TURING 图灵程序设计丛书 程序员修炼系列

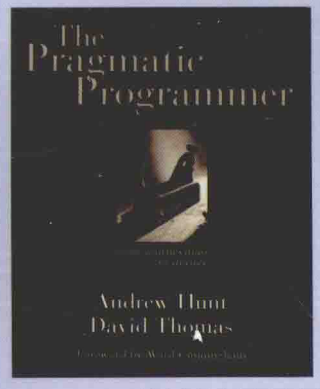
Addison Wesley

The Pragmatic Programmer
From Journeyman to Master

程序员修炼之道

（英文注释版）

[美] Andrew Hunt
David Thomas 著



- 熔知识、哲理、幽默与实践于一炉的编程奇书
- 引导你领悟程序设计的真谛
- 丰富的难词和背景信息注释

人民邮电出版社
POSTS & TELECOM PRESS

URING 图灵程序设计丛书 程序员修炼系列

The Pragmatic Programmer
From Journeyman to Master

程序员修炼之道
（英文注释版）

[美] Andrew Hunt
David Thomas 著

人民邮电出版社
北 京

图书在版编目（CIP）数据

程序员修炼之道：英文注释版 /（美）亨特（Andrew Hunt, A.）（美）托马斯（Thomas, D.）著. —北京：人民邮电出版社，2007.12
（图灵程序设计丛书）
ISBN 978-7-115-15566-5

Ⅰ. 程… Ⅱ. ①亨…②托… Ⅲ. 程序设计—技术培训—手册—英文. Ⅳ.TP311.1

中国版本图书馆 CIP 数据核字（2006）第 146757 号

内 容 提 要

本书是一部令人称奇、耳目一新的著作。书中通过许多有趣的轶事、详实的例子、诙谐的对话和技术细节，从编程一线审视了软件开发以及程序员职业生涯方方面面的最佳实践方案和各种需要注意的前车之鉴。书中给出大量建议，每一条建议都汲取了作者的经验，并与其他建议相互关联而形成系统。

本书适合各层次软件开发人员阅读，也适合高等院校计算机专业学生和教师阅读。

图灵程序设计丛书

程序员修炼之道（英文注释版）

◆ 著 [美]Andrew Hunt David Thomas
责任编辑 傅志红

◆ 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：21.5
字数：410 千字 2007 年 12 月第 1 版
印数：1－3 000 册 2007 年 12 月河北第 1 次印刷
著作权合同登记号 图字：01-2006-3694 号

ISBN 978-7-115-15566-5/TP

定价：49.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223

快速参考

The Pragmatic Programmer

This card summarizes the tips and checklists found in *The Pragmatic Programmer.*

For more information about THE PRAGMATIC PROGRAMMERS LLC, source code for the examples, up-to-date pointers to Web resources, and an online bibliography, visit us at `www.pragmaticprogrammer.com`.

Quick Reference Guide

TIPS 1 TO 22

1. Care About Your Craft xix
Why spend your life developing software unless you care about doing it well?

2. Think! About Your Work xix
Turn off the autopilot and take control. Constantly critique and appraise your work.

3. Provide Options, Don't Make Lame Excuses 3
Instead of excuses, provide options. Don't say it can't be done; explain what *can* be done.

4. Don't Live with Broken Windows 5
Fix bad designs, wrong decisions, and poor code when you see them.

5. Be a Catalyst for Change 8
You can't force change on people. Instead, show them how the future might be and help them participate in creating it.

6. Remember the Big Picture 8
Don't get so engrossed in the details that you forget to check what's happening around you.

7. Make Quality a Requirements Issue 11
Involve your users in determining the project's real quality requirements.

8. Invest Regularly in Your Knowledge Portfolio 14
Make learning a habit.

9. Critically Analyze What You Read and Hear 16
Don't be swayed by vendors, media hype, or dogma. Analyze information in terms of you and your project.

10. It's Both What You Say and the Way You Say It .. 21
There's no point in having great ideas if you don't communicate them effectively.

11. *DRY*—Don't Repeat Yourself 27
Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

12. Make It Easy to Reuse 33
If it's easy to reuse, people will. Create an environment that supports reuse.

13. Eliminate Effects Between Unrelated Things 35
Design components that are self-contained, independent, and have a single, well-defined purpose.

