软件工程实践丛书

PEARSON Prentice Hall

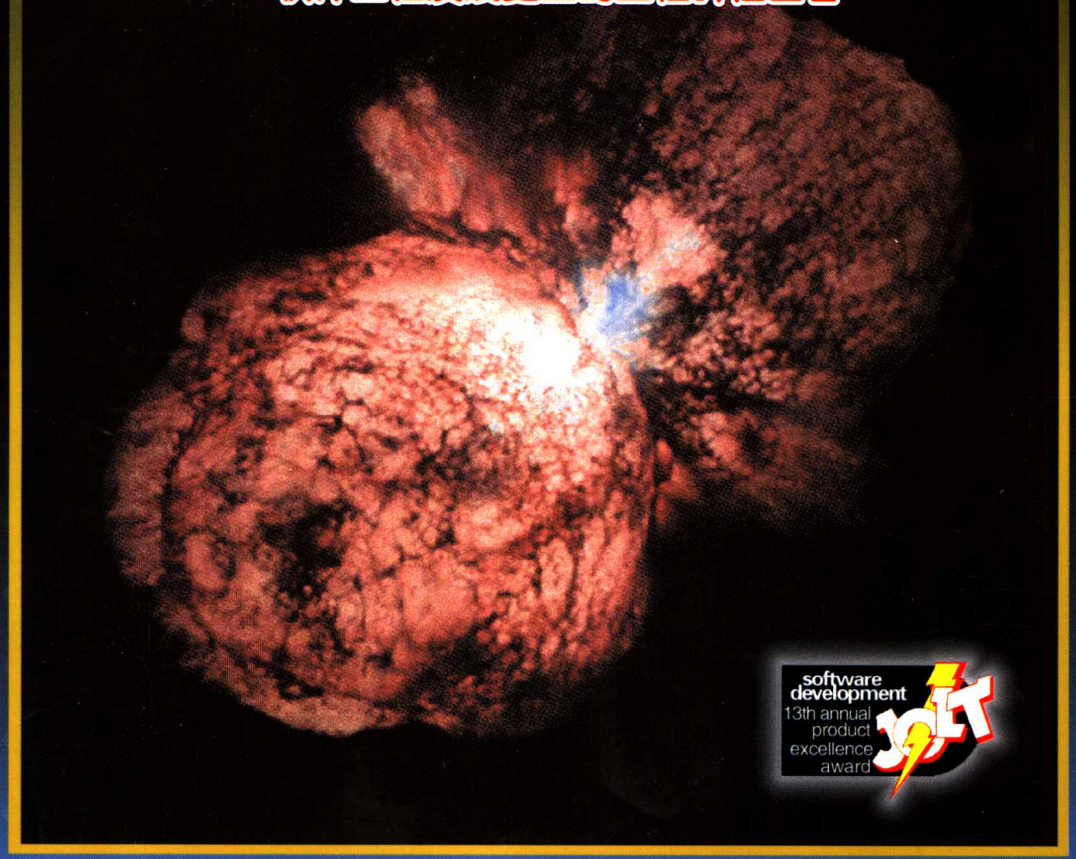
敏捷软件开发

原则、模式与实践

Agile Software Development

Principles, Patterns, and Practices

- 第13届软件开发震憾大奖获奖作品
- 国际软件工程和开发大师最新力作
- 众多名家一致推荐的敏捷开发指南
- 软件工程发展史上的里程碑性巨著



(美) Robert C. Martin 著

邓辉 译 孟岩 审

Pearson Education

清华大学出版社

软件工程实践丛书

敏捷软件开发

原则、模式与实践

（美）Robert C. Martin 著
邓 辉 译
孟 岩 审

清华大学出版社
北 京

内 容 简 介

享誉全球的软件开发专家和软件工程大师 Robert C. Martin 向您介绍如何解决软件开发人员、项目经理及软件项目领导们所面临的最棘手的问题。这本综合性、实用性的敏捷开发和极限编程方面的指南，讲述了在预算和时间要求下软件开发人员和项目经理如何使用敏捷开发完成项目；使用真实案例讲解如何用极限编程来设计、测试、重构和结对编程；包含了极具价值的可重用的 C++和 Java 源代码；还重点讲述了如何使用 UML 和设计模式解决面向客户系统的问题。

本书于 2003 年荣获第 13 届软件开发图书震憾大奖，适于用作高校计算机专业本科生、研究生和软件学院的软件工程和软件开发相关课程的教材或参考书，也适于软件开发和管理人员提高自身水平学习之用。

