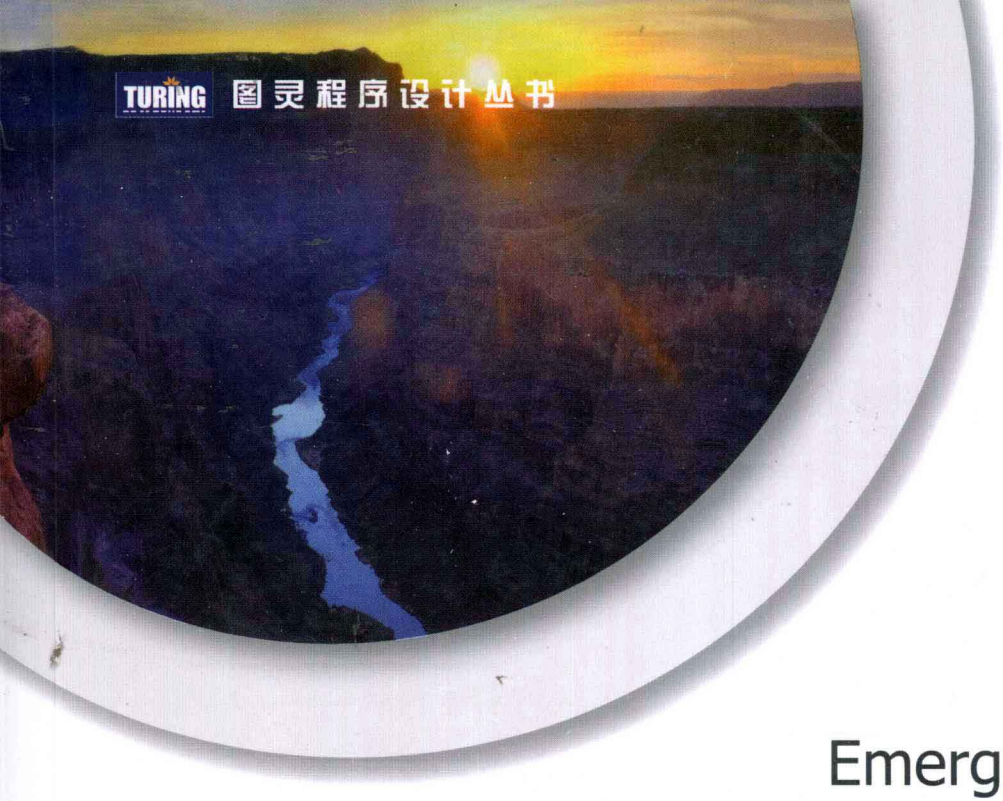


TURING 图灵程序设计丛书

Addison Wesley

Emergent Design

The Evolutionary Nature of Professional Software Development

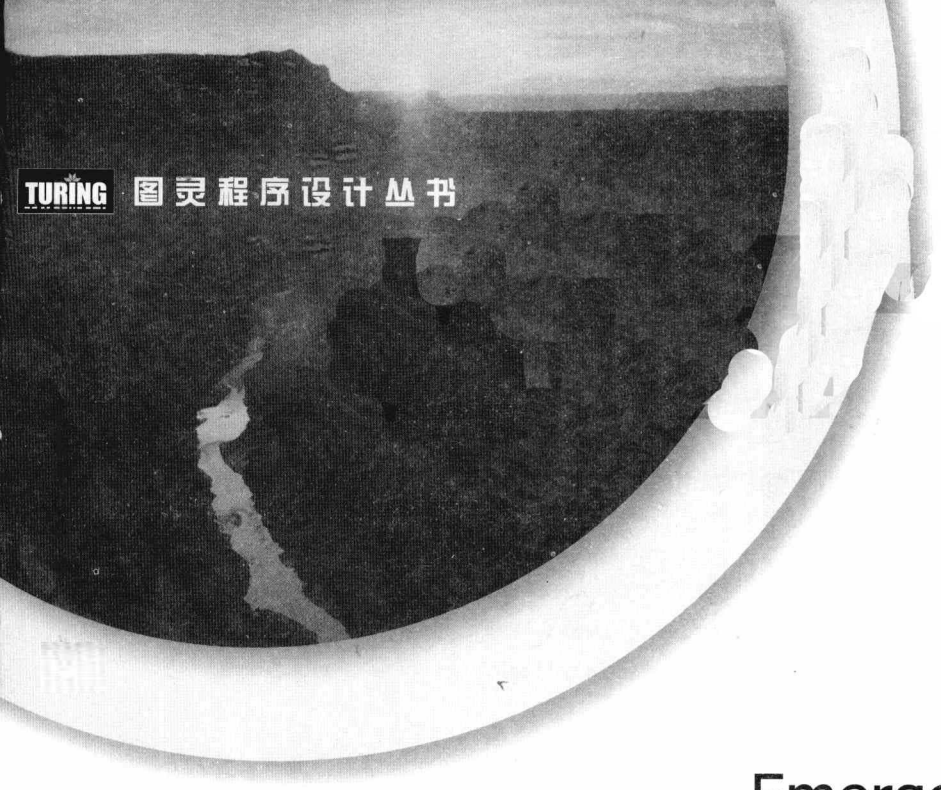
浮现式设计

专业软件开发的演进本质

[美] Scott L. Bain 著
赵俐 华洁 译

- 一部敏捷开发与软件设计的至尊宝典
- 深入剖析软件开发过程的演进本质
- 敏捷开发大师Alan Shalloway鼎力推荐

人民邮电出版社
POSTS & TELECOM PRESS



Emergent Design

The Evolutionary Nature of Professional Software Development

浮现式设计

专业软件开发的演进本质

[美] Scott L. Bain 著
赵俐 华洁 译

人民邮电出版社

图书在版编目（CIP）数据

浮现式设计 : 专业软件开发的演进本质 / (美) 贝恩 (Bain, S. L.) 著 ; 赵俐, 华洁译. -- 北京 : 人民邮电出版社, 2011.8
(图灵程序设计丛书)
书名原文: Emergent Design: The Evolutionary Nature of Professional Software Development
ISBN 978-7-115-25978-3

Ⅰ. ①浮… Ⅱ. ①贝… ②赵… ③华… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2011)第141532号

