DO DESIGN PATTERNS REALLY HELP ?

James W. Cooper

It's just like a political campaign. They get up there and say the same comforting things over and over. Nothing you haven't heard before. *Heard* indeed! Maybe it's just a *herd* mentality. Or maybe as Gilbert said, "The prospect of a lot of dull MP's, in close proximity. / All thinking for themselves is what / no man can face with equanimity." Does this apply to patterns? Are we all assuming they are the magic potion or feather, but without any proof? If patterns are the answer, what was the question?

You've probably heard a lot about Design Patterns, both from me and from lots of the other *Patternostra*. Intuitively, they seem to be elegant ways of describing useful reusable *designs*. But has any one actually tried to see if they make a difference?

Well, I recently heard a talk by Professor Walter Tichy from the University of Karlsruhe, who described work with his colleagues Lutz Prechelt and Barbara Unger, and others. They tried to devise and carry out some experiments to find out if the knowledge of design patterns really helps programmers write better code.

The Exam Experiment

The experiment they set out to measure is whether programmers write better code if they know that design patterns are being used than if they do not. This is to a large degree a real Heisenberg problem, because it is so difficult not to perturb the system you are trying to measure. With that caveat firmly in mind, they set up a final exam situation in which students were given two programming problems to solve. They were given well-commented programs and asked to describe how they would make a modification to the programs. They did not have to write working code, only describe the methods and classes they would add.

In one case, they were told the names of design patterns contained in the base program (in a few additional comment lines), and in the other case they were not. *Otherwise the programs were the same*. The students were divided into four groups. Two of the groups were given problem A with the additional comment lines showing the names of the design patterns (call it Ap) and problem B without the design pattern comments. To avoid a learning effect,

one group got Ap then B and the other group B then Ap. The other two groups were given A then Bp or Bp then A.

As background, all of these 74 students had been given an intensive 6-week course in Java programming, in which they learned about the Composite, Observer, Strategy, Template Method and Visitor patterns, and had assignments in which they practiced using these patterns. The students were uniformly distributed into the 4 groups by their class grades at that point in the course.

Then, before the exam questions were given, the students were asked in a pretest, how well they believed they understood design patterns, and asked in 6 specific programming cases, which pattern would be most appropriate.

The two exam problems utilized the Template Method, Observer and Visitor patterns. One was a simple program for reading in and displaying phonebook style information using a Template Method and Observer pattern, and the other a program for walking through a tree of And/Or nodes using a Visitor pattern. The students took 2 to 4 hours to complete the exam.

In the case of the relatively simple phonebook program, there were no significant differences in the number of correct solutions, but the subjects who were told that a pattern existed solved the problem significantly faster. For the harder tree problem, the number of correct solutions was significantly higher in the group told about patterns. Further, pattern knowledge allowed even the less talented students to produce a correct solution.

I think these results are pretty significant and in themselves tell us a great deal about the value of patterns. However, you could argue that since they didn't actually write complete, working programs they weren't really doing software engineering but only simple puzzle solving.

To try to measure this same information in a context where actual working programs were required, they investigators then joined with Douglas Schmidt at Washington University. Here they performed a similar experiment on a class of 22 students in a C++ programming course. The example programs were, thus all written in C++ and commented as before. The students in this course had to implement their solutions on Unix workstations. About the same number of students came up with the correct solution to the And/Or tree (4 of 8 with and 3 of 8 without). However, the mean time to complete the programs was significantly shorter (52 versus 67 minutes). So again, knowing about design patterns helped.

Looking at the Experimental Design

Before you criticize these experiments as too complex, too simple, too easily perturbed, and so forth, you should read the papers they have written on these experiments. Take a look at Prechelt's web site . Not only have they subjected their data to exhaustive statistical analysis, they have thought through all kinds of possible objections to the experimental design. For example, did the students who were not told of the existence of patterns receive enough information? Was the experiment "fair"? To answer this one, the programs were extensively commented as to the algorithms and at the method level. The only difference was a comment like the following:

```
// **Design Pattern**
// The two displays are registered as Observers
```

You might also be concerned that these problems were too simple and not representative of the kinds of problems working programmers actually encounter. Further, the subjects were students, not experienced professional developers. Might this skew the results in some way?

How did Professionals Do?

One way to find this out was to take a set of professional developers and perform more or less the same experiment. They studied a set of 30 professional developers of varying experience levels at a single software company. They gave them a pretest, gave them a short course on patterns and then gave them a post-test. The tests were carried out on paper, without the need to write complete working code.

They found that in the pretest, developers carried out a module (non-pattern) solution faster than a pattern solution. After the course, some of the developers carried out the pattern solution faster than the modular solution, and that the speed of the modular solution was unchanged from the pretest. And none carried out the pattern solution more slowly. They concluded that patterns in general can help and never hurt.

Communicating About Patterns

Now, remember that the name of a thing is what gives you power over it. If you can provide the name for a pattern in your code, you can tell others about the system in a more efficient way. So is this just a hypothesis, or is it borne out by actual experience**?**

In order to test this, Barbara Unger carried out an experiment using 2-person teams of programmers. She gave one of the programmers a program description and a maintenance task. This first programmer then became the "experienced" programmer, who was then to describe the problem to the other "novice" programmer, and then the two of them were to carry out the solution.

The experiment compared the teams' interactions both with and without pattern knowledge, and studied how they communicated about program design. The teams were given one of the two problems as a pretest. Then they were given a 3-month lab course on design patterns and then they were given the other exercise as a post-test. Some students got problem A before the course and problem B afterwards and others got B, then A. The expert and novice roles were also switched. One exercise used the Observer, Bridge and Singleton patterns and the other the Composite, Visitor, Chain of Responsibility, Observer and Singleton.

Unger measured the communication between the two programmers, looking for questions, use of atomic assertions, and design pattern terminology. She also measured who dominated the discussion and found that, as you would expect, the experienced programmer dominated the discussion at first, and continued to dominate the discussion in the pretest experiment. However, in the post-test experiment when design patterns were now part of their shared vocabulary, the junior team member took a more active role in the discussion and the time each spent speaking became more equal. This took place regardless of how successful the team was in the proposed programming task. This clearly makes the team's communication much more effective. In addition, in teams where the novice dominated in the pretest, even though the novice was considerably less informed, this domination by ignorance did not take place in the post-test. In fact the "ignoramus effect" was completely muted. After the training, the "ignoramus' was no longer ignorant and participated without dominating. This, too is a powerfully compelling result, and one we would all like to see more frequently.

Conclusions

This work seems to provide some convincing proof that design patterns really do help you write better, faster, more correct code, and help in communicating your work to others. I not only use patterns in my daily programming, I tell others they are there. Try it and see if it helps you, too.

References

1. Lutz Prechelt's papers: http://wwwipd.ira.uka.de/~prechelt/Biblio/
2. Barbara Unger and Walter F. Tichy. "Do Design Patterns Improve Communication? An Experiment with Pair Design," http://members.aol.com/_ht_a/GEShome/wess2000/unger-tichy.pdf