**Java – Full stack Assignment**

**Module 1 – Overview of IT Industry**

1. **What is a program?**

* A program is a set of instructions written in a programming language that a computer can understand and execute to perform specific task.

*Simple Definition:* A program is a set of instruction written in a programming language that tells a computer what to do.

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

* C program:

|  |
| --- |
| #include<stdio.h>  main(){  printf("Hello Word");  } |

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

* A program is a set of instructions written in a programming language that tells a computer what to do. It is like given commands step-by-step. The computer reads the program line by line and performs the actions written inside it. Programs are used to solve problems, do calculation, control devices, or create software like games, websites, and apps.

1. **What is Programming?**

* Programming is the process of giving instruction to a computer so it can perform specific task.

*Simple Definition:* Programming means writing code (using a programming language like C, Python, or Java ) to make a computer do what you want.

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

* **Key Steps in programming process:**

1. **Planning & Analysis:**

Define goals, scope, feasibility, and requirements

1. **Design:**

Create system architecture, data models, UI/UX, and prepare design specification.

1. **Implementation (Coading):**

Write code based on design document, following standards and using version control.

1. **Testing:**

Perform unit, integration, system, and acceptance testing to identify and fix defects.

1. **Development:**

Release the software to users-via production environment or live servers.

1. **Maintenance:**

Continuously update, fix bugs, and improve based on feedback.

1. **Type of Programming language?**

* Procedural Programming : C Language.
* Object Oriented Programming : C++ Language.
* Logical Programming : Prolog Language.
* Functional Programming : Python Language.

**THEORY EXERCISE: What are the main differences between high-level and low-level language in programming.**

* **Differences between High-Level and Low-Level Programming Languages are,**
* **Meaning**:
  + High-level languages are closer to human languages and easy to understand.
  + Low-level languages are closer to machine language and difficult for humans to understand.
* **Ease of Use**:
  + High-level languages are easy to read, write, and debug.
  + Low-level languages are harder to write and understand.
* **Hardware Control**:
  + High-level languages give less control over hardware.
  + Low-level languages give more control over hardware.
* **Speed of Execution**:
  + High-level languages are slower because they need to be translated by a compiler or interpreter.
  + Low-level languages are faster as they run directly on the hardware.
* **Examples**:
  + High-level: C, C++, Java, Python
  + Low-level: Assembly Language, Machine Code (Binary)
* **Usage**:
  + High-level languages are used for application and software development.
  + Low-level languages are used for system programming like operating systems, device drivers, etc.

**Conclusion**:  
High-level languages are easy for humans but slow for machines.  
Low-level languages are tough for humans but fast and efficient for machines.

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

**World Wide Web (WWW)**

* The **World Wide Web (WWW)** is a collection of website and web pages stored on the **Internet**. You access the web using a **web browser** (like Chrome Firefox, Safari,).
* It’s a **service** that runs on **top of the internet**.
* Accessed using **URLs** (Uniform Resource Locators) like <https://www.google.com>
* The **Web** is just one part of the **Internet** (like apps, email, etc.).

**How the Internet Works**

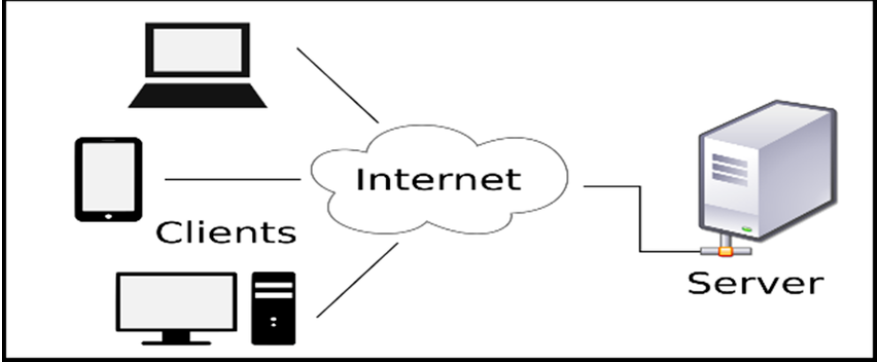
* The **Internet** is a global network of computers connected together to **share data.**

**Internet** is a big network that connects computers worldwide. It lets us send/receive data like web page, videos, massage, etc

1. **You open a browser** and type a website (e.g., www.google.com)
2. **DNS** converts the website name into an **IP address**
3. A **request is sent** to the website’s **server**
4. The **server sends back data** (HTML, images, etc.)
5. Your **browser shows the webpage** on your screen

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

* When a user (Client) sends a request (like opening aa website)
* Appliction Layer (e.g., Brpwser)
* The Clint types URL (like https://www.example.com) in a browser.
* The browser creates an HTTP request.
* DNS Resolution

****

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

* **Clint:**

The client is usually a web browser (like Chrome, Firefox, or Edge) or any app that wants to access information from the internet.

* **Role of the clint:**
  + **Sends a request** to the server (e.g., asking for a web page or data).
  + **Waits for a response** from the server.
  + **Displays** or processes the information received from the server.
* **Server:**

The server is a powerful computer or system that stores, processes, and delivers web content and data to clients.

* + **Role of the Server:**
* **Receives the request** from the client.
* **Processes** the request (e.g., finds the correct web page or performs database operations)
* **Sends back a response**, usually HTML, CSS, JS, or JSON data.

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

* The **TCP/IP** **model** is a **framework** used to explain how data is transmitted over the internet.

It allows different types of Computers and network ti Communication with each other.

It’s Stand For;

**Transmission Control Protocol / Internet Protocol**

* Function:
* **Application Layer** → User-level communication (like using a browser)
* **Transport Layer** → Data safety and order (like parcel tracking)
* **Internet Layer** → Finds destination using IP (like address on parcel)
* **Network Access Layer** → Sends parcel physically (like delivery van)

1. **Application Layer:**

* Closest to user.
* Provides Service like:
* HTTP(Web)
* FTP (File transfer)
* SMTP(Email)
* It prepares the data to be sent.

