## Grasp

### Controller

Our system actually has 3 controllers, one for the offline GUI called Controller, and another one also for the offline GUI called ServerController which is used for the sync methods to the server. The last controller is actually the SliceOfPieService, which is used by the WebGUI class in our system. Our controllers delegate the assignments needed to be carried out when something is queried from the GUI, they serve as a middle-layer classes between our program-logic and the GUI.

### Creator

The creator in our system is the class Storage, which is the one that reads from the file system and instantiates new projects with folders and documents inside, so it has all the initializing information that is needed to construct the objects and pass it on to the rest of the system.

### Indirection

We use the term of indirection, in the way that our GUI, calls to the storage class, but through our controllers. So if we change something in the storage, we don’t have to touch on the GUI parts, since we can just change some parameters in the controllers. That way we make sure that our classes are very loosely coupled.

### Information Expert

Our Expert is the same as the Creator, the Storage class. It holds all the information needed to actually create Documents, Folders, DocumentStructs and Projects, so the classes calls the Storage methods with the correct parameters and it creates the objects, because it has the information needed.

### High Cohesion

Our system practices high cohesion, we don’t have any classes with different responsibilities, rather they all have a certain job to perform, without overlapping with the other elements in the system, a class takes care of the problem or delegates the problem to the appropriate element to handle it. We think that our classes make sense and are well understandable by the way we have named them to represent what their responsibilities are.

### Low Coupling

We believe that our classes exercise low coupling. We have planned our classes in such matter that they have been encapsulated well, and are easy to change without having to change the entire system.

### Polymorphism

We use polymorphism widely in our composite pattern, since we often produce code where we just ask for an IFileComponent, because it could both be a Document, DocumentStruct or a Folder. This meaning that when we have a Folder and want all the children of the folder we can return a list of the folders children as IFileSystemComponents.