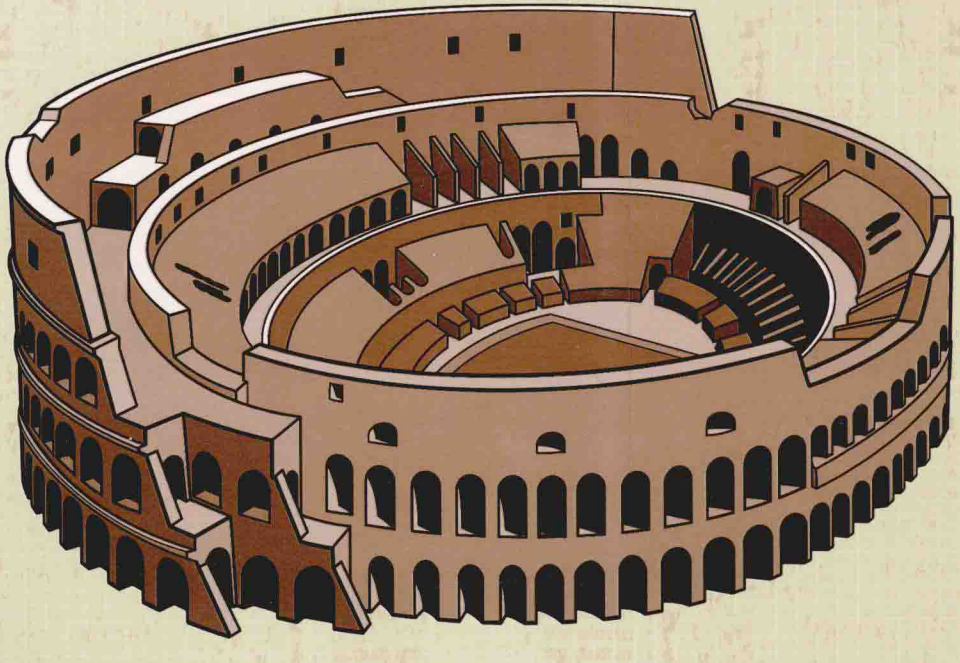
TURING 图灵程序设计丛书

程序员必读之

软件架构

Software Architecture for Developers

[英] Simon Brown 著
邓钢 译



人民邮电出版社
POSTS & TELECOM PRESS

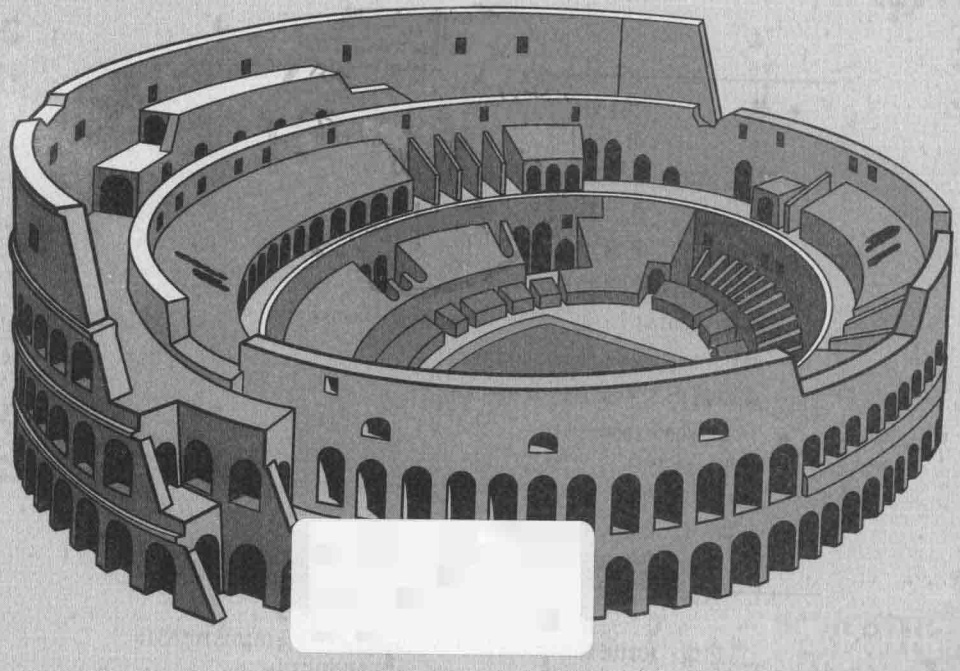
TURING 图灵程序设计丛书

程序员必读之

软件架构

Software Architecture for Developers

[英] Simon Brown 著
邓钢 译



人民邮电出版社
北 京

图书在版编目（CIP）数据

程序员必读之软件架构 / (英) 布朗 (Brown, S.) 著；
邓钢译. -- 北京 : 人民邮电出版社, 2015.1
（图灵程序设计丛书）
ISBN 978-7-115-37107-2

Ⅰ. ①程… Ⅱ. ①布… ②邓… Ⅲ. ①软件设计
Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2014)第219052号

内 容 提 要

通常，人们对软件架构师持两种错误的看法。有人认为软件架构师是一种高高在上的职位；有人认为软件架构师完全不懂开发，只是会画条条框框的指挥家。本书将打破这些传统的认知，模糊软件开发和架构在流程中的界限，进而为软件架构正名。本书是一本强调实践、注重实效、轻量级、面向开发者的软件架构指南。

如果你是一名想成为软件架构师的程序员，那么本书就是为你准备的。

◆ 著 [英] Simon Brown
译 邓 钢
责任编辑 李松峰
执行编辑 李 静 仇祝平
责任印制 杨林杰
◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷
◆ 开本：800×1000 1/16
印张：14.25
字数：343千字 2015年1月第1版
印数：1-3500册 2015年1月北京第1次印刷
著作权合同登记号 图字：01-2014-5610号