图书在版编目（CIP）数据

敏捷软件开发：原则、模式与实践/（美）马丁著；邓辉译，孟岩审. —北京：清华大学出版社，2003

（软件工程实践丛书）

书名原文：Agile Software Development: Principles, Patterns, and Practices

ISBN 7-302-07197-7

Ⅰ. 敏… Ⅱ. ①马… ②邓… ③孟… Ⅲ. 软件开发 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2003）第 078273 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦

http://www.tup.com.cn　　**邮　　编**：100084

社 总 机：010-62770175　　**客户服务**：010-62776969

文稿编辑：尤晓东

封面设计：付剑飞

印 刷 者：北京国马印刷厂

发 行 者：新华书店总店北京发行所

开　　本：185×260　**印张**：31.25　**插页**：2　**字数**：942 千字

版　　次：2003 年 9 月第 1 版　2003 年 9 月第 1 次印刷

书　　号：ISBN 7-302-07197-7/TP・5239

印　　数：1～5000

定　　价：59.00 元

敏捷软件开发宣言

我们正在通过亲身实践以及帮助他人实践，揭示更好的软件开发方法。通过这项工作，我们认为：

个体和交互	胜过	过程和工具
可以工作的软件	胜过	面面俱到的文档
客户合作	胜过	合同谈判
响应变化	胜过	遵循计划

虽然右项也具有价值，

但我们认为左项具有更大的价值。

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

敏捷宣言遵循的原则

我们遵循以下原则：

- 我们最优先要做的是通过尽早的、持续的交付有价值的软件来使客户满意。
- 即使到了开发的后期，也欢迎改变需求。敏捷过程利用变化来为客户创造竞争优势。
- 经常性地交付可以工作的软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。
- 在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。
- 围绕被激励起来的个体来构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。
- 在团队内部，最具有效果并且富有效率的传递信息的方法，就是面对面的交谈。
- 工作的软件是首要的进度度量标准。
- 敏捷过程提倡可持续的开发速度。责任人、开发者和用户应该能够保持一个长期的、恒定的开发速度。
- 不断地关注优秀的技能和好的设计会增强敏捷能力。
- 简单——使未完成的工作最大化的艺术——是根本的。
- 最好的构架、需求和设计出自于自组织的团队。
- 每隔一定时间，团队会在如何才能更有效地工作方面进行反省，然后相应地对自己的行为进行调整。

面向对象设计的原则

SRP　单一职责原则

就一个类而言，应该仅有一个引起它变化的原因。

OCP　开放—封闭原则

软件实体（类、模块、函数等）应该是可以扩展的，但是不可修改。

LSP　Liskov 替换原则

子类型必须能够替换掉它们的基类型。

DIP　依赖倒置原则

抽象不应该依赖于细节。细节应该依赖于抽象。

ISP　接口隔离原则

不应该强迫客户依赖于它们不用的方法。接口属于客户，不属于它所在的类层次结构。

REP　重用发布等价原则

重用的粒度就是发布的粒度。

CCP　共同封闭原则

包中的所有类对于同一类性质的变化应该是共同封闭的。一个变化若对一个包产生影响，则将对该包中的所有类产生影响，而对于其他的包不造成任何影响。

CRP　共同重用原则

一个包中的所有类应该是共同重用的。如果重用了包中的一个类，那么就要重用包中的所有类。

ADP　无环依赖原则

在包的依赖关系图中不允许存在环。

SDP　稳定依赖原则

朝着稳定的方向进行依赖。

SAP　稳定抽象原则

包的抽象程度应该和其稳定程度一致。

极限编程实践

完整团队

XP 项目的所有参与者（开发人员、业务分析师、测试人员等等）一起工作在一个开放的场所中，他们是同一个团队的成员。这个场所的墙壁上随意悬挂着大幅的、显著的图表以及其他一些显示他们进度的东西。

计划游戏

计划是持续的、循序渐进的。每 2 周，开发人员就为下 2 周估算候选特性的成本，而客户则根据成本和商务价值来选择要实现的特性。

客户测试

作为选择每个所期望的特性的一部分，客户定义出自动验收测试来表明该特性可以工作。

简单设计

团队保持设计恰好和当前的系统功能相匹配。它通过了所有的测试，不包含任何重复，表达出了编写者想表达的所有东西，并且包含尽可能少的代码。

结对编程

所有的产品软件都是由两个程序员、并排坐在一起在同一台机器上构建的。

测试驱动开发

程序员以非常短的循环周期工作，他们先增加一个失败的测试，然后使之通过。

改进设计

随时改进糟糕的代码。保持代码尽可能的干净、具有表达力。

持续集成

团队总是使系统完整地被集成。

集体代码所有权

任何结对的程序员都可以在任何时候改进任何代码。

编码标准

系统中所有的代码看起来就好像是被单独一个——非常值得胜任的——人编写的。

隐喻

团队提出一个程序工作原理的公共景像。

可持续的速度

团队只有持久才有获胜的希望。他们以能够长期维持的速度努力工作。他们保存精力，他们把项目看作是马拉松长跑，而不是全速短跑。

中文版序：软件之美

除了我的家庭，软件是我的挚爱。通过它，我可以创造出美的东西。软件之美在于它的功能，在于它的内部结构，还在于团队创建它的过程。对用户来说，通过直观、简单的界面呈现出恰当特性的程序就是美的。对软件设计者来说，被简单、直观地分割，并具有最小内部耦合的内部结构就是美的。对开发人员和管理者来说，每周都会取得重大进展，并且生产出无缺陷代码的具有活力的团队就是美的。美存在于所有这些层次之中，它们都是本书内容的一部分。

