Robert Florence
CS 350
Prof. Wasniowski

HW3

This critique focuses on design patterns and whether or not they are good for software design and use by programmers in general. Both authors have very valid points on both sides of the argument. James Noble argues that every good designer is able to break down the problem into segments and be able to apply some sort of long-standing pattern to solve the problem using tried and tested methods. To use an analogy, James would argue that a good software designer could design a program in such a way as to fit any problem into the basic cookie-cutter solutions and make them work accordingly. On the other hand, Peter Sommerlad argues that good programmers should not rely on old solutions to new problems. He argues that each programming problem is unique and requires each programmer to think everything through and solve it individually. He also says that the tired, old, patterns are from a more simplistic time of programming and they do not encapsulate the true complexity of more modern software design. Peter also argues that using patterns for software design leads to unnecessary complexity which can lead to more issues rather than solving them.

I believe that, like most things, theres no single, correct answer on one side or the other, rather, an average programmer should follow a policy somewhere in the middle between the two. I believe a good foundational programmer can combine his knowledge of patterns whilst being able to analyze certain programming problems from the ground up and building solutions accordingly. On one side, these patterns have been successfully used for about thirty years for a significant reason, which is that they simplify solutions into a cookie-cutter logic thats easy to follow. In that sense, knowing how to use these patterns and being able to solve problems individually makes a good programmer. One of the biggest advantages of using patterns is that because they have been used for many years by many software designers, the power of them are well understood and have a good track record of success. Design patterns are also not language specific and very flexible, so their ideas spread across the domain of software development. On the other hand, the disadvantages of relying on software design patterns makes the skills of the programmer weak, and as a consequence, that lack of expertise can result in bigger software design disasters. The use of cookie cutter design patterns doesn't take into account a lot of the programming problems specifics and could lead to inefficient, over-complex code that can lead to other debugging issues. Peter quotes Kent Beck to argue the point more to the effect of, "think more about the design instead". If I had to choose a side, I would choose Peters, agreeing with him about 65% of the time. James still has a very valid argument.