定价：49.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

版权声明

献给克斯蒂、马修和奥利弗。

推荐序一：架构师真正要学会的事情

1. 要学会去看，然后忘掉

有一本书叫《观止》，写的是微软研发 Windows NT 的一段故事。“观止”在这里的意思是说“看到这些，就无需再看了”，因为世上之物亦无过于此。20 多年过去，如今微软在操作系统上面临着的种种挑战与困境，其实与《观止》所叙的研发方法、理念与目标有着与生俱来的血缘关系。

另一个与“看”相关的词汇是“所见即可得”（WYSIWYG）。这个词以及与此相关的 WIMP（Windows, Icon, Menu and Pointer）曾经主导了整个人机交互的设计理念。也是在 20 多年前，Borland 为 Windows 桌面系统成功地设计了跨语言的 VCL，由此“所见即所得”成为 Borland 对“如何更便捷地构建 UI”的基本假想，以至于这家伟大的公司在互联网时代来临时决定“用 VCL 描述界面的方式来解决‘网站设计’的问题（RadPHP）”。

然而，互联网上的网页是没有 WIMP 的；移动设备上的操作系统也不再采用与 Windows NT 类似的方式开发。

Borland 在几年之前将整个开发工具产品线都卖掉了。当时盛大的一个 Delphi 圈子发起了一次“缅怀活动”，组织者说：“爱民，你应该会为那个时代写点什么吧？”

我在那个缅怀网页上写下了五个字：所见即所碍。

2. 要学会去听，然后忘掉

我通常说架构是一种能力，架构角色则是要求你在具体事务中行使某些行为，而架构师则是用来标识这些能力与行为的一个职务。

当一些人将个人成长定义为“职业发展”时，就表现为“怎样成为架构师”这样的问题。对

此有三种解决方案，第一种是印一张写着这样头衔的名片，而“是与不是”架构师并不重要；第二种是直接否定这个职务的意义，比如声称敏捷天生就是反架构的，于是“架构师”变成了要打倒的对象，所以成不成为这个将被打倒的对象也就不重要了；第三种则干脆声称“人人都是架构师”，既然人人都是了，那么“如何成为”也自然就不重要了。