1. **Transport Layer:**

* **Transport Layer**
* Ensures **reliable delivery** of data
* Protocols:
  + **TCP** (Transmission Control Protocol): Reliable
  + **UDP** (User Datagram Protocol): Fast but less reliable
* Breaks data into **segments** and reorders them at the destination.

1. **Internet Layer:**

* Handles **IP addresses** and **routing**
* Main protocol: **IP (Internet Protocol)**
* Decides how the packet reaches the correct destination.
* **IP (Internet Protocol)** – Assigns logical IP addresses (like 192.168.1.1).

1. **Network Access Layer (Link Layer):**

* Responsible for **actual data transfer** through physical hardware.
* Converts packets into **electrical signals, radio waves, etc.**
* Uses MAC (**Media Access Control**)addresses and device drivers.

A **MAC address** is a **unique ID number** given to every device that connects to a network (like your mobile, laptop, or router).

1. **THEORY EXERCISE: Explain Client Server Communication.**

* Client-server communication is a way for two computers to share information over a network.
* The client sends a request for data or service.
* The server receives the request, processes it, and sends back a response.
* **Example:**

When you open a website:

* Your browser is the **client**.
* The website’s computer is the **server**.
* The browser requests the webpage, and the server sends the webpage data back.

This process uses protocols like **HTTP** and works over the **internet**.

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

**DSL** = via telephone lines  
**Cable** = via TV cables

**Pros:**

* Widely available
* Always on (no need to dial in)
* Faster than dial-up

**Cons:**

* Speed can slow down during peak hours
* DSL depends on distance from provider

**2. Fiber Optic**

High-speed internet using fiber optic cables.

**Pros:**

* Super fast download and upload speeds
* Great for gaming, streaming, large file transfers
* More reliable than copper wires

**Cons:**

* Expensive to install
* Not available in all areas

**3. Satellite**

Uses satellites to provide internet, good for remote areas.

**Pros:**

* Works in rural or remote places
* No need for cables

**Cons:**

* Slower speeds
* High latency (delay)
* Weather can affect signal

**4. Mobile Data (3G, 4G, 5G)**

Internet from mobile networks using SIM cards.

**Pros:**

* Portable — can use anywhere with signal
* 5G offers high speed and low delay

**Cons:**

* Depends on signal strength
* Data plans can be costly
* Speed varies by location

**5. Wi-Fi (Wireless Broadband)**

Local wireless internet from a router.

**Pros:**

* Connect multiple devices
* Easy to use at home or office

**Cons:**

* Signal can be weak in far corners
* Needs proper security (can be hacked)

1. **THEORY EXERCISE: How does broadband differ from fiber-optic internet?**
   * roadband and fiber-optic internet are both high-speed internet connections, but they work differently.

**Broadband** (like DSL or cable) uses **copper wires** to transmit data. It provides good speed and is widely available, but the speed may reduce with distance or during peak usage times.

**Fiber-optic internet** uses **thin glass or plastic fibers** to send data as **light signals**. It offers **much faster** and **more reliable** internet than traditional broadband but may not be available in all areas and can be more expensive to install.

**In short**, fiber-optic is faster, more stable, and better for heavy internet use, while broadband is more common and cheaper but slower.

1. **THEORY EXERCISE: What are the differences between HTTP and HTTPS protocols?**
   * HTTP and HTTPS are both protocols used to transfer data over the internet. The main differences between them are:
2. **Full Form:**
   * HTTP: Hyper Text Transfer Protocol
   * HTTPS: Hyper Text Transfer Protocol Secure
3. **Security:**
   * HTTP is **not secure**.
   * HTTPS is **secure** because it uses **SSL/TLS encryption**.
4. **Data Protection:**
   * HTTP sends data in **plain text**, which can be hacked.
   * HTTPS sends **encrypted data**, making it safe.
5. **Port Number:**
   * HTTP uses **port 80**.
   * HTTPS uses **port 443**.
6. **URL Format:**
   * HTTP URLs start with http://
   * HTTPS URLs start with https://
7. **SSL Certificate:**
   * HTTP does **not** require a certificate.
   * HTTPS requires an **SSL/TLS certificate** to verify the website.
8. **Usage:**
   * HTTP is used for **normal websites**.
   * HTTPS is used for **secure websites** like banking, login pages, etc.
9. **Identify and explain three common application security vulnerabilities. Suggestpossible solutions.**
   * three common application security vulnerabilities along with solutions:
10. **SQL Injection:**

**What happens?**  
Hacker types special code in input boxes (like login) to break into the database.

* **Example:**  
  ' OR '1'='1
* **Fix:**

Never trust user input.

Use **prepared statements** (safe database code).

Check input properly.

**2. Cross-Site Scripting (XSS)**

* **What happens?**  
  Hacker puts bad JavaScript on your website. It runs when someone visits the site. Can steal user data.
* **Fix:**

Don’t show raw user input on screen.

Use **HTML encoding**.

Add **security headers** like CSP.

**3. Broken Authentication**

* **What happens?**  
  Weak login system lets hackers log in as someone else.
* **Fix:**

Use **strong passwords**.

Add **2-step verification (OTP)**.

Use **HTTPS** to keep logins safe.

1. **What is the role of encryption in securing applications?**
   * Encryption is like locking your data with a secret key, so only the right person can unlock and read it.

* **Why Encryption is Important:**

1. Protects Data Privacy:

* Keeps sensitive data (like passwords, credit card numbers) safe from hackers.

1. Secures Data in Transit:

* When data moves between devices (like user to server), encryption hides it.
* Example: HTTPS encrypts your data on websites.

1. Secure Data at Rest:

* Stored data (in database or files) is encrypted, so even if stolen, it’s unreadable.

1. Prevents Unauthorized Access:

* Only users with the correct **key** or **password** can access encrypted data.

1. **Software Applications and Its Types.**
   * A Software application is a program (or group of programs) designed to help users perform specific tasks like writing, drawing, browsing the internet, or managing data.

It runs on top of the operating system (Like Windows, Linux, Android).