内 容 提 要

浮现式设计是一种敏捷技术，强调在开发过程中不断演进。本书的讨论围绕着专业软件开发方法的演进主题展开，强调了让软件成为一个真正专业的重要性，以及以演进方式开发软件的重大意义。书中谈到了如何在演进过程中综合运用设计模式、重构、单元测试和测试驱动开发等实践，以及何时制定耦合、内聚和封装等关键决策，而且通过准确生动的示例说明了如何开发出真正有用的软件。

本书主要面向软件开发者群体，尤其是对敏捷开发感兴趣的程序设计人员。

图灵程序设计丛书

浮现式设计：专业软件开发的演进本质

◆ 著 [美] Scott L. Bain
译 赵 俐 华 洁
责任编辑 傅志红
执行编辑 李 瑛

◆ 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 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：18.75
字数：443千字 2011年8月第1版
印数：1-4 000册 2011年8月北京第1次印刷
著作权合同登记号 图字：01-2010-8064号
ISBN 978-7-115-25978-3

定价：59.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

译 者 序

浮现式设计是一种敏捷技术，其主要思想是把重要的决策推迟到最后时刻，而不是在一开始就把一切确定下来，同时允许设计在开发过程中不断演进。在软件工程领域讲浮现式设计的书并不多见，好书就更是难求了，这也是吸引我接受本书翻译任务的主要原因。作为本书的译者和首批读者，我觉得它无疑是一部石破天惊之作，因此，我向各位读者强烈推荐这本书。