我们大多数人都具有架构的能力，并且也或多或少地行使某些架构角色的行为，唯一缺乏的只是一个叫做“架构师”的头衔而已。问题出在我们总是期望别人通过这样的头衔来认可自己。于是我们为自己贴上这样或那样的标签，然后跟别人持有的同种标签去比对，期求出现一致或找出某种差别。于是我们听到种种声音：某某某真的是/不是、像/不像架构师；如果是架构师，那么就要这样那样，以及怎样怎样；其实这个架构、这样的架构，或某种架构应该怎么做；以及架构是什么，架构师是什么，等等。回顾“三种解决方案”，仍是困在这样的认可求同之中，与之在做着种种斗争罢了。

其实不单是你的所见阻碍了你自己，你还被别人的所见阻碍着。

3. 要学会去做，然后忘掉

朋友跟我聊他家的两岁小孩：我刚把桌子收拾好，一转眼杯子碗筷什么的都全摔地上了。我问：“怎么了？”他说：“小孩子什么也不懂啊，她看到桌布觉得喜欢，就一把抓过去……”

小孩子没能看到桌子上还有杯子，但正因为他们的视线里没有杯子，他们的行动才简单直接，才直达需求，才迅速。而我们的眼睛里有杯子、桌子、桌布等一切，我们经年累月地维护着其中的次序与关系直到这些东西混成一体，然后我们便日日坐守在它们的面前，而又无觉他们的存在。

正是我们自己不知不觉地设定了这些事物之间的界线，并把这些界限、层次与逻辑井然的东西称为“系统”。当我们从那些无序的事物中识别出了这样的“系统”并用一些概念、名词去定义了它们之后，我们对此的一切知识也就固化了。当这种秩序被建立起来之后，我们也就得到了对有序和无序（没有你所设定的“这种秩序”）价值的识别与肯否；当我们设定了种种价值、观念、观察与系统的模型概念之后，也就完成了这个系统的架构。

但这一过程，包括完成这一架构——它可以命名为“世界观”——的方法以及结果，在本质上不过是让你从一个格子跳到了另一个格子而已。我们处在种种界限之中，再也无法回到两岁小孩的、一切无碍的视角：在那个视角下，根本就没有所谓的界线。你之所以时时在寻求跨界，其实是源自你假设了“存在界线”，这就如同全栈的含义其实是“没有栈”，而当有人信心满满地要“成为全栈工程师”时，他的眼里便又有个“这个栈”的存在。

所谓跨界不是指你能力与方法上的变化，你的作为取决于你的格局，你的格局取决于你的所见。

4. 要学会超越

架构师需要超越自己与别人的所见，因为你观察与架构的对象称为“系统”，你看到系统多少的真相，决定了你用怎样的影像去表现它，并进而推进与实现这种影像，亦即是架构。我们既已知道的、理解的、明白的，形成了我们的知识与行为的一切，却也正是阻碍着我们前进的东西。这些障碍正是你以为你最珍视的、最不可放弃的、最鲜血淅沥体验过的那些经验与成就。在这些所得与所碍中挣扎与决策，就是架构师的全部职责。因此作为架构师，你需要能够超越自己对系统的既有认识，看到你在光明中——显而易见之处——所未见的，这是你驱动系统架构进化的主要动力。

所以架构中最难超越的并不是某个大师或前辈，而是你以及你为自己所作的设定。当你设定了“架构师”这个目标，便设定了这个目标所表达的某种影像（角色），你最终可能变得跟这个影像完全一致——成为所谓的“真正的架构师”，但你仍不过是困囿于对这个“角色”的一个假设/设定而已。唯一破局的方法是：超越别人对某个角色的定义，将自己做成这个角色。

