开发人员专业技术丛书

优化软件开发生命周期

HZ BOOKS

Microsoft Press

Solid Code

Optimizing the Software Development Life Cycle

完美代码

（美） Donis Marshall
John Bruno 著

徐旭铭 译

机械工业出版社
China Machine Press

开发人员专业技术丛书

Solid Code

Optimizing the Software Development Life Cycle

完美代码

(美) Donis Marshall
John Bruno 著

徐旭铭 译

机械工业出版社
China Machine Press

本书简单明了地介绍了软件开发中的最佳实践，展示了工程流程在编写优质代码上的重要性以及测试的重要性，总结了众多资深工程师的经验教训，并提供了很多真实案例。书中介绍的经验可以应用到产品开发周期的每个环节，从设计到开发以及最后的发布和维护。本书的中心思想就是要在设计和实现的过程中改进代码质量，包括类建模、性能、安全性、内存使用以及调试，帮助读者构建完美的项目。本书适合各类软件开发人员阅读。

Donis Marshall and John Bruno: *Solid Code: Optimizing the Software Development Life Cycle* (ISBN 978-0-7356-2592-1).

本书版权登记号：图字：01-2009-4156

图书在版编目（CIP）数据

完美代码/（美）马歇尔（Marshall, D.）等著；徐旭铭译．—北京：机械工业出版社，2010.1
（开发人员专业技术丛书）
书名原文：Solid Code: Optimizing the Software Development Life Cycle

ISBN 978-7-111-29240-1

Ⅰ.完…　Ⅱ.①马…　②徐…　Ⅲ.软件开发　Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2009）第 227167 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：刘立卿
北京京北印刷有限公司印刷
2010 年 1 月第 1 版第 1 次印刷
186mm×240mm · 15.25 印张
标准书号：ISBN 978-7-111-29240-1
定价：45.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@ hzbook.com

专家推荐

《完美代码》在管理书籍和技术书籍之间做到了出色的平衡。从如何进行软件建模，到安全性设计，再到防御性编程，本书展示了可以改进开发工作的各种最佳实践。

——Wintellect 联合创始人，John Robbins

《完美代码》不仅仅是一本关于代码的书，它阐述了如何开发一个健壮的项目。这本书简单明了地介绍了软件开发中的最佳实践，并提供了很多实际产品中的案例和经验教训，帮助读者构建完美的项目——从设计到开发，以及最后的发布和维护。

——微软，软件开发工程师，Jason Blankman

作为一名有 20 年经验的软件工程师，能让我每过几年就重读一遍的书并不多见，而《完美代码》就是其中之一，每次温故都能知新。

——微软，软件开发工程师，Don Reamey

对任何专业软件工程师来说，《完美代码》的价值都是无法衡量的，书中到处都是可以立刻投入使用的实践经验。《完美代码》绝对是一本让你爱不释手的必读书籍。

——AJI Software 执行股东，微软区域总裁，John Alexander

《完美代码》对任何 IT 从业人员来说都是必读书籍，特别是如果你打算使用托管代码的话。它不仅涵盖了工程上的最佳实践，还通过已经过实践检验的案例来展示它们。

——微软，发布经理，Andres Juarez

这本书提供了在高效软件开发过程中的最佳实践，因此可以避免很多典型的程序员错误。作者提供了可实践的检测错误的方案，并解释了微软是如何进行软件开发和测试的。

——微软，测试经理，Venkat B. Iyer

无论你是新手还是专家，任何级别的程序员都可以阅读本书。它为优秀的开发实践提供了坚实的基础，不管开发团队的规模是大还是小，甚至只有一名程序员，也应该采用书中的经验。

——独立软件工程师，John Macknight

序

软件工程和传统意义上的工程完全不同。作为一名软件程序员，我非常自豪可以自称是一名工程师。工程师能够通过细致的计划思考并制造出一次就能工作的东西。所以，在我的职业里包含“工程师”这个词实在是一件很酷的事情。

