

# 《代码大全》

## 图书基本信息

书名：《代码大全》

13位ISBN编号：9787121033629

10位ISBN编号：7121033623

出版时间：2006-12

出版社：电子工业出版社

作者：迈克康奈尔

页数：914

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：[www.tushu111.com](http://www.tushu111.com)

# 《代码大全》

## 内容概要

《代码大全(第2版)(英文版)》由电子工业出版社出版。

## 作者简介

Steve McConnell 是Construx公司首席软件工程师。他是软件工程知识体系（SWEBOK）项目构建知识领域的先驱。Steve曾就职于微软、波音以及西雅图地区的一些公司，从事软件工程的研究。Steve McConnell是以下著作的作者：《快速开发Rapid Development》（1996）、《软件项目长存之道Software Project Survival Guide》（1998）、和《专业软件开发Professional Software Development》（2004）的作者。他的书作为杰出软件开发书籍，曾两次获得Software Development杂志的优震撼大奖。1998年，Steve被Software Development杂志的读者评为软件业最具影响力的三大人物之一，与比尔·盖茨（Bill Gates）和李纳斯·托瓦兹（Linus Torvalds）齐名。而且，Steve还是SPC（Software Productivity Center，加拿大软件进程改进公司）的ESTIMATE Professional（的一款计划和估算工具）主要开发者，Software Development Productivity award（软件开发生产力大奖）的获得者。Steve从1984年就开始从事桌面软件产业，现在在快速开发方法论、工程估算、软件架构实施、性能调整、系统整合、和第三方合同管理方面已经具有专业的技术。

