HZ BOOKS
PEARSON Addison Wesley

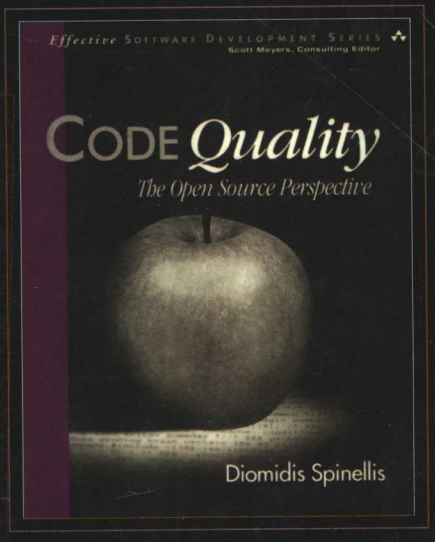
华章程序员书库

# 代码质量

## 注释版

# Code Quality

## The Open Source Perspective




（希） Diomidis Spinellis 著
康 华 齐永升 注释

“如果《Code Quality》和《Code Reading》得到了应有的关注……我认为，在提高代码专业化水平上，它们将取得立竿见影的效果，比其他任何翻滚袭来的浪潮都要迅猛得多。”

《Dr. Dobb's Journal》杂志社 Gregory V. Wilson

机械工业出版社
China Machine Press

程序员书库

# 代码质量

## 注释版

# Code Quality

## The Open Source Perspective

（希） Diomidis Spinellis 著
康 华 齐永升 注释

机械工业出版社
China Machine Press

本书重点讨论代码的非功能特性，深入讲述代码如何满足重要的非功能性需求，如可靠性、安全性、可移植性和可维护性，以及时间效率和空间效率。

本书从Apache Web应用服务器、BSD UNIX操作系统和HSQLDB Java数据库等开源项目中攫取数百个小例子，并以实例为基准点，辅以理论分析，从实用的角度讲述每个专业软件开发人员能立即运用的概念和技术。

本书适合作为软件开发人员、安全工程师及软件测试工程师等参考。



**本书版权登记号：图字：01-2007-3084**

**图书在版编目（CIP）数据**

代码质量　注释版 /（希）斯宾奈里斯（Spinellis, D.）著；康华，齐永升注释. -北京：机械工业出版社，2008.1

书名原文：Code Quality: The Open Source Perspective

ISBN 978-7-111-22671-0

Ⅰ.代… Ⅱ.①斯… ②康… ③齐… Ⅲ.代码-程序设计 Ⅳ.TP311.11

中国版本图书馆CIP数据核字（2007）第167491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：周茂辉

三河市明辉印装有限公司印刷 · 新华书店北京发行所发行

2008年1月第1版第1次印刷

186mm × 240mm · 40.25印张

定价：79.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

# 序　言

要知道能在计算机领域中独树一帜的作品可谓是凤毛麟角，而Diomidis Spinellis的第一本书《代码阅读》（Code Reading）就做到了这点。在计算机教学中教人如何阅读代码而非编写代码的书籍的确是大受欢迎。在教学生如何编写代码前，应首先学习如何阅读代码。因为学习其他语言的方法都是先学阅读，再学写作。而且在新千年，多数开发人员的主要任务是修改已存在的代码，而不是开发新代码。正因为如此，我很感激Spinellis能认识到这点，并且为我们撰写了如何阅读代码的指导书。

就像优秀歌手谢幕后经观众要求而再次歌唱一样，Spinellis为我们再次带来了他的新书《Code Quality》，虽然本书并非是开宗立派，但其精彩在于：它为我们展现了软件工程中软件质量的最最重要，同时也最容易混淆的诸多问题。软件质量是事关软件代码是否有价值的重要因素，不过对于该问题仁者见仁，智者见智，因此对质量定义在多角度、多层次上存在众多技术点。