我们先来看看要是把普通的软件工程方法应用到航空工程里会发生什么事情。一驾飞机正停在登机口等待乘客登机，这时一名航空工程师（不管是一时兴起还是被老板强迫）打算要换掉尾翼部分。毕竟那只是一个尾翼而已，直接把它弄下来换一个上去就好了。绝对没问题，绝对能行！要是航空工程采用和软件工程一样的流程的话，我估计乘客都会瞬间逃离飞机的。但是，这类改变在软件项目的世界里基本上每天都要发生。过去有种自相矛盾的老笑话说是“军事情报”，我觉得“软件工程”也属于这类范畴（这种笑话就是把两个矛盾或者互斥的词放在一起，通常都是为了幽默搞笑——译者注）。而更让人头疼的是，虽然软件真的是在“统治”这个世界，但是几乎所有人在创建它时所采用的方法无一可以称得上是“工程”。

为什么我敢说我现在正在使用的计算机可以正常工作，但是我正在使用的程序（Microsoft Word）却可能会搞乱我列表的自动编号呢？我这么说可能会得罪我那些电子工程的朋友，但是我还是要说，因为硬件比较简单。电子工程师要对付的只是很有限的输入，不像软件工程师要面对几乎无限可能的输入。

管理学也认为电子工程是“真正的工程”，所以管理的时候可以给予适当的时间和资源。而软件业，作为一个独特的领域，还算不上成熟，它真正存在的时间不是很长。其实说起来，我比软件这个行业年轻不了多少，所以我“不成熟”的看法才能透露出这些问题。要是我和电子工程一样老的话，那么现在我就该躺在坟墓里写书了。

软件开发的另一个难题有时候是软件工程师自己。老实说，成为一名软件工程师的门槛还是很低的。我就是一个现成的例子：在获得计算机科学学士学位之前，我就已经是一名全职的软件工程师了。当初获得这份写程序的工作只不过是因为我在面试的时候“特别能吹”而已。我的老板根本就不在乎我没有受过正规的教育，毕竟我要求的工资比较低。

在所有真正的工程领域里，你都必须先获得认证资格后才能在名字里加上专业工程师（Professional Engineer，PE）的头衔。在软件行业里就没这种规定。其中一部分原因是这个行业太年轻了，没有人知道在从业之前应该必须掌握什么知识才能成为一个软件工程师。而在其他领域里，专业工程师通常意味着大量管理上的责任。如果一名有资格的工程师觉得一个设计是行不通的，她就不会在计划书上签名，项目也就不可以进行下去。这就迫使管理层在做计划的过程中更加严

谨。同时，如果签收了一个项目，那就代表专业工程师愿意在出错的时候承担道义和法律上的责任。你有没有准备好承受你的软件设计所带来的道义和法律上的责任呢？只要我们的行业还没有达到那种程度，我们就不能真正自称是传统意义上的工程师。

在我从业近20年的岁月里，软件开发行业已经有了长足的进步。管理高层终于意识到软件项目的失败会给公司带来巨大的损失。有兴趣的话可以看一下 Robert Charette 在《IEEE Spectrum》2005年9月一期（*http：//www. spectrum. ieee. org/sep05/1685*）上发表的文章《为什么软件会失败?》（*Why Software Fails*?）里列出的重大失败。正因为代价是如此之大，所以有些管理高层终于第一次开始正视问题，并为软件项目的启动、计划和实现提供真正的资源。虽然我们要走的路还很长，但是这种来自管理层的真正介入是我在这个行业里看到的最大的变化之一。

在实际工作的层面上，软件开发里最喜人的变化是几乎所有的程序员都开始意识到测试代码的重要性。现在已经很少能听到一个程序员直接把代码扔给 QA 组还期望能一切正常了。这对我们这个行业来说是一个巨大的进步，它让很多团队终于能够按期达到所要求的品质。作为一个把整个职业生涯都投入在调试和性能调节上的人，我真的对我们的行业在测试上变得成熟而感到欣慰。和所有好的改变一样，专注测试虽然是从个人做起，但是它的好处却可以影响到整个组织。

还有一个进步就是我们的工具和环境变得越来越好。在 .NET 下，测试代码变得非常容易，这也就意味着更多人将参与测试。另外，抽象层正在逐渐上移，所以我们不再需要在计算机上处理所有的事情。例如，如果你需要调用一个 Web 服务，你不需要手动打开一个端口，编写 TCP/IP 包，调用网络驱动程序，等待数据返回，然后解析返回数据。这一切现在只要一个方法调用就可以完成。这种更好的抽象层让我们可以把时间花在软件项目中更重要的部分：真正的需求和帮助用户解决问题。