14. There Are No Final Decisions 46
No decision is cast in stone. Instead, consider each as being written in the sand at the beach, and plan for change.

15. Use Tracer Bullets to Find the Target 49
Tracer bullets let you home in on your target by trying things and seeing how close they land.

16. Prototype to Learn 54
Prototyping is a learning experience. Its value lies not in the code you produce, but in the lessons you learn.

17. Program Close to the Problem Domain 58
Design and code in your user's language.

18. Estimate to Avoid Surprises 64
Estimate before you start. You'll spot potential problems up front.

19. Iterate the Schedule with the Code 69
Use experience you gain as you implement to refine the project time scales.

20. Keep Knowledge in Plain Text 74
Plain text won't become obsolete. It helps leverage your work and simplifies debugging and testing.

21. Use the Power of Command Shells 80
Use the shell when graphical user interfaces don't cut it.

22. Use a Single Editor Well 82
The editor should be an extension of your hand; make sure your editor is configurable, extensible, and programmable.

TIPS 23 TO 46

(下转书后)

版 权 声 明

献给 Ellie 和 Juliet，

Elizabeth 和 Zachary，

Stuart 和 Henry

对本书的赞誉

“这本书最棒的地方，是它能使我们对编程工作总是能保持新鲜的感觉。本书能帮助你不断提高，而且显然，它出自经验老到的人之手。”

——**Kent Beck**，*Extreme Programming Explained：Embrace Change* 的作者

“我发现这本书出色地结合了切实的建议和精彩的比喻!”

——**Martin Fowler**，*Refactoring* 与 *UML Distilled* 的作者

“我会买一本，读上两遍，然后让我的所有同事都冲出去买一本。这是一本我决不会出借的书，因为我担心丢失。”

——**Kevin Ruland**，Management Science，MSG-Logistics

“作者的才智和实践经验显而易见。讨论的话题重要而且有益……迄今为止，对我而言它最大的优点在于出色的比喻——曳光弹、破窗，还有拿直升机在危急情况下来说明为何需要正交性的精彩阐述。我几乎毫不怀疑：无论是对编程新手，是专家级的顾问，还是其他专业人员，这本书最终都将成为有用信息的极好来源。”

——**John Lakos**，*Large-Scale C++Software Design* 一书的作者

“这是那种我会在其出版时买上十几本、送给我的客户的书。”

——**Eric Vought**，软件工程师

“现在的大多数关于软件开发的书都没能说清一个杰出的软件开发者究竟应该具备哪些能力，而是把时间花在介绍语法和技术上——在现实生活中，拥有有才华的、真正精通其技艺的开发者，对任何软件团队而言都可能是最大优势。一部杰作。”

——**Pete McBreen**，*Software Craftsmanship* 一书的作者

“读了这本书以后，我实践了书中提出的许多实用建议和提示。所有这些都为我的公司节省了时间和金钱，同时还帮助我更快地完成了我的工作！每一个以编码为生的人，都应该在桌面上放一本做参考。”

——**Jared Richardson**，高级软件开发员，iRenaissance，Inc.

“我想看到这本书被发给公司的每一个新员工……”

——**Chris Cleeland**，高级软件工程师，Object Computing，lnc.

“很难相信编程背后真正重要的思想在这么小的篇幅里就做出了如此清晰地阐述，我非常高兴现在有了一本书，可以让新入行的程序员在获得错误观念、养成坏习惯之前传阅，阻止悲剧的发生。”

——**Luke Kanies**, Reductive, LLC

“本书说它有益已经远远不够，它使人心潮澎湃。”

——**Andy King**, WebReference.com

“20 多年来讲述编程实践的最佳图书。买下它吧，这将是你最超值的投资……。”

——**W.Craig Trader**，软件架构师

出 版 说 明

经过半年多的努力，“图灵程序设计丛书”的新成员——几本英文注释版图书终于与大家见面了。

本次出版的《重构》、《企业应用架构模式》、《敏捷软件开发》（Java 版和 C#版）和《程序员修炼之道》五部著作都是软件行业公认的为数不多的真正的经典之作，如果排除特定于具体语言、工具和技术的图书，它们可以毫无疑问地入选所有软件开发人员必读技术书目的前十名（至于其他候选者，脑中闪过《设计模式》、《人月神话》、《计算机程序设计艺术》、《编程珠玑》、《代码大全》……）。