本书的讨论围绕着“专业软件开发的演进本质”这一主题展开，强调了让软件成为一门真正的专业的重要性，以及以演进方式来开发软件的重大意义。作者不仅教会了我们如何在一个演进的过程中综合运用设计模式、重构、单元测试和测试驱动开发等实践，如何以及何时制定诸如耦合、内聚和封装这样的关键决策，而且还说明了如何才能开发出真正有用的软件，因为不被使用的软件是毫无价值的。此外，作者在现实生活中还是一个能工巧匠，在书中，你会发现他自己动手制作的衣橱和亭子，而所有这些竟然全是靠一把手锯完成的！

本书还有一大特色，就是附录B汇总了书中涉及的所有模式，作者通过他自己多年的理解和感悟解读了这些模式，在讲解时还以现实生活中的例子作为比喻，生动展现了这些模式的精髓和用法。虽然这不是一本讲模式的书，但这部分几乎相当于全书五分之一篇幅的内容无疑对我们理解模式大有帮助。

作者在致谢中说道：“当一个人快要写完一本书时，难免有些当局者迷，我在它上面花了太长的时间，离它太近了，以至于无法真正看清其全貌。”此时此刻，我所感受到的也几乎完全一样，本书的翻译工作历时半年，我真的离它太近了，虽然经过了多次检查，但有些错误和疏漏可能我永远也无法发现，在此恳请各位读者批评指正。最后，感谢人民邮电出版社图灵公司傅志红老师，她的宝贵意见帮助我避免了很多错误和疏漏。此外还要感谢图灵各位编辑所做的大量后期工作。

译 者

2011年4月

丛书序言

Net Objectives的产品开发系列丛书

Alan Shalloway，Net Objectives公司CEO

如果你是像我一样的人，你会跳过这篇序言，认为这里不会讲什么重要内容。这么想是个错误。除非你已经在本系列丛书的另一本中读过这篇序言了，否则请你一定在本书开篇之际花点时间读完这个序言。我将为你讲述一个大多数人都知道却往往没有思考过的故事。这个故事讲的是到底是什么使得我们这个行业备受困扰。这个故事也是我们编写这个系列丛书以及这本书的背景。

我自从1970年就踏上软件开发之旅了。但直到今天，我感到它仍然像40年前一样，给我带来全新的体验。它的无穷魅力让我不断思索如何把工作做得更好，它也让我认识到自己的能力是多么地有限，从而使得我一直保持谦虚谨慎。我热爱软件开发。

在我的职业生涯中，我对其他行业也很感兴趣，特别是工程学和建筑学。工程和建筑学经历了一些重大失败，例如比萨斜塔、塔科马海峡大桥和哈勃望远镜。在工程学的早期，人们对力学作用的认识很肤浅。大部分工程师们尝试改进工程实践，并从失败中学习经验教训。人们花了相当长的时间（几个世纪）才对工程学原理有了扎实的理解。但这是否意味着软件开发人员就有借口花费更多时间来理解软件开发的基本原理呢？不。

工程师们在建造大桥之前，绝对不会不考虑一些众所周知的实践（压力、伸缩，等等），而软件开发人员却可以每天根据自己的喜好来编写软件，而且几乎不会或很少受到其他开发人员的指责，为什么会这样呢？

然而，这还只是故事的一部分。有趣的是，剩下的故事就是我们为什么把这个丛书叫做“Net Objectives产品开发系列丛书”的原因。Net Objectives很好理解，因为这个系列中的所有书要么是由Net Objectives员工写的，要么是由一些与Net Objectives意气相投的作者写的。那么为什么要叫“产品开发”丛书呢？因为当构建软件时，我们应该始终牢记一件事情，就是软件开发实际上等于产品开发。

2003年，Michael Kennedy在他所著的*Product Development for Lean Enterprise*一书中，给出了产品开发的定义：“（产品开发）是一种集体活动或体系，公司利用这个过程把技术和思想转化为一系列满足客户需要和公司战略目标的产品。”

Mary Poppendieck和Tom Poppendieck夫妇在他们合著的*Implementing Lean Software Development: From Concept to Cash*（2006）这本优秀的书中指出：

