http://www.devx.com/architect/OOP-Is-Best-in-Practice.html

OOP Is Best in Practice

By Dave Herren
Jan 26, 2005

In his recent DevX editorial, Richard Mansfield proposes that "OOP Is Much Better in Theory Than in Practice." In reading his article, many of his objections seemed to be based on outdated teachings of what good OOP is. Although I can understand his apparent frustration with making the transition to OOP—the OOP taught in colleges is often a far cry from the OOP in the real world, and many of us must rely on trial and error or help from others to find our way—his arguments are hard to take seriously. As one who has overcome the challenging transition to OOP I can assure Richard that he need not struggle forever, but he first has to let go of his false notions.

Erroneous Arguments
Richard makes the following three main arguments to support his claim:

1.Encapsulation makes Inheritance difficult - i.e., overriding the methods in an inherited class if you don't have access to the underlying code can be difficult, and "Black-box" objects can hide functions critical to creating an inherited class.

While this is true, how often does this situation occur in real-world development? Most of the classes that I inherit from are internal to my company (for which the underlying code is available). Many third-party components are now shipping with their source code as well. In working with the .NET Framework, I usually need to inherit from interfaces, but seldom do I need to inherit from classes. Even when I do, reflection can at least provide access to private methods.

2.Hierarchical class inheritance is not the answer to code reusability that it was once proposed to be.

Although hierarchical class inheritance (specialization) has indeed lost favor—one of the underlying principles of the GoF's Design Patterns book is "favor composition over inheritance"—copying and pasting code as Mansfield suggests is a maintenance nightmare waiting to happen. Most of us have embraced the "code once and only once" philosophy. Class aggregation, as well as interface or abstract class inheritance-based designs, are better solutions.

3.Procedure-oriented design is less complex (and therefore better) than OOP.

Copying and pasting code as Mansfield suggests is a maintenance nightmare waiting to happen.

In response to this assertion, I ask Mansfield to simply consider the .NET Framework itself. It is a replacement for old procedure-based API libraries, which few ever mastered. While daunting, the .NET Framework provides a much more manageable organization of programming functionality. Personally, I dread having to return to API calls for functionality that was not included in the framework. Prior to .NET, many programming shops attempted to create their own OO libraries that mapped to API functionality. These object libraries were often cryptic and sometimes problematic. The problem, however, was not OOP but instead the procedure-based API libraries that were the Microsoft standard at the time and the lack of an OO Microsoft standard (which .NET now provides).

Most applications are far too complex for simple procedure-oriented programming. OOP simplifies and organizes complex functionality. Tradeoffs between complexity and a flexible design are a main consideration in OOP, but good object-oriented design provides for reasonable adaptability for changing specifications while avoiding undue complexity.

I Saw the Light, He Can Too
OOP has a lot more to offer than the author discusses. His arguments should not at all dissuade anyone from OOP. I continue to seek better and more efficient ways of programming. I have studied design

patterns, extreme programming, and refactoring. While none of these programming methodologies was a cure for all my ills, a few key principles did transform my coding:

1. Favor composition over inheritance.
2. Program to an interface not an implementation.
3. Find what varies and encapsulate it.
4. Strive for loose coupling.
5. Strive for high cohesion.
6. Code once and only once.

For anyone struggling with OOP (or for anyone who needs clarification on the above six points), I highly recommend *Design Patterns Explained: A New Perspective on Object-Oriented Design* by Alan Shalloway and James R. Trott. I wish this had been the first book I had read on my quest for good OOP practices, as it put all the pieces together for me.

Becoming efficient and comfortable with OOP is a transition that can take a little while, but it is a worthwhile endeavor. Today, I program faster and more efficiently, and I spend less time on maintenance than ever before. OOP—good OOP—rocks!