

Book Review

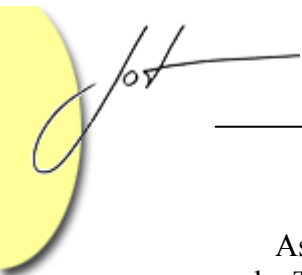
Object-Oriented Reengineering Patterns

by Serge Demeyer, Stephane Ducasse and Oscar Nierstasz, Morgan Kaufmann Publishers, San Francisco, CA, 2003. 282 pp. \$59.95(hardbound). ISBN 1-55860-639-4.

Reviewed by Charles Ashbacher

The authors define a reengineering pattern as one to be globally applied to a large, functioning system that needs to be improved. Their formal definition is “Reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.” Reasons for improvement could be necessity due to poor performance or just the next iteration of the upgrade cycle where major changes are necessary. In any case, the intent is to perform substantial modifications to the code, generally all the way down to the basic design. Their emphasis is not on converting legacy systems without objects to one that is object-oriented. These patterns are used to convert object-oriented systems into systems that are still object-oriented, but where the implementation is more efficient, either in speed of execution or ease of maintenance. In that sense, the book is timely, as object-oriented programming has now been around long enough that the early systems are beginning to show signs of code rot. Furthermore, our understanding of object-oriented programming has matured a great deal in the past several years, and for many it is time to take advantage of this knowledge.

The start point is expected, you begin by setting a general goal, which imposes a generic direction. This involves determining what the inadequacies are perceived to be as well as the expectations for the reengineered product. Once this is done, the next step is a feasibility analysis, which involves the following patterns: Read All the Code in One Hour, Skim the Documentation and Do a Mock Installation. While these patterns are not necessarily to be taken literally, they are very sensible. The source code that currently exists may not be the original, so the most logical first step is to attempt a compilation and install operation. Nothing could tell you more about the seriousness of the potential problems than having an attempted compile fail with a number of errors that exceeds the limits set on the compiler.



Assuming that works, the next step is to perform an hour-long scan of the source code. The idea here is not to read it all, but to examine enough of it to get a sense of how well it is put together. This time limit is of course somewhat arbitrary. It may only take you ten minutes to realize that the code was written by deviants. Finally, a cursory examination of the documentation will help you determine if it is to be of any use. This is the point where you must pay the greatest attention, so it may take longer than the other two. Although the documentation may be wonderful, it is necessary for you to read it in conjunction with the associated code, to verify that the two are synchronized. In this case, one may simply want to randomize the examination in some way, and then probe the selected sections in great detail.

Once you have performed the previous tests without running away in terror, it is necessary to begin the changes by applying more specific patterns. The first set of specific patterns are used to capture a detailed model of the system, followed by the construction of tests, migration strategies, how to detect duplicated code, redistribute responsibilities, and transform conditionals to polymorphism. The last three are standard refactorings, which shows the movement from general strategies to the more specific. However, the authors are still operating at the system level, so the patterns are more general than refactoring. For example, the patterns on duplicated code are how to identify duplicated code rather than the mechanics of how it is removed.

The patterns are described using the structure: The name of the pattern and the problem(s) it addresses, the solution, the list of trade-offs separated into the pros, cons and difficulties; an example, the reasons for applying the pattern and other patterns that are related to it. I found this approach to be refreshing, as many authors give solutions without pointing out the problems their solutions can generate. In general, the patterns form a set of sound advice, but it is up to you to make the transition from the system level to the specifics of code change.

If you are faced with a major reorganization of a large project, then I wish you well. However, like these authors I will do more than just give you sympathy and encouragement. My task is much easier, in that I will simply encourage you to read this book. They did all the hard work of constructing a plan, and you would be wise to listen to them.