PEARSON
Addison Wesley

敏捷软件开发
(英文版)

敏捷软件开发系列

Cockburn • Highsmith 丛书编辑

Agile Software Development

12th Annual SOFTWARE Development JOLT PRODUCT EXCELLENCE AWARD

Alistair Cockburn 著
俞涓 注

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

敏捷软件开发系列

敏捷软件开发

（英文版）

Alistair Cockburn 著

俞 涓 注

人民邮电出版社

图书在版编目（CIP）数据

敏捷软件开发/（美）科伯恩（Cockburn,A.）著. —北京：人民邮电出版社，2003.8
（敏捷软件开发系列）
ISBN 7-115-11364-5

I. 敏... II. 科... III. 软件开发—英文 IV. TP311.52

中国版本图书馆 CIP 数据核字（2003）第 052708 号

敏捷软件开发系列

敏捷软件开发（英文版）

◆ 著 Alistair Cockburn
注 俞 涓
责任编辑 俞 彬

◆ 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
读者热线 010-67132705
北京汉魂图文设计有限公司制作
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：20.25
字数：439 千字 2003 年 8 月第 1 版
印数：1-3 000 册 2003 年 8 月北京第 1 次印刷

著作权合同登记 图字：01 - 2002 - 6016 号

ISBN 7-115-11364-5/TP・3511

定价：35.00 元

内容提要

本书是国际知名软件开发专家 Alistair Cockburn 通过采访项目开发组和总结自己 20 多年的开发和管理经验，撰写的一本介绍软件开发新思想——敏捷软件开发方法学的专著。

本书共 6 章，在第 1 章之前的引言部分，作者阐述了人要正确地认识事物和准确交流是非常困难的这一观点。第 1 章作者通过一个假想的诗歌创作的例子，指出软件开发中常见的问题，并试图揭示软件开发的特点。第 2 章探讨了在软件开发过程中占据决定性作用的人的因素。第 3 章论述了团队的交流与合作，说明哪些因素影响交流的效果，有哪些好的交流方式等等。第 4 章详细列出了方法论的要素、设计原则、词汇术语等内容。第 5 章作者从多个角度论证了一套方法应该是动态的、自适应的。第 6 章阐述了作者自己的水晶系列方法论。附录 A 给出了敏捷软件开发宣言，其主要内容是 4 个核心价值和 12 个指导原则。

本书提供了一个新的角度来看待软件开发活动，以及一个新的思路来设计开发方法。书中提供的材料大部分来自作者丰富的实践经验，对软件开发实践有很高的参考价值，本书适合软件开发人员、项目管理人员、软件工程研究人员，以及所有想要了解敏捷开发思想的各界人士参考。

敏捷软件开发系列

Alistair Cockburn　Jim Highsmith，丛书编辑

在敏捷软件开发宣言中明确提出了敏捷软件开发的 4 个核心观点：

- 较之于过程和工具，更注重人及其相互作用的价值。
- 较之于无所不及的各类文档，更注重可运行的软件的价值。
- 较之于合同谈判，更注重与客户合作的价值。
- 较之于按计划行事，更注重响应需求变化的价值。

敏捷软件的开发需要革新和反馈，基于开发团队和客户之间所产生和共享的知识。敏捷软件开发者利用顾客、用户、开发者的力量，找到能够在质量和敏捷中取得平衡的刚好够用的过程。

这本书是敏捷软件开发系列丛书之一，它侧重于介绍敏捷开发者的共享经验。这套丛书还包括侧重介绍个体技术（如用例）的书籍，群体技术（通过协商来决策）的书籍，为不同组织文化遇到的不同问题提供解决方案。结论是敏捷的核心——实践，它可以丰富你的经验、改进你的工作。

Agile software development centers on four values identified in the Agile Alliance's Manifesto:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

有关更多信息，请参见 http://www.aw.com/cseng

作者简介

Alistair Cockburn 是公认的软件项目管理方面的专家。他是 Humans and Technology 公司的资深顾问，负责帮助客户成功地进行面向对象项目。他在软硬件开发方面有 20 多年的项目管理经验，所涉及领域有保险业、零售业、电子商务公司，并曾在大的组织如挪威中心银行和 IBM 任职。他还著有 Writing Effective Use Cases(Addison-Wesley 2001)和 Surviving Object-Oriented Projects(Addison-Wesley 1998)。