Spinellis就软件质量问题为我们系统化、多层次、多角度地逐步展开分析，而且举例阐述得相当精彩。以前的软件质量书籍多是从管理角度等较高层次夸夸其谈，而Spinellis从大处着眼，小处着手，对很多影响代码质量的技术点各个击破。就我个人意见来看，管理角度讨论质量无异于水中望月，雾里看花，因为最终代码质量的决定因素还是代码本身。比如Spinellis所讨论的代码维护性和可移植性，假如不去分析任何代码，则很难真正理解所谓的可移植性和维护性的要求和措施。

这里我要提醒一下，对于那些所谓“志向高远”的读者，即希望能越过软件技术的细节直接跨入软件管理精英梯队的读者，本书不会帮助理解这样的软件质量；但是对于那些希望在成为管理人员前，能从技术上深刻理解软件质量的读者来说，这正是要认真阅读的书籍。

Robert L.Glass

2006年1月

# 前　言

虽然我很希望告诉大家《代码质量》和《代码阅读》两本书是早已计划出版的姊妹篇，但事实并非如此，《代码质量》一书其实是由于许多偶然因素所促成。

当我签订《代码阅读》一书的写作合同时，其实我已经勾勒出该书的轮廓，甚至完成了部分章节。我天真地按照已完成部分的篇幅和时间推算了该书的总篇幅和最后完成时间，然而我仅完成了预计章节总数的一半时，就发现我已用完了规定篇幅。为了能体面地结束合同，我不得不建议我的编辑将我已完成的部分（除去了移植性一章）作为《代码阅读》的第一卷先行出版，然后再将余下的未完成工作归为第二卷出版。所以《代码阅读》才得以顺利出版，然而令我们鼓舞的是该书广受好评，获得了2004年《软件开发杂志》的大奖，并被翻译成6种其他语言海外出版。

在《代码阅读》一书中，我大量利用开源项目中实际代码片段，尽可能多地囊括开发者可能遇到的代码相关技术点，包括程序结构、数据类型、数据结果、控制流程、项目组织、编码风格、文档和体系结构。我计划第二卷中会讲述程序接口和应用程序相关代码，包括国际化、可移植性、常用库函数和操作系统服务，以及一些底层代码、DSDL语言、脚本语言和混合语言系统。我又分析了《代码阅读》出版后读者的反馈，他们迫切希望第二卷能尽快推出，不过他们并不需要设备驱动细节剖析的内容（这个内容本是我想留在第二卷中写的内容）。2003年，我的编辑Mike Hendrickson建议第二卷命名为《安全代码阅读》（Secure Code Reading），虽说我作为研究人员的确也对IT 安全抱有兴趣，但我还是并不很情愿地跳入这个流行行列当中，因此只写了一章关于安全的内容。当我完成了可移植性和安全性两章后，我对书的命名突发奇想，就叫《代码质量》吧，这个名称不但包含了如何读写代码，而且还强调了代码的质量因素——属于非功能特性。

代码的非功能特性来自于产品的非功能性需求。这些需求虽然和系统功能没有直接关系，但却影响到更普遍、更广泛的系统属性：可靠性、移植性、可用性、交互性、适应性、依赖性以及维护性。另外还有两个重要的非功能特性和系统效率息息相关：时间性能和空间需求。

研究代码非功能特性的重要性基于两点原因：首先，如果不能满足非功能性需求，则在某些环境下会产生严重麻烦，甚至是造成灾难。如果一个系统其功能性需求没有得到满足（多数软件产品都有该问题），那么通过降低操作模式，或让用户避免使用特定功能可以绕过问题。但是非功能性错误则常常会造成程序执行中断（不可绕过）：一个不安全的网络服务器或者一个不可靠的反锁刹车系统中断的代价远高于不去用它；另外非功能性需求很多时候难以验证。我们不可能写一个测试用例去验证系统的可靠性或者安全性。所以非功能特性缺陷带来的灾难和难以验证的特点让我们在处理非功能特性以及对应的软件特性时，需要采取一切可采取的办法。将非功能特性表现在代码中的能力无疑是一个优秀软件工程师所应该具备的。

