TURING 图灵程序设计丛书 程序员修炼系列

Addison Wesley

Code Quality The Open Source Perspective

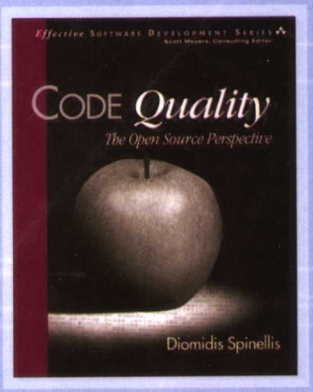
高质量程序设计艺术

[希腊] Diomidis Spinellis 著

韩东海 译

JOLT 17th Annual PRODUCT EXCELLENCE AWARD

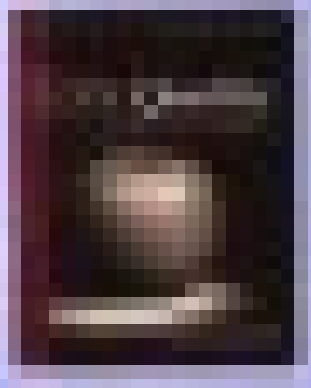
2007年
Jolt生产效率大奖得主



- 深入剖析著名开源软件的质量问题
- 全面阐述C、C++和Java代码中的常见编程错误
- 指导你编写优秀代码的圣经

人民邮电出版社
POSTS & TELECOM PRESS

高质量程序设计艺术



TURING 图灵程序设计丛书 程序员修炼系列

Code Quality The Open Source Perspective

高质量程序设计艺术

[希腊] Diomidis Spinellis 著
韩东海 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

高质量程序设计艺术 /（希腊）斯皮内利斯（Spinellis, D.）著．韩东海译．　—北京：人民邮电出版社，2008.1
（图灵程序设计丛书）
ISBN 978-7-115-16793-4

Ⅰ．高…　Ⅱ．①斯…②韩…　Ⅲ．程序设计　Ⅳ．TP311.1

中国版本图书馆 CIP 数据核字（2007）第 142555 号

内 容 提 要

本书论述阅读与编写软件代码的方法，重点讨论软件代码的质量属性，包括了软件系统的可靠性、可移植性、可用性、互操作性、适应性、可信性以及可维护性等方面。着力培养软件工程师了解这些属性的能力，并能编写出具备这些属性的优质代码。本书研究了来自于现有开源系统的真实示例，并提供了有意义的练习以巩固读者的判断能力和技巧，使用了统一建模语言来绘制所有图表。

本书适合各层次软件开发人员、管理人员和测试人员阅读。

图灵程序设计丛书

高质量程序设计艺术

◆ 著　　[希腊]Diomidis Spinellis
译　　韩东海
责任编辑　陈兴璐

◆ 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
三河市海波印务有限公司印刷
新华书店总店北京发行所经销

◆ 开本：800×1000　1/16
印张：25
字数：585 千字　　2008 年 1 月第 1 版
印数：1-5 000 册　　2008 年 1 月河北第 1 次印刷

著作权合同登记号　图字：01-2006-3691 号

ISBN 978-7-115-16793-4/TP

定价：55.00 元

读者服务热线：(010)88593802　印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

本书赞誉

“每一个软件工程专业的大学生都应该阅读本书，书中的所有内容应该成为每一位专业软件开发人员的必备常识。”

——Greg Wilson，多伦多大学教授，*Beautiful Code*一书主编

“阅读本书真是一种享受……字里行间无不洋溢着思想的光芒。如果你能用心理解书中精髓并在实践中加以运用，前途当不可限量。”

——*;login:*（USENIX协会会刊），Elizabeth Zwicky，国际系统管理协会前主席

“我坚信本书应该是所有程序设计专业和程序设计课程学生的必读之作。从事软件分析、维护和变更控制工作的人，应该将本书作为需要一读再读的圣经，直到真正领会和掌握书中的精髓为止。”

——Rob Slade，ACM comp.risks论坛

“即使你使用的语言与书中讲述的不同，仍然能够获取许多有关编写、阅读和测试代码的思想火花。”

——Scott Duncan，*Software Quality Professional*杂志

“本书关注非功能性特性，讲解了如何满足可靠性、安全性、移植性以及可维护性等这些关键需求。作者运用了来自开源项目中的上百个实例来说明这些概念和技术。推荐阅读！”