* + **Types of Software Applications:**
    1. **Application Software:**
* **Definition:**

Application software is designed to help users perform specific tasks like writing, editing, calculating, browsing, etc.

* **Examples:**  
  MS Word, Google Chrome, Excel, VLC Media Player.

**2. System Software**

* **Definition:**  
  System software helps run the computer hardware and provides a platform for application software to work.
* **Examples:**  
  Windows, Linux, macOS, BIOS**3. Driver Software**
* **Definition:**  
  Driver software allows the operating system to communicate with hardware devices like printers, keyboards, or USB devices.
* **Examples:**  
  Printer driver, Sound driver, Graphics driver (NVIDIA driver).

**4. Middleware Software**

* **Definition:**  
  Middleware software acts as a bridge between system software and application software, or between two applications.
* **Examples:**  
  Database middleware, Web servers, API gateway.

**5. Programming Software**

* **Definition:**  
  Programming software is used by developers to write, test, and debug programs.
* **Examples:**  
  C Compiler, Python IDE, Code::Blocks, Turbo C, Visual Studio.

1. **Identify and classify 5 applications you use daily as either system software or application software.**
   * Below are 5 commonly used applications classified into System Software and Application
   * **Software:**
   1. **Google Chrome:** Application Software.

* It is used for browsing the internet and accessing websites.
  1. **WhatsApp:** Application Software.
* Used for sending massages, voice notes, and making video calls.
  1. **Android OS (Mobile):**  System Software.
* It is the operating system that’s runs your mobile phone.
  1. **YouTube App:** Application Software.
* Used for watching and streaming videos online.
  1. **Windows 10:** System Software.
* It is an operating system that manages all hardware and software in a computer.
* Application software is used to perform specific user tasks, while system software manages the computer system and hardware.

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

* The main difference between system software and application software is based on their function and usage.

**Differences:**

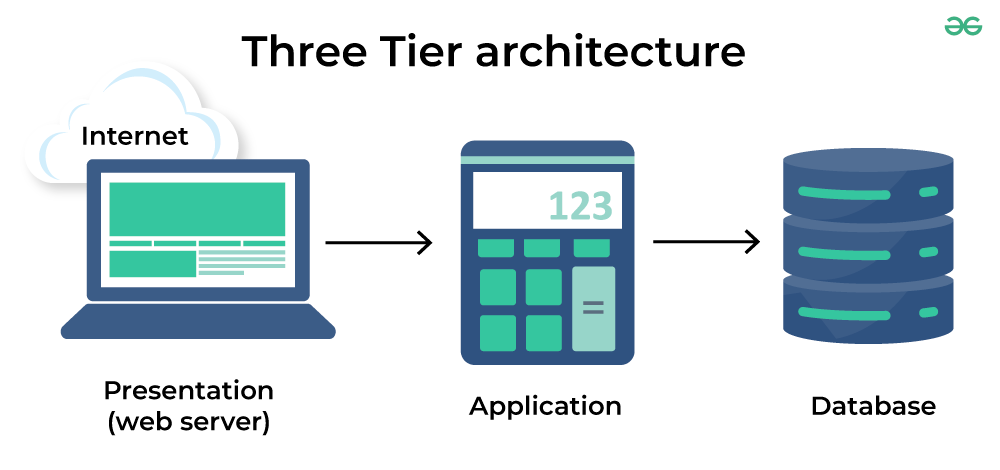
* 1. **Purpose:**➤ System software is designed to run the computer hardware and provide a platform for application software.  
     ➤ Application software is designed to help users perform specific tasks like typing, browsing, or watching videos.
  2. **User Interaction:**➤ System software usually runs in the background and the user does not interact with it directly.  
     ➤ Application software is used directly by the user to complete tasks.
  3. **Installation:**➤ System software is installed when the operating system is set up.  
     ➤ Application software is installed later as per the user’s needs.
  4. **Examples:**➤ System Software: Windows, Android, Device Drivers, BIOS  
     ➤ Application Software: MS Word, WhatsApp, Chrome, VLC
  5. **Dependency:**➤ Application software cannot work without system software.  
     ➤ System software can work without application software, but the user can’t do much without apps.

**Short:**

System software manages the computer and its hardware, while application software helps users do specific tasks. Both are essential in a computer system.

1. **Design a basic three-tier software architecture diagram for a web application.**

* It divides a web application into three layers:



1. **Presentation Tier (Web Server):**  
   ➤ Frontend – what the user sees (like a website).  
   ➤ Example: HTML, CSS, JavaScript.
2. **Application Tier:**  
   ➤ Backend – handles logic and processing.  
   ➤ Example: Java, Python, Node.js.
3. **Database Tier:**  
   ➤ Stores data.  
   ➤ Example: MySQL, MongoDB.
4. **What is the significance of modularity in software architecture?**

* **Modularity** in software architecture refers to dividing a large software system into smaller, independent, and manageable units called **modules**. Each module performs a specific function and can be developed, tested, and maintained separately.
* **Significance of Modularity:**
  1. **Improved Maintainability:**  
     Errors can be easily located and fixed in a specific module without affecting the whole system.
  2. **Reusability:**  
     Modules can be reused in different programs or parts of the same application.
  3. **Simplified Testing and Debugging:**  
     Since modules are independent, each can be tested separately, making the process simpler.
  4. **Parallel Development:**  
     Multiple developers or teams can work on different modules at the same time, increasing development speed.
  5. **Scalability:**  
     New features or changes can be added easily by modifying or adding modules without disturbing the entire system.
* **Better Code Organization:**  
  Code becomes easier to understand, manage, and update due to logical separation.

1. **Layers in Software Architecture**

* Software architecture uses **layers** to organize code into different parts based on what each part does. Each layer has a specific role and interacts with other layers in a clear way.

**1. Presentation Layer:**

Also called the **User Interface (UI)** layer.

It deals with how the software looks and interacts with the user.

**Example:** Buttons, forms, menus.

**2. Business Logic Layer:**

Handles the main logic and rules of the application.

It processes user input and makes decisions.

