第8辑

荟 萃 全 球 软 件 智 慧

Dr.Dobb's JOURNAL CHINA
www.ddjchina.com

# 软件研发



▶建模 ▶管理 ▶安全 ▶模式 ▶科学计算 ▶算法 ▶Java ▶C++ ▶.NET ▶XML

## 【专题】最佳实践

净室软件工程，敏捷过程开发关键任务应用程序，用XP开发语法分析器，激进的重构，如何编写容易理解的代码，增量开发

书摘：《编程珠玑》

in association with
Dr.Dobb's JOURNAL C/C++ Users Journal software development windows developer
独家授权中文版

# 原版一览

2004 年 3 月

【专题：编程语言】

小语言开发环境
*By Clements & Felleisen 等*
程序员们针对特定的应用领域，往往需要自己设计“小语言”。本文中，我们的作者,来自波士顿几所名校的教授和研究生开发了一个公用的编程环境，有漂亮的 IDE，能够根据具体“小语言”自我调整。

Bistro 程序设计语言
*By Nik Boyd*
Bistro 提供了类似于 Smalltalk 的方法语法，但是最后生成的是 Java 类文件。

使用 Python Lex-Yacc 创建解释器原型
*By Shannon Behrens*
为了测试 Python 和 PLY 环境，Shannon 编写了一种称为“Squeamish”的语言，只用了 850 行代码。

集合的枚举：循环、迭代器和嵌套函数
*By Walter Bright & Matthew Wilson*
著名 C/C++ 编译器 Digital Mars 的开发者为了简化 C++ 的复杂度，开发了 D 语言。本文中他和另一位 C++ 名人用 D 语言的 foreach 语句实现了集合枚举。

用 Prolog 构建自定义规则引擎
*By Dennis Merritt*
用基于逻辑和基于规则的工具为逻辑知识编码。

安全和伪随机数生成器
*By Ben Laurie*
在安全领域中，比较弱的随机性有时候比强随机性更好。作者是 Apache 和 OpenSSL 的核心开发人员。

即时消息：程序员的工具？
*By William Wright & Dana Moore*
本文作者考察了 Jabber 的客户端协议，并用 Python, Perl, 和 Ruby 进行开发。

用 OpenMP 标准快速图像处理
*By Henry A. Gabb & Bill Magro*
基于 OpenMP 的工具和多线程可以改进多处理器系统的性能。本文作者来自 Intel 公司并行应用中心。

预览表单和生成 Bean 的一个 Struts 工具
*By Andy Yuen*
Struts 正在成为构造 Java Web 程序的事实标准框架。看作者如何实战运用。

软件安全的业务问题
*By Herbert H. Thompson & James A. Whittaker*
公司在软件安全方面投资是否值得呢？系列文章之二。

添加 .NET 控件属性
*By Phil Wright*
本文提出了处理控件属性的惯用法，解决了属性不能很好地与设计环境集成的问题。

技术点滴
*By George F. Frazier*

事务处理
*By Charles Curley*
事务处理保证了在出现灾难性的故障时，数据的完整性。

【专栏】

嵌入系统专栏：移植 Small-C
*Pete Gray*
Pete Gray 将 Small-C 语言移植到了 Motorola 的 DSP56800 数字处理器上。

Programming Paradigms 专栏：沉默是金
*By Michael Swaine*

Embedded Space 专栏：过时了！
*By Ed Nisley*

Jerry Pournelle 专栏：晦暗的未来与最大性价比
*By Jerry Pournelle*

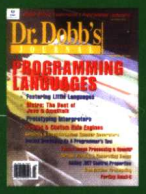
程序员书架：SQL Server 性能与优化
*By Douglas Reilly*
《Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook》和《The Guru's Guide to SQL Server Architecture and Internals》。

社论：浓缩的都是精华
*By Jonathan Erickson*

Verity Stob 的新历险
*By Verity Stob*

Swaine's Flames 专栏：猜字游戏
*By Michael Swaine*

## Dr. Dobb's Journal



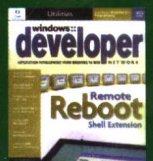

全球软件技术界最具声望、发行量最大的领袖杂志，创办于 1976 年，海纳百川，兼容并蓄，见证和推动了计算机技术的发展(www.ddj.com)。重要作者包括：计算机科学大师Jon Bentley、世界安全权威Bruce Schneier、Python之父 Guido van Rossum 等等。

## Software Development Magazine


软件工程和管理领域的世界顶级杂志，业界大师的论剑道场(www.sdmagazine.com)。重要作者包括：用例之父Ivar Jacobson、IBM/Rational 首席科学家 Grady Booch、AOP 之父 Gregor Kiczales、构件技术大师 Clemens Szyperski、面向对象巨匠 Bertrand Meyer、软件工程新锐 Scott Ambler、敏捷方法大师 Alistair Cockburn、Martin Fowler、项目管理权威 Johanna Rothman、Karl Wiegers。

## C/C++ Users Journal


C/C++ Fans 的圣地(www.cuj.com)，聚集了C/C++几代大师。重要作者包括：C++之父 Bjarne Stroustrup，C++ Guru Scott Meyers、Andrei Alexandrescu、Herb Sutter、Andrew Koenig 等。




## Windows Developer Journal

微软技术精品刊物（www.wdj.com），涵盖了安全、.NET、ASP.NET、Windows 开发、数据库、编译器、操作系统、Web 服务等诸多主题。重要作者包括：Dina Esposito、

2004 年 3 月

【主题：开源软件的胜利】

成功的开源软件
*By John Ravella & Rosalyn Lum*
在 Richard Stallman 发起自由软件运动 20 年后，大量开源产品已经开始成熟。

[访谈]

The Stupid Fun 俱乐部见闻
*By Alexandra Weber Morales*
模拟城市系列游戏的创作者 Will Wright 和著名的制片人 Mike Winter 加在一起，会发明出什么样的机器人？且听 Alexandra 主编道来。

[产品评论]

新品：DtSearch 6.3 等
*By Rick Wayne*

重点：敏捷工具
*By Johanna Rothman*

[会议展望]

第 17 届软件研发大会
*By Nicole Garbolino*
引人瞩目的业界盛会，除了 Jolt 大奖将会揭晓之外，我们还会听到 Steve McConnell 讲述 10 年来软件构造的进展，Ivar Jacobson 讲述用例和 AOP 的结合。此外，大会将请到多位对整个软件行业有重大影响的开发界先驱，回顾 20 年来业界的发展。

【专栏：管理】

项目管理：PERT——敏捷的先驱
*By Robert C. Martin*
学会在小的软件项目中如何运用已经有几十年历史的基本技术。

职业发展中心：字斟句酌
*By Larry Runge*
如何在短短的简历中反映你自己的实力，给招聘经理留下清晰印象？

【专栏：管理】

职业发展中心：简历中的避讳
*By Larry Runge*
懂得在简历中哪些该说，哪些不该说，是非常重要的。

行为性需求：Persona 的力量
*By Matt Stephens*
单纯依靠用例，并不能很好地解决用户界面、人机交互这一类的问题，我们可以结合强大的 Persona（角色），这是 Alan Cooper 发明的概念。