在软件开发真正成为工程领域之前，我们还有很长的路要走，但是这些信号都是很令人鼓舞的。我觉得，当我们最终开始把测试当做是一门专业的时候（把它看得和开发一样，甚至更重要的），就又会发生巨大的变革。尽管在我退休之前大概是看不到软件工程的这一转变了，不过我对现在的发展十分有信心。让我们一起学习，努力推动，这样总有一天我们可以自称是真正的工程师。

本书是把软件纳入工程领域里重要的一步。书店里关于编程的书最多的是两种，一种是泛泛而谈的软件管理类，另一种是面面俱到的技术揭秘类，对于后一种，我总是不太喜欢。当然这种书有它们的用处，有时候也很有帮助，但是我们缺乏的是专门讨论真实世界里团队软件开发的书籍。这种技术在项目中所占的比重很小，但是它对软件项目的发布是最大的挑战。本书在管理书籍和技术书籍之间做到了出色的平衡。从如何进行软件建模，到安全性设计，再到防御性编程，Donis 和 John 向你展示了可以改进开发工作的各种最佳实践。阅读本书就像是在一位顶极的开发经理的带领下体验一个出色的项目，就像是和出色的同事一起工作。

这本书写得非常出彩，我特别喜欢它重点强调的计划和准备工作。我公司（Wintellect）曾经为很多项目救火，大多都是由于它们最初糟糕的计划所导致的后果。细细研读这些章节，你就可以避免那些可能浪费你大量金钱和时间的错误。这本书指出的另一个问题是把性能调节和安全性分析留到项目的最后阶段。正如书中第4章的标题所指出的，“性能也是功能”。在那些章节中，

箴言的价值都是无可估量的。最后，书里还强调了即使代码已经进入了维护阶段，还是能从先前的编码和调试中获取益处。即使像我这样在这个领域里耕耘了将近20年的人，还是从这本书里学到了很多好的方法。

不仅每个程序员都应该读一读这本书，而且公司里的其他人也可以读一读。向你的经理，经理的经理，甚至经理的经理的经理推荐这本书吧！不管是什么公司，我经常会被管理高层问到的一个问题就是："微软是怎么开发软件的?"在这本书里，大多数章节中都有一个微软内幕小节，让你的老板看看就可以知道微软是如何解决那些当今最大型的应用程序中的问题的。开始阅读吧！现在轮到你来帮助我们的行业向真正的工程标准迈进了。

John Robbins

Wintellect 联合创始人

前　言

在过去多年里软件开发已经有了很大的演化。编程语言和快速开发工具（比如 .NET 和 Visual Studio 2008）的进步让软件工业可以构建更高质量的软件，速度更快，成本更低，更新更频繁。尽管这种对软件和工具以及过程演化的需求日益增加，但构建和发布高质量的软件对于软件项目的所有参与者，特别是程序员来说，仍然是一项艰难的工作。幸运的是，本书包含了应用程序工程师在开发健壮的代码中所需要的工程实践、过程、策略和技术等一切最佳要素。

本书探讨了软件开发中几乎每一个方面所需的最佳实践，以达到更高的代码质量。这本书提供了来自资深工程师的实践经验，可以应用到产品开发周期里的每一个环节：设计，构造原型，实现，调试，测试。这些宝贵的资料和建议还进一步得到了来自真实世界实例的支持，这些来自微软内部的工程团队包括了（但不限于）Windows Live Hotmail 和 Live Search 团队。

本书的读者

本书适合软件开发里每一个环节的参与者阅读。特别是那些在构建高质量软件方面寻求最佳实践和建议的应用程序工程师。书中很多部分都展示了工程流程在编写优质代码上的重要性。其他部分则着重于测试的重要性。本书的中心思想就是要在设计和实现的过程中改进代码质量，包括类建模、性能、安全性、内存使用以及调试。

专业和业余程序员都可以阅读本书。我们假设读者对编程概念和 C#面向对象编程具有基本的理解，而水平则没有要求。本书主要是关于托管代码应用程序开发的最佳实践，书里所探讨的话题应该能引起各级托管代码程序员的共鸣。

本书的组织

本书的组织形式和应用程序开发生命周期非常相似。各个章并不是分割成很多部分，而是根据四个关键的原则组织在一起。第 1 章里列出了这些原则：专注设计、防御和调试，分析与测试，以及改进流程和态度。