> 软件被植入其中的产品、活动和过程才是真正意义上被开发的产品。软件开发只是整体产品开发过程的一个子集。由此，从实际意义来讲，我们可以把软件开发称为产品开发的一个子集。因此，如果我们想要理解精益软件开发（Lean Software Development），就需要弄明白优秀的产品开发是由什么构成的。

换言之，软件本身并不重要，重要的是它为业务、客户以及用户贡献的价值。当开发软件时，我们必须始终注意我们的工作增加了什么价值。我们或多或少都知道这个道理。但软件开发组织中经常会出现一种壁垒森严的“筒仓”效应，这妨碍了人们的协作，使人们无法把注意力集中在创造价值上。

在整个组织中实现高效产品开发的最好方式（或许也是唯一方式）是经过缜密的思考，把用于指导企业的精益原则、用于管理团队的敏捷实践以及技术技巧（测试驱动的开发和设计模式等）这三者很好地结合起来。这也正是本系列丛书的创作动机。

长期以来，软件开发行业看起来就像是一个永无休止的钟摆一样，从毫无过程到过程太多，然后又回到没有过程——从极为重视企业控制的严格方法到只关注当前项目的自律团队。现在，该到管理层和开发人员齐心协力共同实现企业价值最大化的时候了。我们相信精益原则将在这方面为我们提供指导。

精益原则告诉我们应该关注自己正在工作的系统，并坚决地改进它们，从而提高工作的速度和质量（这同时也会降低成本）。这需要：

- ❑ 团队拥有他们的系统，并不断改进它们；
- ❑ 管理层培训并支持他们的团队来改进系统；
- ❑ 积极采用有利于提高工作质量的实践。

或许在软件开发行业中，看起来我们距离这个目标非常遥远，但潜力是毋容置疑的。精益方法有助于我们践行前两条原则，而我们对于技术编程和设计的理解已经足够成熟，足以帮助我们实现第三个目标。

随着我们把精益开发、敏捷方法、设计模式和测试驱动开发的原则、思想、技巧以及对它们的价值的重视融入到现有的编码和分析方法中，我们将帮助软件开发从一门单纯的“手艺”转变为一个真正的专业。我们已经具备了实现这一目标的知识，所需的只是一种新的态度。

本系列丛书将帮助大家培养这种态度。我们的目标是帮助管理层和开发人员共同致力于“整体优化”。

(1) **优化整个组织**：整合企业、团队和个人，让他们最好地协作。

(2) **优化整个产品**：不只是产品的开发，而且还要优化产品的维护和集成。

(3) **全程优化**：不只是现在，还有未来。我们需要从付出的努力中获得可持续的投资回报。

浮现式设计：专业软件开发的演进本质

本书从技术角度讨论了产品开发。书中描述了把软件变为一个专业意味着什么。同时，还讨论了以演进的方式来构建软件的必要性。但是，这种设计的演进绝对不是随机发生的，而是在明

确定义且经过事实检验的质量控制措施的指导下发生的。这些质量控制措施也是我们在制定决策时必须要注意的。

虽然一些硬性规则不一定适用于所有地方，但原则是普遍适用的。10年前，人们会以“我们没有一组完善的规则”为借口，但现在这已经不再成立。

早在1984年，我就开始探索高质量的软件意味着什么。那一年发生的一件事以及它所引发的两个问题让我记忆犹新。事情是发现了一个bug，之后我问自己：“我到底做了什么事？我怎么会在代码中制造了一个bug？”我的第一反应是意识到了这样一个事实——我总是在说“发现”了bug，就好像这些bug不是我放在那里的。当早上你走到自家的车位时，肯定不会说：“看，我发现了什么！我的汽车！”（是的，是的，实际上我经常在找到车钥匙时这样说，但这是另外一回事了。）换言之，我们在说起bug时，就好像它们是突然冒出来的，而不是我们在系统中制造的一样。我可以肯定说，小精灵不会在午夜来光顾你的程序，然后把bug插入到代码中。我所认识到的事实是“为什么一直过了14年我才意识到这个问题？”（这就是我记得是1984年的原因。）

