We had a very good experience on pair development, like peer review the initial state machine model, unit test design document, online meeting to discuss task allocation and align team status, code sharing and updating via bitbucket, and refactoring some external interfaces for better design. In current situations, we mainly keep touch by email. If we have any question, we will send email to seek help. If we find any bugs in the code, we will discuss it and fix it. If we have any update, we also will send email to notice everyone. Pair development can help team correct mistakes as early as possible and it also presents an opportunity to improve each other. For example, at first, we thought on Stopped state, the program will do increment by Tick event, when user clicks the button, the program enters the Waiting state. In the middle step of project, thanks to pair development, we found that we made a requirement analysis error. In the requirement list, the program does increment by Click event on Stopped state. This means our model (state machine diagram), unit test design and code all need to refactor. It is a good lesson for us, and we learned much by pair development experience.

We used the state design pattern, it is a behavioral design pattern that allows an object to change behavior when it transitions from state to state. This can be a cleaner way for an object to change its behavior at runtime without resorting to large monolithic conditional statements. So, it is a good fit to our project requirement. The difficult part about this initially was allocating the onX functions to each state. We understood all the functions, but initially we had put some onX functions in the wrong state. Later as we developed our tests we realized where each function belonged in the state diagram.

During the project, we found State Machine Diagram and unit test design are very useful, especially State Machine Diagram. It makes code become easy and clear. Model first is more effective, especially in complex state transition. If the design of State Machine Diagram is correct, it will offer the best ideas and checklist for writing code. Otherwise, the code would become confusion and hard to implement. We had initially made assumptions about the states or number of states that were used for the stopwatch, then have designed our State Machine Diagram. As we implemented the code, we realized that we had to go back and re-adjust our model because of requirement analysis error. After we re-adjusted the model, we finished coding the project.

About the changes we made to our model, initially we were going to use 5 states for our simple timer, we decided against it and we came up with 4 states. The states were Stopped, Waiting, Running, Alarming. Once we understood the functionality of the states, we changed the states to Stopped, Increment, Running, Alarming. From our first to our last model we also changed up the way we control flow of the states. Initially some of the transitions were not inline with the functional requirements.