——*iWeek*（企业技术在线杂志）

“……Diomidis Spinellis全面讨论了各种具体的编程错误，书中处处流露出作者的丰富经验和深刻洞察力。……内容达到了一个相当的高度。……我真希望17年前就能读到这本书，它将是一部永恒的经典，适合广大开发人员。”

——Alan Berg，*Free Software Magazine*杂志

“本书中讨论的原则对使用其他语言的读者也是极有裨益的，尤其是涉及数据结构和算法等方面的问题时。……书中的大量真实开源代码的运用，大大增强了可读性和实战性。”

——Pan Pantziarka，Reg Developer网站

“本书充满了大量有趣而实用的建议。其中许多建议往往要通过多年实践和不断犯错才可能获得。”

——Edwin Fine，Amazon.com

序

在我们所从事的领域中，很少有一个作者能够横空开创一个新的话题，而Diomidis Spinellis的第一本书 *Code Reading* 恰恰就做到了这一点。我们这一行对讨论阅读代码而非编写代码的图书的需求是非常强烈的。软件思潮的一个学派就认为教新手们如何阅读代码比教他们如何编写代码要更重要，理由是：(a)开始写作之前多阅读是教授普通人学习自然语言通常的方法；(b)如今大多数程序员的任务是修改已有代码（这意味着要先阅读），而不是开发新的代码。因此，我非常高兴看到Spinellis认识到这一重要性，并且写了一本非常好的书来告诉我们如何阅读代码。

但是这也带来了一个有趣的问题，那就是Spinellis再次出手的时候会写什么！一个人能开创新话题多少次？有点遗憾的是，他的新书《高质量程序设计艺术》没有做这方面的尝试。但令人高兴的是，Spinellis在第2本书中讨论了软件质量，我敢说这是软件工程领域最重要也最让人困惑的课题。说这个课题重要，因为没有足够的质量，代码可能根本就是没有价值的。同时它很让人困惑，因为软件领域对此众说纷纭，似乎有多少人就此发表著作，就有多少关于质量的定义。

Spinellis不仅讨论了这个让人困惑的重要课题，而且论述得非常好。在其他大多数软件质量的讨论都关注管理并且只是泛泛而谈的时候，Spinellis直入主题，讨论了代码质量所反映的质量技术这一重要课题。在我看来（可能有些偏激），在管理层面上讨论质量基本上是没有意义的，因为只有在具体实现的代码级别上才能够辨别组成质量的各种要素。以Spinellis讨论的两个质量属性——可维护性与可移植性——为例，不对代码进行分析的话，就根本无法了解该软件的可维护性与可移植性。

对于那些渴望跨过技术层面，进入管理层面的读者——在我看来，这样的人太多了——这本书不适合他们理解软件质量。但是，对于那些知道质量的本质首先是技术问题，然后才是管理课题的读者来说，这本书是一个再好不过的起点。作者是这样开始本书的描述的："（你可从中）学到判断软件代码质量的方法。"向他致敬！

Robert L. Glass

ACM会士，软件开发大师

前　言

编程和其他所有事情一样，发生错误就是新的开始。
（In programming, as in everything else, to be in error is to be reborn.）

——艾伦·佩利（Alan J. Perlis），图灵奖得主

我很希望可以在前言中说你手中的这本书是精心策划的系列出版计划的产物，该计划始于《代码阅读方法与实践》（*Code Reading: The Open Source Perspective*，以下简称“代码阅读”），现在以本书作为完结。但是，这么说是在歪曲事实，改变真相来迎合我们这些工程师所喜欢的井然有序的世界。事实上，本书基本上是一系列偶然事件的产物。

在我签下《代码阅读》的出版合同时，手上已经有了大纲并且已经写完了几章。根据这些已完成章节的篇幅以及所花费的精力，我错误地估计了该书最终的篇幅与成书计划。如果你以编写软件为生的话，可能已经猜出，在手稿本应该完成的时候我只完成了大纲中计划章节的一半多一点点，却已经达到了允许的篇幅。为了体面地脱身，我给编辑提了个建议，就是将已经完成的部分（去掉关于可移植性的一章）作为《代码阅读》的第1卷出版，然后在第2卷中继续完成剩下的工作。我们达成了一致，《代码阅读》于是出版[Spi03a]，得到了许多好评，获得了2004年《软件开发》杂志生产效率大奖，并被翻译为6种语言。