这并不是说我从来没有思考过如何才能成为一个更好的程序员，而是说我比以往需要更多的反省和回顾——思考我都做了些什么，如何能够做得更好。我需要用“旁观者的眼光”来研究编程系统。怎样做才能改进它？

很多人已经开始这方面的研究了，并有了很多成果，包括面向对象的语言、正确运用设计模式的阐释[①]以及诸如测试驱动开发这样的敏捷编程方法。显然，对于软件开发人员应该注意的基本方面，我们已经有了足够多的认识。而且，仅仅注重自己的喜好显然是远远不够的。我们必须能够证明我们已经发现了更好的方法，否则就必须遵循业内已经证明的有效方法。

这并不是说实现上述标准的事情，而是说开发人员都要承担起“站在别人肩膀上”的责任。我们必须认识到不能每次总是重复已有定论的事情，而且我们不理解的事物并不意味着它没有价值。我们必须尽一切可能寻找最佳实践，并在必要的时候对其进行调整。

旧时代的终结，新时代的开始

我相信软件行业正处于危急关头。行业正在持续扩张，并且日益成为日常生活的重要部分。但软件开发组织正在面临着可怕的问题。不断退化的代码越来越麻烦，本来就已经超负荷的开发人员的负担看起来也在无休止地增加。虽然敏捷方法使得很多团队有了重大改进，但这还远远不够。通过建立一个真正的软件专业，结合精益原则的指导并采用敏捷实践，我相信我们将给出一份应对方案。

希望你发现本系列丛书是一套真正有价值的指南。为了帮助你学习这套丛书，我们建立了一个资源网站（http://www.netobjectives.com/resources），Scott在本书中多处引用了这个站点。这个站点还包含很多书中没有的信息。在这个站点上，你还会发现我们的精益-敏捷方法的其他资源，包括精益、敏捷、Scrum框架、设计模式，等等。你可以充分利用这个站点所提供的信息。

① 参见Shalloway和Trott合著的书《设计模式解析》（第2版，人民邮电出版社2010年出版）。

前　言

设计和创建软件很难。

我喜欢这一点。我喜欢挑战。我喜欢解决难题。这大概就是最初吸引我投身于计算机和编程事业的原因吧。

我想说的是它的难度有点儿太大了。我并不指望它变得非常简单，而只是希望稍微简单那么一丁点儿，可预测性再提高一点儿，混乱再减少一点儿，这就足够了。

我希望在每个项目开始的时候，能够告诉客户软件在开发完成后能够为他做什么，而且我希望我告诉他这件事的时候能够满怀信心。我还希望能够预测项目需要花费多少时间，它大概需要多少投资。我希望我的这些预测是正确的——至少大部分时间是正确的。

我希望我知道自己在做什么——真正意义上的知道。

任何开发过复杂软件的人肯定都有过这种经历：当一个为期12个月的项目到了第9个月的时候，一切都很好，项目处于正轨。但到了第10个月，我们发现项目落后了4个月。怎么可能发生这种情况呢？显然，项目在第9个月时情况就不好了，只是我们认为它很好罢了。为什么我们会不知道项目出了问题呢？

或者，也许我们有一套工作机制——一套看起来还算不错的机制，这时最终用户提出想要增加一些新功能或能力。这是一项合理的要求。情况总是会变化的，这一点我们都知道。这世界变化快。

但当我们尝试做出客户所要求的改动时，事情开始以无法预料的方式偏离了轨道。这使我们犹豫不决，因为我们知道这种情况会发生。这也使我们不愿做出改动，甚至对适应这样的改变怀有敌意。一个人从事开发工作的时间越长，这种阻力就越大。

