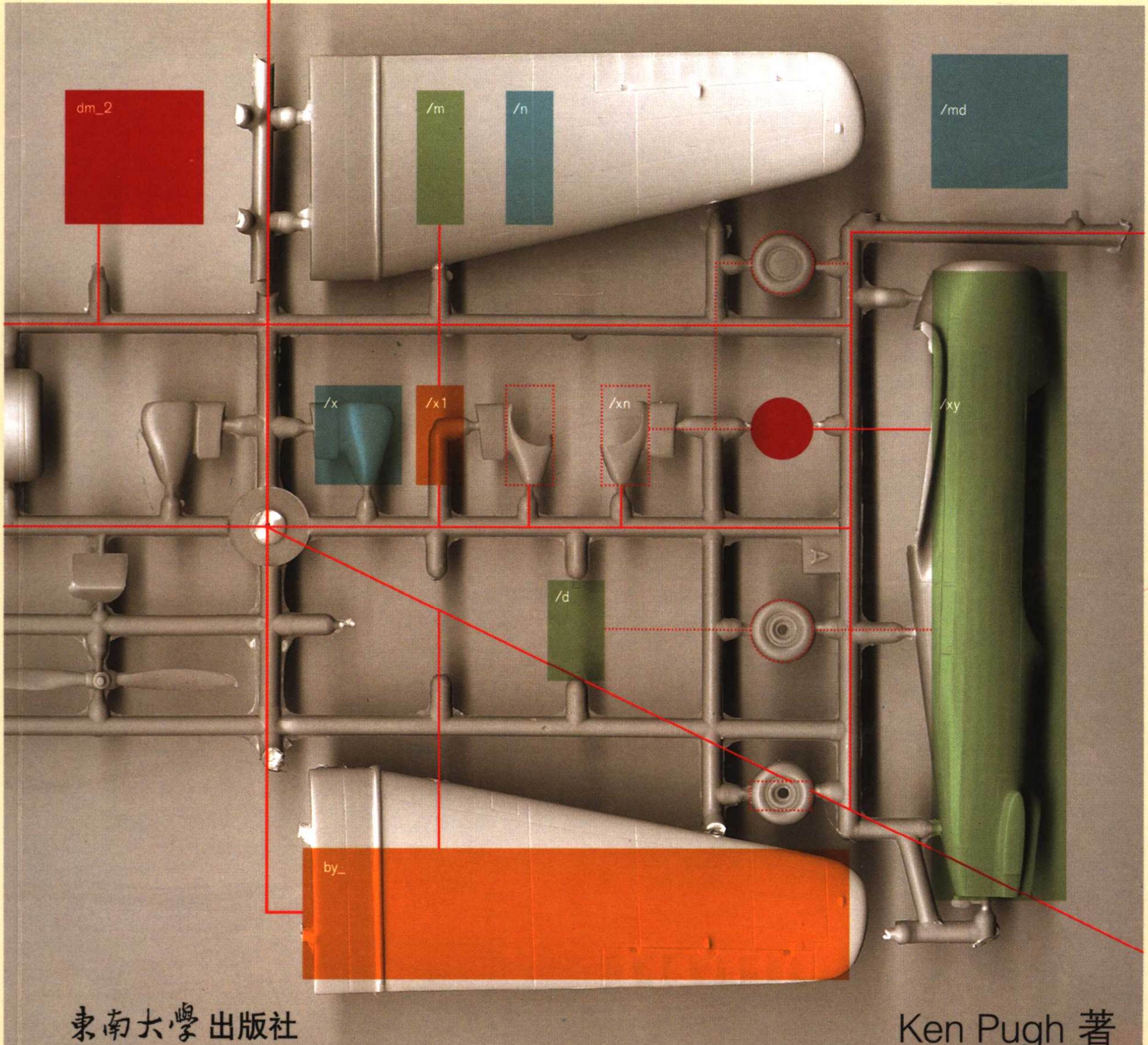
/THEORY/IN/PRACTICE

O'REILLY

软件预构艺术（影印版）

Prefactoring

Extreme Abstraction • Extreme Separation • Extreme Readability



東南大學出版社

Ken Pugh 著

Prefactoring

软件预构艺术(影印版)

