Do Design Patterns Create More Maintainable Systems?

James M. Bieman
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
bieman@cs.colostate.edu

May 19, 2000

Object-oriented design patterns have captured the attention of a great number of software professionals. I find a copy of Gamma, Helm, Johnson , and Vlissides's ("The Gang of Four") popular book on design patterns [1] on the bookshelf of nearly every software developer that I visit. There are now dozens of design pattern books, and several conferences dedicated to design pattern derivation use and/or analysis. The Gang of Four were "honored" by a mock trial at OOPSLA'99, which attracted more than a thousand "witnesses".

Many software developers clearly like the notion of patterns; they are an intuitively appealing design mechanism. Yet, there is little evidence beyond intuition and anecdotes that demonstrates the value of this new design technique. As a scientist, I am by nature a skeptic and seek evidence of the benefits and costs of using design patterns.

Design patterns are idioms for structuring object-oriented software. They provide guidance on how to interconnect components, usually classes, rather than how to build individual classes. I do find the intuitive arguments compelling and, at my institution, we now introduce design patterns in several of our software development courses.

The use of a design pattern actually complicates a design — you must generally add abstract classes and additional associations to employ a design pattern. The key benefit of using a design pattern is that the resulting system should be easier to adapt — you will need to modify fewer classes to add functionality to a system that makes use of design patterns. Thus, maintenance programmers should have to expend less effort to modify these systems.

Do design patterns really improve the maintainability of a system? We are trying to answer this question by investigating the evolution of software systems that make use of design patterns. The goal is to determine the effect of particular design structures on future versions of systems.

We are now in middle of a study of 40 versions of a commercial software system consisting of 230 classes and more than 32,000 lines of C++ code. One objective of this research is to demonstrate the benefits of the design patterns in this system. The operational hypothesis representing this objective is the following:

> Hypothesis: Classes that play roles in design patterns require fewer changes than non-pattern classes as the software evolves.

Research supported by a grant from the Colorado Advanced Software Institute (CASI). CASI is sponsored in part by the Colorado Commission on Higher Education, an agency of the State of Colorado.

We examined the design of an early stable version of the commercial system and identified the classes that played roles in patterns. The patterns were fairly easy to identify through system documentation. We counted the changes made to each class over the next 40 versions of the system. Each change was represented by a check-in on the version control system used during development.

Our results do not support the hypothesis. The classes that play a role in a pattern were among the most change prone classes in the system. They were modified more often than non-pattern classes. This result was significant at the 95% confidence level using a T-Test.

This result directly contradicts our hypothesis, which represents conventional wisdom. However, we are not yet ready to advise people to abandon design patterns. Our skepticism extends to our own initial results.

There are additional factors that may affect the internal and external validity of the results. Internal validity is concerned with the causality of the relationship between variables. Pattern classes appear to represent key classes, and key classes may require more changes. Also, we have not completed an analysis of the kinds of changes, and we may find that corrective maintenance dominates the changes to pattern classes. It may be that patterns make only adaptive maintenance easier.

Concerning external validity, our results may not generalize to organizations and/or application domains outside of the one that we studied. The development team that developed the system was fairly new to object-oriented methods, and certainly to using design patterns. We may get different results when we study the evolution of software developed elsewhere or in different projects.

In spite of any threats to the validity of these initial results, we are quite surprised that pattern classes required more rather than less maintenance effort. It is clearly worthwhile to question conventional wisdom. Empirical research may either support or refute what many of us take for granted.

References

[1] E. Gamma, R Helm, Johnson R., and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading MA, 1995.