**Example:** Calculating bill amount, checking login details.

**3. Data Access Layer:**

Manages communication between the app and the database.

It performs operations like fetching, inserting, and updating data.

**Example:** Running SQL queries.

**4. Database Layer:**

This is where all the data is stored.

It includes the actual database system.

**Example:** MySQL, Oracle, PostgreSQL.

1. **Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.**

* **Software System Name:**

Online Library Management System

**1. Presentation Layer:**

* **Function:**

Interacts directly with the user.

Displays menus, search bar, book details, login/register forms.

* **Example Functionality:**

User logs in using a form.

User searches for a book by title or author.

User clicks on “Borrow” or “Return” buttons.

* **Technologies Used:**

HTML, CSS, JavaScript, React or Angular.

**2. Business Logic Layer:**

* **Function:**

Handles all the core rules and logic of the application.

Validates user inputs and processes requests.

* **Example Functionality:**

Check if the book is available before issuing.

Calculate due date and fines.

Validate login credentials.

Check user’s borrowing limits.

* **Technologies Used:**

Java, Python, C#, Node.js, PHP (any backend language).

**3. Data Access Layer:**

* **Function:**

Communicates with the database to read/write data.

Isolates SQL queries and database handling from business logic.

* **Example Functionality:**

Fetch user details during login.

Insert book issue record.

Update return status and fine details in the database.

* **Technologies Used:**

SQL, JDBC, Hibernate, Entity Framework, MongoDB drivers, etc.

* **Workflow Example: Book Borrowing**

**Step1:** User clicks "Borrow" on a book (Presentation Layer)

**Step2:** System checks availability and user’s borrowing limit (Business Logic Layer)

**Step3:** If valid, it updates records and sets due date (Data Access Layer)

**Step4:** Confirmation message is shown to the user (Back to Presentation Layer)

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

* Layers are important in software architecture because they provide a clear **separation of concerns,** making the system easier to develop, test, maintain, and scale, Each layer handles a specific responsibility, such as:
* **Presentation Layer –** Manage the user interface.
* **Business Logic Layer –** Handles the application’s core functions and rules.
* **Data Access Layer –** Interacts with the database.

The structure allows developers to work independently on different part of the system, reuse code, and make changes without affecting the whole application. It also improves teamwork, testing, and long-term maintenance.

1. **Software Environments:**

* A **Software environment** is the **set of tools, Platform, and settings** where software is developed, tested, and run.

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

* **Types of Software Environment:**
  1. **Development Environment:**
* Used by developers to write, build, and debug (Debug- Debug or Debugging means finding and fixing errors (bugs) in computer program) code.
* Includes toots like IDEs, compilers, and libraries.
  1. **Testing Environment:**
* Used for testing the application to find bugs.
* Separate from the user environment to avoid affecting real data.
  1. **Production Environment:**
* The live environment where the actual users use the application.
* It must be stable, secure, and optimized for performance.
* **Basic Virtual Machine Setup:**
  1. **Pick a Hypervisor:**

Operation include Oracle Virtual Box (free), Hyper-V, V

* 1. **Create a Configure the VM:**
* Create a virtual disk (dynamic allocation is space-efficient).
* Allocate at least 512MB- 2GB RAM depending on your guest OS.
* Attach an OS installation IOS.
  1. **Install the Guest OS:**

Boot from the IOS and go through standard OS setup.

* 1. **Enhance the Setup:**

Install Guest Additions or tools for shared folders and better performance

* 1. **Provision Development Tools:**

Inside the VM, install IDE/editor, compilers, runtime environment, Git, and any dependencies.

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

A **development environment** is a crucial foundation in the software production lifecycle—it’s the dedicated workspace where developers **create**, **test**, and **debug** code before it moves into testing or production. Here's why it matters:

**1. Sandbox for Safe Innovation**

Acts as an **isolated space** where developers can experiment and make mistakes without risking live system.

1. **Consistency Across Team**

Ensures everyone uses the **same tools, libraries, and configurations,** dramatically reducing “it works on my machine” issues.

1. **Boosted Productivity & Efficiency**

* Integrated tools (IDEs, compilers, debuggers, build systems) streamline development, testing, and debugging .
* Automating tasks speeds up workflows and lets developers focus on writing great code .

1. **Early Bug Detection**

* Built-in testing and debugging tools catch issues **before** code moves downstream, reducing costly fixes later
* Version control aids in tracing and reverting problematic changes quickly.

1. **Enable Team Collaboration**

* With shared version control and consisted environments, developers can  **work together seamlessly,** managing code merges and branches more reliably.

1. **Foundation for CI/CD Pipelines**

* Development environment set the stage for **continuous integration,** building and deploying code changes in controlled, reproducible settings.

**Source Code**

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

**Write and Upload Your First Code to GitHub**

1. **Create a GitHub repo**
   * Go to GitHub → **New repository** → name it (e.g., my-first-code) → click **Create**.
2. **Write a source file locally**
   * In a project folder, create hello.c:

c

#include <stdio.h>

int main() {

printf("Hello, GitHub!\n");

return 0;

}

1. **Initialize Git & commit**

bash

git init

git add hello.c

git commit -m "Add hello.c Hello World program"

1. **Link to your GitHub repo**

bash

git remote add origin <YOUR-REPO-URL>

1. **Push the file online**

bash

git push -u origin main

* **Outcome:**
* You’ve created a repo, committed code, connected to GitHub, and pushed your first file.
* Your hello.c is now live and versioned on GitHub!

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

* **Source code:**
* Written by humans in high-level languages (e.g., Python, Java, C).
* Easy to read, modify, and understand.
* Needs translation (by a compiler/interpreter) to run.
* **Machine Code:**
* Binary instructions (0s and 1s) that the CPU executes directly.
* Not human-friendly — tedious and architecture-specific
* Produced by translating source code (or assembly) via compilers/assemblers.

**GitHub and Introduction:**

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

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

* **Collaboration**

Allows multiple developers to work on the same codebase simultaneously without overwriting each other’s work, thanks to branching and merging.