注者介绍及标注说明

本书注者俞涓，计算机专业硕士，从事软件开发及管理工作数年，参与的角色包括编程、系统分析、项目管理、测试等。目前在加拿大工作。俞涓是本书中文版的译者。

在正文中加入了下划线的词汇或句子为本书的标注项，其注释内容列在正文的外侧，以求尽量减少对读者阅读原文造成干扰。本书的标注分两类，一种是解释性标注，对注者认为重要和难理解的词汇、句子或一些术语进行了解释说明，基本与英文原文对应。排版时，用两道横线标出。另一种是特别说明，这类标注有些是对该段内容的简要说明或总结，有些是对作者引用的资料的注解，由于是一些笔记性质的附注，所以用了  图标，以区别于第一种解释性标注。

软件开发新思维

—— 介绍 Cockburn 的 *Agile Software Development*

胡　健

小引

Alistair Cockburn 是一位国际知名的软件开发专家。他在 1991 年受 IBM 之邀对一个 OO 项目设计一套方法。作了一番文献查阅之后，感到非常不满意。失望之余，决定转而请教项目开发组什么是他们认为最好的工作方式，结果就这样讨论出可行的方法过程。同时他也注意到了程序员们所用的语汇和文献上的大不一样。这次经历给了 Cockburn 极大的启发，他从此一发不可收拾，积极造访其他项目，考察这些项目是怎样运作的。在与项目组成员的交谈讨论中，他特别注意他们所用的语汇，并加以吸收。

通过这些采访和总结自己 20 几年的开发和管理经验，Cockburn 发展出了一套软件开发的思想，包括什么是软件开发，软件开发人中人的特点，开发团队应如何组织，怎样选择开发过程等。他在 2001 年底出版的这本 *Agile Software Development* 系统地阐述了他的思想，建立了他的一套独特的话语系统。

此书大致可分为 4 个部分：软件开发的性质，作为开发主体的个人，相互合作的团队以及方法过程。

软件开发是什么

讨论软件开发，首先得对软件开发作个定义。目前有几种代表性的观点：

- 数学观，以 CAR Hoare 为代表[1]：

- 工程观，以 Bertrand Meyer 为代表[2]；
- 艺术/工艺观，许多程序员如是说。

作者认为这些说法都对，但又都不全对。当然，编程本身是一种个体的、富灵感的、逻辑性强的活动，但现代的软件开发更是一种群体活动。

为了进一步揭示软件开发的特点，作者（有些出人意外地）把目光投向了游戏。作者对游戏的种类进行了分析，并详细比较了攀岩与软件开发共有的特点，如合作与目标寻求、团队、技能、训练、工具、资源有限、计划、修正、乐趣等等，最后把软件开发定性为：一种创造与交流的合作游戏（a cooperative game of invention and communication）。游戏的首要目标是交付正确工作的系统，次要目标是生成一些"沉淀"（residue）产品为下一次的游戏进行积累。

游戏的参与者有项目出资人、管理人员、业务专家、软件设计开发人员、测试人员与文档写作人员。他们需要互相帮助合作以达到目标。在这场游戏中我们所见的主要是人的创造性思想以及这些思想在人和人、人和计算机之间的交流。

人啊，人

如此说来，人的因素应当是软件开发中最为重要的了。作者在广泛的项目采访中也发现，一个项目的成功与否，人的因素起着决定性的作用。一个成功的团队总是能做出满足需要的系统，这与使用的技术和软件开发过程没什么必然的联系。但软件开发中对人的研究却出奇地少。根据作者调查，第一本这方面的专著是 Weinberg 在 1969 年写作（正是软件工程的观点兴起之时[3]）、1971 年出版的 *The Psychology of Computer Programming*。在沉寂了 15 年之后，DeMarco 和 Lister 写了 *Peopleware: Productive Projects and Teams*。有趣的是，又是 15 年过去了，作者的这本 *Agile Software Development* 可能是这个领域的第三个"里程碑"了。

