, which is used in many software jobs.*

*4. **Backup Code***
*Code is saved online. If anything goes wrong, students can get the **old version back**.*

*5. **Show Your Work***
*Students can show their **projects to teachers and companies**.*

*6. **Join Open Projects***
*Students can help in **real coding projects** and learn from others.*

*7. **Improve Writing***
*Students learn how to write **project details and instructions** clearly.*

*8. **Learn from Others***
*Students can **see other people's code** and learn new ideas.*

33. Types of Software

1. System Software

- *Controls and manages computer hardware.*
- *Works in the background to run the system.*
- *Acts as a platform for other software.*
- *Examples:*
    - *Operating System: Windows, Linux*
    - *Device Drivers: Printer Driver, Audio Driver*
    - *Utilities: Antivirus, Disk Cleanup*
    - *Firmware: BIOS*

2. Application Software

- *Used by users to perform specific tasks.*
- *Installed on the system as per need.*
- *Easy to use and task-oriented.*
- *Examples:*
    - *MS Word (Document writing)*
    - *Chrome (Internet browsing)*
    - *VLC (Video playing)*
    - *Photoshop (Image editing)*

3. Programming Software

- *Used by programmers/developers.*
- *Helps in writing, testing, and debugging code.*
- *Required to create other software.*
- *Examples:*
    - *Compilers: GCC, javac*
    - *Text Editors: VS Code, Notepad++*
    - *Debuggers: GDB*
    - *Interpreters: Python, Node.js*

4. Middleware (Optional/Advanced)

- *Acts as a bridge between two software systems.*
- *Used in complex or enterprise applications.*
- *Helps different programs communicate.*
- *Examples:*
  - *API (Application Programming Interface)*
  - *Web Server Middleware (Apache)*
  - *Database Middleware (ODBC, JDBC)*

34. What are the differences between open-source and proprietary software? GIT and GITHUB Training

Feature	***Open-Source Software***	***Proprietary Software***
Source Code	*Publicly available*	*Hidden from users*
Usage Rights	*Free to use, modify, and share*	*Restricted by license*
Cost	*Usually free*	*Usually paid or subscription-based*
Customization	*Can be customized by anyone*	*Only company/developer can customize*
Examples	*Linux, LibreOffice, GIMP*	*Windows, MS Office, Adobe Photoshop*
Support	*Community-based support (forums, volunteers)*	*Official support from the company*
License	*Open licenses (e.g., GPL, MIT)*	*Commercial licenses*
Security & Bugs	*Bugs fixed by global developers quickly*	*Fixes depend on company's schedule*

35. GIT and GITHUB Training

1. What is Git?

- ***Git*** *is a* ***version control system (VCS)****.*

- *It helps track changes in your code over time.*
- *You can* ***go back*** *to previous versions of your project.*
- *Useful for* ***individual*** *and* ***team projects****.*

Key Features of Git:

- *Local version control (no internet needed).*
- *Branching and merging support.*
- *Keeps history of all changes.*
- *Allows undo and rollback.*

2. What is GitHub?

- ***GitHub*** *is an* ***online platform*** *to host Git repositories.*
- *Provides a* ***cloud-based*** *place to store and manage your code.*
- *Helps in* ***team collaboration****,* ***project sharing****, and* ***open-source contributions****.*

3. Basic Git Commands:

Command	***Use***
git init	*Start a new Git repository*
git clone [url]	*Copy a repository from GitHub*
git status	*Check status of files*
git add .	*Stage all changes*
git commit -m "Message"	*Save changes with a message*
git push	*Upload code to GitHub*
git pull	*Get the latest code from GitHub*
git log	*View commit history*

4. How Git and GitHub Work Together:

1. *You* ***write code*** *on your computer.*
2. *Use* ***Git*** *to save versions.*
3. *Use* ***GitHub*** *to upload code online.*
4. *Others can* ***view, suggest, and collaborate*** *on your code.*

5. Training Topics Covered:

- *Introduction to Git and GitHub*
- *Installing Git*
- *Creating repositories*
- *Committing and pushing code*
- *Branching and merging*
- *Handling merge conflicts*
- *Collaborating using GitHub*
- *Pull requests and reviews*
- *GitHub Pages (for hosting websites)*

6. Benefits of Git & GitHub Training for Students:

- *Helps in managing academic and personal coding projects.*
- *Encourages collaborative teamwork.*
- *Builds portfolio by sharing public repositories.*
- *Required skill for software developers and open-source contributors.*

36. How does GIT improve collaboration in a software development team?

1. Team Members Work Together Smoothly

- *Git allows **many developers** to work on the **same project** at the same time.*
- *Each person can work **independently** without breaking others' work.*

2. Tracks All Changes

- *Git **remembers** who changed what and when.*
- *This helps in **tracking bugs** and understanding updates.*