- **专注设计**　本书最重要的主题之一就是：经过深思熟虑的设计是提升产品整体质量的重要途径。为了支持这一主题，我们将探讨很多不同的实践，例如类设计和原型开发、元编程、性能、伸缩性以及安全性等话题。
- **防御和调试**　出色的设计是构建高质量软件应用的关键，但是理解那些会阻碍发布无错代码的陷阱的重要性也同样不能忽视。根据这一原则，我们要讨论的话题包括内存管理、防

御性编程技术以及调试。

- **分析与测试** 即使最伟大的程序员遵循了所有推荐的最佳实践，他仍然会写出有 bug 的代码。因此，讨论如何进行代码分析以及测试的方法是进一步提升代码质量的重要环节。
- **改进流程和态度** 除了最佳实践以外，工程的流程和文化也能对产品的质量产生巨大的影响。我们探讨了提升团队的效率以及追求质量几个重要话题。

系统要求

至少要有以下的软件和硬件环境才能在 32 位 Windows 环境下编译执行本书中的示例代码：

- Windows Vista，Windows Server 2003 with Service Pack 1，Windows Server 2008，或者 Windows XP with Service Pack 2。
- Visual Studio 2008 Team System。
- 2.0GHz CPU，推荐 2.6 GHz CPU。
- 512MB 内存，推荐 1GB 内存。
- 8 GB 硬盘安装空间，推荐 20GB 硬盘空间。
- CD-ROM 或 DVD-ROM 驱动器。
- 微软或兼容鼠标。

配套网站

本书在网上设立了一个站点提供示例代码的下载。代码按照章节来组织，可以从以下地址下载：*http：//www. microsoft. com/learning/en/us/books/12792. aspx*。

其他在线资源

本书的补充资料和更新将发布在 Microsoft Press Online Developer Tools Web 站点上。它包含了对本书内容的更新、专栏文章、配套内容、勘误、样章等等。站点的地址是：*http：//www. microsoft. com /learning/books/online/developer*。这个站点会定期更新。

本书的支持

我们尽力确保本书内容的准确性。微软出版社在以下地址提供关于图书的勘误：*http：//www. microsoft. com/mspress/support/search. aspx*。

如果要直接联系微软的帮助和支持，可以在下列网址中输入你的问题：*http：//support. microsoft. com*。

如果你有关于本书的评论、问题或建议，或着在知识库里找不到你问题的答案，请按照以下方式联络微软出版社：

电子邮件：

mspinput@ microsoft. com

通信地址：

Microsoft Press

Attn: Solid Code editor

One Microsoft Way

Redmond, WA 98052 - 6399

请注意：以上联系方式并不提供产品支持。如果你需要支持信息，请访问微软产品支持站点：*http: //support. microsoft. com*。

致谢

牛顿说过，“如果我看得比别人远，那是因为我站在了巨人的肩膀上。”这句名言正适用于这本书，特别是考虑到其中所引述的实践、观点和经验的时候。更进一步地说，那些肩膀属于很多为这个项目做出贡献的人。虽然我们的名字占据封面，但是我们要感谢每一个帮助这本书出版的人。我们要感谢他们在这个项目里做出的贡献和支持，并且希望能一一向他们表示谢意。

首先，如果没有微软出版社的同仁就没有这本书的出版。这里我们要感谢 Ben Ryan、Devon Musgrave 和 Melissa von Tschudi - Sutton，他们保证了本书的高质量和项目的按期进行。另外，我们还要感谢技术编辑 Per Blomqvist 和审稿 Cindy Gierhart 所做出的贡献和反馈。

前面提到过，本书包含了大量的实践、观点和经验。如果没有来自微软和行业里专家们的贡献、支持和反馈，这些元素是不可能出现在书中的。这里我们要特别感谢为此做出贡献的人和帮助我们审校的人：Jason Blankman，Eric Bush，Jacob Kim，Don Reamey，Dick Craddock，Andres Juarez Melendez，Eric Schurman，Jim Pierson，Richard Turner，Venkatesh Gopalakrishnan，Simon Perkins，Chuck Bassett，Venkat Iyer，Ryan Farber 和 Ajay Jha。

还要感谢 Wintellect 公司。Wintellect 是一个咨询、调试和培训公司，专门帮助其他公司专注在 .NET 和 Windows 开发环境上构建更好的软件。它的服务包括短期的 .NET 进阶培训课程，以及开发和咨询服务，包括紧急调试。公司还主持了 Devscovery 研讨会——一个 3 天的多路会议，专门面向中级和高级程序员。更多的关于 Wintellect 的信息请访问 *www. wintellect. com*。