Extreme Abstraction • Extreme Separation • Extreme Readability

Ken Pugh

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

软件预构艺术 /（美）皮尤（Pugh, K.）著. — 影印本. — 南京：东南大学出版社，2006. 5

书名原文：Prefactoring

ISBN 7-5641-0365-5

Ⅰ. 软… Ⅱ. 皮… Ⅲ. 软件开发－英文 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2006）第 047704 号

江苏省版权局著作权合同登记

图字：10-2006-137 号

书　　名 /	软件预构艺术（影印版）
书　　号 /	ISBN 7-5641-0365-5
责任编辑 /	张烨
封面设计 /	MendeDesign，张健
出版发行 /	东南大学出版社（press.seu.edu.cn）
地　　址 /	南京四牌楼 2 号（邮政编码 210096）
印　　刷 /	扬中市印刷有限公司
开　　本 /	787 毫米 × 980 毫米　16 开本　15 印张
版　　次 /	2006 年 5 月第 1 版　2006 年 5 月第 1 次印刷
印　　数 /	0001-2000 册
定　　价 /	45.00 元（册）

O'Reilly Media, Inc. 介绍

O'Reilly Media, Inc. 是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。

出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的一批影印版图书，包括：

- 《深入理解 Linux 内核 第三版》（影印版）
- 《Perl 最佳实践》（影印版）
- 《高级 Perl 编程 第二版》（影印版）
- 《Perl 语言入门 第四版》（影印版）
- 《深入浅出 HTML 与 CSS、XHTML》（影印版）
- 《UML 2.0 技术手册》（影印版）
- 《802.11 无线网络权威指南 第二版》（影印版）
- 《项目管理艺术》（影印版）
- 《.NET 组件开发 第二版》（影印版）
- 《ASP.NET 编程 第三版》（影印版）
- 《深入浅出 Ajax》（影印版）
- 《Ajax Hacks》（影印版）
- 《深入理解 Linux 网络内幕》（影印版）
- 《Web 设计技术手册 第三版》（影印版）
- 《软件预构艺术》（影印版）

This book is dedicated to Denny Bunn, a
longtime friend, teacher, and Ironman. He
lost his last race against cancer.

Preface

THE ART OF PREFACTORING APPLIES TO NEW PROJECTS THE INSIGHTS INTO DEVELOPING SOFTWARE YOU HAVE GLEANED FROM YOUR EXPERIENCE, as well as the experience of others, in developing software to new projects. The name of this book plays upon the term *refactoring*, popularized in Martin Fowler's book, *Refactoring: Improving the Design of Existing Code* (Addison-Wesley Professional, 1999). Refactoring is the practice of altering code to improve its internal structure without changing its external behavior.

This book delineates prefactoring guidelines in design, code, and testing. Applying the guidelines in this book does not guarantee that you will never need to refactor your design or code. However, you might decrease the amount of refactoring that is required.

Many of these guidelines are derived from the experiences of numerous developers over several years. Analyzing how code might have been initially developed to alleviate the need for refactoring produced other guidelines. Like Extreme Programming, some of the guidelines might seem extreme. Many revolve around the concepts of Extreme Abstraction, Extreme Separation of Concerns, and Extreme Readability.

Some guidelines contain references to *design patterns*. Design patterns are standard solutions to common problems in software design. The concept of software design patterns was popularized in *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich

Gamma, Richard Helm, Ralph Johnson, and John Vlissides (Addison-Wesley Professional, 1995). That book discusses patterns of objects and classes that form the basis for the solutions to many common problems.*

A few of the guidelines actually might be called design patterns in pattern circles. According to *Design Patterns*, "Some people's patterns are another's design principle." In this book, I call them guidelines, because they include concepts that reoccur in numerous situations, not just as design patterns. These guidelines have not been put into a pattern format, since they are just suggested practices to follow when developing a program.