3. Branches for Different Features

- *Developers create **branches** to try new features or fix bugs.*

- *Once tested, the branch is **merged** into the main project.*

4. Avoids Code Conflicts

- *Git alerts when **two people edit the same file**.*
- *Developers can then **review and fix conflicts** easily.*

5. History and Backup

- *Git stores **full history** of the project.*
- *You can **go back** to older versions if something goes wrong.*

6. Works Offline

- *Developers can **commit and work locally** without the internet.*
- *When online, they can **push** changes to the central repository.*

7. Better Communication

- *Using GitHub or GitLab, team members can:*
  - *Review each other's code (pull requests)*
  - *Add comments*
  - *Suggest changes*
  - *Approve code before merging*

Example:

- *Alice makes a new login feature on login-branch.*
- *Bob fixes a bug on bug-fix-branch.*
- *Both push their branches.*
- *The team reviews them.*
- *After testing, both are merged into the main branch.*

37. Application Software

What is Application Software?

- *It is a type of software that **helps users do tasks** on a computer.*
- *It works based on the **user's input**.*
- *It helps to **automate** the work.*

Main Features:

- *It can do **one or more tasks** at the same time.*
  (Example: You can write in Word and listen to music together.)
- *There are **many applications** that we use in daily life.*
  (Example: Using WhatsApp, MS Word, Google Chrome, etc.)
- *These apps **follow rules** to complete tasks properly.*
  (Example: Calculator follows rules of math.)
- *It gives a **Graphical User Interface (GUI)** –*
  So users can use buttons, menus, icons easily.
- *People use it for:*
  - *Browsing the internet*
  - *Checking email*
  - *Attending online meetings*
  - *Watching videos or playing games*
- *Application software is made using **high-level programming languages***
  like Java, Python, C++, etc.

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

1. Helps in Daily Business Work

- *Application software is used for tasks like **billing, writing reports, sending emails, and data entry**.*

- *It makes office work easier and faster.*

2. Saves Time and Increases Speed

- *It can do tasks quickly and accurately.*

- *It helps reduce manual work and human error.*

3. Manages Business Data

- *Software like Excel or database tools are used to **store, manage, and analyze data**.*
- *Helps in keeping records safe and organized.*

4. Improves Communication

- *Tools like **Zoom, Microsoft Teams, and Gmail** are used for **meetings, chatting, and sharing information**.*
- *Teams can work together even from different locations.*

5. Handles Accounts and Finance

- *Software like **Tally, QuickBooks** is used to manage **accounts, salaries, invoices, and taxes**.*
- *Helps businesses keep track of money.*

6. Better Customer Service

- *Software helps keep **customer records** and solve their issues faster.*
- *CRM (Customer Relationship Management) tools are commonly used.*

7. Helps in Sales and Marketing

- *Applications are used for **creating advertisements, sending emails, and tracking sales**.*
- *Helps promote products and connect with customers.*

39. Software Development Process

➢ *Software development is the process of designing, building, testing, and maintaining software applications. It follows a series of steps to make sure the final product works correctly and meets the user's needs.*

Key Phases of the Software Development Process:

1. ***Planning and Requirement Gathering:** This phase involves identifying the project goals, defining the scope of the software, and gathering requirements from stakeholders.*

2. ***Analysis:*** *Analysing the gathered requirements to understand the functionality, features, and performance needs of the software.*

3. ***Design:*** *Creating the architectural blueprint of the software, including the user interface, data structures, and algorithms.*

4. ***Implementation (Coding):*** *Writing the code for the software based on the design specifications.*

5. ***Testing:*** *Thoroughly testing the software to identify and fix bugs, ensuring it meets the defined requirements and performs as expected.*
6. ***Deployment:*** *Releasing the tested software to the end users.*

7. ***Maintenance:*** *Providing ongoing support and updates to the software, addressing issues, and incorporating new features.*

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

➢ *The **Software Development Life Cycle (SDLC)** is a step-by-step process used to create software. It helps developers make software in a planned, organized, and efficient way.*

1. Planning

- *Decide **what to build**, **why** it is needed, and **how** it will be done.*

- *Set goals, make a schedule, and choose tools.*

Example: Plan to make a school management system.

2. Requirements Analysis

- *Talk to users and find out **what features** they want.*

- *Understand all **needs and expectations**.*

Example: Users want attendance tracking, fee records, and student reports.

3. Design

- *Create the **structure of the software**.*

- *Design the look (UI) and how the system will work inside (architecture).*

Example: Make screen layouts for login, dashboard, and student list.

4. Coding

- *Developers **write the actual code** using programming languages like Java, Python, etc.*
- *This is where the software is built.*

Example: Write code for login system using Python.