## 书籍目录

Preface Acknowledgments List of Checklists List of Tables List of Figures Part I Laying the Foundation 1  
Welcome to Software Construction 1.1 What Is Software Construction? 1.2  
Why Is Software Construction Important? 1.3 How to Read This Book 2  
Metaphors for a Richer Understanding of Software Development 2.1 The Importance of Metaphors 2.2  
How to Use Software Metaphors 2.3 Common Software Metaphors 3  
Measure Twice, Cut Once: Upstream Prerequisites 3.1 Importance of Prerequisites 3.2  
Determine the Kind of Software You're Working On 3.3 Problem-Definition Prerequisite 3.4  
Requirements Prerequisite 3.5 Architecture Prerequisite 3.6 Amount of Time to Spend on Upstream Prerequisites 4  
Key Construction Decisions 4.1 Choice of Programming Language 4.2 Programming Conventions 4.3  
Your Location on the Technology Wave 4.4 Selection of Major Construction Practices Part II  
Creating High-Quality Code 5 Design in Construction 5.1 Design Challenges 5.2 Key Design Concepts 5.3  
Design Building Blocks: Heuristics 5.4 Design Practices 5.5 Comments on Popular Methodologies 6 Working Classes 6.1  
Class Foundations: Abstract Data Types (ADTs) 6.2 Good Class Interfaces 6.3 Design and Implementation Issues 6.4  
Reasons to Create a Class 6.5 Language-Specific Issues 6.6 Beyond Classes: Packages 7 High-Quality Routines 7.1  
Valid Reasons to Create a Routine 7.2 Design at the Routine Level 7.3 Good Routine Names 7.4  
How Long Can a Routine Be? 7.5 How to Use Routine Parameters 7.6 Special Considerations in the Use of Functions 7.7  
Macro Routines and Inline Routines 8 Defensive Programming 8.1 Protecting Your Program from Invalid Inputs 8.2  
Assertions 8.3 Error-Handling Techniques 8.4 Exceptions 8.5  
Barricade Your Program to Contain the Damage Caused by Errors 8.6 Debugging Aids 8.7  
Determining How Much Defensive Programming to Leave in Production Code 8.8  
Being Defensive About Defensive Programming 9 The Pseudocode Programming Process 9.1  
Summary of Steps in Building Classes and Routines 9.2 Pseudocode for Pros 9.3  
Constructing Routines by Using the PPP 9.4 Alternatives to the PPP Part III Variables 10  
General Issues in Using Variables 10.1 Data Literacy 10.2 Making Variable Declarations Easy 10.3  
Guidelines for Initializing Variables 10.4 Scope 10.5 Persistence 10.6 Binding Time 10.7  
Relationship Between Data Types and Control Structures 10.8 Using Each Variable for Exactly One Purpose 11  
The Power of Variable Names 11.1 Considerations in Choosing Good Names 11.2 Naming Specific Types of Data 11.3  
The Power of Naming Conventions 11.4 Informal Naming Conventions 11.5 Standardized Prefixes 11.6  
Creating Short Names That Are Readable 11.7 Kinds of Names to Avoid 12 Fundamental Data Types 12.1  
Numbers in General 12.2 Integers 12.3 Floating-Point Numbers 12.4 Characters and Strings 12.5 Boolean Variables 12.6  
Enumerated Types 12.7 Named Constants 12.8 Arrays 12.9 Creating Your Own Types (Type Aliasing) 13  
Unusual Data Types 13.1 Structures 13.2 Pointers 13.3 Global Data Part IV Statements 14  
Organizing Straight-Line Code 14.1 Statements That Must Be in a Specific Order 14.2  
Statements Whose Order Doesn't Matter 15 Using Conditionals 15.1 if Statements 15.2 case Statements 16  
Controlling Loops 16.1 Selecting the Kind of Loop 16.2 Controlling the Loop 16.3  
Creating Loops Easily--From the Inside Out 16.4 Correspondence Between Loops and Arrays 17  
Unusual Control Structures 17.1 Multiple Returns from a Routine 17.2 Recursion 17.3 goto 17.4  
Perspective on Unusual Control Structures 18 Table-Driven Methods 18.1  
General Considerations in Using Table-Driven Methods 18.2 Direct Access Tables 18.3 Indexed Access Tables 18.4  
Stair-Step Access Tables 18.5 Other Examples of Table Lookups 19 General Control Issues 19.1 Boolean Expressions 19.2  
Compound Statements (Blocks) 19.3 Null Statements 19.4 Taming Dangerously Deep Nesting 19.5  
A Programming Foundation: Structured Programming 19.6 Control Structures and Complexity Part V  
Code Improvements 20 The Software-Quality Landscape 20.1 Characteristics of Software Quality 20.2  
Techniques for Improving Software Quality 20.3 Relative Effectiveness of Quality Techniques 20.4  
When to Do Quality Assurance 20.5 The General Principle of Software Quality 21 Collaborative Construction 21.1  
Overview of Collaborative Development Practices 21.2 Pair Programming 21.3 Formal Inspections 21.4  
Other Kinds of Collaborative Development Practices 22 Developer Testing 22.1