【专栏：技术】

技术中心：J2SE 1.5 特性
*By Jeff Langr*
为了不在与 C# 的竞赛中落后，Sun 吸收了一些 C# 的特性。系列文章之一

横切：常见的误解
*By Gregor Kiczales*

敏捷前沿：打破宿命
*By Scott W. Ambler*
有时候，那些困扰项目的“不可避免”的约束其实并非不可避免。

编程艺术：拼接
*By Robert C. Martin*

2004 年 3 月

[专题：数据库编程]

SQLite 数据库引擎
*By Michael Owens*
当简单的 B 树程序库不够用，而大型数据库又有些大材小用，该怎么办呢

数据库可移植软件
*By Michael Lutz 等*

RTL：关系模板库
*By Oliver Schoenborn*

STL 容器中的严格所有权
*By Michael Perry*
动态分配对象（DAO）相关的许多问题其实真正的根源是所有权。

异常安全与容器
*By Laurence Marrie*
“Copy, modify and swap”的实用替代解决方案。

XML SAX API 的 C++ 模板包装
*By Yingjun Zhang*
使用 SAX 的时候，代码中很容易出现大量的重复。来自波音公司的华人工程师为我们展示了一种高效的通用方法。

互操作性与 C++ 编译器
*By Joe Goodman*
Intel 编译器实验室的工程师 C++ 编译器实现互操作的秘密。

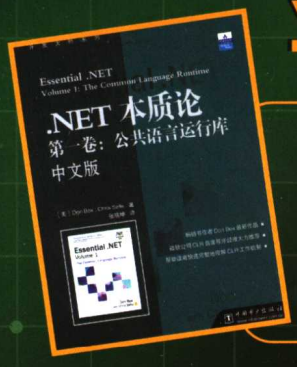
【专栏】

持续集成：D 与 Java
*By Matthew Wilson*
D 语言与 Java 语言的映射。

对话：Impaired Programming
*By Hyslop & Sutter*
std::pair 使用的中庸之道

# 中国电力出版社北京开源智业科技有限公司

# 外版精品　最新登陆！




## .NET 本质论

**作者：**（美）Don Box, Chris Sells　**张晓坤 译**　**定价：**48.00 元

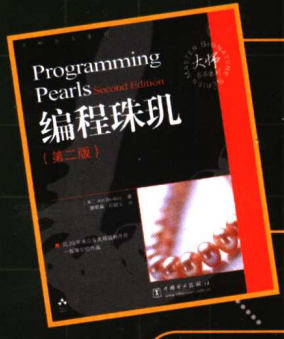
本书为需要充分利用Microsoft.NET强大功能的开发人员提供了详尽的信息，深刻地论述了.NET Framework（.NET 框架）的精髓：公共语言运行库[Common Language Runtime (CLR)]。Box 和 Sells 揭示了 CLR 的内部工作方式——CLR 设计背后的基本原理，它能够解决的问题，以及 CLR 编程中类型的角色。并且，在帮助读者在对 CLR 工作机制有了更为完整的理解的同时，指导他们如何利用 .NET Framework 构建更好的应用程序。


## 测试驱动开发

**作者：**（美）Kent Beck　**孙平平等 译**　**崔凯 校**　**定价：**28.00 元

**喜获最新美国 Jolt 大奖**

本书是软件开发方法泰斗、极限编程（XP）的创始人 Kent Beck 的最新力作，在亚马逊网站上持续热卖，是 Addison-Wesley 出版公司著名的大师签名系列图书之一。测试驱动开发（TDD）是 XP 的重要特点，它以不断的测试推动代码的开发，既简化了代码，又保证了软件质量。本书从头到尾跟踪介绍了两个 TDD 项目，描述了程序员容易上手同时又能大大提高工作质量的技术。在涉及 TDD 最有特色的模式和重构时，后面都附有例子。将侧重点放在灵活的方法和快速开发的策略上，肯定能激发读者接受这些未被充分利用但功能强大的技巧。

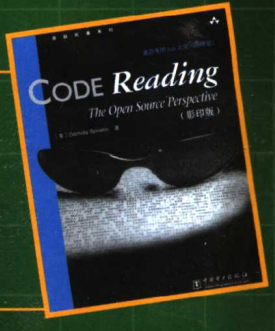



## 编程珠玑（原书第二版）

**作者：**（美）Jon Bentley　**谢君英 石朝江 译**　**定价：**28.00 元

**澜缪™ 名著作的最新版本，众多计算机大师均在其著作中反复引用本书并致推荐。**

本书是作者编程经验的结晶，由发表在ACM杂志上的专栏文章构成，每一章内容相对独立，但都是编程过程中的有机组成部分。本书内容包括问题定义、算法、数据结构、程序验证与测试、程序优化与效率问题，以及这些技巧在排序、查找和字符串处理等方面的几个实际应用。




## CODE Reading　影印版

**作者：**（美）Diomidis Spinellis　**定价：**49.80 元

**如果你是个程序员的话，那么你就需要本书。**

**入围 2004 年美国 Jolt 大奖作品**

阅读代码有着它自身的技巧，并需要我们能够在重要场合对采用何种技术有着判断能力。在这本不可或缺的书中，Diomidis Spinellis 使用了超过 600 个来自现实世界中的例子来向我们展示如何来鉴别好的（或坏的）代码：如何去阅读它，从中去找寻什么，以及如何利用这种技巧来提升我们自身编写的代码的品质。

记住这个事实：如果我们养成了阅读好代码的习惯，我们就能写出更高品质的代码。


**电力出版社有奖征集书评活动，详细内容请查看** http://www.infopower.com.cn


中国电力出版社北京开源智业科技有限公司　地址：北京市西城区三里河路6号（100044）　电话：010-88515918

传真：010-88518169　E-mail:ky_market@cepp.com.cn　网址：www.infopower.com.cn

# 最佳实践

最佳实践本身是一个外来词——译自“Best Practices”，好在译名基本上可以顾名思义；它的历史似乎并不长——Wikipedia把这个流行语的根追溯到了上个世纪80年代的商业管理和专业图书，特别提到了虽然捏造数据却成为经典的Tom Peters名著《追求卓越》；这些年来它一直很时髦——google上搜索中文“最佳实践”，结果有25万条之多，英文更是达到664万。

最佳实践总是一些基本的比较稳定的核心做法，经过前人大量的摸索，值得后人遵循和效仿。当然，一切最佳实践都有其适用的天时、地利与人和。自古而今，某个行业或者技术、方法总结出最佳实践的时候，也应该是它步入成熟的开始。随便举一些例子好了：围棋中的定式；茅台酒和可口可乐的配方；泼墨山水；三相输电线；建筑模式；七律五绝，和曾经的八股文；红绿灯、立交桥；武术套路……

相比之下，软件研发界现在谈论最佳实践多少还有些底气不足。Fredrick Brooks的经典论断“没有银弹”这把达摩克里斯之剑始终悬在我们头上。也许，这正是本期专题文章有些形散（至于神散不散，留给读者评判了）的主要原因。我查了查《Dr. Dobb's Journal》的老底，将“Best Practices”作为杂志的专题居然是今年的首创。原版一月份那一期的文章里，能够与专题拉上关系的只有《用XP和敏捷实践开发关键任务程序》和《激进重构》两篇。显然，这个专题对于DDJ来说也是一个探索。

