TURING 图灵程序设计丛书

Addison Wesley

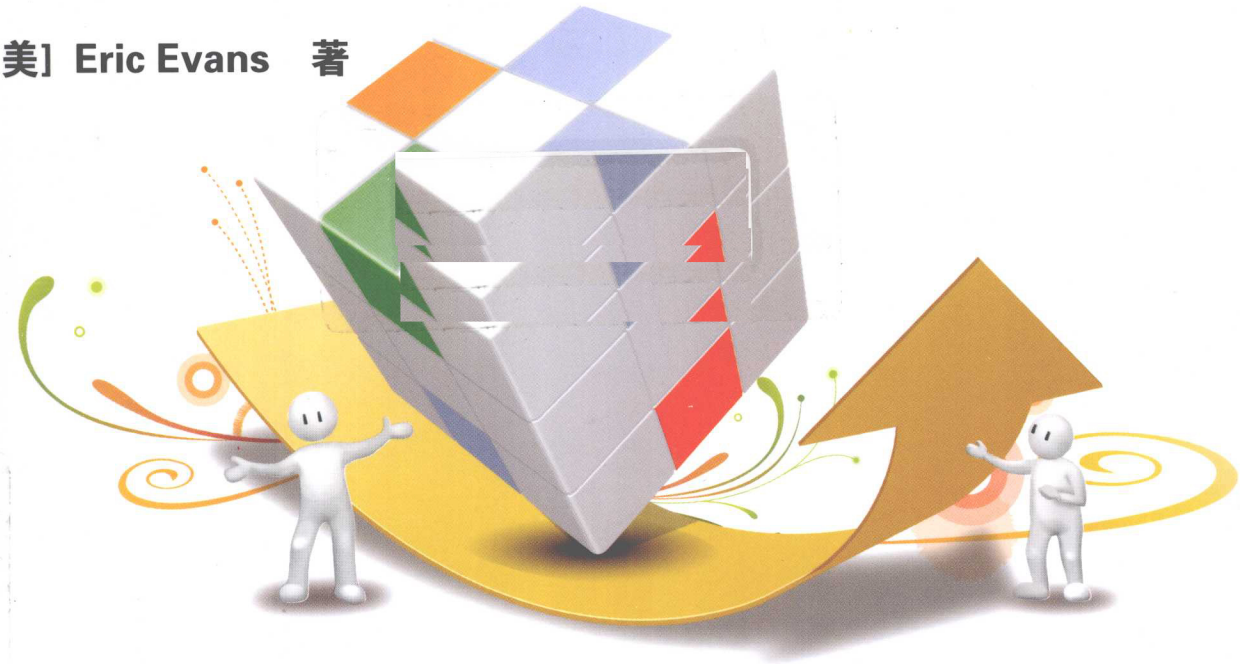
Domain-Driven Design

Tackling Complexity in the Heart of Software

领域驱动设计

软件核心复杂性应对之道（英文版）

[美] Eric Evans 著



- 众多世界级软件大师鼎力推荐
- 凝聚领域建模专家数十年的实战经验
- 深度剖析构建高质量复杂系统的核心技术

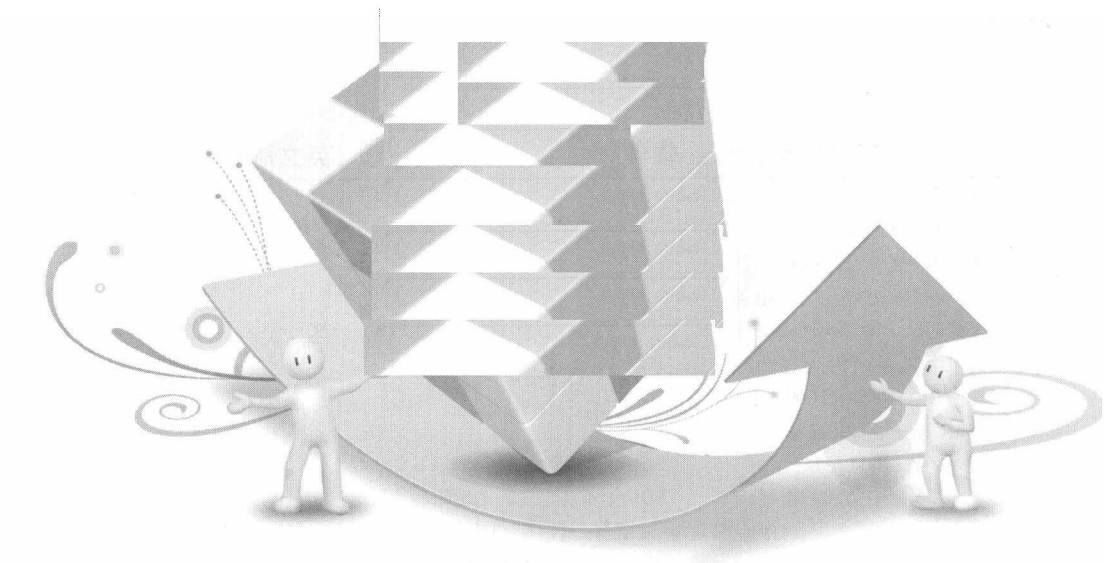
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Domain-Driven Design
Tackling Complexity in the Heart of Software