Role of Developer Testing in Software Quality 22.2 Recommended Approach to Developer Testing 22.3  
Bag of Testing Tricks 22.4 Typical Errors 22.5 Test-Support Tools 22.6 Improving Your Testing 22.7  
Keeping Test Records 23 Debugging 23.1 Overview of Debugging Issues 23.2 Finding a Defect 23.3 Fixing a Defect 23.4  
Psychological Considerations in Debugging 23.5 Debugging Tools--Obvious and Not-So-Obvious 24 Refactoring 24.1  
Kinds of Software Evolution 24.2 Introduction to Refactoring 24.3 Specific Refactorings 24.4 Refactoring Safely 24.5  
Refactoring Strategies 25 Code-Tuning Strategies 25.1 Performance Overview 25.2 Introduction to Code Tuning 25.3  
Kinds of Fat and Molasses 25.4 Measurement 25.5 Iteration 25.6 Summary of the Approach to Code Tuning 26  
Code-Tuning Techniques 26.1 Logic 26.2 Loops 26.3 Data Transformations 26.4 Expressions 26.5 Routines 26.6  
Recoding in a Low-Level Language 26.7 The More Things Change, the More They Stay the Same Part VI  
System Considerations 27 How Program Size Affects Construction 27.1 Communication and Size 27.2  
Range of Project Sizes 27.3 Effect of Project Size on Errors 27.4 Effect of Project Size on Productivity 27.5  
Effect of Project Size on Development Activities 28 Managing Construction 28.1 Encouraging Good Coding 28.2  
Configuration Management 28.3 Estimating a Construction Schedule 28.4 Measurement 28.5  
Treating Programmers as People 28.6 Managing Your Manager 29 Integration 29.1  
Importance of the Integration Approach 29.2 Integration Frequency--Phased or Incremental? 29.3  
Incremental Integration Strategies 29.4 Daily Build and Smoke Test 30 Programming Tools 30.1 Design Tools 30.2  
Source-Code Tools 30.3 Executable-Code Tools 30.4 Tool-Oriented Environments 30.5  
Building Your Own Programming Tools 30.6 Tool Fantasy Land Part VII Software Craftsmanship 31 Layout and Style 31.1  
Layout Fundamentals 31.2 Layout Techniques 31.3 Layout Styles 31.4 Laying Out Control Structures 31.5  
Laying Out Individual Statements 31.6 Laying Out Comments 31.7 Laying Out Routines 31.8 Laying Out Classes 32  
Self-Documenting Code 32.1 External Documentation 32.2 Programming Style as Documentation 32.3  
To Comment or Not to Comment 32.4 Keys to Effective Comments 32.5 Commenting Techniques 32.6  
IEEE Standards 33 Personal Character 33.1 Isn't Personal Character Off the Topic? 33.2 Intelligence and Humility 33.3  
Curiosity 33.4 Intellectual Honesty 33.5 Communication and Cooperation 33.6 Creativity and Discipline 33.7  
Laziness 33.8 Characteristics That Don't Matter As Much As You Might Think 33.9 Habits 34  
Themes in Software Craftsmanship 34.1 Conquer Complexity 34.2 Pick Your Process 34.3  
Write Programs for People First, Computers Second 34.4 Program into Your Language, Not in It 34.5  
Focus Your Attention with the Help of Conventions 34.6 Program in Terms of the Problem Domain 34.7  
Watch for Falling Rocks 34.8 Iterate, Repeatedly, Again and Again 34.9 Thou Shalt Rend Software and Religion Asunder 35  
Where to Find More Information 35.1 Information About Software Construction 35.2 Topics Beyond Construction 35.3  
Periodicals 35.4 A Software Developer's Reading Plan 35.5 Joining a Professional Organization Bibliography Index

# 《代码大全》

## 编辑推荐

“《代码大全》第1版在我看来堪称软件工程领域的经典之作——而第2版则更棒！”——Ralph Johnson, 伊利诺伊州立大学:《设计模式》(Design Patterns)作者之一 “无论您是新手还是经验丰富的开发人员,《代码大全》(第2版)都能教会您思考编程的最佳方法。”——Jeffrey Richter (WWW.wintellect.com),《Microsoft NET框架程序设计》(AppliedMicrosoft.NET Framework Programming)作者 “这本书是讲述软件构建的权威指南——准备孤身前往荒岛的程序员只要带上这本书就足够了。”——Diomidis Spinellis,《代码阅读方法与实践》(Code Reading: The Open Source Perspective)作者 “Steve McConnell是一位既在一线实践,又能把其中奥妙讲个明白的少数人之一。”——John Vlissides, IBM研究院; 《设计模式》(Design Patterns)作者之一 “Steve McConnell比任何人都懂得如何构建软件;我们十分庆幸他能将其所有的深邃见解和实践经验写成这样一本重要而新颖的图书。”——“Visual Basic之父” Alan Cooper,《软件观念革命》(About Face 2.0)作者 Steve McConnell的原作《代码大全》(第1版)是公认的关于编程的最佳实践指南之一,在过去的十多年间,本书一直在帮助开发人员编写更好的软件。现在,作者将这本经典著作全新演绎,融入了最前沿的实践技术,加入了上百个崭新的代码示例,充分展示了软件构建的艺术性和科学性。McConnell汇集了来自研究机构、学术界以及业界日常实践的主要知识,把最高效的技术和最重要的原理交织融会为本既清晰又实用的指南。无论您的经验水平如何,也不管您在怎样的开发环境中工作,也无论项目是大是小,本书都将激发您的思维——并帮助您构建高品质的代码。

