**List of participants:** Neil Dave, Mitchell Comer, Lucas Liebermann, Yu Lu, Tyler Vo

**5 Design Principles:**

From SOLID:

*Single responsibility principle* - each class has only a single responsibility, since TeamMember manages the creation/attributes of members, while Project manages the tasks and members within itself. By separating what each class is concerned about, our code is easier to understand and extend.

*Open/Closed principle* - our design lets us extend our functionality without needing to change our existing code, since we used abstract classes like Task and TeamMember. Thus, we can easily create new concrete implementations of these abstract classes and reuse the shared attributes.

*Liskov Substitution principle* - a superclass can be substituted with its subclasses without affecting the semantic correctness of the program due to our use of abstract classes. For example, any code that uses TeamMember and tries to call workOnProject() will function regardless of if the TeamMember is an instance of Developer or ProjectManager, due to how we designed them as having concrete implementations of the abstract workOnProject() method.

*Dependency Inversion Principle* - high level modules shouldn't depend on low-level modules, but instead both depend on abstractions. Our Project class depends on abstract TeamMember and Task classes rather than specific instances such as Developer/ProjectManager/RecurringTask. This makes our code more flexible since we can easily add new concrete implementations of TeamMember and Task without altering the functionality of the Project class.

From GRASP:

*Information Expert* - responsibilities are assigned to the class that has the necessary information to perform that task. For example, since Project maintains a list of members and tasks, it is the class that has the implementation of the methods to add/remove from these lists. Also, since Task

is what contains the priority state of each task, it makes sense that the getPriority() and setPriority() methods are implemented here and not in another class.