Принципы **S O L I D**

**SRP** Single Responsibility Principle – принцип единственной ответственности

**OCP** Open-Closed Principle – принцип открытости/закрытости

**LSP** Liscov Substitution Principle – принцип подстановки Барбары Лисков

**ISP** Interface Segregation Principle – принцип разделения интерфейсов

**DIP** Dependency Inversion Principle – принцип инверсии зависимости

**Принцип единственной ответственности**

1а - Модуль должен иметь одну и только одну причину для изменения. 1б - Модуль должен отвечать за одного и только одного пользователя - группу пользователей (бухгалтерия, DBA) или заинтересованное лицо (группу лиц)

Проявляется на двух более высоких уровнях: на уровне компонентов – принцип согласованного изменения (Common Closure Principle; CCP) и на архитектурном уровне - принцип оси изменения.

Модуль – связный набор функций и структур данных (возможно, файл с исходным кодом).

**Принцип открытости/закрытости**

Программные сущности должны быть открыты для расширения и закрыты для изменения.

Расширение внедряется либо через наследование, либо через использование абстрактных интерфейсов и полиморфизм.

Система делится на компоненты с иерархией. Верхний уровень – бизнес-логика. Затем контроллеры. Затем презентаторы и база данных. Презентаторы для представлений – более высокий уровень. Компоненты на верхнем уровне защищаются от изменений на нижнем. Зависимости между компонентами должны быть направлены от нижнего уровня к высшему. Для изменения направления зависимостеей и скрытия реализации высшего уровня добавляются интерфейсы. Например, шлюз данных в компоненте бизнес-логики – интерфейс для базы данных.

Чем меньше изменений надо внести для расширения возможностей, тем чище архитектура.

**Принцип подстановки Лисков**

Есть два подкласса, которые есть производными от базового. И есть приложение, использующее метод базового класса по разному реализованный в производных классах. Если поведение приложения не зависит от использования любого подкласса, то дизайн соответствует принципу подстановки.

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом.

**Принцип разделения интерфейсов**

Клиенты не должны зависеть от методов, которые они не используют. Слишком “толстые” надо разделять на более мелкие и специфические. При изменении метода не должны изменяться клиенты, которые этот метод используют.

**Принцип инверсии зависимостей**

К