领域驱动设计
软件核心复杂性应对之道

（英文版）



人民邮电出版社
北　京

图书在版编目（CIP）数据

领域驱动设计：软件核心复杂性应对之道＝
Domain-Driven Design: Tackling Complexity in the
Heart of Software：英文/（美）埃文斯（Evans, E.）
著. —北京：人民邮电出版社，2010.4
（图灵程序设计丛书）
ISBN 978-7-115-22407-1

I. ①领… II. ①埃… III. ①软件设计－英文 IV.
①TP311.5

中国版本图书馆CIP数据核字（2010）第032034号

内 容 提 要

本书是领域驱动设计领域的经典之作。全书围绕着设计和开发实践，结合若干真实的项目案例，向读者阐述如何在真实的软件开发中应用领域驱动设计。书中给出了领域驱动设计的系统化方法，并将人们普遍接受的一些最佳实践综合到一起，融入了作者的见解和经验，展现了一些可扩展的设计最佳实践、经验验证过的技术以及便于应对复杂领域的软件项目开发的基本原则。

本书适合各层次的面向对象软件开发人员、系统分析员阅读。

图灵程序设计丛书

领域驱动设计：软件核心复杂性应对之道（英文版）

◆ 著 [美] Eric Evans
责任编辑 杨海玲
◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
中国铁道出版社印刷厂印刷
◆ 开本：800×1000 1/16
印张：34.75
字数：663千字 2010年4月第1版
印数：1-2 500册 2010年4月北京第1次印刷
著作权合同登记号 图字：01-2006-4173号
ISBN 978-7-115-22407-1

定价：82.00元

读者服务热线：（010）51095186 印装质量热线：（010）67129223
反盗版热线：（010）67171154

序

有很多因素会使软件开发复杂化，但最根本的原因是问题领域本身的错综复杂。如果你要为一家人员复杂的企业提高自动化程度，那么你开发的软件将无法回避这种复杂性，你所能做的只有控制这种复杂性。

控制复杂性的关键是有一个好的领域模型，这个模型不应该仅仅停留在领域的表面，而是要透过表象抓住领域的实质结构，从而为软件开发人员提供他们所需的支持。好的领域模型的价值惊人，但要想开发出好的模型也并非易事。精通此道的人并不多，而且这方面的知识也很难传授。

Eric Evans就是为数不多的能够创建出优秀领域模型的人之一。我是在与他合作时发现他的这种才能的——发现一个客户竟然比我技术更精湛，这种感觉有些奇妙。我们的合作虽然短暂，但却充满乐趣。从那之后我们一直保持联系，我也有幸见证了本书整个“孕育”过程。

本书绝对值得期待。

它实现了Eric的两个宏伟抱负，一是描述并建立了领域建模艺术的词汇库，二是提供了一个参考框架，人们可以用来解释相关活动，并将这门难学的技艺传授给他人。本书在写作过程中，也带给我很多新想法，如果哪位概念建模方面的老手没有从阅读本书中获得大量的新思想，那我反而该惊诧莫名了。