提到最佳实践，很自然就想到了Steve McConnell。这当然首先是因为他在《IEEE Software》当主编（1999~2001）之前曾经为杂志撰写过两年半的专栏，名字就叫“Best Practices”。而且在他的名著《Rapid Development》（中译名《快速软件开发》）中也有一部分共26章，名为“最佳实践”。在刚刚举行的SD West 2004大会上，他发表了名为“Code Complete 2：软件构造10年以来的进展”的主旨演讲。（本次大会的精彩内容我们将在近期向大家介绍。请特别关注。）

这么算起来，圈子里提最佳实践也已经有不少年了。Steve McConnell显然是乐观派。他曾经在1999年回应过Brooks的经典论断，当时他比较了1968年NATO会议以来业界取得的许多进展，包括对goto语句的重新认识、结构化设计与分析、软件度量、面向对象、信息隐藏、评审与审查、增量开发、版本控制、基于构件的开发等等，认为已经总结出来的软件开发所必需的知识已经从1968年的一小部分，发展到1999年的75%左右。在这一次的演讲中，他更将测试为先的开发、重构、开源软件、以VB为代表的图形化编程、标准库的广泛使用等作为10年来的显著进步。如果沿着1999年的思路，我们的最佳实践集合大概超过80%甚至85%了吧。值得注意的是，McConnell并不是敏捷方法的热衷支持者，注重实际的他更愿意把XP之类方法看成是应用范围有限的一组最佳实践子集。在演讲中，他列出的2000年以来的最差实践就包括“没有充分了解就使用XP”、“把什么都叫做敏捷”、“在计划时就考虑以后要重构”等内容。

软件开发的最佳实践总的来看仍有不少争议，还在演进之中。然而，即便如此，正如Brooks观察到的，软件开发中最佳实践与平均实践之间的差距比其他行业都要大得多。McConnell也从自己的咨询经历中痛苦地发现，许多已经证实有效而且易行的最佳实践，偏偏在实践中得不到多少应用。北美的情况尚且如此，我们国内就更是难免了。从这一点来看，也许把功夫花在普及最基本的最佳实践上，比在最前沿、最具争议性的理念、问题上纠缠，更有现实意义。

2004年3月

附记：得到准确的消息，《Windows Developer Journal》的纸面杂志已经正式停刊了。技术刊物总是随着潮起潮落，想想JavaWorld改为纯网站经营后又全部停顿，在这几天才恢复的经历，我们也只能以去留随意的平常心视之。当然，令人欣慰的是，DDJ将扩大自己在Windows和.NET方面的覆盖范围，能够弥补一些缺失。

由于邮局服务质量和通信地址的问题，不少读者不能及时收到杂志，请通过reader@ddjchina.com与我们联系。此外，我们的网站即将开通一些额外的读者服务，请读者到www.ddjchina.com进行注册。

Dr. Dobb's JOURNAL CHINA www.ddjchina.com

# 软件研发

# 目 录

## 软件技术

# Dr.Dobb's JOURNAL CHINA www.ddjchina.com 软件研发

## Feature: Best Practices



## Column



## Special:Tales of Terror



## Management&Engineering



## Technology




执行主编：刘江
责任编辑：贾梅
编　　辑：吴怡 陈剑匡
编辑信箱：editor@ddjchina.com
美术编辑：锡彬
排版制作：金浩
网　　站：周明涛





**读者服务**
电子邮件：reader@ddjchina.com
电　　话：88379061 88379062
网　　站：www.ddjchina.com

**版权登记号** 图字：01-2003-1691
**图书在版编目（CIP）数据**
Dr. Dobb's 软件研发. 第 8 辑 /（美）马丁（Martin, R.C.）著; 刘基诚等译. —北京: 机械工业出版社, 2004.6
ISBN 7-111-14149-0
Ⅰ. D… Ⅱ. ①马… ②刘… Ⅲ. 软件研发
Ⅳ. TP311.52
中国版本图书馆 CIP 数据核字（2004）第 017989 号

**出版发行**
机械工业出版社
（北京西城区百万庄大街 22 号 邮政编码 100037）
印刷：中国电影出版社印刷厂
版次：2004 年 2 月第 1 版第 1 次印刷
889mm × 1194mm 1/16 · 7.5 印张 16 彩页
定价：18.00 元

## 第14届Jolt大奖揭晓

获得大奖的有：

**"图书：通用类"**《Waltzing with Bears》（与熊共舞），Tom DeMarco和Timothy Lister（Dorset House出版）

**"图书：技术类"**《Test-Driven Development: A Practical Guide》David Astels（Prentice Hall出版）

**"变更与配置管理工具"** AccuRev 3.3.1（AccuRev）

**"语言与开发环境"** Eclipse IDE & Framework 2.1（Eclipse，开源）

**"库，框架与构件"** Hibernate 2.1（Hibernate，开源）

**"项目管理工具"** V1: XP 1.0（VersionOne）

**"网站与开发人员网络"** IBM developer Works（IBM）

而Macromedia的Dreamweaver则荣登名人堂。

同时宣布的还有2004年Dr. Dobb's程序设计杰出奖。《C/C++ Users Journal》的现任高级编辑P.J. Plauger因为在C和C++语言方面的开拓性贡献（开发了业界领先的C/C++标准库，标准化工作，经典著作，众多技术文章）获得此项殊荣。这也是继Stepanov之后，C/C++界再次获奖。

## 趋势：开放、融合、集成

本月，SAP开始力推其下一代综合集成与应用平台NetWeaver。NetWeaver基于Web服务，除了作为SAP公司所有产品和解决方案的技术平台以外，还可以供客户使用。新平台能够与Microsoft .NET和IBM Websphere互相操作，以开放标准为基础，容易扩展。除了ABAP语言之外，也同时支持J2EE。与Microsoft、Sun和IBM的策略类似，以后只要购买或者升级SAP的产品，也就得到了NetWeaver。

而BEA方面也传出了正在酝酿一项旨在推进面向服务架构（service-oriented architecture）的计划Project Sierra，包括用户培训、产品改进和技术指导，目的当然也是为了在Web服务集成和应用平台的竞争中立于不败之地。

也是在本月，嵌入系统领域内称雄多年的Wind River公司宣布，其旗舰产品新版本VxWorks 6正在向Linux靠拢，已经实现了与后者类似的进程模型，以方便开发人员移植应用程序。而且，该公司的开发工具Tornado也已经被基于Eclipse的Wind Power所替代，同时支持Linux和VxWorks平台上的开发，而且有免费授权形式。

这些举动意味着，平台之间的融合已经成为大势所趋，各大软件厂商纷纷开始实行整合资源，准备在平台之上以解决方案、以服务决胜。在此趋势下，面对IBM和开源社区要求Java开源，并与Linux进一步融合的呼吁，Sun该何去何从？

## Verilog 20年

目前非常流行的电子系统设计与描述语言Verilog HDL已经度过它的20岁生日。1984年，Gateway Design Automation公司的Phil Moorby设计了这种用软件方式来设计、仿真、验证和实现芯片的语言。1989年Verilog随着Gateway被Cadence公司收购而成为后者的资产。1993年，语言的版权捐献给了IEEE。1995年成为IEEE标准1364，最新版本是2001版。目前工作组正在从事2005版标准的制定工作。按照IEEE的统计，现在光是销售Verilog工具的公司就超过了100家。