在《代码阅读》一书中，通过使用实际开源项目的真实示例，我努力涵盖软件开发人员会碰到的绝大多数代码相关的概念，包括编程结构、数据类型、数据结构、控制流、项目组织、编码标准、文档以及体系结构。对于第2卷，原本我的计划是要涉猎面向接口与应用的代码，包括国际化与可移植性的各种问题、常用库与操作系统的各个要素、底层代码、领域特定的语言与声明性语言、脚本语言以及混合语言系统。不过，《代码阅读》到了程序员们的手上之后，我得到了来自读者的意见。很多读者都在焦急地等待着下一卷，但是彻底剖析某个设备驱动程序（我为下一卷预留的一章）并不是他们所想看的。在2003年7月，当时的编辑Mike Hendrickson建议我写一本名为《安全代码阅读》的书。尽管IT安全是个让我感兴趣的领域，但是我不想赶时髦，就只写了一章。有了关于可移植性的一章，再加上关于安全的一章，我的眼前突然闪现了这本书的主题与书名——《高质量程序设计艺术》将专注于阅读与编写软件代码的方法，专注于软件代码的质量属性，这些通常也被称为非功能性属性。

通过阅读软件系统的代码所能了解的非功能性属性是与产品的非功能性需求相关的，这些需求不与系统特定功能直接相关，而是反映了范围更广的重要系统属性。常见的非功能性属性包括可靠性、可移植性、可用性、互操作性、适应性、可信性以及可维护性。另外两个重要的非功能性属性与系统的效率有关：与时间约束相关的性能和空间需求。

通过阅读代码了解非功能性属性的能力是很重要的，原因有二。首先，无法满足非功能性需求是很危险的，甚至是灾难性的。某些功能性需求有错的系统（大多数软件产品都有这种错误）还能够以降级模式运行，因为可以告诉用户避免使用某些功能。而非功能性属性出错通常是致命的：不安全的Web服务器或不可靠的防抱死系统（antilock brake system，ABS）还不如没有。另外，非功能性需求有时候也很难验证。我们无法写一个测试用例来验证系统的可靠性，或者验证系统没有安全漏洞。因此，非功能性属性既关键又难以验证，意味着在处理非功能性需求及相应的软件属性时，我们需要尽可能利用各种可用手段。给代码与非功能性属性建立联系的能力是软件工程师应该具备的利器。

除了角度不同，本书沿用了《代码阅读》的成功做法：专注于现有代码的阅读，只研究来自于现有开源系统的真实示例，给出所有示例的出处，使用标注来分析代码，提供有意义的练习强化读者的判断能力与技巧，在页边空白处标出代码的惯用法以及陷阱，使用格言的形式来总结每章提出的建议，在“进阶阅读”部分为本书的实践知识提供理论依据，还使用了统一建模语言（Unified Modeling Language，UML）来绘制所有图表。此外，最微妙的成分就是我自己制订的规则：决不使用“玩具”示例，而是从已有的开源项目中提取所有代码示例。遵循这条规则，我常常需要花几个小时才能找到合适的例子，既要能够用来说明我要表示的概念、容易理解，又足够短小到可以放到书中。我发现这种练习既是一种智力模拟，也是使我的写作更加严谨的一种很好的方法。常常是在试图找出代码在某方面的缺陷的时候，我会发现其他值得讨论的有趣的内容。有时候，为某个理论概念查找示例最后被证明是徒劳的：这时我就可以认为这些概念没有那么重要，也就没有必要在书中进行讨论了。

本书背后的原理与动机和《代码阅读》一样：阅读代码可能是计算机专业人士最频繁的活动之一，但是却很少作为一个课程来教授，也很少正式作为一种学习设计与编程的方法来使用。开源软件的流行为我们提供了大量可以自由阅读与学习的代码。基于开源软件的读本可以成为加强编程能力的重要工具。因此我希望这两本书的存在能够引起大家的兴趣，在计算机教学中加入代码阅读的课程、活动和练习。这样，在若干年之后，我们的学生就可以从已有的开源系统中学习编程，就像我们从优秀的文学作品中学习语言那样。