除了立意不同外，《代码质量》一书继承了《代码阅读》一书的成功经验：把重点放在阅读已存在的代码，以开源项目的代码为例进行阐述；采用注释项的方法剖析代码，提供有意义的联系加强读者能力和技巧；在页脚解释编码术语和陷阱；以格言形式在章节尾总结重要事项；在扩展阅读部分给出理论依据；使用UML图示说明等。上述经验中最核心的思想是：摒弃使用无意义的玩具代码做例子，而使用真实的开源项目代码。根据该原则，我花了不少时间寻找合适的能表述书中概念，并且容易理解而长度又适中的例子。通过这种方法，我常在搜索一个特定缺陷时，又发现了其他值得讨论的技术要点。而我在搜索理论概念时，通常都是无功而返：我确实看不出来有哪些在实践中很重要，值得写入本书的概念。

《代码质量》一书的写作初衷和动机无异于《代码阅读》一书：阅读代码可能是计算机专业最常见的实践活动，然而一直以来都很少被当作一门课程或者被看作一个标准方法去用以学习如何设计和实现程序。当前开源软件的流行恰恰给我们提供了充分的阅读代码的机会，尤其是对于新手来说，阅读开源代码无疑可以提高自己的开发技巧。所以我希望这两本书可以在计算机教学中引起对于代码阅读课程、阅读实践和阅读练习的足够重视。希望能在不久以后，我们的学生可以从开源代码中学习编程，就如同从伟大文学著作中学习语言一样普遍。

## 内容以及补充材料

我将在《代码质量》一书中采用《代码阅读》中包含的源代码例子，这样有助于保持两本书连贯性，因为读者可以在同一个实例代码中看到《代码阅读》一书中讲述的功能性、架构性和设计性相关内容，同时又能看到《代码质量》一书中讲述的有关非功能性内容。

本书中的代码虽说多数都过时了，不过这正好给了我们机会，能去前后对比观察实际项目中存在的安全漏洞、同步问题、移植问题、错误使用API问题以及其他错误是如何被发现，并在后来更新版本中解决的。许多代码历史悠久，它们的作者现在有的已经跃升到管理岗位（再回过头来看这些代码定然颇有不快了），或者可能有的已经老态龙钟都看不清字了。不过也正是因为时过境迁，我才能公开地评价这些代码。但是我知道我仍有可能因为批评了某些代码而引起某些作者的不满（这些代码都是作者为了推动开源运动而无私奉献的），或者有人可能会说我应该促其改进，而不应该仅仅停留在批评上。如果我的评论确实冒犯了源代码作者，我在这里提前向您道歉。在绝大多数情况下，我的评论都不会是针对任何特殊的代码片段，而是用于表述一种可以避免的实际缺陷。我常常用作反例的是一些易于攻击的代码，而其实这些代码在编写的那个年代，在技术和其他限制条件下往往都是很保险的（对于有些特殊情况的批判脱离了其运行环境）。任何时候我都希望我的评论可以被慷慨地接受，并且能宽容我自己代码也包含了类似或许更糟糕的过失。

我所选择的实例代码都具有一定的代表性和指导性，我关注的是代码质量、结构、设计、功用、流行性以及许可证（对编辑来说可是一个棘手的问题）。我尽量选择各种语言的代码例子，特别是合适的Java和C++代码。但是同样的道理也完全可以使用其他语言予以示范，我选择C语言作为最基础的典型语言。可以看到书中61%的参考代码是C语言代码，这些例子相对较小，而且有很多是系统程序（系统程序多用C语言开发）；另外有19%参考用例是Java代码，我选择Java语言作为面向对象概念和相关API的典型示范语言，当然绝大多数概念也完全适用

于C#，多数适用于C++语言。

相比起来，我更强调UNIX的API和UNIX的工具，而不是Windows系统，主要原因仍然是由于普遍性：许多Unix工具和API同样在Windows下可用，而反之则不能；另外，许多UNIX兼容系统（如GNU/Linux和BSD的各种变种）都可免费获得，而且作为可启动运行的CD-ROM被使用（live cd），因此任何人都能方便地用它们来做试验；最后一个原因是，UNIX的API和工具已经在过去30年中千锤百炼，相当稳定了，因此它们能够为我们提供讨论和展示普遍原理的最好环境。不过在许多地方，我也引用了Windows的API和命令以讨论代码如何跨平台工作等。但不要让这些参考引用误导你：我没有宣称该书的内容全面覆盖了Windows平台编程问题；对于UNIX系统也一样，本书所讲的并非是所有存在的编程问题。