这并不是我们的错。

在人类技术的长河中，软件开发存在的时间并不算长。其他类似的复杂技术（例如医学、法律、建筑，等等）都已经存在数百年，甚至几千年。在这么长的时间中，人们已经总结出了一整套标准、实践和常识，并一代代传承下来。这帮助一代代的医生、律师和建筑师提高了成功的机会，而且这些技术都已经形成了一种有机的体系，我们将其称作“专业”。

专业有它们自己的生命和存在。例如，木匠专业已在存在几千年了，尽管没有木匠可以活这么长时间。一个专业为从事它的人提供了一个安全网。

本书的目的就是考查一下作为软件开发人员（或者程序员，如果你喜欢这么叫的话），我们如何才能从我们所做的工作、从彼此之间以及从这个行业本身获得这种价值。我打算后退一步，

看看我们的工作性质，并总结出一套最佳实践、常识和专门的工作模式，把我们所做的工作提升到一个真正的专业水平，或者是实现类似的结果，从而获得一个专业能够提供的全部益处。

然而，我的目的并不是单纯从理论上讨论，尽管这很有趣。相反，我想谈一些实在的事情，谈一些软件开发中极难、限制性极大的方面，并提出从事这项工作的更好方式。我想专注于一些真正有价值的东西。

我与你之间的约定如下：我所研究、建议、呈现或演示的每件事情的核心目的都是为了改善软件开发者这个群体。无论一件事情多么有趣，多么吸引人，如果它不能服务于这个目的，那么我绝不会讨论它。

下面开始讨论的一个主题就是：软件开发从本质上讲是一个演进的过程。我们并不是在分析、设计和构建，而是在创建一种能够工作、具有高品质而且有价值的东西，然后经过多个阶段的演变，最终目标是让它成为我们这个世界需要的产品。为了证明这一点，需要一个漫长的过程，我需要一系列支持这一观点的概念和技术。

以下就是我们将要开始讨论的话题。

质量

我们怎么知道软件是好是坏？能工作就是好的吗？我们都知道有很多能工作但却不好的软件。当我们面对两种或三种不同的工作方式时，如何决定哪种工作方式是最好的？最好意味着什么？根据本书中所讲的通用原则，最好的应该是增加开发人员的价值，并提高成功率，最后为客户创造价值。我们要在本书中关注的质量是即刻能为我们提供指导方向的质量，它可以帮助我们更可靠地制定更好的决策：耦合、内聚、消除冗余、可测试性，以及所有这一切的根基——封装。在这些讨论当中，我们还会指出一些负面信号（病状），帮助我们识别一种或多种未达到质量标准的情况。

原则

是什么基本理论定义了好的软件？换言之，采取哪些观点（当我们知道了这些观点之后）就可以更好地帮助我们提高软件的质量？原则指出了“这比那好”或“这比那更重要”。软件需要具有变化的能力，才能满足不断变化的世界的需要，从这个角度来说，原则承诺了更高的质量。

实践

实践就是你日常编程活动的一部分，它们可以为你带来极大的帮助。我最感兴趣的实践是那些能够在多方面提供帮助而不会成为负担的实践。这是典型的花小钱获大利。此外，由于实践在被共享和推广到一个团队（或组织，甚至是整个行业）的所有开发人员时将发挥出真正的价值，因此实践应该是一些易于教其他人去做的事情。

纪律

类似于实践，纪律也是应该去做的事情，但它们的规模更大，需要更长时间去学习，而且并不是没有成本的。然而，它们提供了极为根本且具有深远意义的价值，因此所需的时间和工作量也是值得的。单元测试和重构就是纪律的例子，测试驱动开发的概念亦是纪律。本书都会讨论到它们。

模式

模式表示我们之前做过并成功的事。但我并不是说模式是一个菜谱或一组模板，软件比这要复杂得多。我所说的模式是一组互相关联的“智慧点”，它们反映了开发人员这个群体所掌握的有关某些特定情况的知识，这些情况都是我们一次又一次遇到的。我们作为一个行业整体经历过那些事，即使有些人作为个体尚未经历。模式是一种集体共享经验财富的方式，大家互相支持来获得更大的成功。模式与它的上下文有关，这一点与原则不同。原则是普遍适用的，而模式在不同情形下有不同的应用。我们从每种模式的要素出发来讨论这些概念，并且看看为什么这种模式观点使得它们比单纯的“密封的设计”更有用。我们有很多模式，也有很多讲模式的书，因此本书提供了一个附录，概要介绍了本书中用到的模式，以便说明它们在浮现式设计中的作用。

