As students with little team coding experience, our group started out in our "comfort zones" - working individually on the project. After a few pushes to the repository working separately, we quickly we realized we needed to collaborate with one another so that we were all on the same page about design, progress, and current state of the project as well as not individually working on the same problem within the code structure, resulting in branching issues in the repository.

We began to communicate online via Microsoft Teams, in person class meetings and text message threads which led to higher productivity, efficient delegation of duties and improved the overall structure of the code as a result. This communication also made debugging and problem solving both individually and in group settings a lot easier as we were all familiar with the current state of the project and not working to catch up on the updates another team member had recently made. Open dialog also allowed us to come to an important consensus early on in the project; we needed to refactor the given architecture to more closely match our state diagram. We did so by renaming several classes and methods as well as removing redundant / non applicable code in the original architecture to more closely resemble our model. This allowed us to more clearly see the relationships between the important classes and the overall structure we were trying to implement and set us up on solid ground to begin to build the various requirements of the project

Given our lack of coding experience with big android based projects, we relied heavily on the state diagram we developed in class to get this project completed. It was much easier to look at the diagram as a visual reference for what we were trying to accomplish at a given stage, rather than numerous classes, interfaces, packages and resources in the folder structure. We decided to model first rather than go straight to writing code, as that way we would have a functional roadmap of specifically what code we needed to make sure we had. Because of this our code matches the diagram very closely, with everything accounted for. Now that we have written the code, the diagram does look a little simplified, but wouldn't need any drastic changes to it to make it match the final product. For the most part, our code structure and our diagram are pretty similar. The difficulty came in the actual implementation, which was couched in many patterns and different methods, making it hard to follow at times. One example of this is in the alarm state. Originally, we had trouble fulfilling the requirement of playing the alarm sound because we were having issues following the path of command through the code. In order to solve this problem, we had to go over each step of inheritance and implementation to find where we needed to implement the method.