这其中，《**重构**》曾经与《设计模式》、《反模式》（中文版即将由人民邮电出版社出版）、《解析极限编程——拥抱变化》并称为软件工程四大名著。Martin Fowler 虽然不是重构技术的创造者，但是由于 Kent Beck、John Brant、William Opdyke 和 Don Roberts 等先驱的襄助，本书也成为实至名归的开创性著作。由于近年来工具支持的加强，重构已经越来越成为开发人员日常不可或缺的技术。《**企业应用架构模式**》曾经荣获 2003 年 Software Development 杂志读者选择大奖和 Jolt 生产效率奖，是 Martin Fowler 的另一部名著，某种意义上也是真正奠定他在技术界中地位的一部重要著作。如果说《设计模式》是程序设计层次的圣经，《面向模式的软件架构》是架构设计层次的圣经的话，此书则当之无愧地可以称为企业级应用的圣经。企业级开发已成为主流，其中的困难与挑战是显而易见的，而这一领域的图书一直不多，所以此书更加弥足珍贵。Fowler 的这两部著作都是对业界多年积累的经验的总结，实战性非常强，不仅使你知其所以然，更让你知道如何实现、各种实现方式的适用场合、实现中需要注意哪些问题，等等。书总体上采用教程+参考的形式，教程部分只有 100 页左右，实际阅读时，先精读此部分，参考部分可以通过边干边学方式学习，精华尽在此中。

熟悉最近大红大紫的 Ruby on Rails 的读者，一定知道 Andrew Hunt 和 David Thomas 以及他们创办的 The Pragmatic Programmers 公司，他们撰写和出版的

Programming Ruby 和 *Agile Web Development with Rails* 两本书是RoR关键性的推动力量。而这个如今大名鼎鼎的公司的名字就出自《**程序员修炼之道**》一书（英文版原名即 *The Pragmatic Programmer*）。此书从各个方面来看都令人称奇：它当然是一本技术图书，但是字里行间弥漫着的浓浓的人文气息、哲理性和幽默感，又时常让我们产生疑惑；它的篇幅不大（只有300来页），然而几乎涉及了软件开发和程序员生涯的一切——责任心、学习方法、思考方法、沟通、设计原则、版本控制、代码生成、按契约设计、调试、测试、估算、并发、重构、需求、文档、各种工具……甚至包括如何发邮件、如何在Windows上使用Unix命令；内容层次跨度也极大，既可以高到团队组建，也可以深至代码级的具体细节，甚至还有不少编程习题！阅读此书的时候，常常会好奇，作者是何方神圣，能够将如此之多之丰富的内涵如此完美地熔于一炉。

《**敏捷软件开发**》则是另一位业界大师Robert Martin（人称Bob大叔）的代表作品，Java版（也含有不少C++代码）荣获2003年Jolt大奖，去年年中又出版了C#版，由Robert与他的儿子Micah合作，主要改动除了将代码换成C#之外，还调整了UML建模等部分的内容，脉络更加清晰。此书以“敏捷”命名，其实有很大的误导性，它的副标题“原则、模式和实践”才算名副其实，估计这也是原版C#版书名改为 *Agile Principles, Patterns, and Practices in C#* 的原因。原因很简单，此书只用了100页左右的篇幅概述敏捷开发方法，主体部分主要是对面向对象诸原则和GoF模式的阐释，其实是一部非常优秀的面向对象设计、实现和设计模式图书，语言通俗有趣，而且实战性很强。书中的许多绝妙插图，使我们在会意一笑中，更深地理解一些原本艰涩的技术概念。如果你阅读《设计模式》（机械工业出版社）一书感觉有困难，或者在阅读《设计模式解析》（人民邮电出版社）之后需要在实际开发环境中进一步领悟模式，那么本书将是绝佳选择。