5. Testing

- *Check the software for **bugs (errors)**.*
- *Make sure everything works correctly.*

Example: Test if login works properly or not.

6. Deployment

- *Release the finished software for **real users**.*
- *Install or host it on servers if needed.*

Example: Upload school system to the school's website or computers.

7. Maintenance

- ***Fix bugs**, add **new features**, and give **updates** after the software is released.*
- *Keep the software working smoothly.*

Example: Add a new feature to print student ID cards.

41. Software Requirement

1. Functional Requirements

*These describe **what the system should do**.*
*They include the **functions, features, and actions** the software must perform.*

Examples:

- *The system shall allow users to **log in** using a username and password.*
- *The application shall **generate a sales report** at the end of each month.*

- *The system shall* ***send a confirmation email*** *after a successful registration*

2. Non-Functional Requirements (NFRs)

These describe ***how the system should perform****.*
They focus on ***quality, speed, reliability****, and other performance factors.*

Examples:

- *The system shall* ***respond within 2 seconds*** *after the user clicks.*
- *The software must be* ***available 99.9% of the time****.*
- *The user interface shall* ***support both English and Spanish****.*

42. Why is the requirement analysis phase critical in software development? Software Analysis

- ***Requirement Analysis*** *is the process of* ***understanding*** *what the users or clients need from the software.*
  It involves gathering, analyzing, and writing down all the requirements in a clear and complete way.

1. Clear Understanding of the Project

It helps developers and clients agree on ***what the software should do*** *before starting.*
Example: Whether the app needs login, email, or report generation — all is decided here.

2. Avoids Confusion and Mistakes

With clear requirements, the chances of ***building the wrong features*** *or* ***missing key parts*** *are reduced.*

3. Better Planning and Design

It helps the development team ***plan the project*** *better, choose the right tools, and design the system properly.*

4. Saves Time and Money

Finding mistakes ***early*** *(in the planning stage) is* ***cheaper*** *than fixing them* ***later*** *(after coding).*

5. Improves Communication

*It creates a clear document for **everyone involved** — developers, testers, clients, and project managers.*

43. Software Analysis

1. What is Software Analysis?

- *It is the process of understanding what a software system should do.*
- *It helps in designing the system to meet user needs.*
- *It checks if the software works correctly and meets the goals.*

2. Purpose and Scope

- ***Understand Requirements:***
  Gather what users or clients want from the software.
- ***System Design & Architecture:***
  Plan the system structure and how its parts will work.
- ***Verification & Validation:***
  Make sure the software is doing the right job and works correctly.
- ***Improve Quality:***
  Focus on making the software reliable, secure, and high-performing.

3. Key Activities in Software Analysis

1. ***Requirement Gathering:***
   Understand what the software must do (functional) and how it must do it (non-functional).
2. ***System Modeling:***
   Draw diagrams like flowcharts and UML to visualize how the system works.
3. ***Software Design:***
   Plan how different parts of the software will be built (UI, database, logic).
4. ***Documentation:***
   Write everything clearly for future reference and maintenance.
5. ***Analysis:***
   Check for design problems, performance issues, and areas of improvement.

4. Techniques and Tools

- ***Static Analysis:***
  Check code without running it – find errors, bugs, and bad code.
- ***Dynamic Analysis:***
  Run the software and watch how it behaves during execution.

- ***Software Composition Analysis (SCA):***
  Check if any open-source code is used, and if it’s safe and legal.
- ***Tools Used:***
  - *Visual Paradigm, Lucidchart (for diagrams)*
  - *SonarQube, Fortify (for code analysis)*
  - *Microsoft Visio, StarUML (for modeling)*

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

Software analysis *is the foundation of successful software development. It helps in understanding requirements, reducing risks, and guiding the project from start to finish.*

1. Requirements Gathering and Clarification

- *Understand what users and stakeholders need.*
- *Document clear and complete functional and non-functional requirements.*

2. Feasibility Study

- *Check if the project is possible technically, financially, and legally.*
- *Helps decide whether to proceed and define project scope.*

3. Problem Definition and Modeling

- *Use diagrams (like DFD, ER, Use Case) to visualize the system.*
- *Helps in better understanding and communication.*

4. Risk Analysis

- *Identify possible technical, financial, or operational risks.*
- *Plan ways to reduce or avoid these risks.*

5. System Specification

- *Create detailed Software Requirement Specifications (SRS).*
- *Acts as a blueprint for designers and developers.*

6. Validation and Verification

- *Ensure requirements are correct and meet user expectations.*
- *Use walkthroughs, reviews, or prototypes.*

7. Support for Design and Development

- *Analysis guides the design, coding, and testing phases.*
- *Reduces confusion and rework during later stages.*

45. System Design

1. What is System Design?

System design is the process of planning the architecture of software to meet functional and non-functional requirements.

