CS-499-11684-M01

Professor Fitzroy Nembhard

Fernanda Coghlan

22 May 2025

# CS 499 Module Three

# Milestone Two:

# Enhancement One: Software Design and Engineering

**PART I**

**Artifact Narrative**

### **Artifact Description**

This mobile application, initially created in CS 360: Mobile Architecture and Programming, was developed using Android Studio and Java. The original version allowed users to manually enter their weight and see historical entries in a basic table layout, but it followed a monolithic structure with limited separation of concerns.

### **Justification for Inclusion**

I selected this artifact for the **Software Design and Engineering** category because it provided a foundational base that needed architectural refactoring. By restructuring the app using **MVVM (Model-View-ViewModel)** architecture, I was able to separate the user interface (View), data logic (ViewModel), and data access (Model) components. This makes the app more maintainable, scalable, and easier to test.

The enhanced version now includes:

* A ViewModel for lifecycle-aware data management
* A Repository to abstract data operations
* A Room database with DAO and schema
* LiveData for real-time UI updates

These changes showcase my ability to implement a modern, industry-relevant architectural pattern in mobile development.

### **Alignment With Course Outcomes**

In Module One, I planned to demonstrate skills in **Course Outcome #4 (tools and techniques for design)** and **#5 (security and separation of concerns).** My enhancements show progress in:

* Designing reusable ViewModel and Repository layers
* Managing dependencies and lifecycle across UI components
* Separating business logic from view logic (security by design)

### **Reflection on the Enhancement Process**

Enhancing the application to follow the MVVM pattern required significant refactoring. One major learning curve was integrating the ViewModel in a way that preserved user data through configuration changes, which required the use of LiveData and proper scoping.

Another challenge was redesigning the logic to reside within a Repository instead of the Activity. This included redirecting calls to insert and retrieve weights through ViewModel methods, thus respecting the new architecture.

I also ensured compatibility with API 33+ by offloading database interactions from the main thread using Executors.

1. **Skills Demonstrated**

* Implementation of MVVM in Android
* Use of LiveData, ViewModel, and Room ORM
* Design of reusable architecture components (ViewModelFactory, Repository)
* Thread-safe and lifecycle-aware mobile development
* Refactoring legacy monolithic code into modular components

### **Outcome Coverage Summary**

| **Course Outcome** | **Enhancement Alignment** |
| --- | --- |
| CO #1 – Collaboration | Git versioning and team-ready architecture (MVVM) |
| CO #2 – Communication | Clear code organization, comments, and logical structure |
| CO #3 – Algorithms | Enhanced version also includes analytical features (e.g., weekly trends) |
| **CO #4 – Tools/Design** | MVVM, Room, LiveData, ViewModel, modular refactor |
| **CO #5 – Security** | Separation of concerns, data validation, crash prevention |

**PART II**

**Artifact Submission**

Technical artifact files were zipped and uploaded and I used the check list below to assure the enhancement proposals for Software Design and Engineering were implemented:

* 1. **MVVM Architecture**

My project does implement MVVM (Model–View–ViewModel) through:

* WeightViewModel.java: that contains LiveData, and binds logic away from the UI
* WeightRepository.java: that separates data source logic
* WeightViewModelFactory.java: that provides the ViewModel with required parameters

These changes align directly with your stated plan to restructure the app using MVVM.

* 1. **Separation of Concerns**

My UI activity (WeightTrackerActivity.java) no longer manipulates raw data directly. Instead:

* It observes LiveData from the ViewModel
* It delegates data insertion via viewModel.insertEntry(...)

This demonstrates improved modularity and testability as outlined in your plan.

* 1. **Lifecycle Awareness & LiveData**
* The WeightViewModel exposes LiveData and is lifecycle-aware
* AppDatabase.java initializes Room database cleanly with getDatabase(...)
  1. **Security and Maintainability**

The DAO layer (WeightEntryDao.java) and repository design limit direct DB access, helping meet security and abstraction best practices.