## Draper奖揭晓

美国国家工程院近日宣布，2004年度的Charles Stark Draper奖授予Alan C. Kay, Butler W. Lampson, Robert W. Taylor和Charles P. Thacker，表彰他们在20世纪70年代中期，开发了世界第一台实用的联网个人电脑Alto。这是该奖项继1993年Backus（FORTRAN）、2001年Vinton Cerf等（Internet）之后第3次颁发给计算机界，此外获奖的7项发明还包括集成电路、通信卫星、涡轮喷气发动机、GPS、光缆等等，都是赫赫有名的重大成果。Draper奖于1988年为了纪念惯性导航之父Charles Stark Draper设立，旨在增进公众对工程技术的理解。奖金高达50万美元。

实际上，这更可以看作是对施乐PARC众多研究成果的一种肯定。Alto也的确不愧为著名脑库的代表，它凝聚了PARC计算机研究部门的创始人和灵魂Robert W. Taylor（他以神奇的个人魅力在并不以计算机为主业的PARC中短时间内聚集了众多计算机界的精英，并取得了大量成果）和Alan Kay（Smalltalk之父，现为惠普公司的高级名士）、Butler W. Lampson（1992年图灵奖得主，Microsoft公司杰出工程师，设计了两阶段提交协议、Tablet PC等）等大师的心血。虽然产品本身在商业上并不算成功，但是它却成为今天所有PC真正的先驱，融合了当时PARC出品的鼠标、图形界面（WIMP）、局域网等最先进的技术，等等。如果说Lampson是Alto的系统架构者的话，那么另一位获奖者Charles P. Thacker（现为Microsoft公司杰出工程师）就是真正的幕后英雄，是他负责了硬件的设计、微内码的编写，甚至产品的生产和包装。你能想像吗？在30年前，Alto的黑白屏幕上就能同时画图、输入文本、显示动画甚至联网了！有趣的是，当一位公司领导前来视察的时候，他们居然说："这真的那么好吗？那IBM为什么不干？"

当然，本月带给PARC同人们的，并不只是欢乐。3月4日，PARC的创办者George Pake在家中病逝，离80岁的生日还有不到一个月。Pake是毕业于哈佛大学的核物理学家，对计算机知之甚少。他的卓越贡献来自其低调、宽容的管理风格。据PARC的前雇员回忆说，Pake很少干涉具体的研究，他只是提供良好的氛围——他每天总是在园区里走来走去，和员工随意交谈。虽然许多PARC的成果更多是造福全人类的，但是施乐公司的这笔投资并不亏本，仅仅激光打印机发明一项就带来了滚滚财源。就是本身并不挣钱的Alto，也为后来颇为成功的Star工作站奠定了基础。

（更多新闻和评论，请访问www.ddjchina.com）

# 软件研发恐怖故事

想像我们露宿在旷野，月黑风高，远处似乎有狼嚎，隐隐约约……围拢过来，围坐在篝火旁，让手电筒投射出幽灵般的阴影。我们将一同经历各种令人毛骨悚然的恐怖故事……

在软件研发的历史上，有许多引人注目的失败案例。比如著名的美国联邦航空局空中管制项目，伦敦急救中心项目，等等。最近，《Software Development》邀请杂志的特邀编辑、顾问以及一些热心读者，讲述自己在软件研发中亲身经历的一些恐怖故事。这其中，有的是因为经理的指导无方，有的是出于技术风险，有的则完全是整个团队的无能。感谢他们的故事，感谢他们的勇敢，他们中的大多数人都无畏地署上了自己的名字！而我们，在一边偷着乐，一边看热闹的同时，也不要忘了做一些思考——不要在自己下一个项目中重蹈覆辙！当然，也欢迎广大读者朋友踊跃讲述你类似的亲身经历，也许我们可以为此设立一个专栏。不要担心，你完全可以匿名。请写信到 contribute@ddjchina.com。

# 这样的敏捷

## 强求一致，扼杀了真知灼见

他们邀请我作为顾问帮助一个新项目取得成功，此前，他们的三次尝试都以轰轰烈烈的失败而告终。他们需要的是在迭代地交付软件方面有经验的人。开发经理知道小组的长处和短处，而开发小组也对学习新方法跃跃欲试。但是，在业务经理发表热情洋溢的讲话之前，我其实并不真正了解自己将面临怎样的困难。听听他是怎么说的："是的，我们将进行迭代式开发……在此获得了所有需求以后，马上就开始！"

需求总是会成为避风港。这个小组曾试图将所有需求都夯实。他们雇了9位业务分析师，花了6个月，搞出来一份300页的业务需求规格说明书。当我第一次看到这厚厚的一摞东西时，几乎要晕倒：它洋洋万言居然没有涉及到一点与系统使用相关的需求。每个所谓的"需求"，都是由一个数据流图和一页或者多页叙述一个关系数据库系统 CRUD 功能[1]的文字组成。是的，我知道，这正是功能组的经理们做需求的一贯方式——为什么要改变呢？开发经理甚至在我加入之前，已经多次要求过提供用例，但是他得到的只有数据流图和数据模式。

他和我联手推动用例问题的解决，并取得了一个小小的胜利：我和分析师们合作，找出了系统的主要用例，大约20个。分析人员和系统的目标用户开始编写用例了，而开发经理和我则整理出一份项目计划，准备立即根据最高优先级的用例启动开发工作。可是我们马上就撞到了另一堵南墙上：功能组的经理要求，在所有用例编写好、经过评审和批准之前，一行代码都不能写。

我参加了一次项目人员会议。除了开发经理之外，没有一个人对开发小组依然在原地踏步，尚未写出一行代码表示关注，业务经理反而对功能组经理大加称赞，表扬她赶上了规定的最后期限：他们又拿出了一份300页的文档，其详细程度简直让人难以置信——描述了数据库管理系统中每个表的每个数据字段和数据类型。可是我们连需求都没有，连稳定的领域模型都还没有呢！业务经理的赞扬热情洋溢，他最后是这样结束讲话的："现在他们已经出色地完成了项目中最大的任务。"等到该我发表状态报告时，我说，自己完全同意开发经理的严重关注，最后还加上了一句："我对项目有一点意见，想跟大家说一说，在一个迭代项目中，并没有什么大任务的说法。"可是，只有开发经理理解我在说什么。

## 削足适履

后来怎么样了呢？开发小组和功能小组之间的冲突愈演愈烈。业务经理选择了符合自己个性的冲突解决方法——命令每个人必须团结一致。这其实是意味着，开发小组的一切行动都要得到功能经理的批准。我私下里问业务经理，在一个多层 Web 软件的开发中，功能经理的权限应该是什么。他同意应该没有什么权限，可是他继续暧昧地认为："如果我们能有共识，就一定会成功。"

公司随后决定，加入一个专门的项目经理，以减轻开发经理的负担。当通过电话与他交谈的时候，我问："你的迭代周期一般是多长？"他不假思索地回答说："噢，大概6个月吧。"大概在电话另一端我的沉默让他有些措手不及，他马上问我同样的问题。我的回答是："不超过3星期。"电话另一端也是沉默。开发经理和我都建议不要把他放到这个职位上，可是业务经理还是雇佣了他。

此后不久我就离开了这个项目。公司觉得小组已经可以自己继续进行了。两个月之后，开发经理也离开了，大概也忍受不了这种无聊环境吧。项目后来还是没有逃脱失败的噩运。在我离开以后，早期迭代得到的分析和设计模型都被放弃了，而技术人员却增至三倍。最开始制定的进度被100%地超过，而所交付的系统却完全不能伸缩。他们根本没有进行迭代式开发——他们自始至终都没有获取所有的需求。

