OO Design Rules

1) Design diagram should be in line with the project specification. Add-on classes/plugins can be applied if the relevance of it is seen in the application

2) Dependency is good, but cyclic dependency should be avoided

3) Defensive programming and adherence to good OO programming should be followed

4) Design should be language independent

5) Ensure modularity with the help of classes connected either using Is-A , has-A or Uses relationship

6) Given a choice between adherence to OO way of programming vs ease of programming , always prefer adherence to OO way of programming

7) Given a choice between abstract class/interface/concrete class , prefer an interface in the context of encapsulation/abstraction of implementation details

5) Given a choice between Is-A vs Has-A/Uses , prefer Has-A/uses dependening on the scenerio(reusability)

6) Prefer Is-A relationship when extendability is our need

7) Ensure High Cohesion and low coupling

Cohesion

- achieve the level of clarify in the design with the help of modularity
- sharing of the existing instances and its state to other instances

8) Ensure Refactoring / Pull-Out Refactoring rule of design

Refactoring - changes made in the server code implementation should not enforce the changes to happen in the consumer code implementation

9) Eliminate or reduce class proliferation

class proliferation - The number of classes in the design being more than the number of relevant nouns as identified in the project specification document

10) Reduce conditional delegations

Design Principles (SOLID)

S - Single Responsibility Principle

The Single Responsibility Principle represents a good way of identifying classes during the design phase of an application and it reminds you to think of all the ways a class can evolve. A good separation of responsibilities is done only when the full picture of how the application should work is well understand.

O - Open Closed Principle

The Open Close Principle states that the design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code. The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged.

L - Liskov's Substitution Principle

Liskov's Substitution Principle states that if a program module is using a Base class, then the reference to the Base class can be

replaced with a Derived class without affecting the functionality of the program module.

Derived types must be completely substitutable for their base types.

I - Interface Segregation Principle

It states that clients should not be forced to implement interfaces they don't use. Instead of one fat interface many small interfaces are preferred based on groups of methods, each one serving one submodule.

D-Dependency Inversion Principle

High-level modules should not depend on low-level modules. Abstractions should not depend on details. Details should depend on abstractions.