WHY AREN'T *YOU* USING DESIGN PATTERNS

James W. Cooper

What a nightmare it was. I was in a room giving a talk on Design Patterns I'd given a number of times before, and suddenly I was speaking some other language that even I didn't understand. People started looking at me sort of oddly. No matter what I tried to say or explain, no one understood anything I was saying. Then the audience began sneaking out when they thought I wasn't looking. I kept hoping I would wake up, but no matter what I said or how many lame jokes I tried, nothing was getting across at all.

But it wasn't just a dream. It was real. I was really talking to a room full of about 50 relatively young programmers and web developers at a major firm in New York and I couldn't connect. No one saw why they needed to understand Design Patterns and no one really cared. All they wanted to do was wrangle their code out the door as fast as they could.

Well, maybe this was just an anomaly. Then, just two days ago I was signing copies of my book at a trade show, when a programmer came up and asked what it was about. I told him it was about writing Design Patterns in Java. He immediately said that that was too advanced for him. "No!" I wanted to shout. "This isn't an abstraction!. It's a toolkit of useful tested techniques." But his eyes had already glazed over.

Well, maybe not everyone is ready for Design Patterns. Yesterday I mentioned to a group of Computer Science PhD's that I was thinking of writing an article on why not everyone uses Design Patterns. Uh-oh. Blank stares.

So what is going on here? Are the *patternostra* only talking to each other? Is no one actually using them? There are probably several answers here and they can be helpful to look at, so stay with me a minute here. I recently chatted with John Vlissides (from the Gang of Four) about this, and incorporated some of his ideas into this discussion as well.

My first nightmare group was a group of fairly young and highly focused web lackeys and programmers. Their only goal was to get it out the door. Their motto is like the priorities of the Broadway music copyist: get it there, get it right, get it neat. In that order. Guess which part never gets done? Neat? Right.

Of course they were micro-focused, and would eventually discover that writing code that way was bad for their health. In

other words, they would have to spend untold hours rewriting gobs of code when simpler OO designs might have made it more maintainable. This is the age-old problem of taking some time up-front for good design, since you can't nail it on later. Their manager thought that the talk was a good one since they needed to be exposed to this sort of thing. I wonder how much time he encourages them to spend on design before they commit their thoughts to code?

The second encounter was more problematic. The prospective book reader was worried (scared?) that whatever material I had to present was too hard or too abstract for his technical level. I think it is kind of sad that someone who is a capable programmer believes that there are vast layer of complexity he hasn't grasped. There really aren't, you know. This whole gaff is a single-level scam. You've got the jargon, you own the boat. You're not going to sink. No need to worry. This stuff isn't that hard.

And then there were the 4 Ph.D's. What's with them? Maybe having the degree makes you less secure. Certainly passing a Ph.D. oral makes you know for sure that there are things you don't know. But I think the problem is really one of risk aversion.. You know how to write programs that work. Why learn another way?

Does Anyone Use Design Patterns?

You probably know that Java itself is rife with Design Patterns. Even if you don't consciously include them in your programming, they are there under the covers, thumbing their little noses at you. You'll find factories, templates, command patterns, proxies, adapters and bridges all through Java. Swing classes are particularly infested with them. Since nearly everyone I talked to was using Java, they were certainly using these patterns

Among my closer acquaintances, a lot were using some packages I'd written for manipulating text data and databases. And guess what? I used a lot of design patterns when I wrote these classes. So everyone was using design patterns at one or even two levels, and they didn't even know it.

There are some programmers who probably don't need to use design patterns, though. These are the web lackeys who don't write much besides JavaScript and HTML, and the code maintainers who simply add small changes to existing code bases. However, if the original code were well designed using a few patterns, their job would be easier.

And what about the experienced computer scientists? If they are writing good OO code, and most are, they may well be using some patterns without knowing their names. They may even have developed a few of their own. But there remains a problem here. If they don't know their names, they can't share their coding strategies effectively with others. They can share their *code*, but they can't tell others how it works. And this is one of the reasons for spending a day or two learning about patterns. They provide a convenient high-level shorthand for you to describe how your code actually works.

Do You Use These Patterns?

So OK, which ones are they using and which ones might you be using. If I tell you the names, will you remember them? Give it a shot. Maybe even read a page or two on each of them, and more later if you get interested. The **Adapter pattern** provides a class with an interface between your classes and some other classes that have an inconveniently different interface. Think of the MouseAdapter and WindowAdapter.

- Whenever you write a parent class and leave some methods to be filled in in child classes, you are using the **Template pattern**. It formalizes the idea of defining an algorithm in the base class and leaving some of the details for the child classes.
- You are using the **Factory pattern** if you have a class which decides which of several subclasses to return to the user based on some input data. In more subtle versions, different subclasses of one hierarchy contain different implementations of a class from a different hierarchy.
- You are using the **Command pattern** if you have an interface or class with one or two simple methods like Execute and Undo, and each subclass is used to carry out a different user command.
- You are using the **Mediator pattern** if you have a central class which dispatches method calls to several other classes depending on what you tell it you want to do. It hides the fact that several classes might be involved in carrying out your user's needs.
- You are using the **Proxy pattern** if you have a simple class that stands in for a more complicated one which takes longer to instantiate. The local stub classes generated for use by Java RMI are examples of this, as are local standins for connections to databases.

?? You are using the **Façade** pattern if you write a simple wrapper class to surround some complex array of classes that are too hard to use all the time and you need a simpler interface to them. These are commonly written for database and file access.

?? You are using the **Observer pattern** if you have a data repository and two or more visual representations of that data, such as grids and plots, which both change whenever the data changes.

?? You are using the **Builder pattern** if you have code that constructs a final set of classes from a set of components. This often is used to construct a variable user interface based on differing kinds of data.

?? And finally, you are using the **Iterator pattern** any time you use an Enumeration or Iterator class.

So what did I say? You're probably using them, too. If none of these sound familiar, think a little more about your OO design. If some sound familiar, then just learn the jargon to impress your friends and you'll find you have a new shorthand way to talk about your work. Maybe it will even rub off on those who had feared they couldn't understand these simple tools and tricks. Hang in there and keep using patterns. Remember, magicians and shamans believed that there was great power in just being able to name something. Bring some magic into your life.