During my 35-year career, I have worked with computer languages ranging from IBM 360 assembler to C, C++, Java™, C#, HTML, and XML. I have also worked with frameworks including J2EE, Struts, and MFC. Along the way, I have had the opportunity to meet and interact with hundreds of developers. These developers have provided me with experiences related to many of the guidelines I discuss in this book.

During that same time, I have had the opportunity to learn from numerous well-known practitioners in the software development field. This book coalesces many of the themes they discussed in their writings, talks, and conversations with the practices I developed during my career. These people include Scott Ambler, Chuck Allison, James Bach, Kent Beck, Grady Booch, Alister Cockburn, Larry Constantine, Jim Coplien, Ward Cunningham, Gary Evans, Bruce Eckel, Martin Fowler, James Grenning, Payson Hall, Allen Holub, Andy Hunt, Jason Hunter, Eric Jackson, Ivar Jacobsen, Ron Jeffries, Cem Kaner, Joshua Kerievsky, Robert Martin, Bret Pettichord, P. J. Plauger, James Rumbaugh, Dan Saks, Jim Short, Joel Spolsky, Bjarne Stroustrup, Dave Thomas, Bill Venner, Gerry Weinberg, Karl Wiegers, and Rebecca Wirfs-Brock.

I have attempted to attribute the original proponent of each guideline I cover in the book; however, the source of many of the guidelines is unclear. If you know who originated any of the unattributed guidelines, please let me know. Some of them are listed in the book *201 Principles of Software Development* by Alan M. Davis (McGraw-Hill, 1995), in the book *Code Complete, Second Edition* by Steve McConnell (Microsoft Press, 2004), and on the Web at *http://c2.com/cgi/wiki*.

Enumerating guidelines without a context for them is like talking about curling without referring to hair or ice. Therefore, in this book, I present the guidelines in the context of the development of a CD rental system. To my knowledge, such a store does not exist. The video store example in Martin Fowler's refactoring book inspired this choice.

I love outdoor recreation. In particular, I enjoy windsurfing, backpacking, snowboarding, and biking. *Bitter Java* by Bruce A. Tate (Manning, 2002) gave me the idea to use sports analogies when describing programming. Scattered throughout the book are a few outdoor stories relevant to the topic under discussion.

* See *Head First Design Patterns* by Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra (O'Reilly, 2004) for another look at patterns.

Everybody Is Different

Different people view things slightly differently. The paper clip that shows up in a common word processor can be annoying or entertaining. A seemingly great idea for a user interface can turn out to be not so great since the system's users might have vastly different viewpoints than the system's developers.

People look at problems and solutions differently. I prefer to get an overview of a solution so that I can appreciate how the problem has been addressed. Examining a rough diagram of the major classes and a few sequence diagrams provides me information more quickly than wading through code.

Not everyone feels the way I do. Some people prefer to read the code. Viewing a class diagram provides no benefit to them. It is just noise. Neither approach is necessarily better or worse than the other is. The only time you will run into trouble is if you do not appreciate both approaches when you encounter a person who does it the other way.

Likewise, developers have a spectrum of preferences that run the gamut from strongly typed languages to nontyped languages and from big design up front to no design up front. Those on one end of the spectrum should appreciate the tradeoffs and context issues of those on the other end.

In addition, the guidelines presented in this book might be radically different from your current development paradigm. Comparing them to your own guidelines can help you to understand the tradeoffs made in your paradigm and may spur changes in your guidelines.

The Design Example

This book describes the development of a system that incorporates many features of some of the systems that have been created in the past. The experience of creating those systems forms some of the guidelines for developing new systems. Developers are not perfect. We cannot read our client's mind, and his mind might change. The development story outlined in this book shows where decisions were made that had unanticipated ramifications.