为什么会这样呢？开发经理总结得好："什么叫作愚蠢？就是又一次重蹈覆辙，却期望着这次的结果会有所不同。"这个项目（以及我见过的许多其他项目）的致命缺陷在于，不管条件如何，环境怎样，谁对谁错，盲目追求一致。让业务人员来规定软件开发应该如何进行，显然会南辕北辙。为什么人们会做出这样显然不合理的决策呢？这个项目不是由结果驱动的，相反，它是由一两个绝对无法得到光明正大支持的假定驱动的。因此它又成了一个损失数百万美元的事故。这样的故事仍然在不断发生着——可以说是太频繁了。我也只能这么想了：可别拿这种方式处理自己的家庭。

*Gary Evans*
*Evanetics 公司的创始人*
*《Software Development》杂志特邀编辑*
*《The Rational Edge》杂志撰稿人*
*Jolt 大奖评委*
*南卡来罗纳州，Columbia 市*

译注1：CRUD指的是创建（Create）、复制（Replicate）、更新（Update）和删除（Delete）等数据库必备的操作。

# 注释的缺失

## 前人的暗地使坏使新开发人员不得不选择离开

我始终相信原始的项目源代码中，对上上下下的每个模块都必须有完整的注释和定义，这样才能使维护和修改尽可能简单，既对我自己有好处，也能为以后的开发人员造福。正确的做法是，每个模块都应该有对数据输入和输出的解释，所有外部和内部变量的定义，模块如何完成任务的完整解释，以及任何有助于使代码的维护工作更简单的其他信息。数据字典和所有例程的列表、库文件和版本应该放入程序中，还有谁编写了哪些代码的历史记录。

我曾经作为一名程序员为一家软件公司工作过很短的一段时间。作为一名新雇员，我承担了清理遗留代码的平凡工作。大多数比较新的代码注释都比较好，深刻体现了许多标准实践。不幸的是，当我被派去清理较老的代码，并添加一些新的错误检查例程时，我才发现，这些代码已经被一个相信工作的安全比职业道德更重要的家伙修改过了，他删掉了所有的注释。直到今天，我仍然无法理解为什么这个家伙会干出这样恶毒的事情来。此后我很快离开了这家公司，主要是因为，我感到如果自己连同事都无法信任的话，我是不可能有什么太大成就的。

*匿名*
*佐治亚州，亚特兰大市*

# 无常的窗口

## “这不是 bug，这是一项特性！”

最近，一个公司邀请我就软件的质量问题做一个报告。为了做一些准备，我安装了他们的零售产品（该产品他们每年都能向消费者卖出几百万份——你肯定听说过它的名字，而且很有可能你还使用过），进行简短的测试。

试用这个程序的过程中，我注意到其中各个窗口的关闭方式五花八门。因为一致性是我准备讲述的主题之一，我开始把这一问题记录下来。大约 15 分钟之后，我已经找到了 10 种各不相同的窗口关闭方式，包括 5 种不同的按钮文本，3 个不同的文字颜色，和更多按钮的位置，我都记不清有多少个了。在做报告之前，我同几位工程管理小组的成员交谈了一下，顺便提到了 10 种我碰到不同的关闭窗口的方式。小组的几位成员表示非常惊讶，可能也有些生气。这时候其中有一位老兄冷冷地告诉我，我搞错了。“实际上应该有 11 种，”他说。“我们昨天刚刚开了一次会，又新定义了一种。”

*Scott Meyers*
*咨询，《Effective C++》，《More Effective C++》和《Effective STL》的作者*
*《Software Development》杂志顾问委员会成员*
*俄勒冈州，West Linn 市*

# 大相径庭


## 谁为糟糕的设计负责？

1994年6月，我新任首席分析师和程序员，工作是支持一个大型非盈利机构的捐献处理和捐献者跟踪系统。在我履新之前几星期系统已经投入运行，前任架构师当时已经准备另谋高就，在这个组织的最后一两个星期里，他站好最后一班岗，负责培训我。

在我的第一个星期，星期三早上我很早就来到办公室，把背包放在我空空的椅子上。我的前任摇头晃脑地在各处转悠，一位有些神经兮兮的运营监督跟在他后面。“Allen，你真应该好好弄明白这些东西。”他说。我跟着他走了出去，此后很长时间都没有回办公室。那天，连妻子精心为我准备的午餐我都没有机会享用。

系统的前端由两个古老的支票处理系统组成，可以对来自捐献者的支票进行缩微拍摄并背书（endorse）。机器每天都要记录几万张支票，并将捐献者的ID和支票的捐献金额数据按批次分成100组左右的事务进行处理，并转发给一台PC。然后PC上启动一个批次导入程序，读取文件，在主系统中生成事务。当事务输入到系统中时，程序为每个事务生成一个键码，其中每一批的捐献日期都是键码的一部分。

支票处理机不是太智能。每天早晨，需要有人手工输入公历格式（一种将每年的日期从001到365编号的格式，比如1/1/1999就应该写成99001，而12/31/1999应该写成99365，显然，设计者当时没有考虑到千年虫问题）的当天日期。每个支票处理器然后使用这个数字作为当天处理的每个批次的捐献日期。

可是系统中隐藏着一个问题：一天早晨，一位职员把日期输错了，本来应该是星期一，结果弄成了星期二的日期。批次导入程序没有进行日期验证，并愉快地用星期二的日期作为惟一ID生成了所有事务。到了星期二，这回输入正确了，却引起了大麻烦：这一整天的事务在导入系统时，将前一天的数据全部覆盖了。到了星期二的下午，机构里负责运营的经理们发现出了问题，有几千个事务的捐献数额和账号ID发生了变化，批次的收支平衡报表无法平衡，似乎有一整天的数据都无缘无故地消失了。到了周三一大早，他们找出了问题所在，期间那个神经兮兮的运营监督无数次地跑到大楼的尽头去找我的那位同事。

作为恢复行动的负责人，我经受了“炮火的洗礼”，深入到系统的内部运行机制中。系统关闭了两天，与此同时我们用持续三天的备份对系统实施了回退操作，修改了错误批次中的日期，重新将它们导入到系统中，修改概要表和其他信息。最后，这个错误总共造成了超过100名数据输入人员两天的人力损失。

我给自己定下了任务，要修改批次导入程序，防止此类错误再次出现。我注意到它根本没有数据检查。于是去问我的前任，为什么会这样。他回答说：“因为我们没有时间，程序编写得非常匆忙，是在系统投入运营之前很短时间内赶出来的。”

好嘛，这个架构师被大家当作英雄礼送离开：是他开发了这个系统，设计和实现都取得了很大成功。他还领导开发团队在非常紧张的进度下克服万难让系统运行起来。而另一方面呢，那个愚昧的职员因为错输入一条数据，最终被解雇了。

可是，我要问的是，到底是谁犯了错呢？是那个在五个字符的输入数据中弄错了一位数字的职员呢，还是那个设计的系统允许出现错误使整个机构的运营卡壳的人？他们都只犯了一个错误，但是结果却大相径庭。