内容与补充材料

我决定本书也采用《代码阅读》中使用的同样的系统与发布版作为源代码示例，原因是两卷的连续性是很重要的，这样读者就能够看到同样的源代码既可以用于分析《代码阅读》所涉及的功能、体系结构和设计上的各种软件特性，也可以用于分析本书所涉及的非功能性特性。

本书所用的代码都是以前早些时候的软件代码，目前，它们大多数都只有历史意义了。但是，有了这些代码，我就可以演示在更新的版本中已经被改正的安全漏洞、同步问题、可移植问题、误用的API调用以及其他错误。这些年代久远的代码很有可能意味着，它们的作者现在要么已经进入了管理层，根本不屑于阅读此类图书，要么已经老眼昏花，无法看清楚本书所用的字体。这些变迁让我可以自由地对代码进行评论，而不用担心打击报复。不过，可能会有人谴责我在诋毁这些代码，因为它们的作者怀着促进开源运动的信念贡献了代码，我们应该对其进行改进而不是仅仅进行评论。如果我的某些评论让某位源代码作者感觉不敬的话，我在这里预先真诚地道歉。不过我可以说，在大多数情况下我的评论并不是针对所用的代码片段，引用那些代码的目的只是为了演示在实践中应该避免的行为。我作为反例引用的代码往往是我批评的靶子，其实在它们编写的时候，由于技术等方面的限制那么编写还是合理的，我的批评不过是断章取义。无论如何，我希望你能一笑而过，而且我承认，我自己的代码也有类似的、甚至更糟糕的错误。

在选择本书的示例所用的系统时，我的出发点是这些代码要适合于教学。我关心的内容包括代码质量、结构、设计、应用、流行程度，还有不存在版权问题。我尽量平衡语言的选择，积极寻找合适的Java与C++代码。但是，当多种语言都可以演示类似的概念时，我会选择C作为“最小公分母”。因此，本书所引用代码的61%都是C代码，这包括适用于所有语言的情形以及系统编程（主要用C）方面的示例。另外19%的例子使用了Java代码，我选用了Java代码讲解面向对象的概念及相应的API。这些概念大多也适用于C#，很多还适用于C++（C++语言在4%的例子中用到）。

另外，本书更着重于Unix API和工具而不是Windows，同样是出于最小公分母的逻辑：很多Unix工具与API在Windows上也提供，而反之则不然了。另外，很多Unix兼容系统，如GNU/Linux与各种BSD变种都是免费的，通常还提供可引导的CD-ROM，因此大家可以很容易地使用这些系统进行试验。最后，就本书所涉及的细节程度而言，Unix API与工具过去30年来一直保持相当稳定，这为我们讨论与演示通用的原则提供了极好的平台。不过，在很多地方，我用了Windows API及命令来讨论其他平台上的情况。但不要因此而迷惑：我没有说本书对Windows平台编程问题的讨论比对Unix系统还要完整和详尽。

除了在示例中全部采用开源软件，本书可能还会被（狭隘地）指责为没有加上一些流行的元素，如Java、C#、Windows、Linux以及针对当前各种实际问题来写。事实上我重视所有这些：我家里的机器有29%运行着Linux；我开了一门Java编程的课；我已经在Windows平台上写了很多程序；而且我的书架上至少有10本书布满编了号的、提供具体解决问题的建议的段落。但是我也相信，在当今这个变化无常的世界中，理解背后的原理是很重要的。正如你将在后续章节中看到的，一旦我们专注于原理，那么：