This book presents one of many possible designs that can solve your clients' requirements. There is no absolute measurement method for evaluating the "goodness" of a design. There are obviously good designs and bad designs, but there are many gray areas as well. Counting the number of methods, lines of code, or number of classes seems like a very objective way of measuring, but often the resulting number is not necessarily meaningful, except for the extremes. I measure designs with a gut feeling. This parallels the "code smells" of refactoring. A system's overall design gives me a gut feeling ranging from "warm tummy" to "upset stomach."

Audience

This book will appeal to readers who understand the basic concepts of object-oriented design. It gives them suggested guidelines to create more readable and maintainable code.

The book assumes that the reader has some familiarity with the basic Unified Modeling Language (UML) diagrams (class, sequence, and state), as well as some knowledge of an object-oriented language, such as Java, C++, C#, Ruby, or Python.

Contents of This Book

This book in organized into 17 chapters. The organization of the chapters follows the development of the sample system. It diverts in Chapter 3 to discuss general development issues and in Chapter 6 to examine object-oriented design. The sample system development ends in Chapter 14. Chapter 15 describes a real-life system designed using prefactoring guidelines, and Chapter 16 talks about the design of an antispam system.

Here is a detailed description of the chapter contents:

Chapter 1, *Introduction to Prefactoring*
: This is an introduction to the facets of prefactoring.

Chapter 2, *The System in So Many Words*
: We meet Sam, the client, to get an overall view of the desired system. We discuss creating a shared vocabulary for communication, and we use some extreme abstraction.

Chapter 3, *General Development Issues*
: We look at some general issues in developing a system. This includes the big picture, interface contracts, communicating with code, simplicity, dealing with errors, and the spreadsheet conundrum.

Chapter 4, *Getting the Big Picture*
: We continue talking with Sam to get a clearer understanding of the overall requirements—the big picture. Then we start to create a design for a system.

Chapter 5, *Got Class?*
: We take our system outline and develop the implementation classes. We explore how single or multiple classes can represent concepts.

Chapter 6, *A Few Words on Classes*
: We look at object-oriented design in general. The class maxims of cohesion and coupling are reviewed, along with the three laws of objects. Polymorphic behavior is demonstrated with both inheritance and interfaces.

Chapter 7, *Getting There*
: We address using separation of concerns to create reports. Planning for migration brings up some additional design issues.

Chapter 8, ***The First Release***

We perform a retrospective on how well our design approach worked. We explore issues that were addressed during development and the additional classes that were created during coding.

Chapter 9, ***Associations and States***

Sam presents us with new requirements. We explore using association classes in the system to implement the requirements. We examine how the state of objects in the system can be represented.

Chapter 10, ***Interfaces and Adaptation***

We create interfaces for Sam's catalog-search use case. We explore how to test these interfaces and how to adapt implementations to meet these interfaces.

Chapter 11, ***Zip Codes and Interfaces***

Sam asks that the system keeps track of customer addresses. We determine how to verify the Zip Codes in addresses using interfaces.

Chapter 12, ***More Reports***

Sam decides he needs fancier reports and different reports. We implement his requests using some of the guidelines already introduced.

Chapter 13, ***Invoices, Credit Cards, and Discounts***

Sam decides it is time to add the ability to invoice customers and charge those invoices to credit cards. We explore interfaces to external credit card processors. We add computation of customer discounts in terms that Sam can understand.

Chapter 14, ***Sam Is Expanding***

Sam is expanding his operations. He is opening more stores, both locally and globally. His store is being featured on the Web. We use many of the previously presented guidelines to develop our approach to this expansion.

Chapter 15, ***A Printserver Example***

This chapter presents a case study involving a real-world system used by libraries to charge for printouts of documents from personal computers. This chapter delineates where guidelines were employed.

Chapter 16, ***Antispam Example***

This chapter examines how email is transmitted and received. It presents a proposed design for an email receiver and spam detector.

Chapter 17, ***Epilogue***

We wrap up with some closing thoughts.