过程

一般而言，软件开发是怎么工作的？我们如何发现需要构建什么？如何知道什么时候已经完成？如何知道是否处于正确的轨道上？更重要的是，如何知道何时偏离了轨道？当偏离轨道时，我们需要做什么？前面我已经指出了软件的创建是一个演进的过程，但这显然只是思想的一粒种子，有待萌芽发展。

当然，研究这些问题的人并不是只有我一个。在本书中，我借鉴了其他人的研究成果，这些人包括Alan Shalloway、Martin Fowler、Ward Cunningham、Kent Beck、Ron Jeffries和Robert Martin，等等。我从他们这类人那里学到了很多东西，我在参考书目部分列举了他们的著作，请大家参考他们为这个专业贡献的资源。

像上面提到过的一些同行一样，我曾经被指责“以开发人员为中心”。就我自己来讲，事实的确如此。我关注开发人员，不仅是因为我自己就是开发者，而且因为我相信如果我们想得到更好的软件，就需要更好地支持开发。我认为这意味着要关注开发人员（举例来说，高质量的医疗服务需要培养出好的医生才行）。这并不意味着他们开发的那些不被使用的软件我也认为有价值。相反，我认为不被使用的软件是毫无价值的。因此，虽然我强调那些帮助开发人员获得成功的因素，但我的目的是得到更好的软件和正确的软件，这肯定会使所有相关的人受益。

当然，还有其他的工作需要完成。我不会在仅仅为我的同行们（开发人员）引入了有价值的思想和实践后就假装已经解决了问题，但这确实是解决问题过程中的一部分。

我坚信软件开发就处在临界线上——它马上就要成为一个真正意义上的专业了，走完最后这“一里路”，填补缺失的东西，是我们这个时代要完成的最重要的工作之一。若干年后再来回顾现在这个时候，我们会认识到这是一个软件开发走向成熟从而可靠地满足现代世界需求的时代。能够成为这个时代的一员，我非常高兴。

那么，就让我们开始吧。

致　谢

我以前曾经在加利福尼亚州圣迭戈的KFMB TV/AM/FM做过几年程序员，在那段时间，我从该地区的同行那里学到了很多东西。特别是我的好友Sassan (Sean) Azhadi，他现在是圣迭戈县信用社的高级副总裁，多年以来Sean一直是我的咨询师、调研伙伴和亲密的朋友。他也是我至今仍保留着一头浓密头发的主要原因——我已经记不清有多少次，他的慷慨和离奇的幽默感使我放弃了理光头的念头。

在那些年，我还非常幸运地能够与“Saturday-night游戏小组”的好朋友们保持来往，我的大部分思维技巧都要归功于与以下好友的交流：Francis (Brett) Drake博士、Frank (Todd) Tamburine博士、Doug Hansen、Brenner Roque、Chuck Comfort和我的弟弟Christopher。

像很多长期从事这个行业的开发人员一样，我经历了一段“个人疲劳”期，这主要是由于我试图维护一个在设计时根本没考虑到可维护性的软件（当然，除了我自己外，我不能责怪任何人）。在我离开广播行业之后，我经历了互联网泡沫期，这段时期内我未做任何改善状况的事情。

我的“回归之路”主要归功于两个人：Bruce Eckel和Alan Shalloway。Bruce的书《Java编程思想》[①]帮助我最终理解了OO（并接受了其价值），而Alan则帮助我认清了模式的真正含义，以及模式如何从根本上改变了软件的可维护性。如果没有Bruce的那本书的指导，我永远不可能走近Alan，而且，如果没有Alan的耐心指导和合作，我想我现在可能正忙着其他什么事情来维持生计，根本不会发现软件开发的乐趣了。