*Allen Vander Meulen*
*弗吉尼亚州，Alexandria 市*

# 狂热的度量

我曾经从事过一个大型网络管理软件的开发。当时我们的经理在公司方法小组可怕的软件度量研讨会上被“洗了脑”。我们最担心的事情发生了，因为参加者几乎没有能够毫发无伤，全身而退的。无一例外，他们都带着足够成为自己小组危险分子的满脑子理论知识回来了。果不其然，他开始不断地重复一句话：“无法度量的，也就无法控制，无法度量的，也就……”——最恐怖的噩梦成了现实。这句魔咒让我们一个个簌簌发抖，我们知道，一个本来很好的概念已经被扭曲为罪恶的帮凶工具。

在落入方法小组魔咒的一个星期以内，不出所料，我们的经理编写了一个小程序度量我们的生产率。他选择的度量指标是非议很多的源代码行数（source lines of code count，SLOC）。每个星期，他都会运行他的宝贝程序，得到我们生产率的一个基准值。然后，用这个基准值对我们的表现开始进行逐一考核。如果有谁完成的行数低于这个基准值了，就会被叫到魔窟——他的办公室去解释清楚，为什么生产率下降了。如果你超过基准值，可能受到表扬和奖励。

有一位老兄，我们都称他为编码器，也就是coder，以快速“剪切-粘贴”的工作方式而著称，很自然地成了我们中生产率最高的程序员，并且因此得到了两张职业冰球联赛的门票。与此形成对比的是，一位善于将大量重复代码整理合并，绰号“美学家”的哥们儿，却被叫进了经理的魔窟，因为那个星期他的高超合并技术使自己的代码行数低于了基准值。等他再回到工作岗位上的时候，整个人都变了。

我们都懂得必须自我保护。否则，“美学家”的今天就是我们的前车之鉴，弄不好奖金都会泡汤。不久我们发现了一个秘密，这位“度量超人”——我们的经理所写的那个魔鬼程序，是通过计算分号的数量来统计源代码行数的。很快，所有语句都变成使用双分号来结束：

```
Alarm a = Alarm.mkAlarm(cos);;
a.ack();;
```

我们的经理洋洋得意——他深信自己对度量的有效运用提高了小组的生产率。而我们则不断探索极限，最酷的一位甚至用分号来当注释分隔框的符号：

```
/*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
这里是注释
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;*/
```

很快，我们的小组作为有效运用度量极大提高了生产率的榜样受到了公司的表扬。我们的经理也为我们感到自豪……直到我们的弄虚作假在代码审计中大白于天下。

这个故事的教训是：选择度量指标的时候必须小心，特别是怎样理解它，怎样使用它。将度量指标和程序员的待遇挂钩，还有什么比这更能使度量指标变成罪恶的工具吗？忘记这条教训，你就可能遭到“度量狂热”的迎头痛击。短暂地沉浸在表面的生产率提高的幸福中之后，你很快因为发现自己选择了错误的度量指标，或者错误地解释了它，而陷入打击和惊恐之中。请谨记这一点。

*Steve Adolph*
*高级技术顾问*
*WSA 顾问公司*
*加拿大不列颠-哥伦比亚省，Britannia Beach 市*

# 代码大小问题

## 关系学比数学更强大

几年前，有三个不同背景的小组接到一项任务，要求开发一个算法，在比较短的规定执行时间内解决一个问题。我的小组成功地开发了最精确而且最快的解决方案。可是甲方又增加了几条新的规约——我们很快满足了这些要求。然后项目经理说，因为硬件内存的局限，代码的大小必须少于25KB。这时候我们的可执行文件大小是19KB；而最接近的竞争对手的程序可执行文件大小是47KB。与此相反的是，我们的源文件大小是103KB（注释很多），而他们的是80KB（几乎没有注释）。然而，奇怪的事情发生了：因为那个小组认识甲方的领导，他们使管理层相信，根据源代码的大小，他们的程序更小！

虽然我们力辩二进制文件才是真正有意义的，而这是最简单的数学问题，可是甲方不为所动。最后，关系学取得了胜利，他们把次等的产品推向了市场。

*Mukund Thapa, CEO*
*Optical Fusion 公司，加州 Palo Alto 市*

# 24 小时连轴转

## 对没有脑子的项目经理，就该将计就计

我们曾经在一个记账系统的实现中陷入了危机。眼看着最后期限即将迫近，可是项目还有很多工作没有完成。项目经理在白板上画出了项目的最后五天，将每天分成6个四小时为单位的时间段。然后他根据小组之间的依赖关系，把每个时间段指派给一个小组。最后，预计开发小组在一个子夜 12:00 到凌晨 4:00 的时间段中将应用程序投入运行。

我们发现其实这种安排解决不了多少问题，因此创造出一个称为“最后代码配置”的新任务，并伪称必须在应用程序准备运行之前完成。根据我们的建议，已经毫无头绪的项目经理将这个任务安排在从午夜到凌晨 4:00 的时间段中，这样，系统投入运行就得推迟到凌晨 4:00 到 8:00 了。实际上，我们在凌晨 7:30 才醒来，从宾馆房间拨号进入网络，运行部署脚本，让人看上去好像是晚上 10:00 还在工作，而且熬了一个通宵……

*Gregor Hohpe, 高级架构师*
*《Enterprise Integration Patterns》一书的作者*
*ThoughtWorks 公司，加利福尼亚州旧金山市*

# 条条大陆通罗马

## 三行奇异代码的三种解释

几年前，我志愿前去实施一个 Microsoft Access 前端程序的再工程。本来还以为自己终于接到了真正意义的大项目，可是我很快发现，自己好像是和法国外籍军团签了终身合同。当我披荆斩棘、筚路褴褛地在阴暗杂乱的代码丛林里艰难前行的时候，我发现 三行改变了我程序员生涯（至少是改变了我对程序员的看法）的代码：

```
    Forms!frmMain!txtValue.Enabled
= False
    Me.txtValue.Locked = True
    Forms("frmMain").txtValue.
Value = ""
```

每一行都涉及同一个窗体中的同一个对象，但是写法完全不同。为什么一个脑子正常的人会这样干呢？这个问题在我心中萦绕多年，而挥之不去，渐渐地，我为这种极限编程方式（当然是一种极坏的编程方式了）琢磨出三种可能的解释：

- 阴谋学说。因为担心遍布全世界的 Echelon 间谍网，程序员想尽可能地把自己的真实意图掩饰起来。实际上，没有什么语言系统能够解密这段代码，它看上去闻上去都很像 Visual Basic，但是以这样难懂的方式使用这种语言，最终代码变得完全不知所云了。
- 考古学说。就像几百万年以前曾经存在然后又消失的海洋，现在只留下了层层的沉积岩和化石，不同的程序员曾经修改过这个程序，每个都加了一层代码——因为他们显然没有弄懂前面一层代码的含义。
- 精神分裂学说。程序员受到了复杂的多重人格的困扰。可以看出，各种人格的切换速度是很快的：第一行是化身博士 Jekyll 写的，第二行是代身博士的化身 Hyde先生写的，而第三行，第三行应该是 Anderson 先生写的（他显然吃错了药）。