作者在 1994 年就明确地提出了人是"非线性"的观点。这种非线性表现在，给一个人双倍的输入（如双倍的时间，或双倍的报酬诱惑），你不能期待着他就能做到双倍的输出（双倍的思考质量，或双倍的代码行）。在这本书中，作者更进一步地讨论了人的特点，特别是这些特点在软件开发中的作用。

首先，人是多样化的。有人善抽象思维，有人喜具体细节，有人夸夸其谈，有人沉默寡言，有的领导专行独断，有的上司民主亲善，等等。

对个体而言，人也是不可预料的。作者用了一个有趣的摩登单词 funky（这里有"奇异"，"复杂"，"不可思议"之意）来描述个人。人的 funkiness 表现在

- 自发冲动（spontaneous）：可能会突然心血来潮去干一件事；
- 自相矛盾（contradictory）：如时而口若悬河，时而缄口不语；

- 变化无常：人的个性可能会随时间（每小时、每天、每年）或/和环境（房间、温度、同事）而变化。

.人的哪些弱点会导致一个项目的失败呢？作者归纳了五条：会犯错误，墨守成规，过于追求创新而不愿复用，个人习惯难改，不愿受纪律约束。其中第一条（会犯错误）是最值得注意的。在软件开发中，错误可能出现在任何地方，如：制订进度、需求分析、系统设计、编程、系统安装、测试，甚至打字、校阅文档等。而解决这个问题的最好方法便是采用“往复式”（iterative）或“递增式”（incremental）开发过程[4]。

那么，人有哪些特点会使一个项目成功呢？作者归纳如下：

- 善于寻找（good at looking around）。这是指当一个人在一定的范围内能根据自己的智能和经验迅速地发现需要的或不寻常的东西。
- 学习能力。人们在项目的开发中可以逐步学到新技术、应用领域的知识、开发过程以及如何与同事合作。
- 可塑性。人们在接受了新思想之后能够极大地改变自己的行为。
- 工作自豪（或成就）感。人总是希望在工作中显示能力，取得成绩，获得一种自豪感（pride-in-work）。这种动力也体现在本职工作之外，如利用自己的长处帮助他人或团队（pride-in-contribution），或发现问题主动报告或解决。

作者列出了一些良好的工作方式，它们虽然是对个人而言，但要有效地实施却有赖于个人和管理层的共同努力。偏重个人的方式有：

- 从具体着手。人在开始工作，或学习新东西时，一般喜欢从一个具体的实例入手，通过自己琢磨，或与别人讨论，来理解“这一个”，进而理解“这一类”。所谓“举一反三，触类旁通”是也。
- 使用实物。在项目的某些阶段，如需求分析时，可以用一些实物来帮助人们更好了解系统。这些东西要看得见、摸得着、随手能移、信手便改，一般来说低技术的东西效果更好，如纸张、卡片、纸盒等等。
- 依葫芦画瓢。“拷贝和修改”（copying and altering）是人们在工作中常用的方法。在软件开发中，碰到一个问题时，常用的方法是找到解决类似问题的程序，再加以修改。

而以下这些方式则需要个人与管理层的共同努力：

- 回报激励。编程很大程度上是一种“本能激励”（intrinsic motivated）的活动，因此，要有效地激励程序员，除了物质上的奖励之外，主要是要让他们从工作中获得自豪感和成就感。
- 频繁反馈。频繁的反馈既有利于个人和团队的学习，也有利于系统开发。一个开发过程中应该有频繁的反馈机制，如用户使用反馈，系统测试反馈，开发过程检讨等等。

- 默想与交流。程序员在考虑一个问题的解决方案时，需要一个安静的环境。但软件开发又是一种需频繁交流的活动，需要一个较紧密的环境，因此需要处理好这两者之间的关系。作者建议在设置工作环境时（如座位安排）应加以考虑。
- 工作与个性相匹配。在一个项目中，分配工作时最好能使工作的性质与个人的个性相匹配，这样能使个人的强项得以最大程度的发挥。
- 充分发挥优秀人才的作用。一个优秀的程序员对项目的成功有着非常大的（甚至是决定性的）影响。通常，高手出马，一个顶俩（或更多）。