至此，你是否还在这个角色之中，就是你的觉悟了。

周爱民
现任豌豆荚架构师
前盛大网络平台架构师、支付宝业务架构师

推荐序二

说起架构，想必很多人会认为它离自己太远，我做的事情还远到不了架构这么高的层次。那么什么是架构呢？正如本书作者所做的调查一样，不同的人会给出不同的见解。

我们不妨从平时的项目中来观察一下，技术选型是怎么出来的？团队的分工协作是如何进行的？项目质量和进度是怎么得到保障的？

是的，你会发现，在任何一个项目中，总有些人会在这些事情上付出努力。从他们身上可以看到哪些不一样的特质？他们看起来都很积极，好像整个项目就是他们在负责；他们让事情得到解决，最终让项目得以交付。

可以认为，项目中出现的类似行为都是在对架构的思考，思考架构会是从被动服务到主动服务的 Owner 意识养成过程，会让我们 Get things done！而最终完成的好坏及是否有方法论支撑则是另外讨论的范畴，这也正是本书要为大家呈现的内容！

如果你刚接触项目不久，建议由浅入深，从分清楚什么是库什么是框架开始，带着问题在本书中寻找答案！如果你已经验丰富，同样可以认真思考书中每一部分后面的问题进行自我对照，看看与作者的建议是否有共鸣之处！

从现在开始，认真且有效地去规划完成自己负责的事情！

杜欢

淘宝网高级技术专家

2012 年加入淘宝，曾就职于雅虎台湾及 CISCO

译者序 2.0

在本书行将出版之时读到周爱民老师写的推荐序，感触良多。因而书还未面世，译者序就写了第二稿，倒也跟 IT 行业的风格挺契合：很多软件首次发布时的版本号都是 2.0，甚至更高。

初识软件架构

我是从一个小互联网公司走出来的野生程序员。小公司里没有很细的分工，程序员必须像万金油，什么都会一点。数据怎么分表，后端接口怎么分，URL 结构怎么定，前后端怎么接，这些都得搞定。事情多了，必须想清楚。

我在盛大创新院做的最后一个项目是一个 iOS 垂直社交应用。两个同事合作开发 iOS 客户端，而我在这个项目里的工作是开发一个 REST 架构的数据服务。需求很简单，就是根据客户端的应用场景编写一整套 API。当第一个里程碑的所有工作完成之后，我发现需求开发只占用了一小部分时间，而设计关系型数据库的结构，设计认证、授权和报告，设计应用签名和令牌，设计 REST 风格的 URL 结构，开发 API 调试工具，编写 API 文档，这些事情却耗费了大量的时间。我就想，花了这么多时间做这些事情，并没有增加任何功能，又感觉不能不做，这到底是为什么？对这个问题的思考和学习，应该算是我对软件架构的入门。

怎么会翻译这本书

两年前我进入 IBM，参与的项目是一个适用于大型数据中心的存储资源管理工具。这个工具的规模和复杂程度远远超出大多数面向普通用户的互联网应用，自然对架构的要求更为严苛。而 IBM 作为一家传统软件企业，深厚的技术积累也令我大开眼界，给了我很多学习和思考软件架构的机会和资源。

我们项目的架构师会贡献代码，会参加代码评审/回顾；我们有预先架构设计，也有架构演

化；我们执行 SCRUM 方法；任何人对设计有意见，都可以给架构师发邮件，只要有理有据，就能说服他更改设计。在此之前我从未见过这样奇特的组合。很快适应了以后，我又想，这么好的方式，居然只有我们在用？直到今年三月，在微博上看到图灵的李松峰老师为 *Software Architecture for Developers* 一书征召译者。读过样章后才发现，我们就是作者理想中的团队啊！既然如此，何不尝试翻译这本书？