Wintellect 公司的 John Robbins 和 Jeffrey Richter 都为本书提供了无价的建议和及时的反馈。在此特别感谢！

Donis Marshall 我已经写过好几本书了。但是这是第一本合著的书。在完成本书以后我有一个感想，为什么过去我不找一个人来合著呢？John Bruno 对这个项目来说是无比重要的人物。他广博的知识和深刻的洞察力足以让任何关心 Windows 的技术人员都不能错过本书。John 还具备一个作家少见的可贵品质——遵守时间。

John Bruno 写书是一件劳心劳力的事情，经常会影响到你最亲近的人。所以，我首先要感谢我的妻子 Christa 和我的两个儿子，Christopher 和 Patrick，感谢你们在我写这本书的过程中给予我的耐心、理解和牺牲。你们的爱和支持让我每天都能保持最好的状态。另外，我还要感谢 Donis Marshall 邀请我加入这个项目。我感谢他的友善，并让我有机会和他一起参与这样一个重要的工作。在我的生命里，我有幸认识了很多富有创造力和具有聪明才智的人。对于那些总是启发我、鼓励我、挑战我和支持我的人，在这里感谢你们。

目　录

第 1 章

敏捷世界里的代码质量

我们只是熟能生巧而已。卓越源自习惯，而非一时的行为。

——亚里士多德

作为软件工程师，我们都希望自己的产品是出色的。我们希望能让用户享受到无错的使用体验以及最佳的程序质量。而在同行之间，我们希望能用优雅和稳定的代码展示出自己超凡的技艺。每天我们都在向着这些目标努力工作。我们不断地重复努力开发出高质量的软件，加上我们不断的对改进方法和实践的渴望，凝聚成程序员的习惯。正如亚里士多德所说，卓越是通过重复不断的练习而达到的。

然而，构建高质量的软件是一件非常困难的工作。即便是最基本的程序里也有会 bug。每个人（包括最好的工程师）都会写出有 bug 的代码来。而人类本身就是不完美的，我们经常会犯错。因此当我们将人类语言翻译成按照我们的指令来工作的软件应用程序时，它一定会出错，一定会有意外发生，所以出现质量问题也就没什么好奇怪的了。

但是，仅仅把软件质量的责任归咎于程序员是不公平的。软件工程是一个有很多来自不同领域的人参与的过程。例如在微软，工程团队通常会分成三块：程序管理、开发以及测试。程序管理保证了产品的设计规范是准确的，经过充分考量的，并且他们会明确最终产品所要达到的质量目标。程序员负责创建最有效以及最灵活的设计，保证算法的准确性，以及在代码实现里应用各种最佳实践。测试人员考虑代码和应用程序行为的每一种可能的组合，并且完整检查软件的每一条代码路径。质量属于工程生命周期里的每一个环节，因此每一位参与者都对它负有责任。软件开发组织对此有着深刻的理解，并且花费了大量的时间和精力来实现各种过程和程序以确保每一位团队成员都能专注于质量的保证上。

软件开发过程和方法论自 20 世纪 90 年代末就在业界存在了。随着计算能力的增加，软件程序变得越来越复杂。而在软件开发里增加的复杂度，再加上业界相对的不成熟，导致了软件项目里的诸多问题，这包括了成本超支，缺乏形式化的质量保证过程而导致糟糕的软件质量，以及代码的低可维护性等。因此，诞生了形式化的软件开发过程。它们的主要目标是把一系列的任务放到形式化的结构里，最终能让开发工作产生更好的结果。简单来说，采用形式化的工程流程的目的就是为了产生更优质的产品，让市场或者部署更加容易对其进行预测。使用过程来保证产品的质量是所有提供产品或者服务的产业进行工作的基础。作为软件工程师，在你的职业生涯里说不定已经经历过一些不同的项目管理方法论了，比如瀑布模型、敏捷方法甚至微软解决方案框架（Microsoft Solutions Framework，MSF）。

1.1 软件开发的传统方法

你可能已经听过非常传统的软件开发周期（Software Development Life Cycle，SDLC）里的“瀑布”模型了。这种模型指的是在软件开发周期里的每一个阶段都逐渐向下进行，如图 1-1 所示，它包括的阶段有：