团队的力量

现代的软件开发，几乎不太可能一个人包打天下。软件开发活动是一种群体活动。要使这个群体有效地运转，作者认为最重要的莫过于成员之间的相互交流与合作了。

1. 信息对流

信息对流（convection currents of information）是作者的一个语汇，它用热气流动来比拟信息的流动。一个人的存在、他的姿势、他的声音、他的脚步，都是在向周围环境放射着信息，为别人所接收。同时，他也接收着别人放射出来的信息，从而形成了信息的对流。

作者还提出了“渗透交流”（osmotic communication）的概念。这是指在一个公用的办公室里，你在专注于自己工作的同时，会不由自主地听到一些背景声音，如别人正在讨论某个问题，并能从中不自觉地提出一些感兴趣的信息。这可以使发现信息和传递信息的成本得以下降。

作者认为一个项目组最好能够都在一个（大）办公室之内，这样能最大限度地提高信息交流的效率。更进一步地，如果项目组有不同专业的人员，如程序员和业务专家，他们的座位应该混合安排，这样可以避免分组安排时容易出现的小圈子效应。

布置办公室时要考虑的一个重要事情是放置“信息发射源”（information radiator）。这里，一个信息发射源一般是指挂（贴）在墙上的大尺寸的图表，这样人们经过时都能看见。一个好的信息发射源应该具备两个特点：内容随时更新，看上去一目了然。这种信息发射源最常见的内容有：总体项目进度、开发工作完成情况、系统测试情况、系统运行状态等等。

2. 跃过交流的沟堑

为了提高交流的效率，作者还以两人在一个白板前的讨论为例，详细分析并列出了 11 项交流的要素（modalities in communication）。它们是：物理上的近距离性（physical

proximity)；三维性（three-dimensionality)；气味（smell)；动觉（kinesthetics)；肢体接触（touch)，如拍拍肩膀；声音（sound)；视觉（viusal)；要素同步（cross-modality timing)；低负载（low latency)，因此可以有实时问答；信任与学习（trust and learning)；共用持久性的信息发射器（use a shared, persistent information radiator)，如白板。作者认为具备了所有这些要素的交流是最有效的。

提高交流的有效性还有一个重要的问题就是很多时候需要把讨论中的信息记录下来，如记录在纸张上，或在白板上。作者称之为给信息加上“粘滞性”（stickiness)。这些记录下来的信息可以保存起来，供以后需要时查看。作者还提到了一些具体的方法。例如，把大张的白纸贴满墙上，大家可以随时随地在上面写写画画。一张纸画满了，就把它卷起来，标上日期，然后存放起来。当然，也可以用数码相机拍下来，存在计算机里面。

3. 集体主义放光芒[5]

一个项目组的成员可能来自五湖四海（特别是大项目)，为了一个共同的目标，走到一起来了。如何高效地去达到这个目标呢？很重要的一点是，大家要心（尽量）往一处想，劲（尽量）往一处使。如何做到这一点呢？作者认为可以通过如下方法让每个成员作一些小的、容易做到的改变：

- 告诉每个人现在项目的主攻方向是什么；
- 现在选择这个主攻方向的理由是什么；
- 告诉每个人他现在的行动对项目有什么作用。

这样，每个人可能只是把他的方向往目前项目的方向上调整了一点点，但产生的合力是很大的，能取得事半功倍的效果。

要提高集体（团队）的战斗力（工作力)，还有一个重要的方面，那就是在项目组中建立起良好的人文环境。使大家心情舒畅，愿意互相合作，互相帮助。作者从友好与冲突、团队意识和团队文化等方面讨论了这个问题。

友好与冲突（amicability and conflict)。友好是信任与合作的基础。作者把友好定义为“愿意倾听别人的想法，说话无恶意”，并且认为可以从一个团队内部的友好程度（amicability level)，看出同事间交流时的信息开放或封闭程度。友好（与人为善）能造成一种气氛，使大家畅所欲言，特别是不同的意见。这里作者把交流不同意见称之为冲突，并且认为这样的冲突是必须的，没有冲突或冲突不足对工作是不利的。

