Comp 2005 Group 3 Iteration 4 – Use of Patterns

Design patterns are tried and tested solutions to problems that usually allow for good code structure and good design qualities. These design patterns consist of: Information Expert, Creator, Low Coupling, High Cohesion, Controller, Indirection, Polymorphism, Protected Variations, and Pure Fabrication. In terms of the pattern of Information Expert, this was almost done automatically as we thought about what the classes were supposed to do. For example, we allowed the Board class to be in control of placing the shaped pieces onto the board and while doing this, delegated the changing of the state of the board blocks to the class BoardButton. The main patterns that would need to be looked at I believe would be that of Low Coupling and High Cohesion. Both patterns help with the manageability of the code and allow for easier understanding. In our Blokus implementation, we started out by just trying to get something working not thinking about these patterns. Because of this, we have many classes that work, but consist of logical components, UI components, and many different methods that aren't necessarily cohesive regarding said class. While we continuously worked towards the decoupling of classes towards the end of project, there is still a lot of work to be done to have the ideal coupling. If we had more time to fully think through what classes we needed and what each class would do, it would help to make the code easier to be updated, and less dependent on certain classes. With regard to the other patterns, some of them just aren't entirely relevant to this group project and the game of Blokus. Some things that we could have done however is use the Pure Fabrication pattern to create a class that connects to a database of save states. In our implementation now, only one save state can exist at a time. If there was more time however, we could've implemented a class that saves the states of all objects involved in the game and connects it to a database where that information is stored to be able to retrieve it when needed. We do have a Purely Fabricated class now, "Save Manager", but it overwrites previous saves. All patterns are important and are great solutions to common problems, however, looking back on the project, the patterns that would've needed the most focus would've been Low Coupling and High Cohesion. It would have been very useful to have those two patterns in the back of our heads when designing the initial structure of the code to allow for easier understanding between all group members and a relatively even responsibility between all main classes in the code.