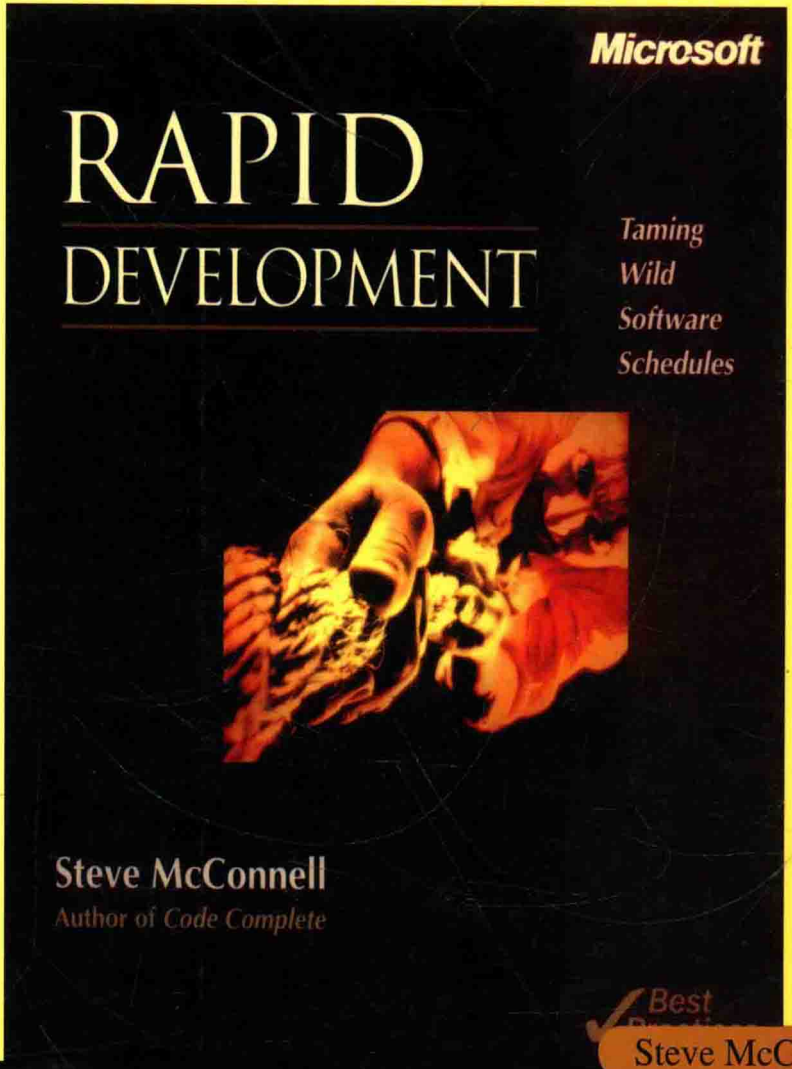
经 典 原 版 书 库

快速软件开发

（英文版）



Steve McConnell 著

机械工业出版社
China Machine Press

Microsoft

经 典 原 版 书 库

快速软件开发

（英文版）

Rapid Development

Steve McConnell 著

机械工业出版社
China Machine Press

Steve McConnell: Rapid Development (ISBN 1-55615-900-5)

Copyright © 1996 by Microsoft Corporation.

Original English language edition copyright © 1996 by Steve McConnell.

Published by arrangement with the original publisher, Microsoft Press, a division of Microsoft Corporation, Redmond, Washington, U.S.A. All rights reserved.

本书影印版由美国微软出版社授权机械工业出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2002-5814

图书在版编目（CIP）数据

快速软件开发（英文版）/麦康奈尔（McConnell, S.）著. -北京：机械工业出版社，2003.3

（经典原版书库）

书名原文：Rapid Development

ISBN 7-111-11750-6

Ⅰ.快… Ⅱ.麦… Ⅲ.软件开发－英文 Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2003）第013546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：华 章

