Solid/Grasp Writeup

Participating Members: Leila Kazemzadeh, Sahadev Bharath, James Gao, Adwaith Ramesh, Matthew Sebastian, Michael Zuo

Design Principles Utilized:

1. Single Responsibility Principle
2. Open/Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion Principle
6. Low Coupling
7. High Cohesion

We use the Single Responsibility Principle by making sure every class has a specific role and doesn't multitask. For example, the Project class just keeps track of the tasks, members, and important dates. The TeamMember class is also able to join or leave any project.

We use Interface Segregation Principle by making sure that our interfaces are specific instead of being generic. For example, the base interface, Task, only worries about the 4 attributes that every task should have - Title, Description, due date and Status. This ensures that classes that implement the interfaces are able to implement all the variables and methods of the interface. Interfaces like Priority and Recurring are responsible for setting Task priority and Task frequency respectively, and are separated from the Task interface for better segregation.

We use the Liskov Substitution Principle by making sure that objects of a superclass shall be replaceable with objects of its subclasses. For example, a TeamMember is like a Member that has a specific name and email. TeamLeader is like a Leader that also has a specific name and email. All the methods implemented in Team Member, take in the same argument types as their counterparts in the

Member interface, and all Member methods are implemented in TeamMember. This is the same for TeamLeader and Leader.

We use the Open/Closed Principle by using interfaces to ensure that our design is open for extension but closed for modification. For example, if we needed to create another type of task, it could implement the Task interface and possibly the Recurring and Priority interfaces as necessary rather than having to change anything in an existing class to allow for this new type of Task to implement the Task methods in its own way.

We also ensure low coupling by keeping unnecessary coupling [dependency] between modules as low as possible. We make sure most classes are self-contained. For example, TeamLeader and TeamMember implement their respective interfaces, allowing for them to not need to depend on each other or create unnecessary coupling.

High cohesion is achieved by ensuring that all methods in each class are functionally cohesive. For example, TeamMember has the addTask, joinProject, and leaveProject methods, but is not connected to unrelated methods like assignTasks that other classes would be better suited to hold.

The Dependency Inversion Principle, where high level modules should not depend on low level modules and instead on abstractions, was adhered to. The class Project has dependency with the interface Task, but does not know anything about the types of tasks that implement it.