Assignment Two
Ryan Lebeau 104535367

1. A Big Ball Of Mud can be caused by a number of factors. These “global forces” are Time, Experience, Turnover, Skill, Complexity, Change, and Cost. They can all easily lead a developer down a path of poor software architecture patterns and eventually turn their system into a big ball of mud.

2. <u>Big Ball Of Mud</u> - A big ball of mud is a term used to define a program with terrible code structure. Even if a program runs its code can still be a big mess, whether it very sloppily formatted or spaghetti strung together.
   <u>Throwaway Code</u> - Throwaway code is code that was designed to be quickly used once and then replaced. More often than not though, throwaway code can get embedded into structure and forgot about until it’s too late.
   <u>Piecemeal Growth</u> - Growth that inevitably takes place in a successful system. These changes can slowly erode away the once neat structure of the system and create a big ball of mud.
   <u>Keep It Working</u> - This strategy of code repair sections off the program so the main parts of the program can continue to run, while side features are fixed. This method attempts to conserve some kind of usability of the system while repairs are being done.
   <u>Sweeping It Under The Rug</u> - As a systems code becomes worse and worse, a simple way to ignore the decline is by putting an attractive facade on the error. In more specific terms, a work around can be created to avoid the error temporarily without solving the original problem.
   <u>Reconstruction</u> - When a program or systems code becomes so bad that there is no way to attempt to repair the many problems. The only option is to rebuild the code from the ground up only salvaging the underlying pattern.

3. <u>Throwaway Code</u>
   Throwaway code is meant to be a temporary quick solution to a problem but iis not ever changed or updated to proper code. The best way to identify throwaway cod eis to analyze the efficiency of code within the program. Since throwaway code is done very quickly to solve the problem with much thought given to time complexity, the quick trick to identifying it is locating inefficient code. Avoiding throwaway code use can be tough in the moment since a time restraint is the cause of this, but making sure to go back and properly solve the problem is at least the best option to avoiding throwaway code related problems. Having a door in your home that doesn’t close properly and using a wedge to hold it closed is a perfect example of throwaway code. It is a quick solution to a problem that may require a lot of time to fix (replacing the door), and even when time becomes available the quick fix may be looked over for far longer than it should be.
   <u>Sweeping It Under The Rug</u>
   Sweeping a problem under the rug is a very simple solution that everyone has been guilty of at one point or another. Whether this problem be something simple like

creating an “organized” mess of clothes in your room instead of doing laundry or quickly working around a problem in your code with a disgusting temporary fix. The best way for identifying where a problem was swept under the rug is locating functions that loosely coupled to the rest of the program. After locating the problem, the main solution is quarantining this section of code away from the main program and trying to rework the initial problem. To avoid sweeping something under the rug the developer has to be willing to spend the time to properly fix their code in the first place instead of working around and hiding the original problem, which is sure to cause problems in the future (either with runtime errors or inefficient code).