- ***High-Level Design (HLD):***
  Focuses on system architecture and major components.
- ***Low-Level Design (LLD):***
  Deals with detailed classes, database schema, and APIs.

2. Key Concepts:

- ***Scalability:*** *Horizontal vs. vertical scaling*
- ***Load Balancing:*** *Distribute traffic evenly*
- ***Caching:*** *Speed up data access*
- ***Database Design:*** *SQL vs. NoSQL*
- ***Sharding & Partitioning:*** *Break large databases into parts*
- ***Asynchronous Processing:*** *Use message queues*
- ***Availability & Fault Tolerance:*** *Keep system running during failures*
- ***CAP Theorem:*** *Balance between Consistency, Availability, Partition Tolerance*
- ***Consistent Hashing:*** *For load distribution*
- ***CDNs & Proxies:*** *Speed up content delivery*

3. Practice System Examples:

- *URL Shortener*
- *Twitter Feed*
- *WhatsApp Messaging*
- *YouTube/Netflix Streaming*
- *Uber/Lyft*
- *Google Drive/Dropbox*
- *Amazon Recommendations*

4. Tools and Diagrams:

- *Sequence Diagrams*
- *Component Diagrams*
- *ER Diagrams*
- *Load Balancing Diagrams*
- *Deployment Diagrams*

5. Tech Stack Examples:

- ***Backend:*** *Node.js, Go, Java*
- ***Databases:*** *PostgreSQL, MongoDB, Redis*
- ***Messaging:*** *Kafka, RabbitMQ*
- ***Infrastructure:*** *AWS, GCP, Kubernetes, Docker*
- ***Monitoring:*** *Prometheus, Grafana*

46. What are the key elements of system design?

System design *is the process of planning the architecture and components of a system to meet functional and non-functional requirements.*

1. Requirements Analysis

- ***Functional:*** *Features and use cases.*
- ***Non-functional:*** *Performance, scalability, security, availability.*

2. High-Level Design (HLD)

- *Choose architecture (monolith, microservices, event-driven).*
- *Break system into major components.*
- *Select tech stack (databases, languages).*
- *Define data flow (REST, gRPC, message queues).*

3. Low-Level Design (LLD)

- *Design classes, objects, and methods.*
- *Create database schema (tables, relationships).*
- *Define API formats and endpoints.*

4. Scalability & Performance

- *Scale horizontally or vertically.*
- *Use caching (Redis, Memcached).*
- *Apply load balancing to handle traffic.*

5. Reliability & Fault Tolerance

- *Use backups, replication, and failover.*
- *Implement retries for failures.*
- *Monitor and alert for issues.*

6. Security

- *Use authentication (OAuth, JWT) and authorization (RBAC).*
- *Encrypt data (TLS, AES).*
- *Validate inputs and rate-limit users.*

7. Maintainability & Extensibility

- *Design modular systems.*
- *Write clean, well-documented code.*
- *Apply unit, integration, and E2E testing.*

8. DevOps & Deployment

- *Automate CI/CD pipelines.*
- *Use Docker and Kubernetes.*
- *Manage different environments (dev, test, prod).*

9. Monitoring & Logging

- *Collect metrics (Prometheus, Grafana).*
- *Centralize logs (ELK, Fluentd).*
- *Set up real-time alerts.*

47. Software Testing

1. What is Software Testing?

Software testing is the process of checking whether a software application works correctly and meets the required specifications.
It helps identify bugs or issues before the product is released.

2. Objectives of Software Testing

- *To find and fix errors in the software*

- *To ensure the software behaves as expected*
- *To check software quality, performance, and security*
- *To improve user satisfaction*
- *To reduce the cost of fixing problems later*

3. Types of Software Testing

A. Based on Execution

- ***Manual Testing****: Done by humans without using tools.*
- ***Automation Testing****: Done using software tools and scripts (e.g., Selenium).*

B. Based on Knowledge of Code

- ***White Box Testing****: Tester knows the internal code and logic.*
- ***Black Box Testing****: Tester only checks functionality without seeing code.*
- ***Gray Box Testing****: Partial knowledge of internal code.*

C. Based on Purpose

- ***Functional Testing****: Checks what the software does (features).*
- ***Non-Functional Testing****: Checks how the software behaves (speed, security, etc.).*

4. Common Software Testing Types

- ***Unit Testing****: Testing individual units or functions.*
- ***Integration Testing****: Checking interaction between modules.*
- ***System Testing****: Testing the complete system.*
- ***Acceptance Testing****: Done by users to approve the final product.*
- ***Regression Testing****: Ensures new changes don't break existing features.*
- ***Performance Testing****: Tests speed and responsiveness.*
- ***Security Testing****: Finds security issues or risks.*