软件开发人员如何学到创造美的知识呢？在本书中，我讲授了一些原则、模式以及实践，它们可以帮助软件开发人员在追求美的程序、设计以及团队的道路上迈出第一步。其中，我们探索了基本的设计原则，软件设计结构的通用模式以及有助于团队融为一个有机整体的一系列实践。由于本书是关于软件开发的，所以包含了许多代码。仔细研究这些代码是学习本书所教授的原则、模式以及实践的最有效方法。

人们需要软件——需要许多的软件。50 年前，软件还只是运行在少量大型、昂贵的机器之上。30 年前，软件可以运行在大多数公司和工业环境之中。现在，移动电话、手表、电器、汽车、玩具以及工具中都运行有软件，并且对更新、更好软件的需求永远不会停止。随着人类文明的发展和壮大，随着发展中国家不断构建它们的基础设施，随着发达国家努力追求更高的效率，就需要越来越多的软件。如果在所有这些软件之中，都没有美存在，这将会是一个很大的遗憾。

我们知道软件可能会是丑陋的。我们知道软件可能会难以使用、不可靠并且是粗制滥造的；我们知道有一些软件系统，其混乱、粗糙的内部结构使得对它们的更改既昂贵又困难；我们还见过那些通过笨拙、难以使用的界面展现其特性的软件系统；我们同样也见过那些易崩溃且行为不当的软件系统。这些都是丑陋的系统。糟糕的是，作为一种职业，软件开发人员所创建出来的美的东西却往往少于丑的东西。如果你正在阅读这本书，那么你也许就是那个想去创造美而不是丑的人。

最好的软件开发人员都知道一个秘密：美的东西比丑的东西创建起来更廉价，也更快捷。构建、维护一个美的软件系统所花费的时间、金钱都要少于丑的系统。软件开发新手往往不理解这一点。他们认为做每件事情都必须要快，他们认为美是不实用的。错！由于事情做得过快，他们造成的混乱致使软件僵化，难以理解。美的系统是灵活、易于理解的，构建、维护它们就是一种快乐。丑陋的系统才是不实用的。丑陋会降低你的开发速度，使你的软件昂贵而又脆弱。构建、维护美的系统所花费的代价最少，交付起来也最快。

我希望你能喜爱这本书。我希望你能像我一样学着以创建美的软件而骄傲，并享受其中的快乐。如果你从本书中略微看到了这种快乐，如果本书使你开始感受到了这种骄傲，如果本书点燃了你内心欣赏这种美的火花，那么就远超过我的目标了。

Robert C. Martin

2003 年 8 月 21 日

The Beauty of Software

Robert C. Martin's foreword for Agile Software Development Chinese Version

Next to my family, software is my passion. It is a medium in which I try to create beauty. The beauty of software is in it's function, in it's internal structure, and in the way in which it is created by a team. To a user, a program with just the right features presented through an intuitive and simple interface, is beautiful. To a software designer, an internal structure that is partitioned in a simple and intuitive manner, and that minimizes internal coupling, is beautiful. To developers and managers, a motivated team of developers making significant progress every week, and producing defect-free code, is beautiful. There is beauty on all these levels, and they are all part of the topic of this book.

How do software developers learn how to create this beauty? In this book I teach the principles, patterns, and practices that can help software developers take the first steps towards beautiful programs, designs, and teams. In these pages we explore basic design principle, common patterns in the structure of a software design, and a set of practices that can help a team knit itself into a functioning whole. Since this book is about software development, it contains a lot of code. The principles, patterns, and practices that this book teaches are learned most effectively by carefully studying that code.

Our world needs software -- lots of software. Fifty years ago software was something that ran in a few big and expensive machines. Thirty years ago it was something that ran in most companies and industrial settings. Now there is software running in our cell phones, watches, appliances, automobiles,toys, and tools. And need for new and better software never stops. As our civilization grows and expands, as developing nations build their infrastructures, as developed nations strive to achieve ever greater efficiencies, the need for more and more software continues to increase. It would be a great shame if, in all that software, there was no beauty.

We know that software can be ugly. We know that it can be hard to use,unreliable, and carelessly structured. We know that there are software systems whose tangled and careless internal structures make them expensive and difficult to change. We know that there are software systems that present their features through an awkward and cumbersome interface. We know that there are software systems that crash and misbehave. These are ugly systems. Unfortunately, as a profession, software developers tend to create more ugly systems than beautiful ones. If you are reading this book, then you are someone who wants to create beauty and not ugliness.

There is a secret that the best software developers know. Beauty is cheaper than ugliness. Beauty is faster than ugliness. A beautiful software system can be built and maintained in less time, and for less money, than an ugly one. Novice software developers don't understand this. They think that they have to do everything fast and quick. They think that beauty is impractical. No! By doing things fast and quick, they make messes that make the software stiff, and hard to understand. Beautiful systems are flexible and easy to understand. Building them and maintaining them is a joy. It is ugliness that is impractical. Ugliness will slow you down and make your software expensive and brittle. Beautiful systems cost the least to build and maintain, and are delivered soonest.