经典之所以成为经典，既在于它们不仅是业界大师的作品，凝聚了软件开发社区集体多年摸索而获得的宝贵经验，拥有不因时光流逝而磨灭的价值；更因为它们很难一遍就领会其所有意蕴和精华，需要反复阅读，而且往往能够常读常新——某种意义上，你的阅读所得与你的经历是直接相关的，在不同阶段会有不同感受和收获，这也是前人说“少不读水浒”和“少不读杜甫”的原因。与此同时，经典的翻译往往难以尽善尽美，事实上这几部著作的中译本都多多少少存在各种问题，有的问题还非常严重。美国大诗人Robert Frost曾经说过，“Poetry is what gets lost in translation（诗歌就是在翻译中失去的东西）”，我们在长期的技术图书翻译和编审工作中也充分体会到，经典原著的许多关键的意境、暗示往往是难以用另外一种文字来表达的。有好的译本，当然能够起到事半功倍的作用，但是即便如此，直接或者对照着阅读原著仍然是有所裨益甚至必不可少的。而且，掌握英语，具备良好的英文技术文献阅读能力，也是今天平坦

世界（读过《世界是平的》吗？）中一名专业软件开发人员必备的素养，而阅读名著原版，绝对是一种一举两得的英语进阶方式。

我们之所以在这几部著作大多出版多年（最早的原版初版于 1999 年），已经有了翻译版，甚至此前曾经还有过影印版行世的情况下，仍然下决心再次出版英文版，主要原因正是它们的经典性和长效性。事实上，这些书原版到现在确实依旧长销不衰，我们手上拿到的原版样书，无不已经是最近的印刷，印次达到 10 几次甚至 20 几次；在 Amazon 等主要的网络书店上，它们也仍然位居销售榜前列，让许多热门的新书后辈也难以望其项背；在巴诺和鲍德斯这样的大型连锁书店，你会看到这些著作依旧被摆放在最显著的位置，心中涌起一种感动。反观国内市场，它们的英文版几乎都已经绝版，更有中文版也绝版的奇怪现象。我们希望这批经典的出版，能够打破这一怪圈。

本次出版的英文注释版，除了按原版版权方的要求翻译了前言、序等文字，并原汁原味地保留了原书的正文部分之外，还在多位业界专家的大力支持下，利用页边和页脚的空白增加了一些注释，算是一种新的尝试，力求为读者能够提供更多的价值。

增加的注释主要是以下几种情况：

1. 直接注释单词。其目的就是方便读者，能够少查字典。事实上，我们在阅读（以及翻译）英语技术文献时，所遇到的直接困难并不是技术上的，而更多来自英语本身。因此我们所注释的词语或者语句，许多并不是术语，而是通用的单词或者习语。在注释时，我们并非简单地按字典的常见义项给出，而是充分参考原著的上下文，给出特定语境下的对应词。

2. 章节标题给出翻译，目录也改为双语对应。

3. 提示主题。其作用类似于一些图书页边的 recap（扼要重述），有助于读者“在脑中读出目录”（这是 Ruby 专家 Obi Fernandez 对阅读技术图书的建议）来。

4. 知识更新和扩展。软件研发技术的发展日新月异，时光毕竟会在这些经典身上留下一些痕迹，我们力争根据注释时的最新技术进展为读者提供新的信息。这个工作并不容易，像《重构》这样作者一直维护着网站（即 refactoring.com）的实在不多，《企业应用架构模式》和《程序员修炼之道》还算有勘误，而《敏捷软件开发》的网页上只有书的一些摘录、评论和到 Amazon 的链接，连勘误都没有。好在，经典毕竟是经典，这方面需要做的工作总量倒是不大。

按我们最初的设计，如果能在注释版中增加一些导读性、读书心得式的文字，应该会更加完美，但是非常遗憾，由于工作量大、难以找到合适的人选、尺度很难把握等等原因，这次虽然做出了尝试，但并不令人满意。我们计划在

图灵网站（www.turingbook.com）上提供类似的信息，构建一个“一起读经典”的社区阅读环境。分享是软件开发的原动力，我们期盼得到有更多的读者和业界专家能够以各种方式加入进来。

这种注释版是一种尝试，由于注释者水平和个人喜好各异，各本书的尺度也会有所差异，由于能力和时间问题，肯定会有各种各样的疏漏和讹误，欢迎大家批评指正。我们的报错信箱是：errata@turingbook.com。同时也可以在图灵网站各本书的配套网页上提交勘误。

图灵编辑部

2007年9月

序

作为审稿人，我有了提早阅读本书的机会。即使还处在初稿阶段，它就已经是一本很好的书了。Dave Thomas和Andy Hunt有话要说，并且深谙阐述之道。我了解他们所做的事情，知道他们的方式是行之有效的。我主动要求来撰写这个序，从而有机会来解释其中的原因。