5. Software Testing Life Cycle (STLC)

1. *Requirement Analysis*
2. *Test Planning*
3. *Test Case Design*
4. *Test Environment Setup*
5. *Test Execution*
6. *Defect Reporting*
7. *Test Closure*

6. Benefits of Software Testing

- *Improves software quality*
- *Reduces development cost*
- *Increases customer satisfaction*
- *Helps deliver bug-free products*
- *Ensures reliability and performance*

7. Popular Testing Tools

- *Selenium (Automation)*
- *JUnit (Java Unit Testing)*
- *Postman (API Testing)*
- *TestNG*
- *LoadRunner (Performance Testing)*

8. Conclusion

Software testing is a critical part of software development.
It helps ensure that the final product is reliable, secure, and works as expected

48. Why is software testing important?

1. Detects Bugs and Errors Early

- *Helps find and fix problems before the software reaches users.*
- *Prevents future failures and crashes.*

2. Ensures Software Quality

- *Validates that the software meets functional and non-functional requirements.*
- *Confirms that it works as expected across different devices and environments.*

3. Improves Security

- *Identifies vulnerabilities that hackers could exploit.*
- *Prevents data leaks and security breaches.*

4. Saves Time and Money

- *Fixing issues early is cheaper than post-release fixes.*

- *Reduces maintenance costs.*

5. Builds User Confidence

- *Reliable software leads to satisfied users.*
- *Builds trust in your brand or product.*

6. Verifies Performance

- *Tests speed, scalability, and responsiveness under various conditions.*
- *Ensures software performs well under load.*

7. Compliance and Standards

- *Ensures the software meets industry or legal regulations.*
- *Especially important in sectors like finance, healthcare, and aviation.*

49. Maintenance

- ***Maintenance** is the process of keeping something in good working condition by regularly checking, repairing, servicing, or replacing its parts when necessary. The goal is to prevent failure, extend the lifespan, and ensure optimal performance of an asset, system, or equipment*

Common Areas Where Maintenance Is Applied:

- ***Buildings and Facilities** (e.g., electrical systems, plumbing)*
- ***Vehicles** (cars, planes, ships)*
- ***Manufacturing Equipment***
- ***IT and Software Systems***
- ***Infrastructure** (roads, bridges)*

50. What types of software maintenance are there?

*There are **four main types of software maintenance**, each serving a different purpose in the software lifecycle:*

1. Corrective Maintenance

- ***Purpose:** Fix bugs or errors found after the software is released.*
- ***Example:** Fixing a crash or wrong calculation in a billing system.*

2. Adaptive Maintenance

- ***Purpose:*** *Update software to work with new hardware, operating systems, or technologies.*
- ***Example:*** *Modifying software to run on Windows 11 or with a new database.*

3. Perfective Maintenance

- ***Purpose:*** *Improve performance, user interface, or add new features.*
- ***Example:*** *Enhancing loading speed or adding a dark mode.*

4. Preventive Maintenance

- ***Purpose:*** *Make changes to prevent future issues or make the code easier to maintain.*
- ***Example:*** *Refactoring code to reduce complexity or upgrading libraries to avoid future bugs.*

51. Development

Development *means progress, growth, or improvement in different areas. It involves moving from a basic state to a more advanced one.*

Types of Development (Short Notes):

1. ***Economic Development***
   – Growth in wealth, jobs, education, and healthcare.
2. ***Human Development***
   – Improving quality of life (health, education, income).
3. ***Personal Development***
   – Self-improvement in skills, mindset, and confidence.
4. ***Software/Web Development***
   – Building and maintaining apps or websites (front-end, back-end).
5. ***Urban/Real Estate Development***
   – Construction of buildings, roads, and public spaces.
6. ***Child Development***
   – Growth of children in physical, mental, and emotional aspects.
7. ***Business Development***
   – Expanding business through strategy, partnerships, and sales.

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

Feature	***Web Application***	***Desktop Application***
Installation	*Runs in a browser, no installation needed*	*Requires installation on the user's device*
Accessibility	*Accessible from anywhere with internet*	*Only accessible on the device where installed*
Internet Requirement	*Requires internet connection (mostly)*	*Can work offline (unless cloud-based)*
Updates	*Easy to update centrally on the server*	*Must be updated on each user’s device individually*
Performance	*Depends on internet and browser performance*	*Usually faster and more powerful*
Security	*Data is stored on the server; needs strong web security*	*Data may be more secure if stored locally*
Platform Dependency	*Platform-independent (runs on any OS with a browser)*	*May be platform-dependent (Windows, macOS, etc.)*
Development Tools	*HTML, CSS, JavaScript, PHP, etc.*	*Java, C++, .NET, Python, etc.*

