**3-2 Milestone Two: Enhancement One: Software Design and Engineering - Narrative**

Matthew R. Chandler

matthew.chandler2@snhu.edu

Southern New Hampshire University

CS 499: Computer Science Capstone

Instructor Joseph Conlan

September 28, 2025

**3-2 Milestone Two: Enhancement One: Software Design and Engineering - Narrative**

The artifact is an Android event tracking app that enables users to create events and receive text reminders. In CS 499, I improved my Java-based CS 360 project by moving the codebase to Kotlin while preserving its original functionality. With its UI layer, Room persistence layer, and integration with platform services like AlarmManager and SMS, this artifact exemplifies a comprehensive yet targeted application, which is why I chose it. As I had intended for this milestone, that combination allowed me to show a modest security hardening of the broadcast path along with a realistic modernization effort.

Because it demonstrates the practical skills required for mobile teams, I included this artifact in my ePortfolio. Experience with Kotlin idioms, such as data classes for entities, explicit nullability of DAO return types, and effective collection and string operations, are all demonstrated by the migration. It also shows that I can change the app glue code without changing the behavior that is visible to the user. For example, Kotlin data classes have taken the place of Java objects for Event and PhoneNumber, and getEventById now returns a nullable type that needs to be verified by the recipient before use. By preventing the alarm receiver from exporting and adding a runtime permission check for SMS, I also decreased the broadcast surface. After adding the precise alarm permission for more recent Android versions at the platform level, I also confirmed that the app continues to schedule and manage reminders correctly. I was able to reduce boilerplate code and improve clarity without sacrificing the stability of the activities and the Room model. My Module One plan made specific reference to these improvements, such as the language migration and receiver hardening, and the outcomes achieve that goal.

In Module One, I achieved the results I had hoped for from this improvement. The code is simpler for another developer to read and edit thanks to the Kotlin rewrite. I created a concise design justification outlining the changes and their reasons for professional communication. I refactored the code to Kotlin while maintaining behavior using contemporary tools and techniques for software engineering and design. By making the SmsAlarmReceiver internal to the application and verifying permissions prior to sending messages, I was able to minimize exposure for security. These were the results I promised to cover for this category, and I will continue to add to in the coming assignments. As originally intended, my algorithm and database improvements will be covered in later work.

I learned a lot about writing safer and more efficient Android code by switching the project from Java to Kotlin. I got a baseline from the automatic converter, but the real effort was in making the output idiomatic. To make the compiler enforce checks that had previously been informal, I changed model classes to data classes, tightened nullability throughout, and substituted Kotlin properties for getters and setters. I had to add early returns in the receiver and switch to a safer pattern for obtaining the SmsManager after updating DAO signatures to return nullable types revealed a few places where the Java code had assumed objects would always be present. To make sure scheduling remained reliable on API 31 and later, I had to examine how alarms broadcast intents on more recent Android versions and add the precise alarm permission. Using explicit annotations and catching SecurityException around SMS sending, I resolved low-level Java and Kotlin differences between the Room and Android APIs, such as default values for function parameters and exception handling. Refactoring the code while preserving the app's functionality and ensuring that nullability changes did not result in errors was the primary challenge. These tradeoff changes improved the application's security and organization while strengthening processes I intend to use on a mobile development team, like encouraging compiler-verified contracts and lowering component exposure.