除了使用开源软件作实例以外，有些人也许认为我似乎回避了很多当前流行的问题，比如Java、C#、Windows和Linux，另外我的写作思路也应该针对当前时弊。我曾统计过：我周围机器中29%是运行Linux的；我教授Java课程，写了许多Windows平台下的程序，我书架上也至少有数十本书长篇累牍地罗列这针对具体问题的解决方案。但是我认为在当今迅速发展的世界面前，最重要的是理解说教背后的真理。就如在后面章节中所见，当我们重视并理解原理时会发现：

- 在技术背后的选择通常是看不见、摸不到的理念。
- 我们所学的都是普遍真理，且会在相当长时间内都成立。
- 具体的建议也不少（看看每章后的建议内容）。

最重要的事情是理解书中的原理，而能否理解原理是区分一个优秀软件工程师和蹩脚代码员的重要标志。

## 感谢

能完成这本书离不开许多人慷慨帮助、建议和评论。首先要感谢我的编辑Scott Myers（是他策划了这一系列丛书），他让我看到了读者的希望和支持，并给出了书的写作方向，同时还科学地指明了使书更通俗易懂且结构紧凑的写书技法。Hal Fulton、Hang Lau和Gabor三位批阅了本书的示范章节，给了我许多很有价值的评论和主张。Chris Carpenter和Robert L.Glass也帮助我审核了示范章节和所有的初稿，并将他们的经验和智慧传授给我。还要感谢Konstantios 、Aboudolas、Damianos Chatziantoniou、Giorgos Gousios、Vassilios Karakoidas、Paul King、Spyros Oikonomopoulos、Colin Perciva、Vassilis Prevelakis、Vassilis Vlachos、Giorgos Zervas和Panagiotis Louridas，他们私下帮我审阅过早期的初稿，为我以后的修改提供了许多细节性的评论和建议。另外还需要感谢stephanos Androutsellis-Theotokis、Lefteris Angelis、Davide P.Cervone、Giorgos Giaglis、Stavros Grigorakakis、Fred Grott、Ghris F.Kemerer、Spyros Kokolakis、Alexandros Kouloumbis、Isidor Kouvelas、Tim Littlefair、Apostolis Malatras、Nancy Pouloudi、Angeliki Poulymenakou、Yiannis Samoladas、Giorgos Sarkos、Dag-Erling Sm rgrav、Ioannis Stamelos、Dave Thomas、Yar Tikhiy、Greg Wilson、Takuya Yamashita、Alexios Zavras和Giorgos Zhouganelis，他们即便常常都不明白为什么我会突然对他们熟悉的领域问一些很含糊的问题，但都会耐心地给予我解答。我也不能不感谢我在Athems大学科技管理

系的同僚们给予我工作的支持。此外还需提到三位引导我写作（其中蕴含了该书的本质）的导师，他们是Mireille Ducasse（科技写作-1990）、John Ioannidis（代码风格-1983）、Jan-Simon Pendry（时间和空间性能-1988）。

在Addison-Wesley出版社，我的编辑Peter Gordon，指导我成书，并处理了很多难题；另外Kim Boedigheimer每天都高效地维护本书开发版本；由于我们之间有7小时的时差，所以在一天中我们差不多每20个小时一轮班。

在该书的开发过程中，Elizabeth Pyan充当我们的总指挥，他有效地沟通和协调我们全球不同领域的人一同工作。

Evelyn Phle，其实很多作者已经描述过她的查错能力如鹰眼一般的敏锐了，是书籍的审稿。不过我还是不能不佩服她的神奇：她竟然在我的初稿中捕捉到了许多我都不可想象的错误，她精确地修正了书中的细节错误，让书内容保持前后一致——只有很少一部分顶级的开发人员才能达到这种功力。还有两位专业人员以其非凡本领缩短了本书的完成周期，Vlovis L.Tondo对于我所使用的软件工具了如指掌，所以排版中保证了代码片样式的准确和美观；而Sean Davey创建了本书别具一格的排版风格。