53. Web Application

*A **web application** is a software program that runs in a web browser. You don’t need to install it on your computer you just open it using the internet.*

Key Features of Web Applications:

- ***Runs in Browsers**: Like Chrome, Firefox, or Edge.*
- ***Needs Internet**: Most web apps require an internet connection.*
- ***No Installation Needed**: Just visit a website (e.g., www.gmail.com).*
- ***Accessible Anywhere**: You can use it from any device with internet access.*
- ***Central Updates**: Developers can update the app on the server side; no user update needed.*

Examples of Web Applications:

- *Gmail*
- *Facebook*
- *Google Docs*
- *YouTube*
- *Online Banking*

Technologies Used:

-
- ***Front-End**: HTML, CSS, JavaScript*
- ***Back-End**: PHP, Python, Node.js, Java*
- ***Databases**: MySQL, MongoDB, PostgreSQL*

Advantages:

- *Easy to access and use*
- *Works across devices (mobile, tablet, PC)*
- *Always up to date*

Disadvantages:

- *Needs a stable internet connection*
- *May be slower than desktop apps*
- *Security depends on the server*

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

*In today's digital world, software applications are essential for various tasks. These applications can be either **web-based** or **desktop-based**. Web applications are programs that run in a web browser, while desktop applications are installed on a local computer. This assignment explains the **advantages of using web applications** over desktop applications.*

Advantages of Web Applications

1. Accessibility

*Web applications can be accessed from **any device** (computer, mobile, tablet) with an internet connection and browser. There is **no need to install** any software.*

2. Platform Independence

*Web apps work on **any operating system** (Windows, Mac, Linux). You just need a browser like Chrome or Firefox.*

3. Automatic Updates

*Web apps are **updated automatically** on the server. Users always use the **latest version** without installing updates.*

4. Low Installation Cost

*Users don't have to download or install anything. This **saves time and space** on the computer.*

5. Easy Maintenance

*Developers can easily fix bugs and update the app from one central server. There's **no need to update each user's system** individually.*

6. Real-Time Collaboration

*Web apps like Google Docs allow **multiple users to work at the same time**, making teamwork easier.*

7. Cloud Storage

*Web apps store data on the **cloud**, which means users can **access their files from anywhere** and don't risk losing data if a device crashes.*

8. Scalability

*Web applications are easier to **scale up** for large numbers of users without installing anything extra.*

55. Designing

***Designing** is the process of **planning and creating something with a specific purpose or function in mind**. It involves combining creativity, problem-solving, and technical skills to make something that is both useful and appealing.*

Key Aspects of Designing:

1. ***Intentionality** – You create something on purpose, with a goal (e.g., solving a problem, communicating an idea, or enhancing usability).*
2. ***Creativity** – You use imagination and innovation to develop ideas.*
3. ***Functionality & Aesthetics** – A good design works well and looks good.*

Types of Designing:

- ***Graphic Design*** *– Creating visuals like logos, posters, branding materials.*
- ***Web/App Design (UI/UX)*** *– Designing user interfaces and experiences for digital products.*
- ***Fashion Design*** *– Creating clothing and accessories.*
- ***Interior Design*** *– Planning and decorating indoor spaces.*
- ***Industrial/Product Design*** *– Designing physical products like furniture, electronics, tools.*
- ***Architectural Design*** *– Planning buildings and structures.*

Steps in a Typical Design Process:

1. ***Research/Understand the Problem***
2. ***Brainstorm and Generate Ideas***
3. ***Create Sketches or Prototypes***
4. ***Test and Refine***
5. ***Finalize and Deliver the Design***

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

*UI/UX design plays a **crucial role** in application development by directly affecting user satisfaction, engagement, and product success.*

1. ***Enhances User Satisfaction***

   – UI focuses on visual design; UX ensures ease of use.

   – Good design improves retention and user feedback.
2. ***Improves Usability***

   – Makes apps intuitive, reducing errors and the learning curve.
3. ***Drives Business Goals***

   – Boosts conversions, brand image, and customer loyalty.
4. ***Reduces Development Costs***

   – Early design/testing finds issues sooner, saving time and money.
5. ***Supports Accessibility***

   – Inclusive design ensures usability for all, expanding reach.
6. ***Differentiates from Competitors***

   – Strong UI/UX helps apps stand out in a competitive market.

