PHP OBJECTS, PATTERNS, AND PRACTICE (SECOND EDITION)

Matt Zandstra, Apress, 2007, \$44.99,

ISBN: 1590599098.

This is yet another book on PHP, but reading it will bring a number of pleasant surprises for many readers.

From the beginning, it's clear that the book is not targeted at beginners. You need a good knowledge of both PHP and Web development in general to read and understand it. That said, author Matt Zandstra has done a great job of writing far more than just a PHP manual.

Developing in PHP can be extremely fast, often at the expense of the maintainability and security of the code developed. The first part of the book focuses on object-oriented programming, providing readers with a good overview of how to stay away from jigsaw PHP development and use object-oriented programming effectively.

The second part of the book is not a light read; it digs deeply into software engineering concepts that are often left aside in PHP books. This is one of the more theoretical sections of the book. While it often fails to be readable enough, it can be helpful if PHP is one of the first programming languages you studied.

The third and final part of the book is focused on practice. It discusses PEAR (PHP Extension and Application Repository), documenting the code, versioning, testing, and building. It's quite focused on PHP-specific tools rather than on more general tools; the reader will have to decide whether this is a downside or an advantage.

This book will give readers an opportunity to become more familiar with the advanced aspects of PHP, look at software engineering methods for Web development, and get a feel for tools that are, or at least should be, well known to experienced developers. —Andres Baravalle

CLEAN CODE: A HANDBOOK OF AGILE SOFTWARE CRAFTSMANSHIP

Robert Martin, Prentice Hall PTR, 2008, \$42.99,

ISBN: 0132350882

As Martin Fowler once wrote, "Any damn fool can write code that a com-

puter can understand, the trick is to write code that humans can understand." Otherwise, we could still be writing code in assembly, and nobody would really complain. Since maintenance takes up a big chunk of the software development budget, however, it is not enough for code to work—it must also be clean.

What exactly do professional programmers mean by *clean code*? Martin conducted a small survey, asking some renowned programmers for their thoughts on the subject. In this book's introduction, you will find their responses. Once the stage is set, the book proceeds in three complementary directions, forming the three parts of the book.

The first part covers the principles, patterns, and practices of writing clean code, from the point of view of Martin and six of his colleagues at Object Mentor Inc. These chapters are written in an opinion-column style and are to be read more as informal blogging rants than as the final word on the topic. While you might not agree with everything they say, you will still find many commonsense coding guidelines and some witty aphorisms.

The second part of the book includes three case studies that illustrate how even well-written code can always be refactored and cleaned using TDD (test-driven development). The premise behind TDD is that code should be continually tested and refactored. Its motto is: first, make it work (pass the automated tests); then, make it right.

The final chapter of the book contains a list of 66 smells and heuristics that the author compiled by walking through different programs and refactoring them. Each time he changed anything, he asked himself why and wrote the reason down. This resulted in a list that, even though it is hardly complete, implies "a value system."

In short, this book makes common sense explicit in the form of coding guidelines, provides three case studies that illustrate the iterative refactoring process needed to produce really maintainable code, and compiles, in a single chapter, many of the underlying heuristics that guide this process. There are better books available on this topic, but this book could spark discussions within your own development team, help polish your differences until you achieve a consensus, and, in agile terms, help develop your own "value system." —Fernando Berzal

Reprinted from Computing Reviews, © 2009 ACM, <http://www.reviews.com>