- 底层技术的选择通常就不重要了；
- 我们学到的知识的应用范围更广而且时效更长；
- 具体的建议自然就存在了（参见第1章之外每章最后的“锦囊妙计”部分）。

最重要的是，理解了编程技艺背后的原理，你才能从无足轻重的编程人员跻身进入身价百倍

的软件工程师行列。

致谢

很多人慷慨地提供了建议、意见，花费宝贵的时间来帮助我完成本书。首先，本系列的编辑Scott Meyers承担了读者代言人的角色，熟练地引导作者，系统地指出本书可以增进可读性、减少薄弱环节之处。Hal Fulton、Hang Lau与Gabor Liptak阅读了本书最初试写的章节，提供了很多有用的意见与观点。Chris Carpenter与Robert L. Glass也阅读了试写的章节，并完整阅读了本书的初稿，让我从他们的睿智与经验中获益。非常感谢Konstantinos Aboudolas、Damianos Chatziantoniou、Giorgos Gousios、Vassilios Karakoidas、Paul King、Spyros Oikonomopoulos、Colin Percival、Vassilis Prevelakis、Vassilis Vlachos、Giorgos Zervas，并特别感谢Panagiotis Louridas，他们非正式地审阅了本书的初稿，并提出了详细的意见与建议，帮助我加以改进。另外，Stephanos Androutsellis-Theotokis、Lefteris Angelis、Davide P. Cervone、Giorgos Giaglis、Stavros Grigorakakis、Fred Grott、Chris F. Kemerer、Spyros Kokolakis、Alexandros Kouloumbis、Isidor Kouvelas、Tim Littlefair、Apostolis Malatras、Nancy Pouloudi、Angeliki Poulymenakou、Yiannis Samoladas、Giorgos Sarkos、Dag-Erling Smørgrav、Ioannis Stamelos、Dave Thomas、Yar Tikhiy、Greg Wilson、Takuya Yamashita、Alexios Zavras和Giorgos Zouganelis也都提供了很多有价值的建议，我常常是突然袭击，向他们打听其专业领域中的晦涩问题。我还要感谢雅典经贸大学管理科学与技术系的同事们，感谢他们对我工作的支持，还要感谢下面3位老师给我上的课，这对我写作本书至关重要：Mireille Ducassé（技术写作——1990）、John Ioannidis（代码风格——1983）和Jan-Simon Pendry（时间与空间性能——1988）。

在Addison-Wesley出版社，我的编辑Peter Gordon熟练地指导了本书的创作过程，解决了很多困难。Kim Boedigheimer以非凡的效率处理了日常事务，7小时的时差通常让我们得以在一天内以20小时轮班的方式来进行本书的写作。

在制作过程中，Elizabeth Ryan像一个专业的指挥一样，高效地协调和组织着我们这个全球跨学科的创作小组。

其他作者对本书的文字编辑Evelyn Pyle的评价是目光犀利。我非常赞同这一说法，并且还要加上一句，她的工作简直太棒了：她发现了我从来就没有想到在手稿中会出现的错误，并且她还以对细节与一致性的高度关注对行文进行了修正，只有少数顶尖程序员才能与她匹敌。另外两位同样具有编程艺术才能的人也为本书付梓做出了巨大贡献。Clovis L. Tondo以对本书所用工具的深刻理解和对书中代码的极大尊重完成了排版工作，而Sean Davey以其超凡能力力保本书风格一致。

本书所用例子的绝大多数来自于已有的开源项目。使用真实代码让我能够演示人们实际可能遇到的代码，而不是简化的玩具程序。因此，我要感谢所有对本书引用的开源项目做出贡献的人，感谢他们与编程社区共享他们的工作。书中代码的作者名字，如果在相应的源代码文件中列出的话，会包含在附录中。

目　录

第 1 章 概　述

看乃天生之能力，见，艺术也。
（Sight is a faculty; seeing, an art）

——乔治·珀金斯·马什（George Perkins Marsh），美国作家

本书的目的是帮助读者学到判断软件代码质量的方法。一旦掌握了这项艺术，就能够将刚刚获得的判断能力应用在自己或者他人编写的代码上，以评估代码质量的方方面面，并对不足之处进行改进。我们还可以在与同事们讨论各种备选实现方案时，使用所获得的关于代码质量的知识，让我们的软件项目向着最正确的方向发展。

1.1 软件质量

我们可以从规格说明的角度来看软件质量，把它定义为软件与需求之间的符合程度，或者还可以考虑到人的因素，将质量定义为软件满足客户或用户的需求或预期的程度。不管我们怎么看待质量，它都是重要的。质量、时间和成本是衡量所有软件项目成功与否的3个核心要素，而质量则是这些要素中唯一不能马上通过管理手段来改变的。而且，软件质量低下所带来的后果可能是相当严重而且难以挽回的：如果航天探测器（space probe）的机载软件算错了一个变量，因而坠毁在所探测的行星上，那么结果就是我们又回到了起点（还损失了探测器）。虽然本书关注的是程序代码的质量，但是在我们讨论诸如如何处理无效引用（null reference）之前，很有必要对软件质量的全貌做更广泛的了解，以确定我们将遵循的方法的适用性和局限性。