哪一种解释最合适呢？那得您说了算。比我更直截了当的人，可能会干脆地管这叫做暗中破坏。

*David Dossot, 软件工程师和架构师*
*法国 Thionville*

## 完美的工作

1990 年，当 DOS 恐龙还在统治世界，软件开发的工作还不好找的时候，我通过报纸上的分类小广告得到了一份工作。听上去他们在干的事情还不错：CAD 和绘图、三维建模甚至包括动画。研发总监和他的得力工程师都是富于热情、才华横溢、幽默风趣的人。（他们到现在还是我最亲密的两个朋友。）

管理层怎么样呢？等一会儿吧，我们等一会再说。研发总监他们给我演示了他们开发的软件。他们还给我看了一个类似 Microsoft Windows 3.0 beta 版的产品。我的眼睛都看直了。不知不觉中，我们谈话的口气就从“如果你能在这里工作，如何如何”变成“下个星期一你就开始怎么怎么样”了。

公司的总经理虽然在技术细节上爱犯了些小错误，但总的说来还是一个相貌堂堂、善谈的人，他不停地介绍“公司成长”、“新式营销”之类的事情。公司的股东是瑞士人，公司前不久刚刚改了名字。谈话中有些地方让我感到有点儿奇怪，可是谁还关心这些呢？他们正在开发伟大的产品，他们也许会创造历史！

他们同意聘用我了。我喜出望外，立即冲向他们的代码，冲向其他的工程师。哇，这些人真的很棒！我想，我们将抢走 Autodesk 的午餐！

只不过……这个公司不止改了一次名字，而是四次或者五次了。公司的股东总是把公司资产转移到瑞士，然后将美国公司破产，再开一家“新”公司，以此来逃避债务。可是，谁管这些呢？“我们能开发出伟大产品就行了！”我想。

但是，股东还信奉一种奇怪的新产品开发过程——“守株待兔法”：等待好想法跑过，自入瓮中，然后把它们抓起来。这看上去真像是一个笑话。拿出一张纸：“这一条你需要么？”客户回答：“不。”在纸上涂涂画画，一把弄皱，扔掉，然后在拿出一张新的：“这个呢？”“不。”士气开始低迷。我们的 Windows 产品从来没有成为现实。

而总经理的所谓“新式营销”的理念原来就是通过冷冰冰的电话直销来卖我们复杂的 CAD 软件。在被几千个电话受访者“哼！”过之后，股东把他给开掉了。他通过一纸传真，将董事长告上了法庭。我们渐渐都开始害怕起传真机的声音来。

研发总监也另谋高就了。他的得力干将接替了位置。“办公室主任还在管账，”他说。“她很正直，只要她还在，就不会太离谱。”可是不久办公室主任就离职了。一天，我们去上班的时候，发现办公室的门上有一把模样怪怪的大锁，锁边上站着模样酷酷的警察，警察身边站着的是我们模样凶凶的房东。显然，有人已经告诉他我们公司改名的把戏了。

真奇怪，我们居然又改了一个公司名称。新的研发总监也离开了。我们的新办公室空空的，有 20 个格子不满。而胜利大逃亡还在不断升级，很快就只剩下我们四个人在那里发呆了。我继续呆了很长时间，甚至帮助公司招聘过新的员工，最后还是不得不离开。当然了，我们伟大的产品从来也没有成为现实。

*Rick Wayne*
*软件工程师*
*威斯康星大学麦迪逊分校*
*《Software Development》产品评论编辑*

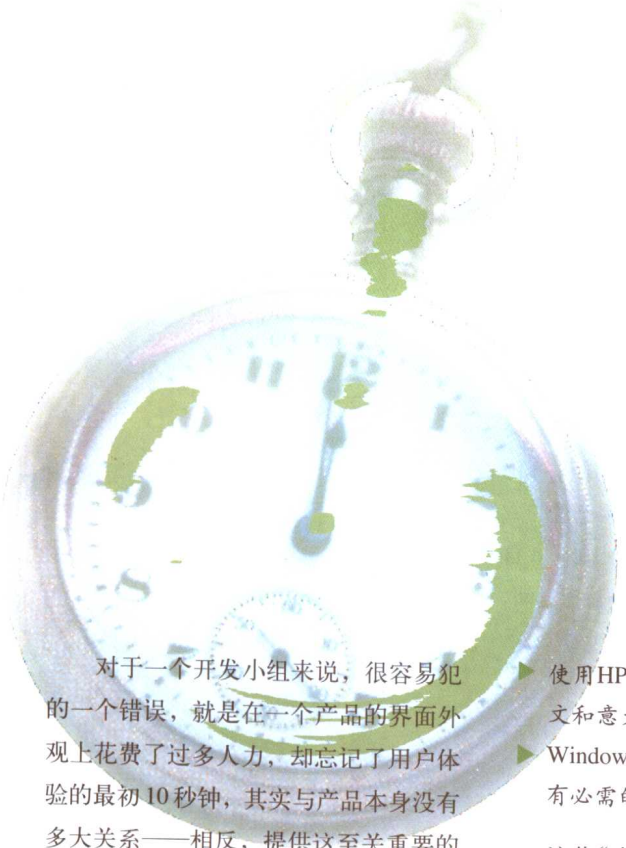
## 尖端的教训

我要讲的恐怖故事，是说好东西未必能成功——这样的事情太多了，我敢肯定每个人都曾经见过类似的 YYY 产品。

有一小组软件开发者，每个人的生产率都高于业界平均水平，他们经常讨论自己每天使用的各种开发工具如何如何不够用。很自然地，他们着手开始开发更好的工具。在此过程中，他们还创办了一家小的始创公司，完成并且商业化这些工具。他们的产品很出色，而且差不多已经按时间进度完成了。

几年以后，YYY 工具仍然活着，而且仍在周期性地升级。但是，它的市场份额太小了，几乎可以忽略不计，当然，也无法使投资获得什么回报。什么地方出了问题呢？市场营销太差了？超前的技术需要时间被人接受？非也。这些都不是问题所在。事后再来看，最大的问题其实在于，这些明星开发人员所开发的尖端工具只适用于很小的目标市场：他们自己。

*Clemens Szyperski*
*微软研究院*
*《Software Development》专栏作家*



# 最初的10秒钟
## ——安装工具大比武

产品的安装阶段经常被人忽视，但是对于用户而言这却是成败攸关的时刻。了解并使用三种顶级的安装程序，有助于我们改善至关重要的用户第一印象。

Roland Racko

对于一个开发小组来说，很容易犯的一个错误，就是在一个产品的界面外观上花费了过多人力，却忘记了用户体验的最初10秒钟，其实与产品本身没有多大关系——相反，提供这至关重要的第一印象的，是安装过程。

随着产品和所处环境的需求越来越复杂，一个产品或者服务的安装是预先设定好的用户体验要素。因为安装会影响最终对质量的评价和对产品的满意度，开发人员最为关切的问题应该首先是：你的安装过程怎么样？

安装过程并不仅仅和塑封盒装供公共消费的那些产品有关。任何规模的公司都要为多种桌面的软件构件、配置调整和服务升级开发漂亮的安装过程。想像一下这样的场景：经理要求你基于Java开发一个新的内部使用的搜索引擎，部署到公司每个桌面上。桌面包括 Windows 2000、Linux、HP-UX 和 Solaris。而且，和盘古开天辟地以来所有老板一样，经理轻描淡写地说，这个引擎需要有几个“小小的”细节：