而且，正是因为有了Alan，才有了Net Objectives，因此我才有机会与一些杰出的人士共事并向他们学习，这些人包括Rob Myers、Rod Claar、Dan Rawsthorne、Jeff McKenna、Ed Lance、Amir Kolsky、Jim Trott和David Bernstein。

我还从其他许多作者的书中受益匪浅，我尽量在书中提到他们以示感谢（以脚注的形式）。此外，当一个人快要写完一本书时，难免有些当局者迷，我在它上面花了太长的时间，离它太近了，以至于无法真正看清其全貌。因此，别人的仔细审阅可以说是无价的。我衷心感谢Donna Davis、Jeremy Miller和Matt Heusser对本书手稿的细心审阅以及他们大量有帮助的改进建议。

这是我第一次出书，我真的不知道能够期盼什么样的结果。Addison-Wesley的朋友们为我提供了莫大的帮助，他们非常耐心而且专业。还要特别感谢Christopher Zahn博士、Raina Chrobak和Christopher Guzikowski，感谢他们为我所做的一切。

---

① 本书第4版由机械工业出版社2007年出版。——编者注

最后，我最大的幸运是拥有一位像我一样对技术感兴趣的妻子。Andrea是一名程序员，同时也是一名优秀的艺术家。因此，我不仅能借助她的天才来为这本书配上插图，而且我这些年来所写的每一篇文章、参编的每一章节、制作的每个PPT文件都得益于她的细心审阅，她为我提供了无数的改进建议。如果没有她的协作和支持，我的每项工作成果都将大打折扣。

目　录

第1章 软件开发这个职业

本章我们来考查一系列有趣的问题。有时，一个问题的真正价值并不在于找到答案，而在于通过考查这个问题引出其他或许更有价值的问题。另外，有时候发现一个无人问津的问题，也可能会帮助我们看到一些未被发现的机会，从而引出更深远、更有价值的发现。

我已经“搞软件”很长时间了，我觉得我们这个行业已经到了“回头看看”的时候了，此时回顾一下我们工作的基本性质可能是一件非常有用的事情。

1.1 人类制作软件已经有多久的历史了

像很多问题一样，这个问题的答案是“要看情况”。

制作软件的概念都包括什么？是否包括最早期由绕线PC板和交换管构成的编程？是否包括提花织机[①]？

也许不包括。但使用穿孔卡片和大型主机进行数据处理的那段时期是否应该包括进来呢？那时人们使用穿孔卡片或磁带输入，使用打印机输出，没有交互，没有VDT，为了看看程序是否正确运行，需要通宵等待。

为了便于本书的讨论，我认为软件制作应该从20世纪70年代中后期开始算起，那时小型桌面系统刚刚问世，我们开始开发供人们直接使用的交互式软件。

这并不是说我认为数据处理不够重要、不够有趣，或不够复杂，只是过去他们[②]的数据处理方法与我们现在的数据处理方法之间的差别实在是太大了，以至于我看不到其间存在智慧的传承。这就像是骑马与开车一样：二者都是复杂的活动，都需要知识和技巧，但学会了一样实际上并不能帮助你学会另一样。

但是先等一下！面向对象（OO）语言和技术的出现又该如何看待呢？它们与过去一度盛行的过程语言（例如C和Pascal）可是正好相反啊。OO出现之前的过程语言时代也应该被看作是一种完全不同的活动吗？

---

① Jacquard loom，电脑为其提供指令信息，由机械来完成一系列动作的纺织机械。提花机与操作专业纺织软件有关系，但主要是纺织专业知识方面的程序，与软件的编程没有联系。——编者注

② 我猜想这里用“我们”也许更恰当一些。当我刚开始工作时，我需要把穿孔卡片插到键控穿孔机上，必须用锉刀在铅棒上刻出凹槽来改变定位点。我真的一点儿也不怀念那个年代。