1.1.1 用户、制造者和管理者眼中的质量

你的新自行车真是棒极了。它看起来坚固耐用却很轻盈，易于操控但又很稳定，时尚却不浮华，舒适而且可靠。你骑着它全速冲下一条平坦、空旷的乡间坡路，感觉自己是世界之王。这种感觉的背后有什么魔法呢？我们怎样才能构造让人有同样感觉的软件呢？让我们逐一研究关于自行车质量的4个方面，这同样也适用于软件的质量。

第一个也是最广为人知的方面就是使用中的质量（quality in use），也就是真实的最终用户体验。一般来说，它反映了用户在特定环境中能够实现自己的目标的程度。在自行车的例子中，你的确在一个特定的环境中（空旷的下山路上）实现了自己的目标——一次完美的骑车旅行。如果你骑行在

崎岖的上山路上，与一群互相攀比车架中双锂（dilithium[①]）含量的骑手在一起较劲的话，你的体验可能就完全不同了。同样的思路也可以用在软件上。在检查软件使用中的质量时，我们关心用户是如何感知软件的[图1-1（左上角）演示了全世界所有不幸的最终用户都会经历的程序崩溃]。我们不关心用户永远不会碰到的程序错误，不关心难以理解的代码，也不关心虽然效率不高但是对于用户所处理的数据量来说没什么影响的算法。

在准备骑车出行时，你已经知道自己会很享受这次旅行，因为你认为它是一个高质量的产品。在出发之前，你略微提起自行车，让它在人行道上落下，以检查轮胎是否正常以及是否有不牢靠的部件；你切换档位，并捏闸刹住车轮。此外，车架底部的一个小标签证明了装配完成后有人在工厂的地面上试骑过这辆车，以确认它确实是所宣称的好产品。自行车的这些外部质量属性（external quality attribute）无疑会影响使用中的质量。如果车闸反应迟钝，你可能就会在一个树干上结束你的下山之旅了。在软件世界中，外部质量方面包括我们可以通过运行软件确定的方方面面，一般是在一个测试环境中——图1-1（右上角）演示了运行中的JUnit回归测试（regression test）框架。通过彻底测试并修正软件外部质量方面的各种问题，我们可以将最终用户会碰到的错误数减到最少。

