Group 4 Development Journey – Kaufman, Kidanu, Yared, Rizvi, Ogrodny, Jahin

Our group's development journey revolved around creating a functional timer system with specific features and state-based behavior. The process started by breaking down the timer's functionalities into smaller components, such as managing the display, button actions, timing logic, and alarm behavior. We ensured clarity and prevented the project from becoming overwhelming by solving one feature at a time.

Pair programming was a cornerstone of our collaboration. Team members paired up to implement key functionalities such as button event handling, state transitions, and the alarm system. This approach facilitated peer reviews and ensured that the code maintained consistent standards while allowing us to address challenges collaboratively.

The repository served as a shared workspace and was essential to our workflow. To maintain synchronization, we only pushed working modules to the repository after thoroughly testing them locally. This practice helped other team members seamlessly pick up where others left off and reduced the risk of conflicts during integration. By using descriptive commit messages, we kept the history organized and made it easier to track changes.

Testing and refactoring were ongoing activities throughout the development process. We wrote tests to verify the timer's behavior in various states stopped, running, and alarming and iteratively refined the code for readability and efficiency. These practices ensured that the final product met the expected requirements and handled edge cases effectively.

The relationship between the model and the code is that the extended state machine model provided a conceptual framework for the timer's behavior and significantly influenced the implementation.

- **Similarities**: The code closely followed the model's states and transitions, such as **stopped**, **running**, and **alarming.** Each state's behavior like decrementing the time when stopped, or activating the alarm when the time reached zero was directly derived from the model.
- **Differences**: The model captured the high-level behavior of the timer but did not address the implementation-specific details, such as button debounce handling or the precision of time measurement. These aspects had to be carefully addressed in the code to ensure smooth operation.

- **Code vs. Model First**: Starting with the model was beneficial as it provided a roadmap for implementation and helped us avoid ambiguities in the timer's behavior. However, during coding, we gained insights into practical constraints and edge cases that required iterative refinement of both the model and the code.
- **Improvements to the Model**: Based on the completed code, some adjustments to the model could enhance its accuracy and utility. For instance:

Adding transitions for edge cases, such as a button press during the transition from running to stopped. Including a representation of time constraints, such as the 3-second delay before the timer starts running.

These refinements would improve the model's alignment with real-world usage and make it more robust for future projects.