I hope you enjoy this book. I hope you learn to take as much pride and joy in the creation of beautiful software that I do. If this book can show you just an inkling of that joy, motivate you to feel just the beginnings of that pride, and provide just a spark of appreciation for that beauty, then it has more than accomplished my goal for it.

七年等待换来的经典(代序)

孟 岩

Robert C. Martin 的经典著作《敏捷软件开发》(*Agile Software Development*)中文版面世，这是计算机技术出版领域的一件大喜事。即使在技术图书非常繁荣的今天，本书的问世也仍然是值得广大开发者格外留意和关注的事件。这不仅是因为它刚刚荣获 2002 年度 Jolt 震撼大奖，更因为这本书本身的价值和独特魅力所在。

Robert Cecil Martin 是软件开发领域里响当当的名字。20 世纪 70 年代，当他还是个年轻小伙子的时候就已经是一位有名的 UNIX 黑客。经过长期的开发实践后，他成了软件开发领域中的知名专家。20 世纪 90 年代初，Rational 软件公司首席科学家 Grady Booch 邀请他加盟，希望借助他丰富的实践经验，结合 Booch 自己的软件设计理念，开发一套创新性的软件产品。这就是大名鼎鼎的 Rational ROSE 的早期版本！在 Rational 公司期间，Martin 丰富的实践经验与 Booch 深厚的理论功底形成了完美的组合，把面向对象设计的理论与实践推向了高峰。1994 年，Grady Booch 的经典著作 *Object-Oriented Analysis and Design with Applications* 问世，几乎同时，Martin 的第一本著作 *Designing Object-Oriented C++ Application Using the Booch Method* 也由 Prentice Hall 出版。这两本书相互辉映，当时引起了很大的反响。Martin 的著作结合了当时最流行的面向对象语言 C++和最出色的面向对象设计建模方法 Booch Method，以大量实例讲解技术概念和应用方法，分析透彻、讲解务实、技术精妙，在读者中声誉极佳。该书不仅为 Martin 确立了软件设计领域顶尖专家的地位，而且奠定了他独特的写作风格。Martin 本人对该书非常有感情，多年后，在我给他写信谈到这本书时，他还很得意地宣称：这是他的代表作，其中的大部分内容即使在今天也不过时。没有十年磨一剑的功力，又怎能创造出长盛十年而不衰的经典？

大约 2001 年 10 月，我有机会复印了 Martin 那本 1995 年的名著，阅读之后，大为震撼，并决定不惜重金从国外购买原版，同时尽力向其他朋友推荐。不少朋友阅读之后亦非常兴奋，甚至有人说："读此书方知什么是面向对象设计"，可见对该书的评价之高。惟一可惜的是，该书年代久远，不少内容有些陈旧了。很自然地，我和我的朋友们对于该书的第 2 版都是翘首以待。

大约在 2002 年 4 月，我从 Amazon 网站上看到了该书第 2 版的预订信息。当时的名字叫做 *Designing Object-Oriented C++ Application Using UML*。通过因特网上的调查，我了解到整个面向对象设计开发社群都在高度关注这本书的写作进展情况。已经被冠以"鲍勃大叔"昵称的 R. C. Martin 曾经允诺在 1998 年推出此书，但结果一拖再拖，搞得整个开发社群的人们"群情激愤"。正如本书前言开头有一幅漫画插图所示，一位叫做 Claudia Frers 的女士在 1999 年的 UML World 大会上愤怒地声讨鲍勃大叔："可是你说过，去年就可以完成这本书啊！"。面对读者爱极而怒的反应，我想鲍勃大叔虽然表面上不免尴尬，内心却一定是暗喜的。而包括他在内的所有人，当时可能都想不到，这本书居然还要拖上 3 年，才会最终以全新的面貌出现在读者面前。

看到该书的预订页面后，我立刻请求一位师长帮我从 Amazon 网站上预订。之后，我就开始了漫长的等待过程。这是多么奇异的一段等待啊！我先是发现这本书的名字变了，变成了 *Designing Object-Oriented C++*，没有"UML"了。书名变短了，不过看来内容是扩充了，因为预告篇幅增大到了 700 多页，这倒也不错。可是到了大约 2002 年 8 月份，我突然发现，这本书不见了！在原来的网页上出现了一本名叫"*Agile Software Development*"(《敏捷软件开发》，简称 ASD)的书，虽然作