Eric还对我们多年以来学过的知识进行了归纳总结。首先，在领域建模过程中不应将概念与实现割裂开来。高效的领域建模人员

不仅应该能够在白板上与会计师进行讨论，而且还应该能与程序员一道编写Java代码。之所以要具备这些能力，一部分原因是如果不考虑实现问题就无法构建出有用的概念模型。但概念与实现密不可分的最主要原因在于，领域模型的最大价值是它提供了一种通用语言，这种语言是将领域专家和技术人员联系在一起的纽带。

我们将从本书中学到的另一个经验是领域模型并不是按照“先建模，后实现”这个次序来工作的。像很多人一样，我也反对“先设计，再构建”这种固定的思维模式。Eric的经验告诉我们，真正强大的领域模型是随着时间演进的，即使是最有经验的建模人员也往往发现他们是在系统的初始版本完成之后才有了最好的想法。

我衷心希望本书成为一本有影响力的著作，并希望本书能够将如何利用领域模型这一宝贵工具的知识传授给更多的人，从而为这个高深莫测的领域梳理出一个结构，并使它更有凝聚力。领域模型对软件开发的控制有着巨大影响，不管软件开发是用什么语言或环境实现的。

最后，也是很重要的一点，我最敬佩Eric的一点是他敢于在本书中谈论自己的一些不成功经历。很多作者都喜欢摆出一副无所不能的架势，有时着实让人不屑。但Eric清楚地表明他像我们大多数人一样，既品尝过成功的美酒，也体验过失败的沮丧。重要的是他能够从成功和失败中学习，而对我们来说更重要的是他能够将所有经验传授给我们。

Martin Fowler
2003年4月

试读结束：需要全本请在线购买： www.ertongbook.com

前　言

至少20年前，一些顶尖的软件设计人员就已经认识到领域建模和设计的重要性，但令人惊讶的是，这么长时间以来几乎没有人去写出点儿什么，告诉大家应该做哪些工作或如何去做。尽管这些工作还没有被清除地表述出来，但一种新的思潮已经形成，它像一股暗流一样在对象社区中涌动，我把这种思潮称为领域驱动设计。

过去10年中，我在几个业务和技术领域开发了一些复杂的系统。我在设计和开发过程中尝试了一些最佳实践，它们都是面向对象开发高手用过的一些领先技术。有些项目非常成功，但有几个项目却失败了。成功的项目有一个共同的特征，那就是都有一个丰富的领域模型，这个模型在迭代设计的过程中不断演变，而且成为项目不可分割的一部分。

本书为作出设计决策提供了一个框架，并且为讨论领域设计提供了一个技术词汇库。本书将人们普遍接受的一些最佳实践综合到一起，在这些最佳实践中还融入了我自己的见解和经验。面对复杂领域的软件开发团队可以利用这个框架来系统性地应用领域驱动的设计。

三个项目的对比

谈到领域设计实践对开发结果的巨大影响时，我的记忆中立即就会跳出三个项目，它们就是鲜活的例子。虽然这三个项目都交付了有用的软件，但只有一个项目实现了宏伟的目标——交付了能够

满足组织后续需求、可以不断演进的复杂软件。

我要说的第一个项目完成得很迅速，它提供了一个简单实用的Web交易系统。开发人员主要凭直觉开发，但这并没有妨碍他们，因为简单软件的编写并不需要过多地注意设计。由于最初的这次成功，人们对未来开发的期望值变得极高。我就是在这个时候被邀请开发它的第二个版本的。当我仔细研究这个项目时，发现他们没有使用领域模型，甚至在项目中没有一种公共语言，而且项目完全没有一种结构化的设计。项目领导者对我的评价并不赞同，于是我拒绝了这项工作。一年后，这个项目团队陷入困境，无法交付第二个版本。尽管他们在技术的使用方面也值得商榷，但真正挫败他们的是业务逻辑。他们的第一个版本过早地变得僵化，成为一个维护代价十分高昂的遗留系统。

要想克服这种复杂性，需要非常严格地使用领域逻辑设计方法。在我职业生涯的早期，我幸运地完成了一个非常重视领域设计的项目，这就是我要说的第二个项目。这个项目的领域复杂性与上面提到的那个项目相仿，它最初也小获成功，为贸易机构提供了一个简单的应用程序。但在最初交付之后紧跟着又进行了连续的加速开发。每次迭代都为上一个版本在功能的集成和完善上增加了非常好的新选项。开发团队能够按照贸易商的要求提供灵活性和扩展性。这种良性发展直接归功于深刻的领域模型，它得到了反复精化，并在代码中得以体现。当团队对该领域有了新的理解后，领域模型也随之深化。开发人员之间、开发人员与领域专家之间的沟通质量都得到改善，而且设计不但没有加重维护负担，反而变得易于修改和扩展。