建立团队意识（team building)。现代企业一般都很重视团队意识或协作精神的培养。有些公司在开始一个项目时把所有成员拉到一个地方去活动一两天，以使大家增进了解，在工作中能很好地合作。这种做法在实践中也很有效。但有些程序员却不太以为然，说道：“我对我们是不是能一起烧烤或一起爬墙不感兴趣，我只对我们是否能

一起做出软件感兴趣”。那么，什么是更有效的方式呢？作者引用了一些相关文献的建议，认为最好的活动是能让大家一起完成一些只有在一起才能干好的工作。

团队文化（team culture）。一个项目组往往具备己独有的团队文化。它会受到所在的企业或组织的文化的影响，也会受到国家民族的文化的影响。团队文化还有一个重要方面便是职业亚文化（professional subculture），也就是每一个职业特点形成不同工作方式与习惯，如项目经理、程序员、需求分析人员和市场营销人员就有他们各自的工作特点与方式。

法无定法

Methodology 这个词一般译作“方法学”或“方法论”，即“study of methods”。而作者称他使用这个词是用的 Merriam-Webster 字典中的定义：“a series of related methods or techniques。”即“一组相关的方法或技术”。而 method 的定义是“systematic procedure”，类似“technique”。作者也提到一位同行在 1993 年告诉他的一句话：“methodology is a social construction。”过了两年，他才开始理解这句话的含义，并给出自己的定义为：“A methodology is the conventions that your group agrees to。”即你的团组同意遵循的一组习惯做法。这实际是个“social construction”（社会建设），社会性的一个重要的方面便是协调团组成员之间的活动，促进交流与合作。

1. 要素与原则

作者对一套方法需考虑的要素进行了详尽的分析。作者并用了一张结构图来描述 13 个结构要素和它们之间的关系。我认为其中最主要的有：

- 角色（role）。每个团组成员的工作要求、他的技能以及他的个性特点。
- 个性（personality）。人员的个性特点（似乎为角色的一部分）。
- 技能（skill）。角色需要的技能，如在一个项目中，程序员需要具备或学习 OO、Java 和单元测试技能。
- 团队（team）。一个项目可能只有一个组，如开发组，或多个组，如开发组、分析组和测试组。
- 技术（technique）。完成一项任务所需的知识和过程，如编写用例（use case），项目回顾（project retrospective）。

另外，作者还提出了一些描述方法本身的概念要素，主要的有：

- 方法大小（methodology size）：所控制的要素的数量，要素如系统发布（deliverable）、标准、活动等。
- 正规性（ceremony）：方法的精确度与控制的松紧度。如对一项使用情形作检

讨回顾，正规性低的方法允许利用午饭时间边吃边做，而高正规性的方法则可能要求花两三天来做。

- 方法重量（methodology weight）：这是方法大小与正规度的乘积，这只是一个概念性的，并非一个精确数字。
- 问题大小（problem size）：问题中的元素与它们的交叉复杂性。它很大程度上取决于估测者的知识与看问题的角度。
- 项目大小（project size）：项目组人员数量。但需注意，这个测度与"问题大小"是不同的，不可混淆。
- 系统要紧性（system criticality）：系统出错时会造成的影响，如饮料机（本来要咖啡，结果出来可乐）和飞机（可能机毁人亡）的系统要紧性显然不同。

这些要素为设计一套方法提供一个系统的思路。作者还根据自己的经验，提出了设计一套方法的七大设计原则：

（1）面对面的交流是费用最低、速度最快的信息交换方式。

如果可能，项目组成员应该坐在一起，这样交流起来方便，使开发工作能有效进行。

（2）方法超重则所付代价昂贵。

前面提到"方法重量"是方法控制的结构要素与正规性的乘积。所以，当控制要素数量增加（典型的如分析、规划、设计、图表等文档这样的工作产品），正规性提高时（典型的如频繁开会、报告），开发进程将受到很大影响。