架构离我们并不遥远

写给程序员的软件架构，这是一个很有趣的出发点。长久以来，架构师在程序员群体中声名狼藉，软件架构被很多人认为是一项脱离现实、高高在上的工作。其实对程序员来说，架构近在眼前！下至接口设计，上至技术选型，不论你是否意识到，每个程序员或多或少都接触和参与过一些架构工作。架构师也自然而然成为相当一部分程序员的职业发展方向：你看，我们努力想要成为自己咒骂的人。

本书的作者是一位经验丰富的架构师。他从最简单的基本概念入手，对软件架构进行了层层深入的细致讲解，结合自己的实践经验，总结出很多实用的准则和方法，并且附上一个完整的开源项目来对这些内容加以佐证，帮助读者学习和理解。翻译这本书，在我看来更是对软件架构的一次系统学习，不仅丰富了我对软件架构的理解，更改变了我对架构师这个角色的一些固有印象。这本书令我获益匪浅，希望更多有志成为架构师的程序员朋友也能从中有所收获。

周爱民老师的序

十一假期结束后打开邮箱，收到图灵的李静老师的邮件，得知周爱民老师会为这本书撰写推荐序。惊喜，因为爱民老师是包括我在内的很多人敬仰的资深前辈。惶恐，因为我在软件架构方面还是个菜鸟，写作能力更与爱民老师相去甚远。

要来爱民老师的文章一读，“然后忘掉”，写得真好！我从未受过任何计算机科学或软件工程的专业训练，大学所学的化学专业也跟计算机毫无关联。因此，去看去听去做，并且牢牢记住，使我得以在这个行业一步步走到今天。自从对软件架构产生兴趣，也是如此逐渐学到很多概念、方法。虽然清楚架构必有权衡，不能十全十美，然而了解的知识越多，就更想面面俱到，反而放不开。殊不知，牢牢记住，也给自己画地为牢，陷入爱民老师所说的困局。

如果说这本书帮我画了一个更大的圈，那么爱民老师的文字则告诫我要跳出这个圈。我面前的架构之路还很长，不知何时能走出圈外，走到爱民老师今日所处之地。

谢谢你们

这本书的翻译能够完成，我最想感谢的人是成都七中的高中语文老师王正可。王老师是我近20年学生生涯中最重要的一位老师：她是第一个让我对语文产生兴趣的人。王老师教给我文字的艺术，帮助我找到阅读和写作的乐趣，对我而言这是一笔巨大的财富。如果不曾有幸成为她的学生，我想我不会养成写作的习惯，更不可能有翻译图书的想法。

为了让我能够安心地翻译，家里的领导承担了洗衣、做饭、扫地（以及数钱）等繁琐的家务。对一些难以理解的字句，她也和我一起讨论。作为一个非球迷，她还陪我熬夜观看了好多场世界杯比赛，大大减轻了我因为拖延翻译而产生的负罪感。图灵公司的李松峰和李静两位老师，对我翻译这本书给予了极大的肯定和支持，并且纵容了我逾期未完工的行为。这本书也有他们的一份付出。

2014年10月于上海

序

信息技术行业不是大步前行，就是剧烈动荡。一方面，我们奋力前行，重新发明软件构建方式，时时处处精益求精。而另一方面，我们不断遗忘过去的好处，软件开发团队常意想不到地把事情搞砸。

软件架构在一个成功的软件交付中扮演关键角色，然而令人沮丧的是，很多团队都忽视了这一点。即使在最敏捷的团队中，软件构架这一角色也都是必需的，不管是由一个人还是整个团队共同扮演，但要寻求到预先和演化两种构架理念的平衡，往往还只是人们美好的意愿而并没有变为现实。

软件架构的坏名声

当我介绍自己是软件架构师时，对方通常会有两种反应。要么觉得这非常酷，想了解更多；要么就是露出不屑的神情，意思是说“我想跟实际开发软件的人聊，而不是跟只会画框框线线的指挥家聊”。软件架构的角色在 IT 行业中名声很差，出现这种想法自然不难理解。