本书中大量实例来源于开源代码。这使我能提供给大家实际中可能遇到的真实代码，而非简单的玩具demo。因此我必须感谢所有开源软件的贡献者，这些贡献者的名字都在书中保留，可以在最后的附录中查到他们。

# Contents

# 1

# Introduction

概 述

远见是天赋，而观察是一门艺术。

— George Perkins Marsh

本书的目的是学习如何判断软件代码的质量。有了这些知识，我们就可以对我们自己写的代码或者别人写的代码做出评价，借此提高我们的技能，稳步推进我们所从事的项目，向着正确的方向前进。

In this book, we set as our goal to learn how to judge the quality of software code. Having mastered this art, we'll then be able to apply our newfound sense to the code we write ourselves and to the code written by others, aiming to assess its quality aspects and improve what we find lacking. We can also use our acquired knowledge of code quality when we discuss implementation alternatives with our colleagues: ideally, nudging our project toward the most appropriate direction.

## 1.1 Software Quality

### 1.1 软件质量

什么是软件质量？可以定义为软件满足需求的程度，或者满足客户期望的程度。无论在哪种情况下，质量都很重要。质量、时间和费用互相制约。鱼与熊掌，不可兼得。但只有质量是不能强制的，质量不好的代码可能造成灾难性后果，而且修改起来非常麻烦。

We can view software quality from the point of its specifications and define it as the degree to which it meets the specified requirements, or we can also take people into account and define quality as the degree to which the software meets customer or user needs or expectations. No matter how we look at quality, it is important. Quality, time, and cost are the three central factors determining the success or failure of any software project, and quality is the only one of those factors that cannot be changed on the spot by management fiat. In addition, the effects of poor software quality can be dramatic and difficult to undo: If our space probe's software miscalculates a variable and crashes onto a planet, we are back to square one (minus the probe). Although this book focuses on the quality of program code, before we discuss, say, the treatment of `null` references, it is worthwhile to take a broader look at the software quality landscape to see the applicability and the limits of the approach we will follow.

1.1.1 用户、构建者和管理者眼中的质量

软件如商品，商品质量如何？我们应当从哪些角度来分析？

## 1.1.1 Quality Through the Eyes of the User, the Builder, and the Manager

Your new bike is truly exceptional. It feels sturdy yet light, maneuverable yet stable, trendy but not flashy, comfortable but also dependable. You ride it at full speed down a smooth, empty, downhill country road and feel like the king of the world. What is the magic behind this feeling? How can we build software that feels like that? Let's examine one by one four views of your bike's quality, which also apply to the quality of software.

第一点就是要分析使用质量，也就是最终用户体验。当我们检查使用质量时，我们在设想用户的使用感受。

The first and most commonly perceived view of quality is *quality in use*, the actual end-user experience. Broadly speaking, this view reflects the extent to which users can achieve their goals in a particular environment. Thus, in our bike example, you do achieve your goal—a Perfect Bike Ride—in a particular environment: an empty downhill road. Maybe if you were riding on a rocky uphill trail with a group of bikers comparing the quantity of dilithium contained in each bike's frame, your experience would be completely different. We can apply the same thinking to software. When we examine quality in use, we care about how the user perceives the software—Figure 1.1 (top left) illustrates a program crash as experienced by hapless end-users all over the world. We're not interested in bugs that the user never encounters, in unreadable code, or inefficient algorithms that don't matter for the amount of data the user will process.