从本书可以了解到如下这些经久不衰的技术与策略:

- 做出具有最小复杂度和最大创造性的设计
- 从协作式的开发中获益
- 应用防御式编程技术来减少并排查错误
- 发掘重构或改善代码的机会,并安全可靠地进行代码重构和改善
- 结合项目的规格合理选用恰当的构建技术
- 快速而有效地排除问题
- 尽早地正确解决关键构建问题
- 分别在项目的早期、中期以及后期加强代码的质量

# 《代码大全》

## 精彩短评

- 1、重点的读了10多章，还有一些部分是快速阅读的。这本书解决了我对写代码的大部分疑惑。而且深感降低复杂度，以人为本的重要性。
  - 2、这本书已经被我读了2遍，还想读。最好结合项目研读。
  - 3、读来像一个循循善诱的老者！
  - 4、几乎涵盖了软件开发过程中的各方面，整体认识。
  - 5、程序员的行为规范，通过分析问题，解决问题的方式，更好的理解代码的本质和程序员文化。
  - 6、好书
  - 7、外国的书质量就是好，
  - 8、阐述设计模式、编程习惯。必读
  - 9、篇幅很长，个人也只是初略翻过一遍。
- 总体而言，这本书非常经典，但你可以有其他更好的选择，比如代码层面的《编写可读代码的艺术》，软件工程方面的《构建之法》。
- 10、教你如何构建代码
  - 11、经典
  - 12、有些内容不喜欢，有些用来举例的语言也不是很喜欢，关于性能和代码布局的章节保留意见，不过大部分见解还是认同的=。=
  - 13、软件工程，是一门技艺，这个是指导说明
  - 14、在历经6~8个月后，第一遍终于读完了。。好累啊，感觉不会再读第二遍了。不过即使这样懒散地读，收获也超出了想象，当有了方向，有了专业内的价值观后，看一些事情会清晰不少。
  - 15、读了文前部分 版权取得有点小曲折 不过结果不错 经得起时间的检验 经典就是经典！
  - 16、很好的书，包括了软件工程里面所有的东西。
  - 17、修内功
  - 18、很好地利用你的聪明要比你的聪明有意义的多。。。
  - 19、前面5章收获不大，更多的是 中间的部分，如何实际写出一种高效优美的代码，如何封装类，构建子程序，如何定义好的命名。同重构有很多部分的重叠。其中感触最深的一节，软件工程最首要的核心技术：控制复杂度！！！控制复杂度！！！！
  - 20、1000页的代码大全也没有看起来那么厚,如同小说一般的看完了,颇为受益.
  - 21、很早就知道这书不错，这次看完了发现看晚了，如此提纲之作，越早读越好。
  - 22、点到即止，还不能完全指导实践，很多东西还是要自己摸索体会，比如复杂度的管理
  - 23、突然发现，我等弱想进步，还是有迹可循的。// mark，56部分待读。
  - 24、111
  - 25、基本上可以列为基础必读书了。除了少数过时的内容（例如把快排当作有难度的高级算法），要注意的就是不要陷入近一千页的篇幅中去，有思考、有快慢地读，don't panic，这本书会带来益处的。
  - 26、涵盖了编程实践的方方面面，满满的干货，看起来很过瘾。就像作者说的，大多数程序员在工作中遇到的难题和疑问，都已经有了很多讨论和实用经验，然而这些基本都存在于论文里，极少人会去翻看，这本书是对软件行业几十年来优秀智力成果的归纳。对于程序员来讲，越早看这本书，获得的收益越大。
  - 27、太厚了，没有看完，只能以后在积累项目经验的时候反复拿具体的例子来实践，我觉得《构建之法》是迷你版的《代码大全》
  - 28、第二次开始 这次是看风格与布局 31-34章，已书排版比喻代码风格，新颖 贴切
  - 29、不错，很多小细节
  - 30、好厚，终于断断续续看完。后面几章还可以。
  - 31、软件构建之实践指南
  - 32、进可砸人，退可装X；好吧，说真的，我不知道能不能看完它。
  - 33、看起来很舒服的一本书。