者也是 Robert C. Martin，但只有 500 多页了。书名变短了，内容也缩水了！我当然很激愤，想不到 Amazon 也会来偷梁换柱这一招！忍耐了一段时间之后，我在 comp.object 上见人就问："您知道 Bob 大叔的那本 C++书第 2 版到哪儿去了吗？那本 *Agile Software Development* 又是怎么回事？我的 money 还能要得回来吗？……"其实我很清楚，Bob 大叔本人就经常在这里活动，我这招"敲山震虎"迟早会引得 Bob 大叔本人出来解释的。果然，两天后我就收到署名"Uncle Bob"的来信，信中客气地说，这本 *Agile Software Development* 就是原来那本书的第 2 版。不过内容、风格已经完全变化了，应该算是一本全新的书。虽然书中大多数代码是用 Java 写的，但是 Bob 大叔向我保证，这本书的内容绝对对得起我付出的每一分银子。

OK，我还有什么可说的呢？Bob 大叔亲自出来解释，我当然满意了。2002 年 10 月底，一个大大的包裹放到了我的桌子上——ASD 到了。其时不单国内很少有人拿到这本书，就是在美国，该书签名首发的活动也还在筹备中。因此，我大概可以算是全国最早阅读此书的前 5 人之一了吧。

从 1995 年第 1 版面世算起，整个面向对象设计社群经过了 7 年的漫长等待，终于等到了这本书。7 年中，斗转星移，物是人非，多少英雄升起又落下，多少神话创生又破灭！当这本硬壳封面的精美图书沉甸甸地放在手上，怎能不让人感慨！

阅读本书的过程，是一个充满了发现、领悟和兴奋的过程。我只读了其中几个章节，就已经知道本书大大超过了事前的预期，是当代软件开发领域最杰出的著作之一。此时，著名的美国 *Software Development* 杂志每年一度的 Jolt 评奖工作正式开始，我毫不犹豫地给本书投了一票。后来我愉快地得知，本书果然如我所愿夺得了 2002 年度技术图书类最高奖项——第 13 届震憾大奖（Jolt 大奖）。我在 Amazon 网站上对本书的评论中，我的结论写道：本书不但肯定是 2002 年度最好的设计类技术图书，而且在今后的几年中，也很难出现超过它的作品。那篇短评也算是为本书的宣传做了一点点事，为此我感到非常高兴。

听说在台湾省，3 位资深工程师联名上书某出版公司要求引进此书，并且自告奋勇担任翻译工作。这些资深工程师收入丰厚，根本不是为了那点微薄的翻译酬劳而来，完全是希望能够把这本极有价值的好书介绍给更多的业内人士，造福大众。一本书能够引起这么大的热诚，可见它的魅力了。

那么本书究竟好在哪里呢？真正关心这个问题的读者，应该先阅读原著。Bob 大叔是位实践大师，读者必须通过阅读实践，才能够真正体会到本书的精妙之处。我在此只谈谈本书给我印象最深的三个特点。

首先，该书延续了 Bob 大叔著作的一贯特色，以实例为本，不尚空谈，因此格外真实，摄人心魄。Martin 从多年的写作中提炼出一种独特的风格，就是"引领式传授"。与别的作者不同，面对问题，Martin 并不是把最后那个完美的答案一下子放在你面前，让你拜倒在他脚下，被他的睿智折服。他很清楚，当读者在大师面前拜倒的同时，除了挫折感外，同时也会丧失自信。而自信是一个设计者由成功走向成功的最关键因素。因此，Martin 还原了一个真实的设计过程——带领首读者一起设计。读者在书中能够看到，作者也会犯错误，也要面临痛苦的选择，也会做一些愚蠢的决定，也会被一些"激动人心"的想法诱惑，但是，最后他能够跨越重重障碍，看透重重迷雾，得到优秀的设计。他跟我们一样，是一个有血有肉的设计者，而不是什么天赋异禀的天才。正因为如此，他能够达到的，我们也能够达到。这样的风格，使我们从阅读中能够逐渐体会到软件设计最精髓的东西：张力与平衡。如何选择，如何思考，如何面对困境，这些才是最宝贵的财富。这些财富，就在我们自己的大脑里，而 Martin 的书，就是打开这个财富宝库的钥匙。此外，读者在阅读本书的时候会发现，Martin 在教授具体技术的时候，始终以实例教学为主。比如关于 UML 的附录，就是通过两个

实例来教授 UML 的基础知识。诸位不妨从这两个附录开始阅读此书，体会一下作者的写作风格，看一看在 UML 学习方面，究竟是传统的罗列式教材给你的印象更深，还是以实例为依托的 Martin 式风格更有效率。

其次，详略得当。Martin 很清楚哪些东西应该讲得细致一些，哪些东西可以讲得抽象一些。这一点也是大师风范。软件设计这个领域，有很多东西是应该讲清楚，却很少有人能够讲得清楚的，比如设计中的权衡，实例的综合运用等等。还有很多东西，是没有必要像有些书那样长篇大论的，比如说一些基本的设计原则或模式，一些开发过程思想，一些技术理念等等。这些东西讲得简明扼要能够给人以智慧的启迪，讲得冗长拖沓，定下几十上百条规则方法，则必然脱离实际，堵塞和束缚读者的思想。Martin 的书可说决无此弊，这也应当归功于他多年的实践经验。从本书中我们可以看到，他对于思想和原则一般是言简意赅，意到为止。而对于有助于提高读者实际设计能力的实例，则不惜笔墨，详加阐述，可谓煞费苦心。