（3）大的团队需要"重"一些的方法。

团队大起来以后，成员之间的直接交流便减少，如果没有一套有效的协调与交流的方式，工作易于陷入混乱，影响开发效率与质量。

（4）系统越要紧/重要，正规性应越高。

如上所述，正规性提高将导致费用上升。但另一方面，正规性提高的确有助于少犯错误。

（5）增加反馈与交流可以减少中间产品。

这里的"中间产品"是指在一个组内部使用的工作产品，主要是指那些描述计划中的活动的文字材料，如分析、设计、测试计划等。

（6）纪律、技能、理解不同于过程、形式、文档。

这个原则来源于 Jim Highsmith 的三句话[6]。纪律（discipline）是指一个人（主动）选择要求的工作方式，而过程（process）是指人（被动）听命而行；技能（skill）含有创造性的能力之意，而形式化或手续化（formality）则是要求按步就班；理解（understanding）是指知识，包括那些隐藏的知识，如在项目中谁知道什么、那些讨论成为设计的一部分等，而文档（documentation）则只是理解（知识）的一部分。探索

性的活动需要强调纪律、技能和理解，而软件开发很大程度上正是一种探索性的活动。

（7）从非瓶颈活动中求效率。

有一种提高效率的做法常被忽视，就是在瓶颈的“上游”处多花些时间把结果尽可能地稳定下来，再提交给瓶颈，这样可以减少瓶颈处的“重做”（rework）负荷。

在设计方法时，应充分考虑这些要素与原则。作者提出了用一个三维坐标系统来直观地表示要素和方法。项目组人数作为 x 坐标，系统要紧性作为 y 坐标，系统优先因素作为 z 坐标。x 坐标取值为 1～6 人，20 人，40 人，100 人，200 人和 500 人；y 坐标取值 C（loss of comfort），D（loss of Discretionary money），E（loss of Essential money）和 L（loss of Life）。这样 xy 不同的组合就形成了不同的坐标格，表示一个项目的基本特征。例如，“C6”表示一个项目有 6 人，系统出错时会引起一些不方便。而“D20”则表示一个有 20 人的项目，如果系统出错会导致损失一些金钱。对每一个 xy 坐标格，z 实际上说明了项目的一些其他特征。作者用这个思路来设计了他的水晶方法系列（Crystal methodology family），“D6”类称之为 Crystal Clear，“D20”类为 Crystal Yellow，“D40”类为 Crystal Orange。在此框架下，XP 可归入 C4 至 E14。总而言之，要记住“不同的项目应该有不同的方法”。

除了以上这些较为“理论”的分析，作者还从实践经验中归纳出一些对项目成功有着极大影响的“秘诀”，如：

- 2 到 8 人坐在一起。
- 专家用户在场。
- 1 月递增周期。
- 全自动回归测试（单元或/和功能测试）。
- 有经验的开发人员。

2. 构造与生长

一套方法应该是动态的、自适应的，也就是说，一套方法要在适应开发环境中构造和生长。作者把一个软件项目比拟为一个生态系统。系统中的各组成部分如人员个性、坐位、工作语言等都在相互作用、相互影响之中。当然，一个良好而稳定的系统的形成与发展主要取决于项目组成员的交互作用。如果大家能理解个人、团队和方法的特征，那么他们就能更好地选择（或设计出）一个适合于自己这个项目的初始方法并不断地发展之，而推动发展的一个关键之处在于经常性的检查与调整。这是一个随时间而进行的过程，作者提出的具体步骤为：

- 项目开始时。要做的是设计出一个初始方法，主要思路是考虑项目人数、系统要紧性、优先因素并选择一个相近的现成方法（如 XP、RUP 等），然后由整个项目组充分讨论候选方法，并加以修改、“定稿”，成为项目的初始方法。

- 第一个递增周期的中间。讨论目前的初始方法是否工作，如有必要，则需作一些调整。
- 每个递增周期之后，应该有一个专题检讨会（reflection workshop）。要讨论的两个主要问题是“学到了什么”和“什么可以做得更好”，据此对方法进行一些修改，作为下个周期的方法。
- （第二个周期以后的）每个递增周期的中点。这个检讨的目的主要是尝试新的想法和做法。但作者认为如果周期为三周或更短，则这个中点检讨可以不做。