## 《代码大全》

- 34、控制编程复杂度，面很广，有些可以跳过
- 35、nice
- 36、大致扫了一遍，适合一点点读，体会每一个细节。
- 37、这部大块头确实经典，涉及到了软件开发的方方面面。有点后悔没有早些阅读，值得推荐给还没读过的朋友。它并不是针对某种语言的武林秘籍，应该可以看作是基础内功修炼吧
  
- 38、这本书应该叫《软件构建指南》《大鸟带你飞》
- 39、计算机大牛的编程方法.
- 40、great 如果你是个程序员的话，都应该读一读
- 41、清晰，有内容的设计书，每一个观点都有实证研究或强有力的理由。基本上，一个讲设计的书必须写得清晰有条理，否则你很难相信写书的人能够设计出清晰有条理的软件。
- 42、五星已经无法体现这本书到底有多好。反复的读了若干次后，如今我遇到任何关于软件的问题还是会去这本书中寻找答案。
- 43、软件工程中的《九阴真经》
- 44、初级程序员要多翻翻。
- 45、非常厚的巨著，包括许多有效的代码片段。
- 46、一本特别厚的书，适合有编程经验的人看
- 47、和想象的不同,形而向上的东西,看看就好,不建议没有编程经验的看这个,头很大
- 48、讲的非常好，其中的大部分问题，在以前的工作当中都遇到过。
- 49、经典 程序员必看书籍之一
- 50、这本书没有讲具体的技术，讲的是一些原则和要求，按照这些方法做，能够让代码构建更轻松，错误更少~~这本书不太适合从头读到尾，直接看自己关心的章节即可~