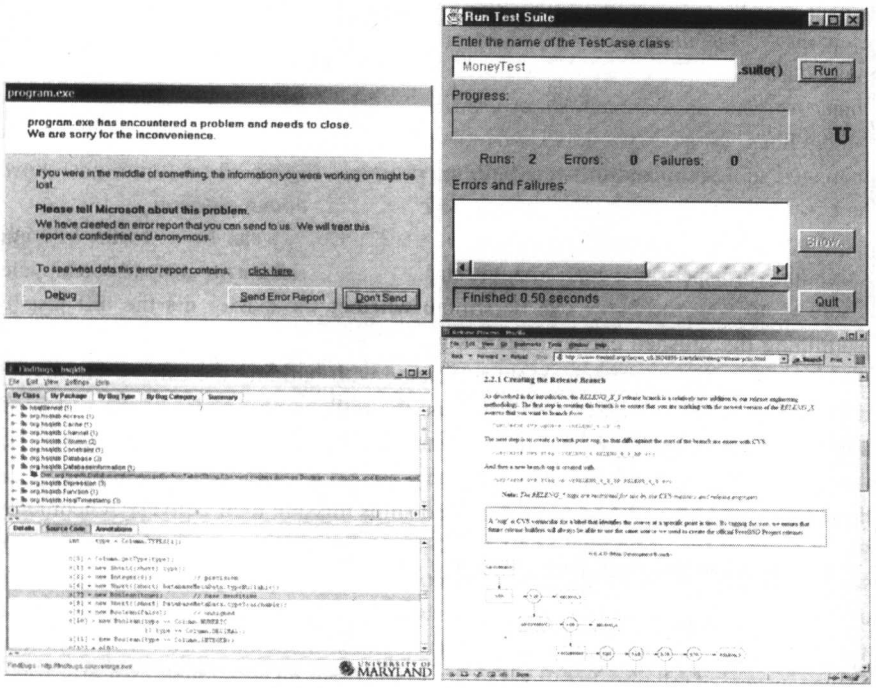


图1-1　软件质量各个方面的例子：使用中质量、外部质量、内部质量以及过程质量

现在，让我们回到自行车的生产厂。实际上，在装配线末端的试骑中很少甚至不会有自行车

① 美国的一部著名电视剧集“星际迷航”中所假想的一种用于提升飞船动力的水晶矿物质。——译者注

会当场散架。对于一辆精心设计和制造的自行车（例如你所拥有的这一辆），它的框架形状以及合金成分提供了必需的强度和硬度，所有生产出来的部件都满足规定公差，所有适当的（最贵的）部件也都正确安装，并且所有传动部件的活动范围也都精确地校准了。这些可以通过检查（而不是骑）自行车而确定的特性被称为内部质量属性（internal quality attribute）。对于软件来说，这些都是可以通过检查而不是运行来确定的软件属性，这也正是本书所关注之处。图1-1（左下角）演示了被FindBugs程序所发现的一个有问题的源代码结构。

当我们再次造访自行车工厂的时候，我们意识到自行车的品质不是偶然得到的。生产这辆自行车的公司上下一心要反复地制造让你享受完美骑车过程的自行车。各门类的工程师们都会花上几个星期的时间来仔细研究每一个设计，确保所有的细节都是正确的。所有入厂的原料都会被仔细检查，确保它们没有潜藏的故障；而所有的工人也都被允许（并且鼓励）在发现某辆自行车有质量问题的时候暂停流水线。所有这些措施（以及其他措施）都是过程质量（process quality）的元素，它们存在的目的是确保工厂生产自行车的方式是一种能够一贯产生几近完美的自行车的方式。与此类似，很多软件开发商都使用成熟的开发框架，如CMMI（Capability Maturity Model Integration，能力成熟度模型集成）或ISO 9001，来组织文档化（documented）、可重复（repeatable）、已定义（defined）、已管理（managed）和已优化的软件制造过程——图1-1（右下角）演示了FreeBSD发布工程过程文档的节录。

关注过程而不是产品本身有可能是值得的，而且有时候还是不可或缺的。想象一下不是制造自行车而是建造航空母舰。美国海军绝不可能只通过检查完工的航母就确定它是否符合需要（“这就是您的船，司令员！”）。海军需要做的是检查与航母的建造过程相关的过程元素。这可能包括设计师的资质证明、焊点的X光板以及所有组装好的部件的验收测试结果。但是，这个过程也有可能走了极端，结果成为一个所谓完美的航空母舰建造过程，这个过程的产物就是昂贵、超标准设计的、延期交货数年的航母。想想那出名的640美元军用标准马桶座圈吧[①]。在软件开发领域中，有些机构似乎过分强调了过程，其代价是损害了产品本身以及用户，结果必然是负面的。对此，软件工程界的反应就是出现了一些敏捷软件开发（agile software development）方法论，它们欢迎变化而不是死板地遵循计划；关注个体胜于过程；强调能用的软件胜于详尽的文档；强调客户协作胜于精确的需求定义。如果你正在遵循某种敏捷实践或使用某种敏捷方法，例如特性驱动开发（Feature Driven Development，FDD）、精益软件开发（Lean Software Development）、Scrum[②]、测试驱动设计（Test-Driven Design，TDD）或者极限编程（eXtreme Programming，XP），那么本书对于软件内部属性以及这些内部属性与代码关系的关注将帮助你提升代码的水平。

练习1.1 浏览某个开源软件的问题数据库，针对我们所讨论的质量的4个方面中的每一个，找出5个有代表性的条目，作为对应质量问题的证明。

① $640 Mil.Spec.toilet seat，这里的Mil.Spec.是美国军用标准，通常称为Military standard, MIL-STD或者MIL-SPEC。这里作者的意思是说在这种标准中有类似于640美元的马桶座圈这种“超标准”、昂贵的东西。——译者注

② Scrum一词原指橄榄球中的争球队形。这种敏捷开发流程将软件开发团队比作橄榄球队，强调阶段目标，沟通合作，确保每天、每个阶段都朝向目标有明确的推进。——译者注