“软件架构”给人的印象通常是架构师闭门造车，提前做好大型预先设计，然后好像接力赛跑时传递交接棒一样，把庞大的 UML（Unified Modeling Language，统一建模语言）模型或 200 页 Word 文档丢给毫不知情的开发团队。当然，这是假设架构师实际参与了软件设计。似乎很多人都认为，只要做一个 PPT，而且幻灯片中有一页出现了“企业服务总线”框线图，就算是做完了软件设计。哦，千万别忘了，这个 PPT 里毫无疑问也少不了对 ROI（Return on Investment，投资回报）和 TCO（Total Cost of Ownership，总体拥有成本）的陈述。

很多组织对软件开发普遍都有一个有意思的看法。比如，他们看到了离岸外包可以节省成本，因而把软件开发流程中的编码工作也看作一种可以买卖的商品。其结果往往是本地开发者被推向所谓“高价值”的软件架构职位，而编码则交由其他人完成。多数情况下这只会让软件架构和开发更加脱节，还常常让人像赶鸭子上架一样不得不去承担架构工作。这些组织也常倾向于把架构

师看作一种职位级别而非工作角色。

敏捷愿景

“敏捷”已经出现了差不多十年，但它仍是“外来的时髦小子”。很多软件团队都有“实现敏捷”的愿景。毫无疑问，敏捷有很多好处，人们都想让你相信它是灵丹妙药，但事实并非如此。IT行业的每件事，都伴随着铺天盖地的宣传和天花乱坠的炒作。如今，开始一个新的软件项目，总能听到自组织的团队、自动化验收测试、持续交付、回顾、看板、浮现式设计，还有一大堆你可能都没听过的新名词。这很奇葩，但团队往往急于赶时髦，就将原来的东西不分好坏一起丢掉。“非功能需求”听起来虽然不酷，但这并不是你能忽视它们的理由。

这堆老古董软件架构的东西都是什么？很多软件团队似乎认为他们不需要软件架构师，张口闭口都是“自组织团队”、“YAGNI”（You Aren't Going to Need It，你不会需要它）、“演化架构”和“最后责任时刻”这些词。如果他们确实需要架构师，也许会去找个“敏捷架构师”。我不完全确定这些词都是什么意思，但我猜它有点像用便利贴替代UML，或用TDD（Test-Driven Development，测试驱动开发）替代画图。也就是说，假设他们已经不是只使用高层次系统隐喻的概念，而且也不把“浮现式设计”作为盲目乐观的借口。

那么你觉得自己是架构师吗

看起来这个行业里有很多人自称是软件架构师，而他们实际上完全在做别的事。我能够原谅那些在大企业里实践软件架构，却误以为自己是“企业架构师”的人。总之我们这行的术语就是经常把人搞糊涂。

但那些夸大自己在软件团队里作用的人又如何呢？这些不负责任的架构师通常担任技术领导，却连基本能力都不够格。我见过一些面向公众的网站在进入用户验收测试环境时，还有一堆安全问题，没有基本性能测试，常用功能也有问题，死链，并且完全没有文档。这只是我能看到的软件外在的问题，天知道代码会是什么样子！如果你承担了软件架构的角色，最后却交付这样的东西，你做得就不对。这不是什么软件架构，这也只能算是盲目乐观。

失意的架构师

必须承认，不是所有软件团队都像这样，但我前面讲的也不是空穴来风。糟糕的是很多组织确实就是这样干的，因此软件架构有这样的名声并不奇怪。

想在这个行业里有所作为，就需要克制对新鲜玩意的迷恋，开始问一些问题。敏捷需要架构吗，或者架构真的需要敏捷吗？相比近些年学到的东西，我们是不是忘掉了更多好的软件设计方法？对于我们现在构建的软件系统，只有盲目乐观就够了吗？如果我们不是从培养未来软件架构师的角度考虑，这些问题还有意义吗？我们要如何从失意走向平和？