## 精彩书评

- 1、推荐刚毕业的看看，之后的就算了。不亏是大全，从项目开始到结束，面面俱到，但对我来讲有用的东西就不多了。什么都涵盖，难免不够深入。
- 2、每天读6页，读了三四个月，中间有时候偷了一下懒，终于读完了。回想过来，有些东西是可以一下子就明白的，例如关于具名常量、格式等等。有一些需要慢慢在以后的实践中慢慢推敲。而且这本书信息量超乎寻常的大，不仅仅指书本身还有每个章节中的引用和书本后面的引用。还有最后一章，专门有一章的推荐读物，全部展开，可能穷尽一生都未必能够读完。所以需要好好的细读，慢慢的斟酌。
- 3、作者说:创建一个软件的最彻底的办法并不是创建——而是去购买一个软件，你可以购买数据库管理系统、屏幕生成程序、报告生成程序和图形环境。在苹果公司 Macintosh 或者微软公司.Windows 环境下编程的一个主要优点是你可以自动获得许多功能；图形程序，对话框管理程序，键盘输入与处理程序，可以自动与任何打印机或者显示器工作的代码，等等...那能写出这么一大本书的人不知道蚊子部分有参考别人的呢,因为这本书的内容实在是太丰富了,而且几乎面面俱到,太神奇了
- 4、最近在《代码大全》这本书,包括的内容非常多,从软件设计到代码开发,团队管理都有,更像是一个软件编程领域的百科全书.但是,对于书中提到的一点印象最为深刻,其实在《人月神话》和《卓有成效的程序员》这两本书都有提到,那就是:软件设计与开发的核心就在于控制复杂度这句话的核心其实包括几个问题:软件开发的本质问题性难题是复杂度?如何可以一定程序的降低复杂度?杂耍抛球其中,书中对于软件设计必须控制复杂度的解释原因是:没有谁的大脑能容得下一个现代计算机程序,也就是说,作为软件开发人员,我们不应该试着在同一时间把整个程序都塞进自己的大脑,而应该试着以某种方式去组织程序,以便能在同一个时刻可以专注于一个地方.这么做的目的是尽量减少同一时间所要考虑的程序量.你可以把它想做是一种心理上的杂耍(边抛边接:通过轮流抛接使两个或者两个以上的物体同时保持于空中)-程序要求你在空中保持的(精神上的)球越多,你就越可能漏掉其中的某一个,从而导致设计或者编码上的错误——代码大全当我读到这一段的时候,感觉这本书的作者真是说出了软件开发者心中的痛点,我也认为这段话可以说真本书的一个核心,其实代码大全的所有部分都是在围绕『如何降低软件开发中的复杂度』.作者用这种,杂耍抛球的方式非常形象的比喻了,我们的大脑(生物结构上)本质的局限性导致的.曾经美国人有一个非常有名的调查,人类的大脑短期记忆能够容纳最多的不连续信息数就是7,加而或减二具体可以参考心理学上被引用最多的一篇文章之一:魔数七,加二或者减二:人类处理能力的局限性.而现实问题域中,我们要处理的变量何止是7!所以我们根本不可能同时让这么多变量一起出现在我们大脑中.没有银弹从哲学的角度来说,柏拉图认为,任何事物都有两个属性:本质属性与偶然属性.通过此我们可以将软件行业遇到的问题也分为两类,那么软件开发过程中本质性的难题是什么?《人月神话》的作者认为,软件行业中遇到的非根本问题(偶然属性)都会随着时间发展,技术的提升,会逐步解决.但是开发中的根本性问题-对于现实复杂世界本质的概念的复杂性是无法降低的.一个软件系统有大量的状态,存在大量不同元素的相互叠加.这使得软件系统的复杂性以指数的形式增长.而且,这些复杂度是软件系统的根本属性,而不像数学和物理中那样可以建立简化的模型而忽略复杂的次要因素.《人月神话》中对于本质的复杂性无法避免,起了一个名词,日后基本成了这个行业的标志:木有银弹.不要期望通过一种万能药能解决软件行业中所有的复杂性问题.所以当每次一种新的技术/语言/框架等出现的时候,当有些人大喊可以颠覆以前软件开发中的所有问题的时候,这时候你就要小心了.心理默默的告诉自己,没有银弹!降熵其实,软件的复杂度从某种意义上,用物理学第二定律来理解和加强.物理学第二定律又叫做,熵定律:自然过程中,一个孤立系统的总混乱度(即“熵”)不会减小.换成是软件行业的背景就是,用《程序员修炼之道》里面的解释就是:软件的熵总是倾向于最大化的,程序员们称之为“软件腐烂”.程序员只有在开发过程中,不断的通过外部做功(思考,主动性的思考和改代码),不断进行代码重构与整理,通过外部系统注入能量,来降低整个软件系统的熵,是整个软件达到有序的状态.为此软件开发行业提出了一系列的原则和指导方法,重构/单元测试/模块化设计/KISS原则/面向接口编程/模式设计/分布式系统...等等如此,其实你都会发现,这些方法和指导原则,都是告诉程序员,在软件开发的过程中,通过这些方法降低软件系统整体的复杂度,以便后期更好的维护与开发.当软件复杂度可以得到很好的控制,而不是让软件的熵无限的增长,那么这个软件系统的寿命也就会很长,得到更好的维护性.所以,最后,我们知道:软件开发中的本质难题是复杂度,那么我们在之后开发中应该时刻的主动思考和做功:如何通过不断的代码重构降低整体的复杂度.保持我们的







## 版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:[www.tushu111.com](http://www.tushu111.com)