57. Mobile Application

- ➢ *A **Mobile Application** (or **mobile app**) is a type of software program specifically designed to run on **mobile devices**, such as **smartphones** and **tablets**. These apps are developed for various operating systems, the most common being:*

- ***Android** (developed by Google)*
- ***iOS** (developed by Apple)*

Key Characteristics of Mobile Apps:

1. ***Platform-Specific**: Many apps are built specifically for either Android or iOS, though cross-platform development is also common.*
2. ***Touch Interface**: Designed for touchscreen interaction (tapping, swiping, pinching).*
3. ***Portability**: Allows users to access services and features on the go.*
4. ***Access to Device Features**: Mobile apps can use device hardware like the camera, GPS, microphone, and sensors.*
5. ***Distribution**: Usually downloaded from app stores:*
    - ***Google Play Store** (Android)*
    - ***Apple App Store** (iOS)* ***Types of Mobile***

Applications:

1. ***Native Apps**: Built specifically for one platform using languages like Swift (iOS) or Kotlin/Java (Android).*
2. ***Hybrid Apps**: Built using web technologies (like HTML, CSS, and JavaScript) and wrapped in a native shell.*
3. ***Web Apps**: Mobile-optimized websites that behave like apps but run in a browser.*

Examples of Mobile Apps:

- ***Social Media**: Instagram, Facebook, TikTok*
- ***Messaging**: WhatsApp, Telegram*
- ***Banking**: PayPal, Venmo, mobile banking apps*
- ***Productivity**: Microsoft Office, Google Docs*

- ***Games**: Candy Crush, PUBG Mobile, Clash of Clans*

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

***Native Apps**:*

1. *Developed for a specific platform (Android or iOS).*
2. *Uses platform-specific languages like Java/Kotlin (Android), Swift (iOS).*
3. *Offers high performance and smooth user experience.*
4. *Full access to device features (camera, GPS, sensors, etc.).*
5. *Follows platform UI/UX design guidelines closely.*
6. *Requires separate codebases for each platform.*
7. *Longer development time and higher cost.*
8. *Better suited for complex or performance-heavy apps.*

Hybrid Apps:

1. *Built using web technologies like HTML, CSS, JavaScript.*
2. *Runs inside a native shell (using frameworks like React Native, Ionic).*
3. *Slower performance compared to native apps.*
4. *Limited or plugin-based access to device features.*
5. *One codebase works for both Android and iOS.*
6. *Faster development and lower cost.*
7. *May not match native UI perfectly.*
8. *Suitable for simpler or content-based apps.*

59. DFD (Data Flow Diagram)

*A **DFD (Data Flow Diagram)** is a visual representation of how data flows through a system. It helps to understand how inputs are transformed into outputs through processes, data stores, and external entities.*

Key Components of a DFD:

1. ***External Entity (Source/Sink)**:*

   - *Represents outside systems or users that interact with the system.*
   - *Symbol: **Rectangle** o Example: Customer, Bank, Government Agency*

2. ***Process**:*

   - *Represents operations that transform data. o Symbol: **Circle** or **Rounded rectangle** o Example: "Verify Login", "Generate Invoice"*

3. ***Data Store**:*

   - *Represents where data is stored for later use. o Symbol: **Open-ended rectangle** (like two parallel lines) o Example: "User Database", "Product Inventory"*

4. ***Data Flow**:*

   - *Represents the movement of data between components.*
   - *Symbol: **Arrow** o Labelled with the name of the data moving through the system.*

Levels of DFDs:

- ***Level 0 (Context Diagram)**:*

  Shows the system as a single process and how it interacts with external entities.

- ***Level 1 DFD**:*

  Breaks the main process into sub-processes to show internal operations.

- ***Level 2 (and beyond)**:*

  Further decomposes Level 1 processes into more detailed subprocesses.

Example: Online Shopping System (Level 0)

***Entities**:*

- *Customer*
- *Payment Gateway*

***Process**:*

- *"Online Shopping System"*

***Data Flows**:*

- *Order Information, Payment Details*

Scenario: Online Library Management System (Level 0 DFD – Context Diagram)

Components:

- ***External Entities**: o Student o Librarian*
- ***Main Process**: o Library Management System*
- ***Data Flows**:*
  - *Book Request, Issue Confirmation, Book Return, User Info*
- ***Data Stores**: (Not shown in Level 0) o Shown in Level 1 and beyond*

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

1. Visual Representation of the System

- *DFDs provide a clear and concise graphical representation of how data moves through a system.*
- *They illustrate the flow of information between processes, data stores, external entities, and data flows.*

2. ***Improved Communication***

- *DFDs act as a communication tool between system analysts, developers, and non-technical stakeholders (e.g., clients or end-users).*
- *They help ensure everyone has a shared understanding of the system's functionality.*

3. ***System Decomposition***

- *DFDs support **top-down decomposition**, allowing analysts to break down complex systems into simpler, manageable parts.*
- *This helps identify system components, functions, and their interactions in a structured manner.*

4. ***Requirement Analysis***

- *By showing how data is input, processed, stored, and output, DFDs help in identifying and validating system requirements.*
- *They aid in spotting redundancies, inefficiencies, or missing elements in data processing.*

5. ***Documentation***

- *DFDs serve as part of the system documentation, useful for future maintenance and upgrades.*
- *They offer a stable reference model for understanding the system's data processes.*

6. ***Facilitates System Design***

- *In the design phase, DFDs assist in translating requirements into logical and physical designs.*
- *They provide a foundation for database design, user interface planning, and process specification.*

7. ***Error Identification***

- *DFDs make it easier to detect logical errors or incomplete processes in the system before development begins.*

61. Desktop Application

Definition:

*A **Desktop Application** is a software program that runs on a **personal computer (PC)** or **laptop**, installed directly on the operating system (e.g., Windows, macOS, Linux).*

Key Features:

1. ***Platform Dependent***
   – Designed for specific operating systems.