简而言之，本书将告诉你怎样以一种训练有素的方式编程。也许你认为这不是一件困难的事情，但事情往往并非如此。为什么？原因之一是，并非所有的编程图书都是由程序员撰写的。其中许多图书是由语言设计者，或是与他们有合作关系的专业技术作者们编撰而成的，意在推销其作品。那些书所讲的只是怎样使用某种编程语言进行表达——这当然很重要，但这只是程序员工作的一小部分而已。

除了使用编程语言进行表达之外，程序员还需要做些什么呢？嗯，这是一个更深入的问题了。大多数程序员都不太容易说清楚自己的工作。编程是一项需要兼顾各种细节的工作，留意这些细节需要花费精力。时间流逝，代码被编写出来，看看吧，其中全是语句。如果你不假思索，也许会以为编程不过就是输入某种编程语言的语句。这当然错了，但搜遍书店的编程专柜，你却讲不出错在何处。

在本书中，Dave和Andy告诉我们怎样以一种训练有素的方式编程。他们怎么会如此聪明呢？他们不也和其他程序员一样，需要专注各种细节吗？答案是，他们在工作时，会留意自己所做的事情，然后试着加以改善。

设想你正在参加一个会议。你或许在想：这个会议没完没了，还不如去编写程序呢。而Dave和Andy会想，他们为什么要开会，是否有其他方式能够取代开会，而且一些事情可以自动决定，以使开会推后。然后他们就会这样去做。

这就是Dave和Andy的思考方式。开会并不是阻碍他们编程的事情。开会就是编程，而且是能够改善的编程。我之所以知道他们这样的思考方式，是因为这是书中的第二条建议：思考你的工作。

想像一下吧，他们就这样思考了几年，很快就积累了一套解决方案。而他们又在随后的几年中在工作中使用这些解决方案，放弃了其中太过困难或者不

能总是产生结果的解决方案。于是，这样的做法就基本上定义了pragmatic（注重实效）的含义。现在他们又用了一、两年来写出他们的解决方案。你也许会想，这些信息肯定是座“金矿”。完全正确。

本书中，两位作者讲述了他们的编程之道，而且是以一种可以实践的方式叙述的。这种方式本身的意义更为深远。让我来解释一下。

作者一直在小心地避免提出什么新的软件开发理论。这是值得庆幸的，因为如果他们那么做，就不得不为了证实自己的理论而对各章进行“处理”。这样的“处理”在物理这样的学科中是一种传统，在这些学科中，理论最终不是成为定律，就是被静静地抛弃。而编程所具有的定律非常少（如果说还有一些的话），所以围绕那些要成为定律的东西所形成的编程建议写出来也许很好，而在实践中却无法令人满意。这也是那么多方法学图书误入歧途的原因所在。

我研究这个问题已经十多年了，发现一种称为模式语言（pattern language）的方法最有前途。简单说来，模式（pattern）就是解决方案，而模式语言就是彼此相辅相成的若干解决方案组成的系统。围绕对这些系统的探索，已经形成了一个整体社区。

本书并不只是许多建议的汇集，它是另外一种形式的模式语言。我之所以这样说，是因为其中每条建议都源自作者的经验，是非常具体的，而且与其他建议联系起来形成了一个系统。这些都是能够学习并遵循的模式语言的特征。在本书中它们同样发挥着作用。

你可以遵循书中的建议，因为它们是具体的。你不会发现含混不清的抽象之辞。Dave和Andy紧贴你的需要，每条建议仿佛都是使你的编程生涯如虎添翼的必备策略。他们建议尽量简单，通过讲故事来解释，使用轻松的笔触，然后还给出了各种问题的解答。

不仅如此。在阅读了10或15条建议之后，你将开始看到另一层面的工作。我们有时称之为QWAN，即quality without a name（无名特质）的简称。本书的理念将渗入你的意识，并与你自己的理念融合在一起。它并不宣扬什么，只是讲述行之有效的东西。这正是本书精彩之处：它体现了自己的哲学，而且是以朴实无华的方式。