* **Change History & Accountability**

Every change is recorded with details on who made it, when, and why—making it easy to review and trace issues

* **Backup & Recovery**

If something breaks, you can revert to a previous stable state—minimizing risk and downtime

* **Experimentation via Branching**

Work on new features or bug fixes in isolated branches. Merge back only when ready, keeping the main codebase clean

* **Code Quality & CI/CD Integration**

Supports code review processes and automated testing pipelines that enforce quality before deploying to production

**Student Account in GitHub**

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

**1. Get a GitHub Student Account**

1. **Create a GitHub account** (if you don't have one).
2. **Apply for the Student Developer Pack**:
   * Visit GitHub Education → Student Developer Pack → click **Get benefits**.
   * Use your school email or upload a valid student ID/transcript
3. Once verified, you receive free GitHub Pro, private repos, and premium tools

**2. Collaborate on a Small Project with a Classmate**

1. **Create a repository** in your GitHub account.
2. **Add your classmate as a collaborator** in the repo settings .
3. Both of you:

bash

git clone <repo-url>

git checkout -b feature-branch

# make changes

git add .

git commit -m "Your message"

git push -u origin feature-branch

1. On GitHub:
   * Open a **Pull Request** from feature-branch to main.
   * Review, then click **Merge**. Delete the branch afterward

* **Benefits of This Workflow**
* Safe **branching** prevents issues in your main code.
* **Pull requests** support code review and collaboration.
* Keeps your history **clean and organized**.

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

* GitHub gives students free access to powerful tools, teaches real-world workflows, promotes collaboration, boosts professional portfolios, and enhances employability—all while saving money and growing skills.

**Types of Software**

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

* **Software Classification:**
  1. **System Software:**
* Definition: Manages hardware and provides a platform for running other software.
* **Examples**: Operating systems (Windows, macOS) and device drivers.
  1. **Application Software:**
* Definition: Programs designed to perform specific tasks for users.
* **Examples**: Web browsers (Chrome), word processors (MS Word), media players (VLC).
  1. **Utility Software:**
* Definition: Helps maintain, optimize, or protect the computer system.
* Examples:
* Antivirus Software
* Disk cleanup/defragmentation tools.
* File compression tools (7-Zip)
* Backup Untilities
* Classification ensures proper **organization**, **functionality**, and **maintenance** of software, helping users and developers understand each tool’s purpose and scope within a system.

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

* **Open-source software** grants freedom to use, modify, and share—encouraging collaboration, transparency, and flexibility.
* **Proprietary software**, controlled by companies, offers polished interfaces, official support, and predictable updates, but locks users into restricted licensing and limited customization.

**Git and GITHUB Training**

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

**1. Clone  
Creates a full local copy of a remote repository, including all files, commit history, and branches, so you can work offline and experiment independently**

**2. Branch  
Enables separate development lines (“feature branches”) so you can add new features or fix bugs without affecting the main codebase**

**3. Merge  
Combines changes from one branch into another (e.g., merging a feature branch into main). It’s a non-destructive operation that preserves history or allows “fast‑forward” when possible**

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

**How Git Improves Collaboration in Teams**

1. **Distributed Repositories**
   * Every developer has a full local copy (history + branches), enabling offline work and reducing server dependency.
2. **Parallel Development with Branching**
   * Lightweight branches let team members work on features or fixes independently without affecting the main codebase.
3. **Pull Requests & Code Reviews**
   * Changes are shared via pull requests, allowing colleagues to review, discuss, and validate code before merging.
4. **Detailed Change History & Accountability**
   * Commits record who changed what and when, supporting traceability, rollback, and auditability
5. **Conflict Resolution & Integrity**
   * Git offers robust merging tools and maintains data integrity using secure hashes (SHA‑1)

* Git supports **parallel workflows**, ensures **code quality through reviews**, provides **full history and accountability**, and enhances **reliability and efficiency in team collaboration**.

**Application Software**

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

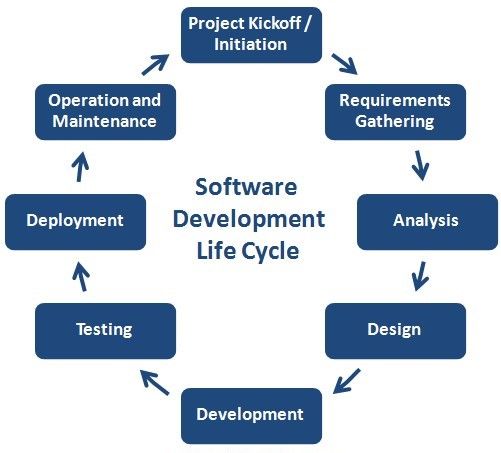
* + 1. **Office/Productivity Suites:**
* Includes: Word processors, spreadsheets, presentation tools.
* Benefit: Automates writing, calculations, formatting, graph creation; enables real-time collaboration.
  + 1. **Database Management Systems (DBMS)**
* Includes: Systems like MS Access, SQL-based tools.
* Benefit: Organizes and queries large datasets efficiently, supporting faster reporting and decision-making.
  + 1. **Graphic & Multimedia Software**
* Includes: Adobe Photoshop, video/audio editors.
* Benefit: Speeds up creation of visual and audio content, saving creative time.
  + 1. **Communication & Collaboration Tools**
* Includes: Slack, MS Teams, Zoom.
* Benefit: Enables seamless team messaging, meetings, and file sharing—boosting coordination.
  + 1. **Project & Workflow Management Apps**
* Includes: Asana, Trello, Monday.com.
* Benefit: Visualizes workflows, automates task assignment, tracks progress—reducing project overhead.
  + 1. **Specialized Business Apps**
* **Includes**: ERP (e.g., Dynamics 365), CRM (e.g., Salesforce), Inventory & Helpdesk systems.
* **Benefit**: Tailored to business tasks—integrates operations, secures data, and supports scalability.
* **How They Boost Productivity:**
* **Automate repetitive tasks** → saves time (e.g., auto‑calculations, macros, workflow routing).
* **Enhance accuracy** → reduce human errors with validation and consistency checks.
* **Improve collaboration** → shared documents and real-time updates.
* **Support informed decisions** → via dashboards and data analytics.
* **Scale with business growth** → cloud‑based SaaS adapts to user/team size changes.