遗憾的是，仅靠认真使用模型并不会使项目达到这样的良性循环。我要说的第三个项目就是这种情况，它开始制订的目标很高，打算基于一个领域模型建立一个全球企业系统，但在经过了几年的屡战屡败之后，不得不降格以求，最终“泯然众人矣”。团队拥有很好的工具，对业务也有较好的理解，也非常认真地进行了建模。但团队却错误地将开发人员的角色独立出来，导致建模与实现脱节，因此设计无法反映不断深化的分析。不管在什么情况下，业务对象设计得再详细，也无法保证它们能够严丝合缝地被整合到复杂的应用程序中。反复的迭代并没有使代码得以改进，因为开发人员的技术水平参差不齐，他们没有认识到他们使用了非正式的风格和技术体系来创建基于模型的对象（这些对象也充当了实用的、可运

行的软件）。几个月过去了，开发工作由于巨大的复杂性而陷入困境，而团队对项目也失去了一致的认识。经过几年的努力，项目确实创建了一个适当的、有用的软件，但团队已经放弃了当初的宏伟抱负，也不再重视模型。

复杂性的挑战

很多因素可能会导致项目偏离轨道，如官僚主义、目标不清、资源缺乏，等等。但真正决定软件复杂性的是设计方法。当复杂性失去控制时，开发人员就无法很好地理解软件，因此无法轻易、安全地更改和扩展它。而好的设计则可以为开发复杂特性创造更多机会。

一些设计因素是技术上的。软件的网络、数据库和其他技术方面的设计耗费了人们大量的精力。很多书籍都介绍过如何解决这些问题。大批开发人员很注意培养自己的技能，并紧跟每一次技术进步。

然而很多应用程序最主要的复杂性并不在技术上，而是来自领域本身、用户的活动或业务。当这种领域复杂性在设计中没有得到解决时，基础技术的构思再好也是无济于事。成功的设计必须系统地考虑软件的这个核心方面。

本书有两个前提：

（1）在大多数软件项目中，主要的焦点应该是领域和领域逻辑。

（2）复杂的领域设计应该基于模型。

领域驱动设计是一种思维方式，也是一组优先任务，它旨在加速那些必须处理复杂领域的软件项目的开发。为了实现这个目标，本书给出了一整套完整的设计实践、技术和原则。

设计过程与开发过程

设计书就是讲设计，过程书只是讲过程。它们之间很少互相参考。设计和过程本身就是两个足够复杂的主题。本书是一本设计书，但我相信设计与过程这二者是密不可分的。设计思想必须被成功实现，否则它们就只是纸上谈兵。

当人们学习设计技术时，各种可能性令他们兴奋不已，然而真

实项目的错综复杂又会为他们泼上一盆冷水。他们无法用所使用的技术来贯彻新的设计思想，或者不知道何时应该为了节省时间而放弃某个设计方面，何时又应该坚持不懈直至找到一个干净利落的解决方案。开发人员可以抽象地讨论设计原则的应用，而且他们也确实在进行着这样的讨论，但更自然的做法应该是讨论如何完成实际工作。因此，虽然本书是一本有关设计的书，但我会在必要的时候穿越这条人为设置的边界，进入过程的领域。这将有助于将设计原则放到一个适当的语境下进行讨论。

虽然本书并不局限于某一种特定的方法，但主要还是面向“敏捷开发过程”这一新体系。特别地，本书假定项目必须遵循两个开发实践，要想应用书中所讲的方法，必须先了解这两个实践。

(1) 迭代开发。人们倡导和实践迭代开发已经有几十年时间了，而且它是敏捷开发方法的基础。在敏捷开发和极限编程（XP）的文献中有很多关于迭代开发的精彩讨论，其中包括*Surviving Object-Oriented Projects*[Cockburn 1998]和*Extreme Programming Explained*[Beck 1999]。

