**Layers according to SOLID and GRASP Principles.**

***GRASP*** helps us in deciding which responsibility should be assigned to which object/class.

**Principles**:

* ***The Creator*** decides who can be creator based on the objects association and their interaction.

***The Creator*** principle can be applied inside the **DAO** layer. When working with our entities, we decide which one is the container object and which is the contained one.

For example, when creating a *Video store*, we also do have to create inside that entity a *Video* entity (as a video store has many videos).

* ***The Information Expert*** says to assign those responsibilities to an object for which that object has the information to fulfill that responsibility.

***The Information Expert*** principle can be applied in the Business Layer (which implement the core functionality of the system and encapsulates the relevant business logic) level up to Data Access Layer.

Example: A service which implements the functionalities of the application, needs to have access to the data to be manipulated. If we want to rent a video we need to have access to the entities involved (video store and video).

* ***The Controller*** deals with how to delegate the request from the UI layer objects to domain layer objects.

Example: requesting a list of all the available videos from the video store.

* ***Coupling*** is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is an evaluative pattern that dictates how to assign responsibilities to support:
* lower dependency between the classes,
* change in one class having lower impact on other classes,
* higher reuse potential.

***SOLID PRINCIPLES***

***S – Single Responsibility Principle***

The idea is to focus on a single class, the smallest part of any application logic, so one class can have only one reason to change.

Applying the single responsibility principle in the layered architecture, means to focus on the connection between the business logic layer and the data logic layer.

The way we achieve this in an application is creating a class which handles the functionalities of only one entity. (A StudentBLL handling the StudentDAO).

***O – Open/Closed Principle***

***L- Liskov’s Substitution Principle***

These two principles will be applied at all layer levels.

***I - Interface Segregation Principle***

It consists of the fact that all the interfaces that we create are responsible for the minimum functionality. This principle is applied at the Business Logic Layer, where we implement the functionalities of the entity.

For example, we should have an interface containing the methods specific to a certain entity (a student which can create an account or see exam dates etc.)

***D - Dependency Inversion Principle***

This principle follows directly from the so-called layered application development. Code of the layers lying above may not be dependent on the code of the lower layers. But both layers are dependent on abstraction. For example, the business logic layer should not depend on the data access layer. We should make them depend on an abstract class.