- Linux 用户已经有一个beta版的搜索工具了，因此他们另外需要一个 UPS 包裹费用计算软件。
- 使用HP-UX 的人要求搜索工具有英文和意大利文的错误信息。
- Windows 2000 和 Solaris 用户可能没有必需的 Java 虚拟机。

这些“小小的”细节在有经验的开发人员眼中就会转换为长长的一串问题：

- 所有目标平台的部署体系结构是什么？
- 文件系统方面的细节——快捷方式、文件权限、执行权限、文件别名和分区该怎样处理？
- 每个桌面已经有共享文件吗？哪些平台需要用到注册表键？
- 每个桌面上现有（如果有的话）的 Java 虚拟机都是什么版本？
- 应该给每个用户什么样的图形化安装体验？
- 图形化安装体验应该怎样匹配搜索工具的界面外观？
- 程序的安装过程本身需要帮助文件吗？
- 初始界面有可选子组件（帮助文件，语言选择等等）供用户选择吗？
- 程序图标能够安装到桌面上吗？
- 初始界面需要搜索工具链接到桌面上或者桌面之外任何公司数据库、用户账号或者服务吗？
- 有办法知道以前安装了什么程序吗？怎么发布程序，CD-ROM，网络下载还是远程（可能是在后台进行）安装？
- 怎样组织所有平台安装过程的回归测试呢？

这些问题对于无论盒装产品还是内部使用，都是存在的。大多数公司开发小组都已经创建了某种自动化的构建过程来处理这些问题。但是在许多情况下，这种过程是自己定制的——结合使用从源代码控制系统收集源代码和二进制文件的批处理文件或者脚本，将它们集中，然后构建出产品安装程序来。但是，其中涉及的过程非常复杂，使我们很容易想到应该采用强大的工具处理这些问题方面。

## 竞争者

目前业界处于领先地位的安装程序（即打包工具）有三种，我们将对其进行全面的考查（参见文后的比较表）：其中两个是多平台的，可以满足前面我们设想的场景中跨平台的需求；另外一个不是多平台的，但是它能与 Microsoft 的 Visual Studio .NET 紧密集成。每种产品都具有其他竞争者所不具备的特性，但

是在解决前面提到的问题方面，就各自适用的平台而言，它们都很合格。Zero G公司的InstallAnywhere 6能够构建基于Java的安装程序，因此能够运行在装有Java虚拟机的任何平台。InstallShield软件公司的InstallShield MultiPlatform 5，与InstallAnywhere一样，也是一个Java程序，能够构建基于Java的安装程序。最后一个，Wise Solutions公司的Wise for Visual Studio .NET 5企业版，可以用来创建和编辑Windows安装程序(.MSI)包。说得更具体一些，它为Windows环境专门提供了一些功能，而这些是前面的多平台产品无法自动提供的。

为了评估这些产品的使用手册和IDE，我采用了一些经典的指标，它们最初是为评估语音技术产品而设计的，但是也可以应用于更广泛的任何产品（参见www.voicewizard.com/testing_usability.html）。有一些更严格的指标对于这些产品尤其重要：

- 肌肉事件数：一个需要眼球移动、手臂动作或者手指动作的操作（比如鼠标点击）算做三次肌肉事件。此指标越小越好。
- 无指示使用情况：不阅读使用手册或者使用帮助屏幕能够向前走多远。朝任务完成的方向越远越好。
- 记忆负担：开发人员要完成当前操作必须记住多少以前操作的信息。越少越好。
- 屏幕实用部分比例：屏幕专门用于当前任务的部分与其他帮助屏幕、提示或者此刻不需要的界面元素相比有多少。与任务相关的实用部分越多越好。
- 奇怪数：使用户无法理解的任何GUI设计都算做一次奇怪数。奇怪数越少越好。

根据这些指标，我们的三个竞争者的表现就显出不同来了。

## 各就各位 ……

一个安装程序自己的安装过程显然为产品提供了显示其水平（包括外观和实质内容）的机会。Wise产品本身的安装过程非常容易，但是它的最简主义方法却没有达到我们称为“安装程序艺术”的图形化或者用户体验标准。而InstallAnywhere呢，安装界面非常华丽，是开发人员值得推荐的表率。它为所要经历的各个步骤都提供了优秀的图形化反馈，还在安装过程中以非常稳重的方式提供了产品特性列表。InstallShield自我安装的方式介于前两个竞争对手之间，界面华丽，动静很大，但是缺乏InstallAnywhere信息丰富的特点。

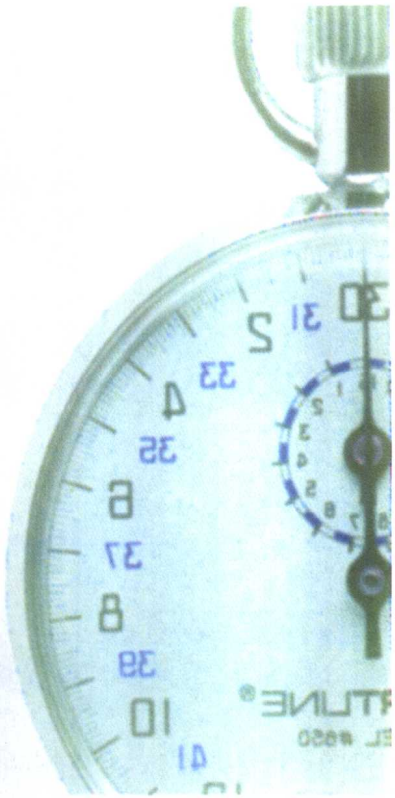
如果你以前没有使用过安装程序，我大力推荐你在实际运用之前先看一看产品的使用手册。安装包的构造是非常琐细的事情，不做一些准备，理解起来是非常困难的。

## InstallShield

InstallShield的使用手册在三个产品中是最臃肿的。有趣的是，它的帮助文件并不缺乏，而且很详细，篇幅也很大，有丰富的精心选择的超链接和大量细节的相关主题。问题出在安装程序IDE的用户界面设计上。

InstallShield的IDE表面上看好像是以对象/属性的模式为基础的，似乎过多使用了Java术语。根据这种模式，开发人员多多少少需要在脑子里组织实际的安装事件。构建安装包时，开发人员必须想着所有包中的对象，然后在多个窗口之间跳来跳去设置所有对象的属性，经常是通过下拉菜单。有些属性的下拉面板不太明显，因此开发人员必须用鼠标移来移去，才可能找到。这种活动的肌肉事件数是所有三种产品中最高的。

众多的帮助屏幕实际上使过程变复杂了，因为它们也是屏幕上不以任务为中心而且使人分心的东西。它使我想起了本人的用户界面规则第23条：“如果一个GUI需要大量帮助文件的话，那帮助文件也帮不了它。”InstallShield IDE还有一个快捷向导，奇怪的是使用手册中居然没有提到。这个快捷向导涵盖了基本的功能，但是在简化更多复杂安装的细节方面并没有做太多工作。要处理这些细节，还是得求助于对象/属性模式。



## Wise

Wise的使用手册相对好用一些，因为它是以任务为中心的。其实在循序渐进的过程中，犯错误并不容易，即使犯了错，使用手册也基本预料到了，提供了相关信息。

IDE本身比InstallShield的要容易理解一些，主要的驱动菜单面向开发人员构建安装包要执行的操作单元。但是，因为严格遵守Microsoft IDE布局和设计的