**Day 22 – August 1st**

**TASK 1 – Good Code Vs Bad Code**

**Good Code**

**-----------------------------------**

| **Criteria** | **Description** |
| --- | --- |
| **Readable** | Easy to understand at a glance. Clear variable names and comments. |
| **Modular** | Broken into reusable functions or classes. |
| **Follows SRP** | Each class or method does one thing (Single Responsibility Principle). |
| **Uses Proper Naming** | Variables and methods clearly reflect their purpose. |
| **DRY (Don’t Repeat Yourself)** | Avoids code duplication; uses loops, functions, or inheritance. |
| **Consistent Style** | Follows consistent formatting, indentation, and naming conventions. |
| **Handles Errors Gracefully** | Includes exception handling and input validation. |
| **Scalable** | Easy to extend or modify with minimal impact on other parts. |
| **Testable** | Can be easily tested with unit tests. |

**Bad Code**

**------------------------------------**

| **Criteria** | **Description** |
| --- | --- |
| **Hard to Read** | Confusing logic, poor formatting, no comments. |
| **Monolithic** | Long methods doing too many things. No separation of concerns. |
| **Bad Naming** | Uses vague names like a, temp, doStuff. |
| **Repeats Logic** | Same code written in multiple places (copy-paste everywhere). |
| **No Error Handling** | Crashes on edge cases, no validation. |
| **Tightly Coupled** | Components heavily depend on each other—hard to modify independently. |
| **Inconsistent Style** | Different styles in indentation, naming, etc. |
| **Not Testable** | Difficult to write tests due to tightly coupled and complex code. |

**TASK 2 - What do you understand by DataBinding?**

**Data Binding** is a programming technique used to **connect the data (logic)** of an application with its **UI (User Interface)**, so that changes in one automatically reflect in the other.

Data binding allows to **bind UI elements to data sources**, so that when the data changes, the UI updates automatically — and sometimes, vice versa.

**TASK 3 - What do you know about continuous development?**

**Continuous Development (CD)** is a modern software engineering practice that emphasizes **frequent, automated updates to software**, ensuring faster, safer, and more reliable product delivery.

### Key Concepts of Continuous Development:

1. **Automation** – Builds, tests, and deployments are all automated using tools (like Jenkins, GitHub Actions, CircleCI).
2. **Version Control** – Uses Git (or similar) to manage frequent code changes.
3. **Testing Pipeline** – Unit, integration, and end-to-end tests validate every change.
4. **Feedback Loops** – Quick error detection and resolution.
5. **DevOps Culture** – Developers and operations collaborate closely.

### Benefits:

* Faster feature delivery
* Reduced bugs in production
* Improved team productivity
* Early detection of integration issues
* Happier users with frequent updates

**TASK 4 – Conditions of PolyMorphism?**

### ****Conditions for Polymorphism in Object-Oriented Programming (OOP)****

Polymorphism means **"many forms"** — it allows the same interface or method to behave differently depending on the context (object or data type).

There are **two main types**:

* **Compile-time polymorphism** (method overloading)
* **Runtime polymorphism** (method overriding)

## **Compile-Time Polymorphism** (Method Overloading)

### Conditions:

1. Methods must have the **same name**.
2. Methods must differ in **number or type** of parameters.
3. Can be in the **same class**.
4. Return type **can be same or different** (but not only difference).

## **Runtime Polymorphism** (Method Overriding)

### Conditions:

1. There must be **inheritance** (parent-child relationship).
2. Method must be **overridden** in the subclass with:
   * **Same method name**
   * **Same parameters**
   * **Same return type** (or covariant)
3. Method must be **non-static** and **not final/private** in the superclass.
4. Access modifier in child class should be **same or more accessible**.

**TASK 5 – TTD and BDD**

## **TDD – Test-Driven Development**

### ****What is TDD?****

TDD is a **development approach** where tests are written **before** writing the functional code. It follows a **Red → Green → Refactor** cycle.

### ****Why is TDD used?****

* To **write clean, bug-free code**.
* To ensure that **every feature is testable**.
* To catch bugs **early** in the development process.
* To create a **safety net for future refactoring**.

### ****Where is TDD used?****

* In **unit testing**: Small components/modules.
* Used in **backend development**, logic-heavy services, microservices.
* Popular in **Agile/DevOps teams** for CI/CD pipelines.

## **BDD – Behavior-Driven Development**

### ****What is BDD?****

BDD is an **extension of TDD** that focuses on the **behavior of the system** from the user’s point of view. It uses **natural language (Given-When-Then)** to describe how the app should behave.

### ****Why is BDD used?****

* Encourages **collaboration** between developers, testers, and business.
* Makes requirements more **readable and understandable**.
* Serves as **living documentation**.
* Automates **acceptance testing**.

### ****Where is BDD used?****

* In **integration or end-to-end testing**.
* In **cross-functional Agile teams**.
* Used in **product development** to align business needs with code.

**TASK 6 - List down the Manual and automated testing tools**

**Manual Testing Tools**

| **Tool Name** |  |
| --- | --- |
| **JIRA** |  |
| **TestLink** |  |
| **qTest** |  |
| **PractiTest** |  |

**Automated Testing tools**

Selenium

Cypress