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

* Application software is essential in business by:
* **Automating core operations**—like accounting, payroll, order processing—to boost efficiency and accuracy.
* **Streamlining communication and collaboration** through email, messaging platforms, and shared documents—vital for modern workplaces.
* **Supporting data-driven decisions** with tools that generate analytics, reports, and dashboards.
* **Enhancing competitiveness** by enabling better customer relationship management, faster responsiveness, and improved service delivery.

**Software Development Process**

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



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

* **Key SDLC stages include:**
* **Planning & Feasibility:** define scope, objectives, and technical/financial viability.
* **Requirements Analysis**: gather, document, and validate stakeholder requirements.
* **System Design:** devise architecture, UI, and technical specifications.
* Implementation: developers write and debug the code.
* **Testing & Integration:** perform unit, integration, system testing to detect and fix defects.
* **Deployment:** release software for end-users via rollout or pilot.
* **Maintenance:** ongoing updates, bug fixes, and feature enhancements.

**Software Requirement**

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

* **Purpose & Scope:** Web-based system to manage book catalog, member info, borrowing/return, fines, and reporting.
* **Actors:** Librarian and Member.
* **Functional Requirements:**
  + *Librarian:* Add/edit/delete books; register/update members; issue, return, and renew books; overdue fines; generate inventory/loan reports.
  + *Member:* Search catalog; view availability; request book; view loan status and history; renew loans.
* **Non-functional Requirements:**
  + *Performance:* Respond to searches in ≤2 seconds; support 5,000 records.
  + *Security:* Login authentication; enforce role-based access.
  + *Usability:* Intuitive UI; web-browser support (Chrome, Firefox).
  + *Reliability:* Daily backups; 99% uptime.

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

* **The requirements analysis phase is vital because it:**
  + Aligns stakeholder expectations—setting a shared understanding of what will be built.
  + Prevents costly rework by catching missing or ambiguous requirements early.
  + Guides design, development, and testing, serving as a clear foundation.
  + Mitigates project risks by identifying constraints, traceability, and feasibility issues up front.

**Software Analysis**

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

* **Functional Analysis Process:**
* **Define system goal:** Enable customers to browse, purchase, and manage orders; enable admin tasks.
* **Identify actors & use cases:**
  + ***Customer:*** Browse/search products, add to cart, checkout, pay, track order, request returns.
  + ***System:*** Inventory checks, price calculations, order processing, payment handling, notificat**ions.**
  + ***Admin:*** Add/edit products, manage orders, handle returns, view analytics.
* **Decompose functions:**
  + Browsing & search (filters, sorting)
  + Cart management (add/remove/update)
  + Checkout (capture shipping/payment details)
  + Payment processing (integration with payment gateway)
  + Order confirmation and tracking
  + Returns/refunds workflow

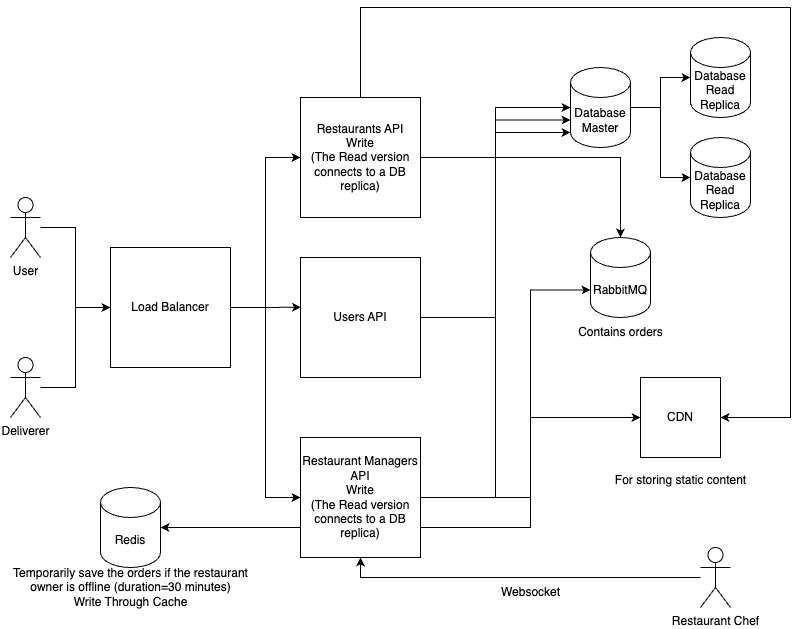
**THEORY EXERCISE: What is the role of software analysis in the development process?**

* Software analysis bridges **what** needs to be done and **how** to do it. Its main roles are:
* **Clarifying requirements** in functional terms.
* **Identifying user interactions, inputs/outputs, and constraints** early.
* **Enabling feasibility checks**, guiding architectural and design decisions.
* **Reducing ambiguity and rework**, by resolving issues before coding starts.
* **Providing documentation and traceability** that underpins development, testing, and maintenance.

System Design

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

* **Design a basic system architecture:**
* **Users & Delivery Drivers** hit a **Load Balancer**, which directs requests to the right backend service.
* **Users API** handles sign-up, login, profile actions.
* **Restaurant APIs** (Customer-facing & Manager-facing):
* **Writes** go to the master database.
* **Reads** go to read replicas.
* Order submissions are pushed into RabbitMQ for processing (e.g., notifications, updates).
* **Restaurant managers** receive real-time updates via WebSocket, even if offline—using **Redis** cache (30 min expiry).
* **Static files** (images, menus) are served from a CDN.
* **Database replicas** help scale read traffic and reduce load on the main DB.

