# Solid principles in layers architecture

## Single responsibility

Since each layer is grouped logically, we can conclude that related responsibilities will be grouped in the same layer. For example, data access layer has the responsibility of fetching data and saving data.

## Open-closed principle

If abstract interfaces are defined between the layers, then this principle is respected: we may change the implementation of a layer (such as store objects in a file instead of a database) and all of the other layers should not be changed.

In this scenario, we may easily extend (add another from of representation of data -> open, but do not change existing layer implementations of the interface -> closed.

## Liskov substitution principle

Layers architectural pattern do not guarantee that subclass will be able to replace their superclass.

## Interface segregation principle

It helps design better interfaces, since the responsibilities of each layer are predefined. If you don’t mix business logic with data access then the clients which use the interface will not depend on both of them, but only one.

## Dependency inversion principle

If the interfaces between layers are well defined, details will depend on them and there will be no need for high level layers to depend on level layers.

# Grasp principles

## Coupling

Coupling between layers will be lower, since there is a clear separation between the

## Reuse/Release equivalency principle

If you want to reuse a class, you may reuse the whole layer.

## Common Reuse Principle

Since common functionality is grouped most probably classes that should be reused together will be reused together.

## Common Closure principle

A no change should ripple trough the whole application, layers help to localize change by grouping related functionality.

## Acyclic dependency principle

No circular dependencies should be present. Layers should be