## SRP, the Single Responsibility Principle

Meaning: A class should have only one reason to change.

In MVC, every class belongs to one of the three parts: one part is the m for model which stores the data model, the second part is the v for view which is the user interface and the last is the c for controller which manages the logic or business part, handles requirements and so on. Among each part we separate each class according to the objects they are going to deal with. In a word, in our design each class provides a specific server to only one kind of object which may be a set of data or an actor. For example, the use case “Sign In”, the user wants to sign in. The controller use the function “encode” to encode the password, then the controller use the function to check the user with the encoded password and username. In the controller’s “check” function, it will use the function “check” in the class “UserInfo”. If everything is right, the user signs in successfully.

## LSP, the Liskov Substitution Principle

Meaning: Subtypes must be substitutable for their base types. If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged. The objects which is in the same inherit architecture should have behavior features.

In the Wave, the design follows the LSP. We use the “Abstraction” and “Polymorphism” in the design. In the design, there are interfaces and abstract classes, which reflect the “Abstraction”. Some classes realize the abstract class or interface. This is the reflection of “Polymorphism”. For example: in the use case “Manage Org Information”, there is an interface named “Execute”, which has the operation named “execute”. This is the reflection of “Abstraction”. Three entity classes realize this interface. Each entity class has an operation named “execute”. In the runtime, the program will decide which “execute” to use. This is the reflection of “Polymorphism”. The example descript above gives expression to LSP.

## DIP, the Dependency Inversion Principle

Meaning: High-level modules should not depend on low-level modules. Abstraction should not depend on details.

Let us see this principle this from a lower level: the language we use and the design. We chose to use C# as our programing language which provides interface and all the features that object-oriented programming need. We all know that interface which represents a higher level of abstraction is more stable than implements. We designed many interfaces and many classes are just their implement. Thus strategies and implements details depend on interface which is a higher level of abstraction and stable. If we want to follow this principle, we should analysis our system to find which part is the easiest to change and in which part; it is always stable in a layer system. According to our system, the user interface is easy to change rather than business logic or something else. The data and user interface has already been separated in MVC framework. With the help of ASP.NET MVC 2, we can even change the style and the way the data is represented using Master model. With these two levels of abstraction are provided, details, here is the user interface are free the change without affecting low-level modules.

## ISP, the Interface Segregation Principle

Meaning: Clients should not be forced to depend on methods that they do not use.

This principle says that in the interface, there can’t be unnecessary methods. That is every method in the interface is implemented in the classes, which realize this interface. The design of the Wave follows this ISP principle. For example: in the use case “User Account Management”, there is an interface named “Account Management”, which has two methods “modify” and “delete”. The controller class realizes this interface. This class also implements the two methods of the interface. There are no used methods in this interface. We check the whole design all the interface follow this LSP.

## OCP, the Open-Closed Principle

Meaning: Modules should be both open (for extension; adaptable) and closed (the module is closed to modification in ways that affect clients).

As is described above, we should follow the OCP principle. When we want to extent some classes or use cases, we need not to change the original code. The Wave follows the OCP principle. For example: we use the strategy pattern. As is said in the PPT, strategy pattern is both open and closed. In the use case “Sign In”, there is an interface named “encode”, which has a method named “encode”. There are two classes implement this interface. This is the strategy pattern. So this is the example of OCP.