(2) 开发人员与领域专家具有密切的关系。领域驱动设计的实质就是消化吸收大量知识，最后产生一个反映深层次领域知识并聚焦于关键概念的模型。这是领域专家与开发人员的协作过程，领域专家精通领域知识，而开发人员知道如何构建软件。由于开发过程是迭代式的，因此这种协作必须贯穿整个项目的生命周期。

极限编程这个概念是由Kent Beck、Ward Cunningham和其他人共同提出的（参见 [Beck 2000]），它是敏捷过程最重要的部分，也是我使用得最多的一种编程方法。为了使讨论更加具体，整本书都将使用XP作为基础讨论设计和过程的交互。本书论述的原则很容易应用于其他敏捷过程。

近年来，反对“精细开发方法学”（elaborate development methodology）的呼声渐起，人们认为无用的静态文档以及死板的预先规划和设计加重了项目的负担。相反，敏捷过程（如XP）强调的是应对变更和不确定性的能力。

极限编程承认设计决策的重要性，但强烈反对预先设计。相反，它将相当大的精力投入到沟通和提高项目快速变更能力的工作中。具有这种反应能力之后，开发人员就可以在项目的任何阶段只利用“最简单而管用的方案”，然后不断进行重构，一步一步做出小的

设计改进，最终达到满足客户真正需要的设计。

这种极端的简约主义是解救那些过度追求设计的执迷者的良方。那些几乎没有价值的繁琐文档只会为项目带来麻烦。项目受到“分析瘫痪症”的困扰，团队成员十分担心会出现不完美的设计，这导致他们根本没法进展。这种状况必须得到改变。

遗憾的是，这些有关过程的思想可能会被误解。每个人对“最简单”都有不同的定义。持续重构其实是一系列小规模的重新设计，没有严格的设计原则的开发人员将会创建出难以理解或修改的代码，这恰好与敏捷的精神相悖。而且，虽然对意外需求的担心常常导致过度设计，但试图避免过度设计又可能走向另一个极端——不敢做任何深入的设计思考。

实际上，XP最适合那些对设计的感觉很敏锐的开发人员。XP过程假定人们可以通过重构来改进设计，而且可以经常、快速地完成重构。但重构本身的难易程度取决于先前的设计选择。XP过程试图改善团队沟通，但模型和设计的选择有可能使沟通更明确，也有可能会混淆沟通。

本书将设计和开发实践结合起来讨论，并阐述领域驱动设计与敏捷开发过程是如何互相增强的。在敏捷开发过程中使用成熟的领域建模方法可以加速开发。过程与领域开发之间的相互关系使得这种方法比任何“纯粹”真空式的设计更加实用。

本书的结构

本书分为4个部分。

*第一部分“让领域模型发挥作用”*提出领域驱动开发的基本目标，这些目标是后面几部分中所讨论的实践的驱动因素。由于软件开发方法有很多，因此第一部分还定义了一些术语，并给出了用领域模型来驱动沟通和设计的总体含义。

*第二部分“领域驱动设计的构建块”*将面向对象领域建模中的一些核心的最佳实践提炼为一组基本的构建块。这一部分主要是消除模型与实际运行的软件之间的鸿沟。团队一致使用这些标准模式就可以使设计井然有序，并且使团队成员更容易理解彼此的工作。使用标准模式还可以为公共语言贡献术语，使得所有团队成员可以使用这些术语来讨论模型和设计决策。

但这一部分的主旨是讨论一些能够保持模型和实现之间互相协调

并提高效率的设计决策。要想达到这种协调，需要密切注意个别元素的一些细节。这种小规模的仔细设计为开发人员提供了一个稳固的基础，在此基础上就可以应用第三部分和第四部分讨论的建模方法了。

第三部分“通过重构来加深理解”讨论如何将构建块装配为实用的模型，从而实现其价值。这一部分没有直接讨论深奥的设计原则，而是着重强调一个发现过程。有价值的模型不是立即就会出现的，它们需要对领域的深入理解。这种理解是一步一步得到的，首先需要深入研究模型，然后基于最初的（可能是不成熟的）模型实现一个初始设计，再反复改进这个设计。每次团队对领域有了新的理解之后，都需要对模型进行改进，使模型反映出更丰富的知识，而且必须对代码进行重构，以便反映出更深刻的模型，并使应用程序可以充分利用模型的潜力。这种一层一层“剥洋葱”的方法有时会创造一种突破的机会，使我们得到更深刻的模型，同时快速进行一些更深入的设计修改。

