# Term Paper: Design Patterns and their Side Effects for Gang of Four (GoF) patterns

## Introduction

It is well understood that design patterns provide a lot of value in the creation and refactoring of software solutions. This statement is re-enforced with the recognition that most class libraries delivered in support of various Operating System (OS) platforms draw heavily on some very recognisable patterns to deliver core OS functionality. A very recognisable example of this would be the various collection classes in .Net which all derive from IEnumerable <T>, an implementation of the GoF iterator pattern. Many more patterns are used extensively throughout the .Net class libraries. This pattern is replicated across many class libraries and core language features as demonstrated with Observable and Singleton mixins for Ruby.

It is highly conventional to consider knowledge of design patters and their intents to be a great strength as a developer. However, as with all strengths, if they are over played, and in certain circumstances, they can switch from being a strength to being a weakness. In most structured courses on design patterns, the focus is spent on the pattern itself and when to use it. And whilst many courses will outline the consequences of each pattern, frequently this does not get the same levels of focus and attention, by design you could say.

In the following sections, we shall outline some of the GoF patterns with their intents and focus on the consequences of each pattern.

## General Downsides to Overuse of Design Patterns

Before looking at specific patterns, it could be useful to look at some of the general downsides to a pattern based mind set. One issue that immediately comes to mind is seeing patterns that don’t actually exist.