****

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

1. **Client-Server Architecture –** separates UI and backend logic.
2. **APIs / Communication Protocols** – REST/gRPC over HTTP(S).
3. **Load Balancing** – distribute traffic to multiple servers.
4. **Caching** – reduce latency and server load.
5. **Databases & Partitioning** – storage with sharding/replication for scale and availability.
6. **Message Queues / Asynchronous Processing** – reliable event handling outside user flow.
7. C**DN & Edge Optimization** – deliver static assets quickly.
8. **Fault Tolerance & Scaling** – redundancy, auto-scaling, circuit breakers.
9. **Security** – HTTPS, encryption, authentication, authorization.
10. **Monitoring & Observability** – metrics, logs, alerts to maintain health.

**Software Testing**

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

* **Calculator Program**

**#include<stdio.h>**

**main() {**

**int a=20, b=10, ans;**

**float div;**

**ans = a + b;**

**printf("\n addition=%d", ans);**

**ans = a-b;**

**printf("\n sub=%d", ans);**

**ans = a\*b;**

**printf("\n mult=%d", ans);**

**div = a/b;**

**printf("\n div=%f", div);**

**}**

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

**Software testing is critical because it:**

1. **Identifies bugs early,** greatly reducing fix costs.
2. **Ensures reliability and performance,** validating behaviour meets requirements.
3. **Boosts confidence—**developers and stakeholders trust the product.
4. **Prevents regressions** with automated testing to catch future breaks.
5. **Controls risk and compliance,** reducing security flaws and meeting standards.

**Maintenance**

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

* **Case:** A popular mobile banking app malfunctioned after an OS update.
* **Issue:** New iOS version broke the session timeout logic, causing app crashes and login failures.
* **Response:** Within 12 hours:
  + **Corrective fixes** to patch the bug.
  + **Adaptive maintenance** to ensure compatibility with updated OS APIs.
  + **Preventive measures** added to test on future beta OS releases.
* **Outcome:** Restored critical services and prevented recurrence.

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

* **According to IEEE/ISO standards, software maintenance falls into four categories:**

1. **Corrective –** Fix bugs and defects.
2. **Adaptive –** Update software to work with changed environments (OS, APIs).
3. **Perfective –** Improve performance, usability, or add enhancements.
4. **Preventive –** Refactor code, update docs, anticipate and prevent future problems.

**26.Development**

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

* **Platform dependency:** Desktop apps are OS-specific; web apps run on any browser.
* **Installation: Desktop requires install;** web apps are accessed via URL—no install needed.
* **Updates:** Desktop updates are manual per device; web apps are updated centrally and immediately.
* **Performance:** Desktop apps can be faster and resource-rich; web apps rely on network and browser capabilities.
* **Security:** Desktop stores data locally (more private); web apps store data in the cloud with inherent online risks.

**27.Web Application**

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

* **Web apps offer:**
* **Cross-device access:** Anytime, anywhere via browser.
* **No installation or local storage required:** Saves space and setup hassle.
* **Automatic, centralized updates:** Users always access the latest version.
* **Lower development and maintenance costs:** One codebase supports all platforms, simpler to maintain.
* **Collaboration features:** Real-time editing and shared data (e.g., Google Docs).

**28.Design**

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

* **Role of UI/UX Design in App Development**

**First Impressions & Trust**

* A clean, intuitive UI creates a positive initial experience, leading users to trust and stay with the app

**Usability & Efficiency**

* Good UX ensures users can reach their goals quickly and without frustration—reducing errors and support needs

**User Engagement & Retention**

* Engaging design elements and smooth flows keep users coming back, boosting retention and satisfaction .

**Brand Identity & Loyalty**

* A consistent, visually appealing UI reinforces brand image and encourages loyalty

**Cost Reduction & Faster Development**

* Early UI/UX prototyping catches problems before coding, saving time and money later on

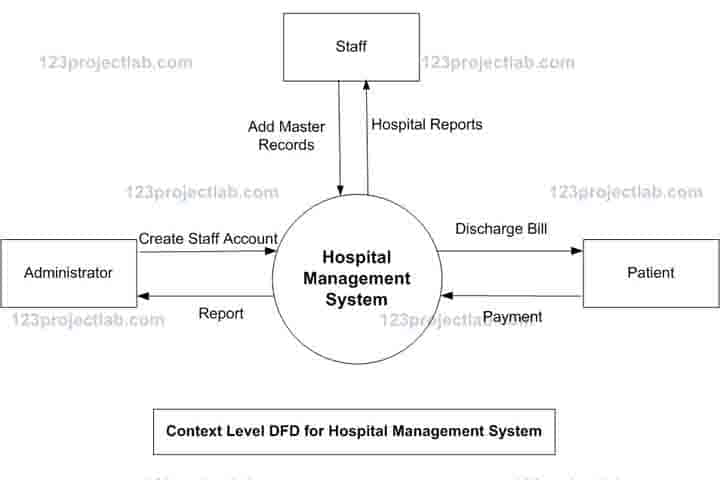
**29. Mobile Application**

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

* **Development:**
* **Native:** Build using platform-Specific language (Swift/Obj-C for iOS, Java/Kotlin for Android).
* **Hybrid:** Uses web technologies (HTML/CSS/JavaScript) inside a native wrapper like Cordova, lonic, Flutter, or ReactNative
* **Performance & UX:**
* **Native** apps have full access to hardware (Camera, Sensors, Bluetooth, AR).
* **Hybrid** apps rely on plugins, sometime limiting access or stability
* **Cost, Speed & Maintenance:**
* **Native** requires separate codebases for each platform—slower development, higher cost, more maintenance.
* **Hybrid** allows one codebase for both platforms—faster to market, cheaper, and easier to update

**30. DFD (Data Flow Diagram)**

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

****

* Admin creates staff accounts and gets reports.
* Staff adds records and receives hospital reports.
* Patients pay bills and receive discharge bills.