它已经在你手中了：一本易读也易用的、完整讲述编程实践的书。我一直在不断讲述它为何有效，而你关心的也许只是它是否有效。它确实有效，你马上就会看到。

——*Ward Cunningham*

软件模式和极限编程的先驱，Wiki之父

前　言

本书将帮助你成为更为出色的程序员。

无论你是单独的开发者，大型项目团队中的一员，还是同时面对许多客户的咨询师，本书都将帮助你作为个体更好地完成工作。本书不是理论图书——我们专注于实践性的话题，专注于使你凭借自己的经验做出更明智的决策。pragmatic（注重实效）一词来自拉丁语的 pragmaticus，即“精于事务”，后者又源自希腊语的πρατειν，意为“to do”。是的，这是一本关于“实践”的书。

编程是一门技艺。用最简单的话来说，编程就是让计算机做你（或你的用户）想要它做的事情。作为程序员，你既是倾听者，又是指导者；既是解释者，又是发号施令者。你努力捕获难以表述的需求，并找到表达它们的方式，让机器能够正确处理；你努力记录你的工作，以便其他人能够理解它；你还努力进行设计，以便其他人能够在此基础上进一步开发；另外，你还努力在项目时钟不停歇的“嘀嗒”声中完成这些工作。你每天都在创造小小的奇迹。

编程是一项艰难的工作。

有许多人要帮助你，工具商吹嘘其产品的神奇效果，方法学专家承诺他们的方法肯定有效。每个人都声称他们的编程语言是最好的，而每一种操作系统都能解决所有问题。

当然，所有这些都不是真的，并不存在简单的答案，也不存在最佳解决方案，无论工具、语言还是操作系统。只有在某些特定情形下更为适用的系统。

这正是注重实效的用武之地。你不应该局限于任何特定的技术，而是应该拥有足够广博的背景和经验，以让你能在特定情况下选择好的解决方案。你的背景源自对计算机科学基本原理的理解，而你的经验来自广泛的实际项目。理论与实践的结合将使你无比强大。

应该调整方法，适应当前的环境。判断影响项目的所有因素的相对重要性，再利用你的经验制定适合的解决方案。随着工作的进展不断这样做。注重实效的程序员不仅完成工作，而且要漂亮地完成。

读者对象

本书的目标读者是有志于提高自己编程效率的程序员。或许你正在为自己没有发挥出真正的潜力而感到灰心；或许你注意到同事们似乎在使用一些工具，使他们比你效率更高；或许你现在的工作使用的是些老技术，你希望知道怎样把新的思想应用于手头的工作。

我们不会假装自己知道所有的（或者大部分）答案，我们的想法也并不适用于所有情形。我们要说的是，如果你遵循我们的方法，将迅速获取经验，生产效率将会提高，并且还能对整个开发过程有更好的理解，最终，你能够编写更好的软件。

注重实效程序员的特征

每一位开发者都是独特的，有自己的强处和弱点、喜好和厌恶。随着时间的流逝，每一位开发者都会营造出自己的个人环境。这个环境会强烈地反映这位程序员的个性，就像其业余爱好、衣着或是发型一样。但是，如果你是一位注重实效的程序员，就会具有许多下列特征：

- **早期的采纳者/快速的适应者**。你具有技术上的直觉，喜爱试验各种事物。给你一样新东西，你能很快地掌握它，并与其他知识结合起来。你的自信源自经验。
- **好奇**。你喜欢提问。“那真漂亮啊——你是怎么做的？”“你用那个库时有问题吗？”“我听人说起过 BeOS，它是什么东西？”“符号链接是怎样实现的？”你孜孜不倦地收集各种小知识，而每一条小知识都有可能影响到以后的某项决策。
- **审慎的思考者**。你很少在没有了解事实之前人云亦云，当同事说“因为以前就是这么做的”或者厂商承诺提供全部问题的解决方案时，你就应该在心中打一个大大的问号。
- **现实**。你会努力理解所面临的每个问题的本质。这种注重现实的态度会使你更好地认识到任务的难度和工作量，认识到困难和艰巨性能够给予你坚持不懈的毅力。
- **广博**。你会尽力熟悉广泛的技术和环境，并且与各种新发展俱进。虽然现在的工作也许只要求你某方面的专才，但你将总是能够转向新的领域，迎接新的挑战。

我们把最基本的特征留到了最后。所有注重实效的程序员都应具备这些特征。这些特征是如此重要，以至于需要用提示的方式来陈述：