For Assignment 2, we had highly cohesive classes and somewhat high coupling, primarily due to the large amount of classes. On the whole we avoided low cohesion and high coupling, but with a couple exceptions that we could not work around.

As far as assignment 3, we had to make some changes to our project structure to make things function. One of the main differences was splitting up our controller class into multiple smaller methods to allow better interfacing with our swing GUI and increasing the cohesion of our classes. Originally string input from the command line was passed to arguments in a switch statement, but that would create a flag coupled method which would not be optimal. We originally were pursuing adding buttons for each individual command and a command pattern esque way of calling commands. We ran out of time to fully implement this so we reverted to using OptionPanes in a somewhat late decision. This allowed us to pass in string arguments in a similar fashion to the command line version, allowing us to reuse code.

Some of the patterns that we did implement include MVC, Builder, and Singletons. Builder classes can be found with fluent setters in multiple parts of the model including the model class itself. We chose to utilize these where we were instantiating complex objects that didn't make sense to split up, things like boardLocations that have 5 attributes to set but that all relate to the location. We implemented this for any class that would have to take more than 3 arguments in the constructor, including Upgrade, BoardLocation, Part, Model, and Card.

MVC is obviously a more overarching design pattern that encapsulates the entire program, but the pattern can be seen in the classes Model, Controller, MainFrame (GUI), and CLIView. While we struggled to have a truly detachable view, we mostly avoided making changes to the model and while we made some larger structural changes to the Controller, most of the functionality was retained and most of the code was able to be reused. If we were able to refactor our program we would have liked to wrap both MainFrame and CLIView in a View interface and better define the methods. We also would have liked to make the input more detached, as the way we did input originally was very much linked to command line style prompts. Our latest revision does detach the input more, but it would be nice to implement a fully featured Command pattern because of the number of possible commands and the complexity of some commands like upgrade.

Singletons can be found with classes like Bank and Dice, where they almost act as just a set of methods. We chose to use Singletons here because we wanted to split up the functionality of admin into more specific classes, but it didn't need multiple instantiations per model. It would have also been good to use an observer to update multiple classes of changes like currentPlayer, it would have minimized a lot of redundant code.