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

* **Data Flow Diagrams (DFDs) are vital in system analysis because they:**

1. **Visualize information flow** – Show how data moves between processes, stores, and external entities, in a clear, standardized format
2. **Bridge stakeholders** – Simple and graphical, DFDs help both technical and non-technical users understand system design
3. **Enable layered detail** – From high-level context (Level 0) to detailed subprocesses, they support progressive elaboration.
4. **Expose inefficiencies & risks** – Highlight bottlenecks, redundancies, and potential data-security gaps.
5. **Serve as documentation** – Form lasting system documentation that aids maintenance, audits, and future modifications

**31. Desktop Application**

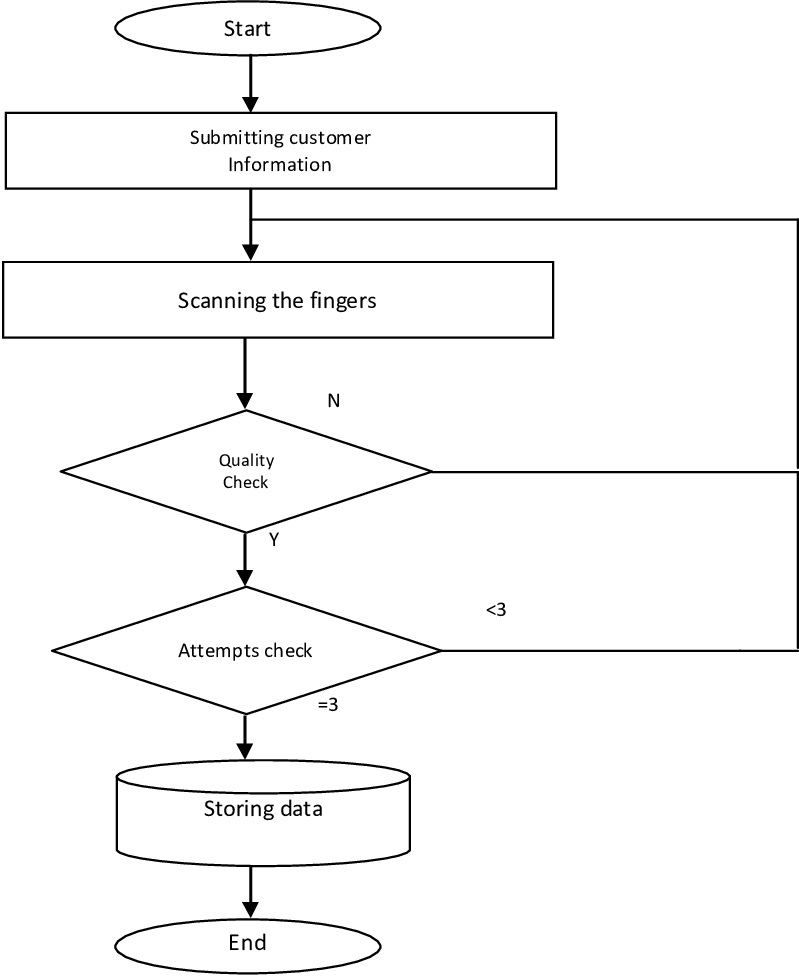
**THEORY EXERCISE: What are the pros and cons of desktop applications compared to web applications?**

* **Desktop Applications**
* **Pros:**
* **Better performance & speed** – Runs locally, offering fast responsiveness and smooth UX
* **Offline availability** – Fully functional without internet
* **Rich features & deeper system integration** – Can leverage hardware and advanced capabilities
* **Enhanced privacy & security control** – Data stored locally, less exposure to online threats .
* **Cons:**
* **Device-specific installation** – Needs separate builds per OS; not portable
* **Manual installation & updates** – Inconvenient and error-prone
* **High development cost** – Cross-platform apps require multiple codebases
* **Consumes local resources** – Requires disk space and computing power
* **Limited collaboration** – Harder to share or collaborate in real time
* **Web Applications**
* **Pros:**
* **Universal access** – Available on any browser, device, or OS
* **No installation required** – Fast setup via URL
* **Automatic updates** – Always current and secure without user effort
* **Cost-effective & scalable** – Easier to scale and often cheaper to develop and deploy
* **Real-time collaboration** – Multiple users can work together simultaneously
* **Cons:**
* **Internet dependency** – Requires network; offline use limited .
* **Lower performance** – Heavier resources or latency affect response speed
* **Security & privacy concerns** – Data hosted externally; browser vulnerabilities
* **Browser compatibility quirks** – Extra effort needed to support multiple browsers
* **Ongoing costs** – Subscription fees may accumulate over time .

**32. Flow Chart:**

**EXERCISE: Draw a flowchart representing the logic of a basic online registration system.**

* **representing the logic of a basic online registration system**

Top of FormBottom of Form

1. **Start –** User enters personal info.
2. **Scan Fingerprint –** The system scans the fingerprint.
3. **Quality Check –** If the scan is poor, you try again (loops back).
4. **Attempts Check –** After up to 3 tries:
   * If under 3 attempts and good scan → move on
   * If exactly 3 tries, still proceed
5. **Store Data –** Save user info and fingerprint.
6. **End –** The process finishes.

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

**1. Visualizing Logic & Processes**

Flowcharts graphically represent workflows, algorithms, and decision points, making complex logic easier to **see and follow**

**2. Planning & Design**

They serve as **blueprints** during system design—helping map modules, data flow, and component interactions before writing any code .

**3. Debugging & Quality Control**

By tracing execution paths visually, flowcharts aid in pinpointing logical errors, inefficiencies, or bottlenecks during development and testing .

**4. Communication & Collaboration**

A universal diagramming language, flowcharts allow technical and non-technical stakeholders to align on process steps, system requirements, and design decisions

**5. Documentation & Maintenance**

They form clear, visual documentation that supports onboarding, legacy system understanding, and future modifications .

**6. Limitations**

**Scalability issues**: Becomes hard to manage for very large systems.

**Maintenance burden**: Changes require updating the chart manually, which can be time-consuming and costly

* **Theory**

Flowcharts are foundational “boundary objects” in systems engineering—supporting **analysis**, **design**, **communication**, and **documentation**. While they may not suit every modern codebase, they’re especially valuable in early design stages, multidisciplinary projects, and when clarity matters most

* Top of Form
* Bottom of Form