High-Level Design

We will use the Layer/Tiered Architecture to structure our project. More specifically, the Model-View-Controller / 3-layered architecture would be best suited for our project. This is mainly due to the MVC architecture's inclusion of UI. Our project will be reliant on a well made UI system that can take various inputs to process. The application will then have to process these inputs to manipulate a database of all of the team members (users) and all of the tasks for the team. It will then have to access this database and update the UI to show the users the correct information about the tasks.

Low-Level Design

The Behavioral Design Pattern Family would probably be helpful for our project. Besides designing the UI, this project would revolve around two main types of objects, team-members and tasks.

- State Pattern: Status patterns can be used to model the different states of a pending task (e.g., unfinished, completed, in progress). The behavior of a task can change based on its current state. For example, you can use different text colors to mark work in different states.
- Template Method Pattern: The template method pattern can define a skeleton algorithm for managing tasks, and the specific steps are implemented by subclasses. For example, the interaction and task between user software and third-party software can be realized through subclasses.
- Memento Pattern: Memento can be used to capture the status of a to-do list, allowing users to undo or redo changes. For example, the user's work requirements when creating a to-do event list are only approximate requirements and are not precise. It can be used to store the status of the list before and after modification, so that the user can adjust according to the process.
- Strategy Pattern: Strategy patterns can be used to encapsulate different sequencing strategies for tasks. Users may want to sort tasks by deadline, leader requirement, priority, creation date, difficulty to overcome. The Strategy pattern encapsulates the algorithms, and makes them interchangeable. It lets the algorithm vary independently from clients that use it allows Easily switch between strategies

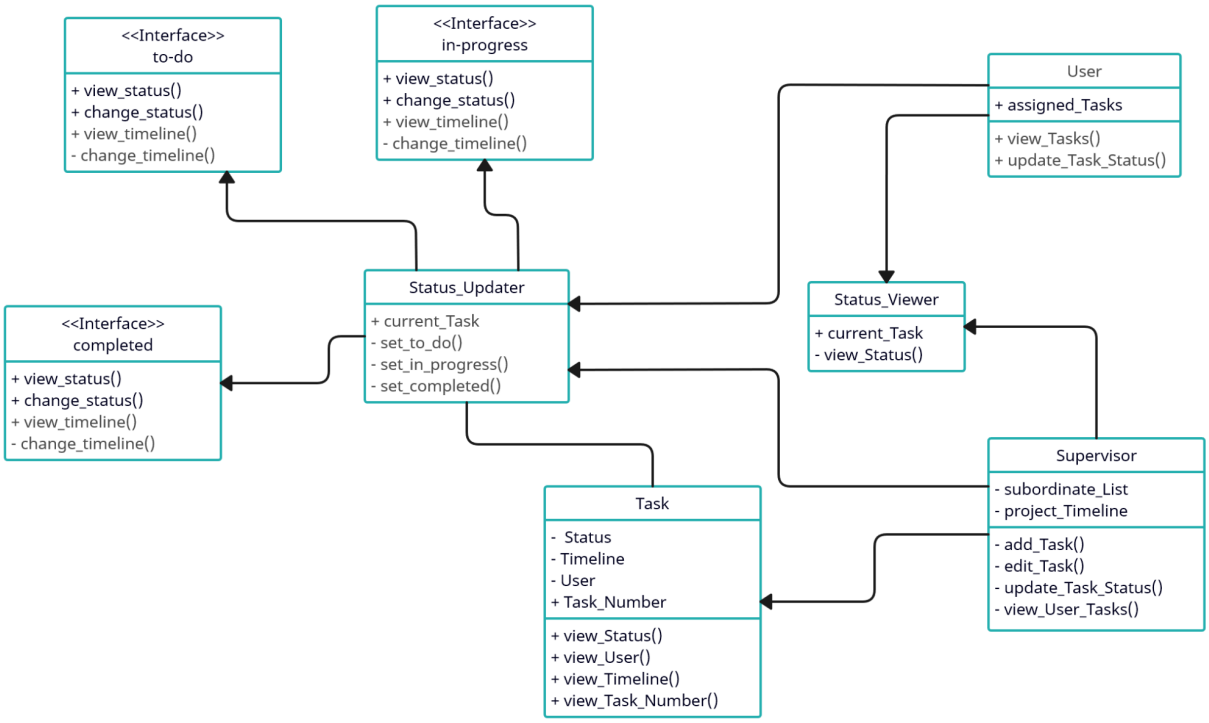
For our program, we would make the most use out of the State Pattern as our program relies on tasks being implemented, then having their "state" changed from "to-do", to "in progress", to "completed". Making use of the State Pattern allows us to accomplish this with our project.

Pseudocode:

create Task “Design UI”;
set Task_Status “to-do”;
set Task_Description “The program’s user interface needs to be developed in a way that allows users to easily view tasks, while also allowing supervisors to view all lower level task assignments and manage them with ease.”;
if Task_Status is “to-do”
    Check if user has started task;
    If user has started task, change status to “in progress”;
Else, if Task_Status is “in progress”
    Check if user has completed task;
    If user has completed the task, change status to “completed”;

push Task to Supervisor;

Class Diagram:



Design Sketch

This is a wireframe we created for our TODO app. This app contains the feature of task creation, time checking, and filters. Users could check correspondent events based on their needs. Users could create different types of tasks. Below is the link to this wireframe.
https://balsamiq.cloud/s5x1gc1/p2fgrz0

Process Deliverable