最后一点，也是相当重要的一点，本书选材匠心独到，精彩绝伦，特别是还有大量独创性和创新性的技术，来自 Martin 多年的研发实践，十分珍贵。比如本书的 Acyclic Visitor、Extension Object、Monostate、Taskmaster 等模式，并不属于经典的 23 模式，很多都是来自 Martin 自己的创造，配合实例解析，有助于大幅提高读者的设计实践能力。

总之，本书是近年来难得的佳作，希望大家都能好好阅读和体会这本书，它将为你的设计能力带来一个质的提升，同时，也有助我们对“怎样算是一本好的技术图书”有更进一步的认识。

译 者 序

关于软件开发方面的书真是不少，有过程方法的、有最佳实践的、还有设计原则的。但是当你真正进行软件开发实践时，却会发现这些书中告诉你的知识在实际运用的时候总是和期望的效果有一定的差距。我在这方面有深刻的体会。经过一段时间的反思，我隐约觉得，应该还有某个存在于过程方法、最佳实践以及设计原则之外的东西来有机地把它们结合起来，才能真正地发挥它们的最大效用。这种东西不是可以形式化的条条框框，而是活跃于人的大脑中的某种思维方法。看完了Robert C. Martin 的《敏捷软件开发：原则、模式与实践》之后，我有一种豁然开朗的感觉。本书把这种思维方法阐述得再清晰不过了。

本书具有两大特色。第一，很多讲述软件开发的书籍，要么是仅仅涉及过程方法方面的内容，要么是仅仅涉及设计原则方面的内容。这些做法相对于整个软件开发活动来说都是片面的。其实，过程方法、设计原则以及最佳实践是一个不可分割的整体。孤立地去使用任何一部分都无法获得最佳的效果。最有效的方法应该是根据自己开发团队的实际情况，找出一种能够有效地把这三者结合起来并使它们相互支持的方法。比如，大家都知道每周（或每日）构建是一种得到广泛认可的最佳实践方法，但并不是只要你每周/日都去构建了就能得到好的效果，有时，结果可能会更糟。要想使这种方法有效，还需要其他方面的支援。每周/日构建的前提是软件必须是易于每周/日构建的。也就是说，你需要对软件中的依赖关系进行管理，使之具有每周/日构建的基础。而这种依赖关系的管理是需要设计原则来指导和度量的。这只是其中一个例子，本书中到处都体现着作者的这种主导思想和实践。如果读者能够在这个方面好好体会的话，肯定会对软件开发有一个更为全面、深入的理解，从而可以更加有效地去使用这些过程方法、设计原则以及最佳实践。

第二，本书的核心是软件设计，但是它对软件设计的理解以及讲解方式非常的特别。许多有关软件设计的书籍中，要么先讲述一些设计原则、模式，然后再给出几个简单的在理想情况下的应用；要么是拿一个最终的设计结果来剖析，然后告诉你它们是多么的优美。当时，你可能真会那么认为，但是当你试图在自己的实际开发中应用时，总会发现情况是完全不一样的。此时，你要么束手无策，要么会误用设计原则、模式。究其原因，主要是因为，在此类书中所讲述的不是真正的设计，只是设计的部分内容，而忽略了设计中最为重要的方面。设计是人的思维的一种动态活动，是设计者针对自己的问题的思索、权衡、折衷、选择的过程。其中会出现很多在理想情况下不会出现的问题，对这些问题的处理水平才是真正的设计水平。同样，本书中到处都是这样的思考过程。针对每个案例，作者都会和你一起思索、一起探讨、一起权衡、一起验证。本书中所展示的是一个个完整的设计活动过程。通过这些案例的学习，相信读者肯定会对设计有一个更深刻的理解。此外，本书中也讲述了很多的设计模式，但是和很多其他讲述模式的书不同的是，它更多的是在告诉你什么时候不要去使用模式，去抵制模式的诱惑，以免带来不必要的复杂性。在对模式狂热吹捧的今天，本书无疑是一剂纠偏良药，可以让你更加合理、有效地使用模式。

其实，这些内容正是软件开发活动中最本质，同时也是最难以琢磨的内容。要把这些内容通过文字表达出来更是非常困难的，这也是这方面的书籍凤毛麟角的原因。然而在本书中，Robert C. Martin 先生能把这些内容编写得如此清晰、如此易于理解，充分展示了作者深厚的技术功底和卓越的表达能力。因此，本书能从众多优秀书籍中脱颖而出获得第 13 届 Jolt 大奖，就是意料之中的了。