The Cover

The cover picture is a 1/72 scale model of an Azur NAA-57 airplane. The NAA-57 was based on the North American T-6 Texan, which was licensed to European manufacturers. Jeremy Mende of MendeDesign designed the cover. His design choice is appropriate to my hobbies. I am a private pilot, although I have not flown much in recent years, except for an expedition to Alaska a few years back.

Conventions Used in This Book

Pseudocode examples explain many of the guidelines in a concrete manner. I try to make the examples as generic as possible, as this is not a language guide. The code uses a combination of conventions from a number of languages. Class names use uppercase separation (e.g., `ClassName`). Attributes, variables, and method names use lowercase with underscores (e.g., `method_name( )`), à la the C++ Standard Template Library and Python.

For classes for which you must access an object's attributes, I show assignment to and from the attributes, as if they were properties of Eiffel or C#. In other languages or with other conventions, you probably will use get and set methods.

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, and the output from commands. It also indicates class outlines in pseudocode.

`User input`
: Shows user input or message transactions.

Boldface
: Indicates states or enumerated values.

NOTE
This design element signifies a tip, suggestion, or general note.

This icon identifies a guideline.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you are reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Prefactoring* by Ken Pugh. Copyright 2005 O'Reilly Media, Inc., 0-596-00874-0."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/prefactoring

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Safari Enabled

When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top technology books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Acknowledgments

I thank my wife, Leslie Killeen, for putting up with my talking about this book for hours on end. She also read my drafts and suggested changes. She is a weaver, not a programmer, so the material was not in her field. For that reason, I doubly thank her.

I appreciate Jim Batterson, Pam Brown, Eric M. Burke, Gary K. Evans, Michael Green, Jason Hunter, Mark Kozel, Nitin Narayan, Graham Oakes, David Rasch, Ronald E. Thompson III, and many others, for reviewing the book and making numerous suggestions. I thank Rob Walsh, the co-founder of EnvisionWare, for permission to use his system as an example in Chapter 15. I thank John R Levine and Pete McNeil for reviewing Chapter 16. I thank Lisa Pulignani and McCabe and Associates for providing analysis reports on the sample program.

I thank Scott Ambler, Larry Constantine, Ken Estes, Danny Faught, Don Gray, Payson Hall, Gary McGraw, Ken Ritchie, Paul Strack, Dan Saks, and many others for attribution of many of the guidelines. I thank Gerry Weinberg for his suggestions.

Finally, I thank Jonathan Gennick, my editor at O'Reilly, Audrey Doyle, my copyeditor, and Sarah Sherman, my production editor, for helping me to produce this book.

ABOUT THE AUTHOR

KEN PUGH has extensive experience in the area of software analysis and design. He has worked on systems ranging from goat serum process control to financial analysis to noise recording to satellite tracking. His previous books were on C and Unix, and he is a former columnist for the *C/C++ Users Journal*. He has taught programming courses for Wellesley College and the University of Hawaii, as well as numerous corporate courses, and he frequently presents at national conferences. As an independent consultant for over 20 years, he has served clients from London to Sydney. As an expert witness, he has provided testimony in both civil suits and criminal cases.

When not computing, he enjoys snowboarding, windsurfing, biking, and hiking the Appalachian Trail.

COLOPHON

OUR LOOK IS THE RESULT of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

Sarah Sherman was the production editor and proofreader, and Audrey Doyle was the copyeditor for *Prefactoring*. Mary Anne Weeks Mayo and Claire Cloutier provided quality control. Lydia Onofrei provided production assistance. Johnna VanHoose Dinse wrote the index.

MendeDesign designed and created the cover artwork of this book. Karen Montgomery produced the cover layout with Adobe InDesign CS using the Akzidenz Grotesk and Orator fonts.

Marcia Friedman designed the interior layout. Melanie Wang designed the template; Phyllis McKee adapted the template. The book was converted by Keith Fahlgren to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Adobe's Meridien; the heading font is ITC Bailey. The illustrations that appear in the book were produced by Robert Romano, Jessamyn Read, and Lesley Borash using Macromedia FreeHand MX and Adobe Photoshop CS.

TABLE OF CONTENTS