多余的话

我在 2001 年 7 月左右第一次读到这本书时的草稿（draft version 2b1）时，第一感觉就是耳目一新。对书中所列举的对个人和团体的描述都有似曾相识之感（相信干过几年软件开发的同行都会有同感），但从未被这样全面而系统地分析过。读第一遍产生的“冲击波”消退后，我开始重读并作了一些笔记，以求更好地理解并掌握作者的主要思路，但在阅读中却觉得材料有些零乱和重复。这本书在 2001 年底正式出版以后，我把书对照着草稿又读了一遍，发现基本材料（我想有 90%）没变，但次序和结构有些变化。很明显，作者在尽力把这些材料有机地组织起来并表述出来。

总地说来，我觉得这本书提供了一个新的角度来看待软件开发活动，以及一个新的思路来设计开发方法。书中提供的材料大部分来自作者丰富的实践经验，对软件开发实践有很高的参考价值，值得软件项目参与者，包括出资者、管理者与开发者反复研读。我认为美中不足的是，作者试图建立来统帅这些材料的框架并非严谨完美，从结构上说，这本书给我的感觉更象是一本结构化的技术散文集，而非严格意义上的学术著作。不过，从另一方面说，这也许是一个优点，让读者能从这些丰富的材料中，根据自己的知识和经验来构造出自己的“理论”框架。

在这篇介绍文章中，我试图根据我的理解来勾勒出作者的主要思路，使读者诸君对这本书有个大概的印象。文章基本依据书的顺序，为方便读者阅读原文，再简要说明如下：

- “Introduction: Unknowable and Incommunicable”：引论，主要说明人要正确地认识事物和准确交流是非常困难的，甚至是不可能的。本文未作介绍。
- “Chapter 1: A Cooperative Game of Invention and Communication”：在本文的“软件开发是什么”中介绍。
- “Chapter 2: Individuals”：在本文的“人啊，人”中介绍。
- “Chapter 3: Communicating, Cooperating Teams”：在本文的“团队的力量”中介绍。

- “Chapter 4: Methodologies”：在本文的“法无定法”中的“要素与原则”中介绍。
- “Chapter 5: Agile and Self-adapting”：在本文的“法无定法”中的“构造与生长”中介绍。
- “Chapter 6: The Crystal Methodologies”：是作者对他的水晶方法系列的简介。本文未作介绍。

这本书和 Jim Highsmith 的 *Agile Software Development Ecosystems* 是计划中的 *“Agile Software Development Series”*（系列丛书）的两本基础书。作者称这套丛书的核心思想是：

- 不同的项目需要不同的一套方法。
- 注重技能、交流和团队的项目要比注重过程的项目有效。

这套丛书可分为三大类：

- 个人技术，如作者的 *Writing Effective Use Cases*。
- 群体技术，如作者的 *Surviving Object-Oriented Projects*。
- 方法实例，如作者的 *Crystal Clear*。

从这里可以看出，作者关于软件开发的思路是技能、交流与团队为纲，方法过程为目，“纲举目张”。

注释

1. Tony Hoare 是计算机科学的奠基者之一。1999 年从牛津大学退休。此公老骥伏枥，志在一统。最近几年从事的研究之一便是计算机科学的统一理论。

2. Bertrand Meyer 是现代软件工程的先驱之一，他的著作 *Object Oriented Software Construction* 被视为 OO 技术的经典。“契约式设计”（Design by Contract）是他对软件开发技术的重要贡献。他有一篇文章 Software Engineering in the Academy（IEEE Computer 2001 年 5 月）是谈院校的软件工程教学的，也可从中较全面地了解他对软件工程的看法。

3. 有趣的是，20 世纪 60 年代末 70 年代初，也是结构化程序开始盛行之时。一篇著名的经典文献是 1968 年 Edsger W. Dijkstra 的“Go To Statement Considered Harmful”（Communications of the ACM, Vol. 11, No. 3, 1968）。

4. 作者书中的 iterative 的含义和现在常用的意思不太一样。书中给的定义是：a scheduling and staging strategy that allows rework of ieces of the system，所以，译成“反复”似乎恰当一些。而 incremental 在书中的定义为：a scheduling and staging strategy in which peices of the system are developed at different rates or times and integrated as they