关于本书

这是一本强调实践、注重实效、轻量级、面向开发者的软件架构指南。你将从中学到：

- 软件架构的本质；
- 为什么软件架构角色应当包含编码、指导与合作；
- 开始编码前真正需要思考的事情；
- 如何用简单的草图让你的软件架构可视化；
- 为软件生成文档的轻量方法；
- 为什么敏捷和架构并不冲突；
- “恰如其分”的预先设计是什么意思；
- 如何通过风险风暴来识别风险。

这部短文集推倒了传统的象牙塔，模糊了软件开发和架构在流程中的界限，将教会你软件架构、技术领导力以及它们与敏捷之间的平衡。

本书写作初衷

跟很多人一样，我的职业生涯从软件开发开始，从前辈那里得到指导，和团队一起工作，交付软件系统。久而久之，我也开始设计软件系统中的一小部分，最后我的职务变成了这样：承担我现在认为是设计软件架构的任务。

我的职业生涯多数是为 IT 咨询机构工作，这意味着我参与过的大多数项目要么是为客户构架软件系统，要么是和客户一起完成构建。IT 咨询机构要发展壮大，就需要更多的人和团队。要组建更多团队，又需要更多的软件架构师。这就是我写这本书的理由。

(1) 软件架构应该容易理解。第一次设计软件架构时，尽管有一些优秀的导师，但我还是搞不清自己该干些什么。的确，有很多软件架构方面的书籍，但它们的写作视角不一样。我发现其

中大多数都偏研究方向，甚至完全是学术派，而我是一个寻求现实建议的软件开发者。我想写一本对我职业生涯的那个阶段有用的书，即面向软件开发者的软件架构书。

(2) 所有软件项目都需要架构。我真心喜欢敏捷方法，但其中很多方法缺乏对软件架构的明确重视，这让我如坐针毡。敏捷方法不是说不应该做任何预先设计，但它们通常也不明确探讨这一点。我发现这会让人们得出错误的结论，我也看到了缺乏预先思考可能造成的后果。我非常清楚大型预先设计也不能解决问题。我感觉适当地做一些预先思考能提供一种愉快的中间状态，而这特别适合与不同经验和背景的团队一起工作的情形。我更喜欢轻量的软件架构方法，这样我就可以尽早让一些结构单元到位，从而提高成功率。

(3) 传播轻量级软件架构实践。这些年我学习和实践了很多对设计软件架构很有帮助的做法。这些实践涉及软件设计流程，并通过发现技术风险来沟通和记录软件架构。我总是认为这些实践都合理，但情况并非如此。过去几年，我向上千人教授这些实践，并见证了他们的变化。写书可以帮助我把这些想法传递给更多人，希望其他人也能从中受益。

软件开发的新方法

这本书不谈创造软件开发的新方法。传统软件开发经常会过度地预先思考，而初次接触敏捷方法的团队往往又缺乏架构思维，本书就是想要在这两者之间找到一个很好的平衡点。最终要告诉大家，预先设计与演化架构是可以共存的。

关于软件架构，每个开发者都应该知道的五件事

为了帮助你大致了解本书的内容，这里有每个开发者都应该知道的五件有关软件架构的事。

1. 软件架构不是大型预先设计

软件架构历来被认为跟大型预先设计和瀑布式项目有关，团队要周全地考虑软件设计的所有细节，然后才开始编码。软件架构就是关于软件系统的高层次结构，以及你如何理解它。它是影响软件系统形态的重要决策，而非理解数据库每个字段应该有多长。

2. 每个软件团队都需要考虑软件架构

不论产品的大小和复杂性，每个软件团队都需要考虑软件架构。为什么？简单地说，尚未发生的坏事往往都会发生！如果软件架构是关于结构和愿景的，不考虑这一点就可能产出结构糟糕、