本书主要包含 4 部分内容，这些内容对于今天的软件工程师都非常的重要，它们是：

- 敏捷方法：主要讲述了如何去使用敏捷方法，其中有很大一部分内容是告诉你为什么要这样做。
- 面向对象设计原则：本书包含了 12 个面向对象设计原则，涵盖了包的设计和类的设计。这是我所见过的对这方面内容讲解得最清晰、最彻底、最深刻的惟一的一本书。
- 设计模式：本书中讲述了 23 个设计模式，并都有具体的实例。讲解的重点如何在实际的应用中去使用模式，如何根据当前问题的上下文以及约束力去选择最适合的模式，以及何时避免使用模式。
- UML：本书不是关于 UML 的，但是为了让读者更好地理解书中的内容，作者使用了一些 UML 图来展示设计思路。同时，本书中也对如何有效地使用 UML 做了深入的阐述。本书中有两个附录专门对 UML 进行了简介。

总之，本书是写给一线软件工程师的。如果你想学习 UML，如果你想学习如何去设计软件，如果你想学习设计模式，如果你想学习最好的软件开发实践，那么请阅读本书。

感谢 Robert C. Martin 先生为我们写了一本如此优秀的著作，此书必将成为经典。我从本书中学到了很多。我相信本书也不会令您失望。

邓 辉，2003.8.7 于上海

又及：

- 感谢本书作者 Robert C. Martin 先生总是非常及时、耐心地回答我的询问，他的回答使本译本更加贴近作者的本意。
- 感谢本译本的责编尤晓东和审校孟岩先生，他们的努力使本译本增色不少。
- 最后要感谢的是在我的翻译过程中一直陪伴在我身边，给了我很大帮助的人。她就是我的妻子孙鸣。我把最诚挚的感谢献给她。任何珍宝都没有她的陪伴重要。

译者简介

邓辉，软件工程师，对软件设计、面向对象、泛型以及模式有深入的研究和实践，尤其喜欢敏捷软件开发。在 IBM developerWorks 上发表过多篇关于设计、模式以及测试方面的文章。联系信箱：dhui@263.net。

原英文版序

刚刚交付了 Eclipse 开放源码项目的一个主要的版本之后，我就立即开始写这篇序言。我仍然还在恢复之中，思维还有些模糊。但是有一件事情我却比以往更加清楚，那就是：交付产品的关键因素是人，而不是过程。我们成功的诀窍很简单：和那些沉醉于交付软件的人一起工作，使用适合于自己团队的轻量过程进行开发，并且不断地去适应。

如果去了解我们团队中的开发人员，就会发现他们都是一些认为编程是开发活动的中心的人。他们不仅编写代码，而且还经常对代码进行融会贯通，以保持对系统的理解。使用代码验证设计所提供的反馈，对于获得对设计的信心来说是至关重要的。我们的开发人员知道模式、重构、测试、增量交付、频繁构建以及其他一些改变了我们看待当今方法学方式的 XP（极限编程）最佳实践的重要性。

对于那些具有高技术风险以及变化的需求的项目来说，熟练地掌握这种开发风格是它们取得成功的先决条件。虽然敏捷开发对于形式和文档保持低调，但是当涉及重要的开发实践时，它却表现出极度的关注。让这些实践“活”起来正是本书的中心内容。

Robert 是面向对象社团中一位长期的积极参与者，对于 C++实践、设计模式以及面向对象设计的一般原则都有贡献。他是一位 XP 和敏捷方法的早期的和直率的提倡者。本书就构建于这些贡献之上，覆盖了敏捷开发实践的全部内容。这是一项了不起的成就。不仅如此，Robert 在说明每一件事情时，都使用了研究案例和大量的代码，完全和敏捷实践相符。他阐明编程和设计的方式就是实际去编程。

这本书中充满了对于软件开发的明智的建议。不管你是想成为一个敏捷（Agile）开发人员，还是想提高已有的技能，它都是同样有用的。我一直在期盼着这本书，它没有令我失望。

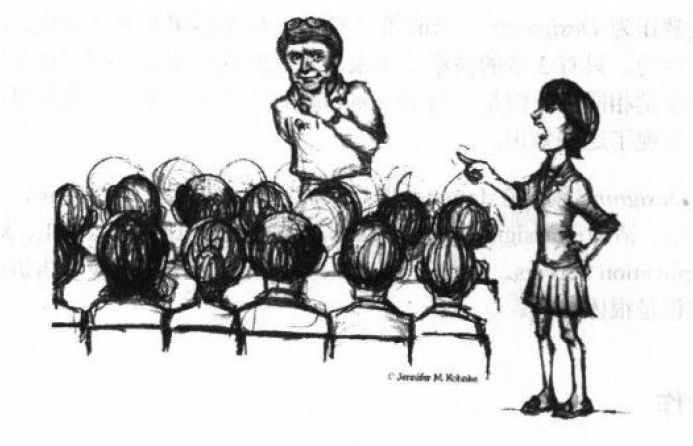
Erich Gamma
Object Technology International