第二点要分析外部质量属性，通常会采用类似JUnit的测试工具对软件进行外部测试，尽可能从外部质量的角度发现问题，减少最终用户将来遇到的问题。

When you set out for your bicycle ride, you already knew you would enjoy the trip, because you felt you had on your hands a quality product. Before departing, you lifted the bike and let it fall on the pavement to see that the tires were in good order and no parts were loose, you changed the gears up and down, and you used the brakes to bring the bike to a standstill. Furthermore, a little sticker on the bottom of the frame certified that someone at the end of the factory's assembly line rode the bicycle around the factory floor to verify that it was indeed the fine product advertised. These *external quality attributes* of your bike certainly influence the quality in use. If you found the breaks responding sluggishly, you might end your downhill ride on a tree trunk. In our software world, the external quality view consists of what we can determine by running the software, typically in a testing environment—Figure 1.1 (top right) shows the JUnit regression testing framework in action. By thoroughly testing and correcting the problems associated with the software's external quality view, we will minimize the errors the end-user will encounter.

Let us now return to your bike's factory. In practice, few if any bikes collapse into pieces when test driven at the end of the assembly line. In a well-designed, well-built bike, such as yours, the frame's shape and alloy provide the requisite strength and

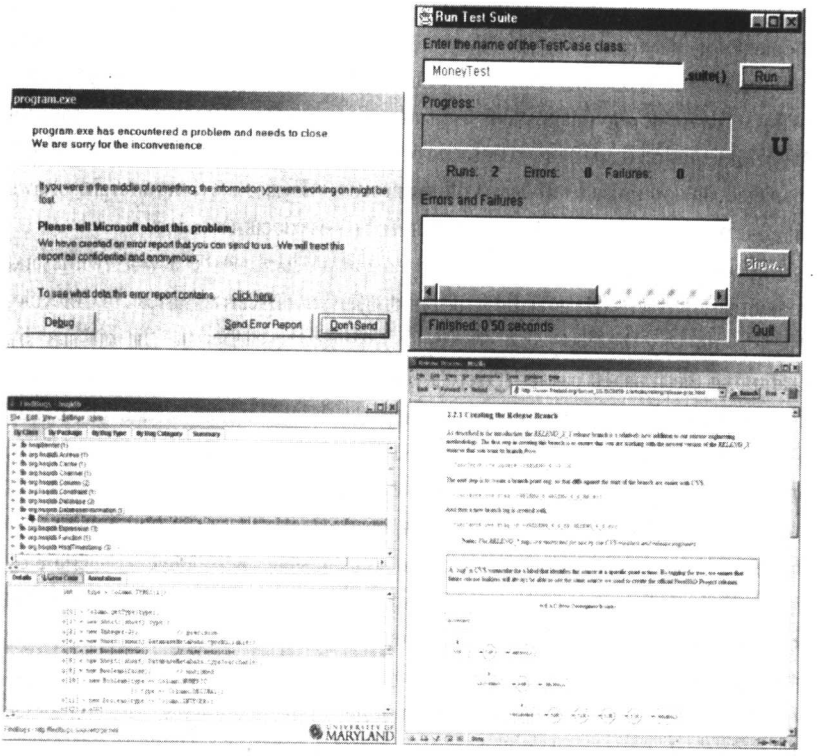



**Figure 1.1** Examples of the various software quality views: in use, external, internal, process

stiffness, all manufactured parts follow the prescribed tolerances, all appropriate (top-of-the-line) components are correctly mounted, and the ranges of all moving parts are precisely calibrated. These characteristics, which we can determine by examining (rather than riding) the bike, are termed *internal quality attributes*. In software, these attributes are the ones we can determine by examining, rather than running, the software, and they are the main focus of this book—Figure 1.1 (bottom left) illustrates a problematic source code construct identified by the *FindBugs* program.

第三点是要分析内部质量属性，使用静态分析工具对软件代码进行分析，检查可能存在的软件缺陷。

In our fourth and last visit to our bike's factory, we realize that the bike's quality is no accident. The company producing this bike is geared toward making—time and again—bikes that will grace you with a Perfect Ride. Engineers of various disciplines pore over each design for weeks to ensure that every detail is correct. All materials coming into the factory are carefully inspected to make certain that they have no hidden faults, and all workers are allowed (and encouraged) to bring the assembly line to a

第四点是分析过程质量，通过严格的质量过程来组织管理，对软件过程进行定义。让软件过程重复利用，处于可控状态，并且持续优化。