- **需求** 对要构建的软件进行详细的书面描述。
- **设计** 计划软件项目实现细节的过程。
- **实现** 编码、测试以及调试软件的各个独立部分的过程。
- **集成和验证** 将软件的各个部分集成在一起进行更大规模的测试阶段。
- **安装和维护** 软件开发周期的末尾，新软件完成部署并且进入发布后的修正和调整阶段。

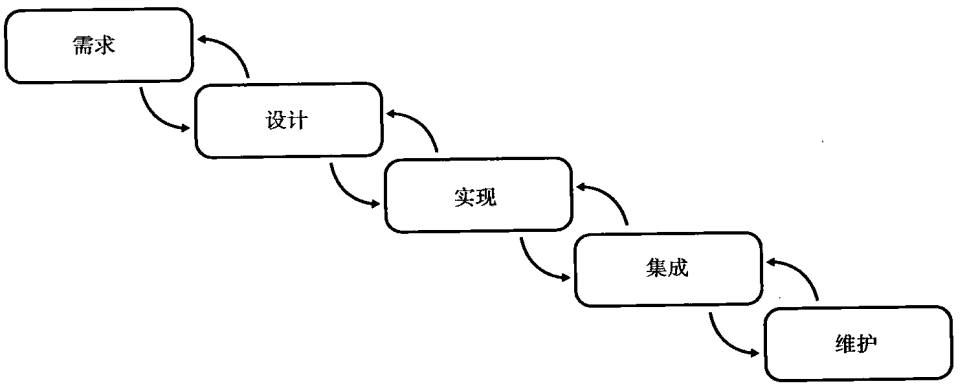

图 1-1 传统的软件开发周期

这个模型呆板的地方在于它要求每个步骤都必须按顺序进行，直至软件完成，并且要求在一个阶段完成之前不可以开始下一个阶段。需求分析必须在进入设计阶段之前全部完成。一旦设计完成，准备好让工程师实现需求的计划后，编码阶段就开始了。集成跟在编码的后面，将程序的各个部分组合起来作为一个完整的系统来测试。当集成完成、软件被部署以后，就进入维护阶段。这里的每一个阶段里，都要求详细的书面文档，这就让瀑布型的项目变得十分臃肿。

瀑布模型保证质量的方法是假设你在分析和设计软件时花费更多的时间，在整个过程中尽早地发现纰漏，才能在随后的开发阶段避免错误。批评指出这种方法在实际中并不好用，因为程序或应用里很多关键的细节不到实现阶段是不可能发现的。而且，由于这个模型强制要求在编码开始之前要冻结规范文档，它就很难在发布阶段将早期的用户以及测试反馈包含进来，这样就限制了团队及时响应客户需求的能力。最后，完整的端到端测试被一直留到了整个周期的最后，这就在集成关键部位开始的时候，给应用程序的质量增加了风险。

注解：

“瀑布”方法通常被认为是由 Winston W. Royce 博士在 1970 年所发表的一篇论文，“管理大型软件系统开发（Managing the Development of Large Software System）”里提出的。具有讽刺意味的是，Royce 博士实际上在论文里是在试图指出这种方法是有风险的，而且有可能导致项目失败。他提到了在项目周期的末尾进行测试，即当系统里所有的关键部分第一次被组装到一起的时候才进行测试是不现实的。他还进一步指出在周期的后期所发现的错误会由于随之对设计做出的修改而导致成本不可避免的超支。虽然他从概念上是认同这种分阶段的开发周期模式的，但是更倾向于将它进化成一种更加迭代式的方法，他认为那样会比流水线模型更有可能达到良好的结果。

尽管毁誉参半，软件项目管理的瀑布型方法为很多今天存在的衍生模型提供了一个框架。正如 20 世纪 80 年代末以及 20 世纪 90 年代初兴起的快速应用开发方法，以及近年来很多公司逐渐转向的敏捷开发方法论，都是衍生自瀑布型方法。这些更新的模型在软件开发过程中更加迭代化，而且它们鼓励更短、更快、更集中的发布周期。

1.2 软件开发的敏捷方法

基于对瀑布模型这样臃肿严格的方法论的不满，业界逐渐兴起了向更轻型的软件开发方法论演化的运动。在 20 世纪 90 年代中叶出现了好几种这样的轻型方法，专注于类似自我管理团队、提倡面对面交流、轻型文档和频繁发布等概念。最终，这些方法论的缔造者们，包括 Kent Beck、Martin Fowler 以及 Ward Cunningham，组成了敏捷联盟（Agile Alliance）[⊖]，统一了敏捷软件开发的原则，发布了敏捷宣言（Agile manifesto）[⊜]。