献给 Ann Marie，Angela，Micah，Gina，Justin，Angelique，Matt 和 Alexis……

没有任何珍宝比我的家庭的陪伴更重要，

没有任何宝藏都比她们爱的安慰更丰富。

前　言



Bob，你说过会在去年写完这本书的。

——Claudia Frers, *UML World*, 1999

敏捷开发（Agile Development）是一种面临迅速变化的需求快速开发软件的能力。为了获取这种敏捷性，我们需要使用一些可以提供必要的纪律和反馈的实践。我们需要使用一些可以保持我们的软件灵活、可维护的设计原则，并且我们需要知道一些已经被证明针对特定的问题可以平衡这些原则的设计模式。本书试图把所有这 3 个概念编织在一起，使它们成为一个有机的整体。

本书首先描述了这些原则、模式以及实践，然后通过学习许多的案例来演示是如何应用它们的。更重要的是，研究案例介绍的并不是最终已完成的结果，而是设计的历程。你会看到设计者犯错误；你会看到他们是如何识别出错误并最终改正错误；你会看到他们对于难题的苦苦思索以及对于一些权衡和含糊问题的苦恼；你会看到设计的行为。

隐藏在细节之中

本书包含有许多的 Java 和 C++代码。我希望你能够仔细地学习这些代码，因为在很大程度上，代码正是本书的要旨。代码就是本书要讲的内容的实现。

本书采用了一种重复的讲解方式。它由一系列不同规模的案例研究组成。有一些非常的小，有一些却需要用几章来描述。每个案例研究之前都有一些针对该案例研究的预备内容。例如，在薪水支付案例研究之前，就有一些描述在该案例研究中用到的面向对象设计原则和模式的章节。

本书首先对开发实践和过程进行了讨论，其中穿插了许多小的案例研究以及示例。从这些穿插点处，本书转移到设计和设计原则的主题上，接着是一些设计模式、更多的管理包的设计原则以及更多的模式。所有这些主题都附有案例研究。

因此，请准备好学习一些代码并钻研一些 UML（统一建模语言）图。你将要学习的书籍是非常技术性的，并且其中要教授的知识和恶魔一样，也隐藏在细节之中。

一段小史

大约6年前，我写了一本名为：*Designing Object-Oriented C++ Application using the Booch Method* 的书。它是我的一部主要作品，并且它的效果和销量都令我非常满意。

本书开始时是被作为 *Designing* 一书的第2版的，但是结果却并非如此。在本书中所保留的原书中的内容是非常少的。只有3章的被维持下来，并且对这些章节进行了重大的修改。书的意图、精神以及许多的知识是相同的。但是，自 *Desinging* 出版6年以来，在软件设计方面我又学到了非常多的知识。本书表现了这些知识。

几年过去了！*Designing* 刚好在 Internet 爆炸式流行之前出版。从那时起，我们要面临的缩略词的数量已经翻了一倍，诸如：Design Patterns、Java、EJB、RMI、J2EE、XML、XSLT、HTML、ASP、JSP、Servlets、Application Servers、ZOPE、SOAP、C#、.NET 等等。我要告诉你，使本书的内容跟得上最新的技术知识是很困难的。

与 Booch 的合作

1997年，Booch 和我联系，让我帮他撰写他的非常成功 *Object-Oriented Analysis and Design with Applications* 一书的第3版。以前，我和 Grady 在一些项目中有过合作，并且我是他的许多作品（包括 UML）的热心读者和参与者。因此，我高兴地接受了。我邀请了我的好朋友 Jim Newkirk 来帮助完成这项工作。

在接下来的2年中，我和 Jim 为 Booch 的书撰写了许多的章节。当然，这些成果意味着我不可能在这本书中按照我本来想的那样投入同样多的努力，但是我觉得 Booch 的书值得我这样做。另外，当时本书完全只是 *Designing* 的第2版，并且我的心思也不在其上。如果我要说一些东西的话，我想说一些新的并且不同的东西。

糟糕的是，Booch 的这个版本的书并没有完成。在正常的时间里很难抽出空来撰写一本书。在浮躁的“.com”泡沫期间，这几乎是不可能的。Grady 也更加忙于 Rational 以及一些像 Capapulse 这样的新风险投资企业的事务。因此这项工作就停止了。最后，我问 Grady 和 Addison-Wesley 是否可以把我和 Jim 撰写的那些章节包含在这本书中。他们很有风度地同意了。于是，一些案例研究和 UML 的章节就从此而来。

极限编程的影响

1998年晚期，XP 崭露头角，并向我们所珍爱的关于软件开发的观念进行挑战。我们是应该在编写任何代码前先创建许多 UML 图呢？还是应该避开任何种类的 UML 图而仅仅编写大量代码呢？我们是应该编写大量的描述我们设计的叙述性文档呢？还是应该努力使代码具有自释义能力以及表达力，这样辅助性的文档就不再必要了呢？我们应该结对编程吗？我们应该在编写产品代码前先编写测试吗？我们应该做什么呢？

我是凑巧接触到这次革命的。在20世纪90年代的中后期，Object Mentor 公司在面向对象（OO）设计以及项目管理问题上帮助了许多公司。我们帮助这些公司完成它们的项目。作为帮助的一部分，