探索本身是永无止境的，但这并不意味着它是随机的。第三部分深入阐述一些指引我们保持正确方向的建模原则，并提供了一些指导我们进行探索的方法。

第四部分“战略设计”讨论在复杂系统、大型组织以及与外部系统和遗留系统的交互中出现的复杂情况。这一部分探讨了作为一个整体应用于系统的3条原则：上下文、提炼和大规模结构。战略设计决策通常由团队制定，或者由多个团队共同制定。战略设计可以保证在大型系统或应用程序（它们应用于不断延伸的企业级网络）上全面实现第一部分提出的目标。

本书通篇讨论使用的例子并不是一些过于简单的“玩具式”问题，而是全部选自实际项目。

本书的大部分内容实际上是作为一系列的“模式”编写的。但读者无需顾忌这一方法也应该能够理解本书，对模式的风格和格式感兴趣的读者可以参考附录。

补充材料可以参考http://domaindrivendesign.org，该网站提供了示例代码和社区讨论内容。

本书面向的读者

本书主要是为面向对象软件开发人员编写的。软件项目团队的

试读结束：需要全本请在线购买： www.ertongbook.com

大部分成员都能够从本书的某些部分获益。本书最适合那些正在项目上尝试这些实践的人员，以及那些已经在这样的项目上积累了丰富经验的人员。

要想从本书受益，掌握一些面向对象建模知识是非常必要的，例如UML图和Java代码，因此一定要具备基本读懂这些语言的能力，但不必精通细节。了解极限编程的知识有助于从这个角度来理解开发过程的讨论，但不具备这一背景知识也能读懂这些内容。

一些中级软件开发人员可能已经了解面向对象设计的一些知识，也许读过一两本软件设计书，那么本书将填补这些读者的知识空缺，并向他们展示如何在实际的软件项目上应用对象建模技术。本书将帮助这些开发人员学会用高级建模和设计技巧来解决实际问题。

高级软件开发人员或专家可能会对书中用于处理领域的综合框架感兴趣。这种系统性的设计方法将帮助技术负责人指导他们的团队保持正确的方向。此外，本书从头至尾所使用的明确术语将有助于高级开发人员与他们的同行沟通。

本书采用记叙体，读者可以从头至尾阅读，也可以从任意一章的开头开始阅读。具有不同背景知识的读者可能会有不同的阅读方式，但我推荐所有读者从第一部分的引言和第1章开始阅读。除此之外，本书的核心是第2、3、9和14章。已经掌握一定知识的读者可以采取跳跃式阅读的方式，通过阅读标题和粗体字内容即可掌握要点。一些高级读者则可以跳过前两部分，而重点阅读第三部分和第四部分的内容。

除了这些主要读者以外，分析员和相关的技术项目经理也可以从阅读本书中获益。分析员在掌握了领域与设计之间的联系之后，能够在敏捷项目中作出更卓越的贡献，也可以利用一些战略设计原则来更有重点地组织工作。

项目经理感兴趣的重点是提高团队的工作效率，并致力于设计出对业务专家和用户有用的软件。由于战略设计决策与团队组织和工作风格紧密相关，因此这些设计决策必然需要项目领导者的参与，而且对项目的路线有着重要的影响。

领域驱动团队

尽管开发人员个人能够从理解领域驱动设计中学到有价值的设

计技术和观点，但最大的好处却来自团队共同应用领域驱动设计方法，并且将领域模型作为项目沟通的核心。这样，团队成员就有了一种公共语言，可以用来进行更充分的沟通，并确保围绕软件来进行沟通。他们将创建出一个与模型步调一致的清晰的实现，从而为应用程序的开发提供帮助。所有人都了解不同团队的设计工作之间的互相联系，而且他们会一致将注意力集中在那些对组织最有价值、最与众不同的特性的开发上。

