Subject: Some Object Oriented Programming Books
Posted by gambler_1650 on Thu, 18 Nov 2004 15:07:41 GMT
View Forum Message <> Reply to Message

Below is a list of 4 books that I've found useful in my relatively recent foray into 'professional programming'. Disclaimer... I gain nothing from my promotion of Four Addison Wesley books. ;)

"Test Driven Development" by Kent Beck
"Refactoring" by Martin Fowler
"Design Patterns" by Gamma, Helm, Johnson and Vlissides
"Refactoring to Patterns" by Joshua Kerievsky

I've had training (mostly self training) in languages like C++ in the past, but am now programming in IDL for the Acoustics group in NEFSC, National Marine Fisheries.

One of the things that instantly attracted me to IDL was the recent object oriented extensions, as in my previous dabbling I discovered I much prefered object oriented programming (OOP) over the procedural methods I'd learned in college. However, when I started here I wasn't a good object oriented programmer. I could hack objects together and get something to work. I read numerous books dealing with OOP and began to get a handle on some of the better ways of doing things. Then I ran across a couple of books called "Test Driven Development" and "Refactoring". Neither of these is a tutorial on how to do OOP but they have a new take on how to write OO programs. The first basically advocates writing tests first. You create the code to run the (initially) non-existent objects. This forces you to think about what interface (in OO terms, an interface is how you call a function or procedure, or even more OO.. how you send a message to an object) you want to use before you write the code. Then you write the code to the interface until the test works.

"Refactoring" was a book I picked up at the same time, mainly because I was dealing with legacy code that wasn't Object-Oriented but needed to be made so. It's basically a book that describes how to change areas of code to become cleaner, more understandable and more object-oriented. The 'bad smells in code' section is worth the price of admission (code duplication is an example of a bad smell). It tells you what to watch out for in old code, but also helps you pick up on these things while you're writing new code. The above two books together lead to a different way of designing. You write the test. You make it work no matter how ugly the code looks. When it works, you refactor into something cleaner. The key is that the tests should be small, as should the steps be when you refactor. It may seem

wasteful to be writing tests, changing code that 'works' to be cleaner, but I find it a much more natural way of writing code than trying to design the whole thing before beginning coding. The real benefit of the tests is that you find bugs more or less instantly, and since the amount of code you write for each test is small you can usually find it right away.

"Design Patterns" is a book I'd always looked at but just recently picked up. It takes problems that appear over and over in an OOP environment and gives you the patterns that have been proven to work. I've already used a couple to get me through a "ok, how the _heck_ do I solve this issue' when I'm really stuck, and now that I actually have said book in possession it's a valuable resource.

"Refactoring to Patterns" combines the above three books into an overall approach to writing code. Write the tests, make them work, refactor the code towards the patterns most appropriate.

What these books won't do is teach you OOP... Taken together they teach a whole new way of designing and coding.

Hopefully this message was of some interest to this group.

Robert Gamble