2. ***Offline Access***
   – Works without an internet connection (in most cases).

3. ***High Performance***
   – Can handle complex tasks like video editing, programming, etc.

4. ***Installed Locally***
   – Requires download and installation on the user’s system.

5. ***Direct Access to Hardware***
   – Can use system resources like RAM, processor, and storage efficiently.

Examples of Desktop Applications:

- *Microsoft Word, Excel, PowerPoint*
- *Adobe Photoshop, Illustrator*
- *VLC Media Player*
- *Visual Studio, NetBeans*
- *Tally ERP*

Advantages:

- *Fast performance*
- *Better control over system resources*
- *Works without internet*

Disadvantages:

- *Must be installed separately on each device*
- *Limited to the machine it is installed on*

- *Updates are manual (in many cases)*

62. What are the pros and cons of desktop applications compared to web applications?

Pros of Desktop Applications:

1. ***High Performance:***
   – Faster and more powerful, ideal for heavy tasks (e.g., video editing, coding).

2. ***Offline Access:***
   – Can work without an internet connection.

3. ***Full Access to Hardware:***
   – Better control over RAM, CPU, storage, printer, etc.

4. ***Security Control:***
   – Data is stored locally, offering more control (if secured properly).

Cons of Desktop Applications:

1. ***Platform Dependent:***
   – Separate versions are needed for Windows, macOS, etc.

2. ***Manual Installation & Updates:***
   – Must be installed and updated on each machine individually.

3. ***Limited Accessibility:***
   – Only works on the computer it's installed on.

4. ***Higher Storage Requirement:***
   – Takes up local storage space.

Pros of Web Applications:

1. ***Accessible Anywhere:***
   – Runs on any device with a browser and internet.

2. ***No Installation Needed:***
   – Open in browser, saving time and storage.

3. ***Automatic Updates:***
   – Always up to date for all users.

4. ***Cross-Platform Compatibility:***
   – Works across Windows, macOS, Linux, mobile, etc.

Cons of Web Applications:

1. ***Requires Internet:***
   – Most web apps need a stable connection to work.

2. ***Limited Hardware Access:***
   – Can't use system resources fully (e.g., for intensive tasks).

3. ***Slower Performance:***
   – Depends on browser speed and network.

4. ***Security Risks:***
   – Data is stored online, which may be vulnerable if not properly secured.

63. Flow Chart

Definition:

*A **Flow Chart** is a **diagram** that visually represents the **step-by-step flow of a process or system** using **symbols** connected by **arrows**. It helps in understanding how a process works from start to finish.*

Purpose of a Flow Chart:

- *To **visualize logic or workflows***
- *To **identify problems or bottlenecks***
- *To **document, design, or improve processes***
- *Useful in **programming**, **business processes**, and **system design***

Advantages of Flow Charts:

- *Easy to understand*
- *Helps in identifying errors or inefficiencies*
- *Makes communication and documentation clearer*

Disadvantages:

- *Can become complex for large systems*

- *Not suitable for all types of analysis*
- *Requires updating when processes change*

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

1. Visualizing Logic Clearly

- *Flowcharts show the **step-by-step logic** of a program.*
- *Makes it easier to understand how data and decisions flow.*

2. Easy to Plan Before Coding

- *Helps programmers **plan the structure** of code before writing it.*
- *Reduces confusion and guides the coding process.*

3. Improves Communication

- *Non-programmers (like clients or team members) can also understand the system flow.*
- *Useful in **team collaboration** and **presentations**.*

4. Identifies Errors and Loopholes

- *By tracing the flow, logic errors or missing steps can be **spotted early**.*
- *Helps in debugging and refining algorithms.*

5. Aids in System Design

- *Helps system designers **map out processes**, data flow, and decision points.*
- *Useful for creating **DFDs**, **algorithms**, or **process diagrams**.*

6. Saves Time in Development

- *A well-drawn flowchart provides a **blueprint** for the final program.*
- *Prevents unnecessary coding and rework.*

7. Helpful in Documentation

- *Flowcharts serve as **technical documentation** for future reference and mainte*