领域驱动设计是一项艰巨的技术挑战，但它也会带来丰厚的回报，当大多数软件项目开始僵化而成为遗留系统时，它能为你敞开一扇机会的大门。

致　谢

本书的创作历时4年多，其间经历了诸多工作形式，在这个过程中很多人为我提供了帮助和支持。

感谢那些阅读本书书稿并提出意见的人。没有这些人的反馈意见，本书将不可能出版。其中有几个团队和一些人员对本书的评阅给予了特别的关注。由Russ Rufer和Tracy Bialek领导的硅谷模式小组（Silicon Valley Patterns Group）花费了几周时间详细审阅了本书完整的第一稿。由Ralph Johnson领导的伊利诺伊大学的阅读小组也花费了几周时间审阅了本书的第二稿。这些小组长期、精彩的讨论对本书产生了深远的影响。Kyle Brown和Martin Fowler提供了细致入微的反馈意见和宝贵的建议，也给了我无价的精神支持（以及我们坐在一起钓鱼的时候）。Ward Cunningham的意见帮助我弥补了一些重大的缺陷。Alistair Cockburn在早期给了我很多鼓励，并和Hilary Evans一起帮助我完成了整个出版过程。David Siegel和Eugene Wallingford帮助我避免了很多技术上的错误。Vibhu Mohindra和Vladimir Gitlevich不厌其烦地检查了所有代码示例。

Rob Mee看了我对一些素材所做的早期研究，并在我尝试表达这种设计风格的时候与我进行了头脑风暴活动，帮我产生了很多新的想法。他后来又与我一起仔细探讨了后面的书稿。

本书在写作过程中经历了一次重大转折，这完全归功于Josh Kerievsky。他劝说我在写作本书时借鉴“亚历山大”模式[①]，后来本书正是按这种方式组织的。在1999年PLoP会议临近时的忙碌时刻，Josh还帮我收集第二部分的材料，首次将它们组织为更严密的

① 克里斯托弗·亚历山大（Christopher Alexander），1936年10月4日出生于奥地利的维也纳，是一名建筑师，以其设计理论和丰富的建筑设计作品而闻名于世。亚历山大认为，建筑的使用者比建筑师更清楚他们需要什么，他创造并以实践验证了“模式语言”，建筑模式语言赋予所有人设计并建造建筑的能力。亚历山大的代表作是《建筑模式语言》，该书对计算机科学领域中的“设计模式”运动产生了巨大的影响。亚历山大创立的增量、有机和连贯的设计理念也影响了“极限编程”运动。

——编者注

形式。这些材料成了一粒种子，本书大部分后续内容都是围绕这些内容创作的。

还要感谢Awad Faddoul，我有数百个小时坐在他的咖啡厅中写作。咖啡厅宁静优雅，窗外的湖面上总有片片风帆，我正是这样才坚持写下去。

此外还要感谢Martine Jousset、Richard Paselk和Ross Venables，他们拍摄了一些非常精美的照片，用来演示一些关键概念（参见本书附录后面的照片出处）。

在构思本书之前，我必须先要形成我自己对软件开发的看法和理解。这个过程得到了一些杰出人员的无私帮助，他们是我的良师益友。David Siegel、Eric Gold和Iseult White各自从不同方面帮助我形成了对软件设计的思考方式。同时，Bruce Gordon、Richard Freyberg和Judith Segal博士也从不同角度帮助我找到了项目的成功之路。

我自己的观念就是从那时的思想体系中自然而然发展形成的。有些内容我在正文中清楚地列了出来，并且在可能的地方标明了出处。还有些可能是十分基础的知识，我甚至自己都没有意识到它们对我产生了影响。

我的硕士论文导师Bala Subramanium博士是我在数学建模方面的引路人，当时我们用数学建模来进行化学反应动力学方面的研究。虽说建模本身没什么稀奇，但那时的工作是引导我创作本书的一部分原因。

在更早之前，我的母亲Carol和父亲Gary对我思维模式的形成产生了很大影响。还有几位特别值得一提的教师激发了我的兴趣，帮助我打下坚实的基础，在此感谢Dale Currier（我的高中数学老师）、Mary Brown（我的高中英文写作老师）和Josephine McGlamery（我上6年级时的自然科学老师）。

最后，感谢我的朋友和家人，以及Fernando De Leon，感谢他们一直以来给我的鼓励。

CONTENTS

试读结束：需要全本请在线购买： www.ertongbook.com