敏捷方法论试图通过小型的，自我管理的团队用短小合作式的发布周期来鼓励迭代式的软件开发。质量在敏捷软件开发的每一个阶段都是及其重要的，并且采用了很多关键的原则（例如结对编程、测试驱动开发、重构和持续集成）来保证能在每一个发布周期里及早及时地发现并消灭错误，这和原本传统的方法论非常的不同。

今天广泛使用的不同敏捷方法论有很多，包括（但不限于）Scrum、极限编程（eXtreme Programming，XP）、测试驱动开发（TDD）、重构和持续集成。另外，还有很多优秀的资源来帮助理解每一种方法背后的原则和实践（请参见附录 A 里的图书列表）。本章准备介绍其中一些最流行的敏捷方法，并向你展示这些方法中那些微软工程文化里喜欢用来改进软件质量的原则和实践。

⊖ *http：//www. agilealliance. org*。

⊜ *http：//www. agilemanifesto. org/principles. html*。

1.2.1 Scrum

Scrum 大概是目前敏捷方法里最出名，至少也是最脸熟的一个。从本质上来讲，Scrum 本身并不算是真正的方法论，而是一组实践和为在其过程中的参与者所定 义角色的框架。和绝大多数敏捷方法一样，Scrum 鼓励小型的、自我管理的团队，在短的发布周期里完成一系列良好定义的开发任务。

尽管 Scrum 为管理软件开发过程提供了一个很有价值的框架，但它自身却几乎没有规定任何明确的方法来管理开发软件的质量。不过这并不一定会产生问题，因为 Scrum 把这个责任下放给 Scrum 团队里的每一个人，让他们自己自由选择要实现什么样的质量控制实践。然而其他的敏捷方法论则比较少关注项目的框架，而是更多地对管理代码质量做出了具体的实践规定。

每一个 Scrum 项目周期都被表示为一个 Sprint，它表示完成一组功能开发所需要的时间。Sprint 从计划阶段开始，即从 Scrum 团队决定要把精力放在一组功能上开始。这组功能是由团队在源自 Product Backlog 的计划阶段期间选择的，所谓的 Product Backlog 是指一张关于软件开发要实现的所有可能功能的优先级列表。在计划阶段的末尾，所有从 Product Backlog 里选出来的功能都会被加入并在 Sprint Backlog 里跟踪，Sprint Backlog 表示了团队要开发的具体功能的细节。一旦 Sprint Backlog 定义完成，Sprint 就开始了。通常一个 Sprint 会持续 30 天，在这期间，团队每天都要聚在一起检查当天的工作状态并且帮助每一个团队成员保持工作效率。在 Sprint 的末尾，之前在 Sprint Backlog 里定义的功能将全部完成并且可用。这个过程如图 1-2 所示。

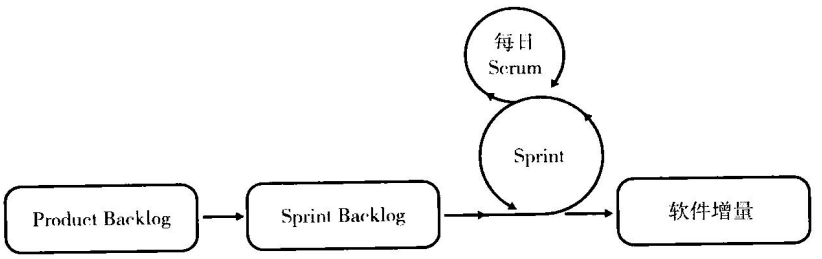

图 1-2 Scrum 过程流

注解：

Scrum 是一个非常有趣的框架。除了一些花里胡哨的参与角色名词，诸如“猪”和“鸡”之外，Scrum 所提供的这个框架还是有很多有价值的地方的。如果你想更多地了解它的内容，推荐读一下 Ken Schwaber 所著的《Agile Project Management with Scrum》（Microsoft Press，2004）。

Scrum 可能是微软工程团队里使用的最广泛的敏捷项目管理实践形式了。Scrum 为管理所提供的过程事实上对于小型的、相对自治的开发团队来讲非常有效。大多数团队都已经努力把 Scrum 应用到我们的核心工程实践里来了。