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003年3月第1版第1次印刷

787mm×1092mm 1/16·42.25印张

印数：0 001-3 000册

定价：58.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国

家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

此为试读，需要完整PDF请访问：www.ertongbook.com

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

Case Studies

2-1. Rapid Development Without a Clear Strategy 6

2-2. Rapid Development with a Clear Strategy 25

3-1. Classic Mistakes 29

4-1. Lack of Fundamentals 52

5-1. Lack of Contractor Risk Management 82

5-2. Systematic Risk Management 103

6-1. Wandering in the Fuzzy Front End 124

7-1. Ineffective Lifecycle Model Selection 134

7-2. Effective Lifecycle Model Selection 159

8-1. Seat-of-the-Pants Project Estimation 164

8-2. Careful Project Estimation 200

9-1. A Successful Schedule Negotiation 229

10-1. The Requirements Club 234

10-2. The Requirements Club Revisited 246

11-1. A Disheartening Lunch with the Boss 250

11-2. A Highly Motivational Environment 270

12-1. You Call This a Team? 274

12-2. A High-Performance Team 277

12-3. Typical Team-Member Selection 282

12-4. A Second High-Performance Team 294

13-1. Mismatch Between Project Objectives and Team Structure 297

13-2. Good Match Between Project Objectives and Team Structure 315

14-1. Managing Change Effectively 342

15-1. Ineffective Tool Use 346

15-2. Effective Tool Use 368

16-1. An Unsuccessful Project Recovery 372

16-2. A Successful Project Recovery 385

Reference Tables

2-1. Characteristics of Standard Approaches to Schedule-Oriented Development 18

2-2. Code-Like-Hell Approach Compared to This Book's Approach 24

3-1. Summary of Classic Mistakes 49

5-1. Levels of Risk Management 84

5-2. Most Common Schedule Risks 86

5-3. Potential Schedule Risks 87

5-4. Example of a Risk-Assessment Table 92

5-5. Example of a Prioritized Risk-Assessment Table 95

5-6. Means of Controlling the Most Common Schedule Risks 98

5-7. Example of a "Top-10 Risks List" 101

6-1. Approximate Activity Breakdown by Size of Project 122

7-1. Lifecycle Model Strengths and Weaknesses 156

8-1. Estimate Multipliers by Project Phase 169

8-2. Function-Point Multipliers 176

8-3. Example of Computing the Number of Function Points 177

8-4. Example of a Risk-Quantification Estimate 180

8-5. Example of a Case-Based Estimate 181

8-6. Example of a Confidence-Factor Estimate 182

8-7. Exponents for Computing Schedules from Function Points 185

8-8. Shortest Possible Schedules 190

8-9. Efficient Schedules 194

8-10. Nominal Schedules 196

8-11. Example of a Single-Point–Estimation History 197

8-12. Example of a Range-Estimation History 198

9-1. Scheduling History of Word for Windows 1.0 208

11-1. Comparison of Motivators for Programmer Analysts vs. Managers and the General Population 252

11-2. Team Performance Ranked Against Objectives That Teams Were Told to Optimize 256

12-1. Practical Guidelines for Team Members and Leaders 295

13-1. Team Objectives and Team Structures 301

15-1. Example of Savings Realized by Switching from a 3GL to a 4GL for 50 Percent of a 32,000 LOC Project 361

15-2. Example of Savings Realized by Switching from a 3GL to a 4GL for 100 Percent of a 32,000 LOC Project 362

III-1. Summary of Best-Practice Candidates 396

III-2. Summary of Best-Practice Evaluations 400

26-1. Examples of Kinds of Measurement Data 470

26-2. Example of Time-Accounting Activities 472

28-1. Vendor-Evaluation Questionnaire 497

28-2. Contract Considerations 498

30-1. Differences in Office Environments Between Best and Worst Performers in a Programming Competition 512

31-1. Approximate Function-Points to Lines-of-Code Conversions 517

31-2. Approximate Language Levels 519

36-1. Example of a Staged-Delivery Schedule for a Word Processor 552

37-1. Project Stakeholders and Their Objectives 560

37-2. Steps in Theory-W Project Management 562

Preface

Software developers are caught on the horns of a dilemma. One horn of the dilemma is that developers are working too hard to have time to learn about effective practices that can solve most development-time problems; the other horn is that they won't get the time until they do learn more about rapid development.

Other problems in our industry can wait. It's hard to justify taking time to learn more about quality when you're under intense schedule pressure to "just ship it." It's hard to learn more about usability when you've worked 20 days in a row and haven't had time to see a movie, go shopping, work out, read the paper, mow your lawn, or play with your kids. Until we as an industry learn to control our schedules and free up time for developers and managers to learn more about their professions, we will never have enough time to put the rest of our house in order.

The development-time problem is pervasive. Several surveys have found that about two-thirds of all projects substantially overrun their estimates (Lederer and Prasad 1992, Gibbs 1994, Standish Group 1994). The average large project misses its planned delivery date by 25 to 50 percent, and the size of the average schedule slip increases with the size of the project (Jones 1994). Year after year, development-speed issues have appeared at the tops of lists of the most critical issues facing the software-development community (Symons 1991).

Although the slow-development problem is pervasive, some organizations are developing rapidly. Researchers have found 10-to-1 differences in productivity between companies within the same industries, and some researchers have found even greater variations (Jones 1994).

The purpose of this book is to provide the groups that are currently on the "1" side of that 10-to-1 ratio with the information they need to move toward the "10" side of the ratio. This book will help you bring your projects under control. It will help you deliver more functionality to your users in less time. You don't have to read the whole book to learn something useful; no matter what state your project is in, you will find practices that will enable you to improve its condition.

Who Should Read This Book?

Slow development affects everyone involved with software development, including developers, managers, clients, and end-users—even their families and friends. Each of these groups has a stake in solving the slow-development problem, and there is something in this book for each of them.

This book is intended to help developers and managers know what's possible, to help managers and clients know what's realistic, and to serve as an avenue of communication between developers, managers, and clients so that they can tailor the best possible approach to meet their schedule, cost, quality, and other goals.

Technical Leads

This book is written primarily with technical leads or team leads in mind. If that's your role, you usually bear primary responsibility for increasing the speed of software development, and this book explains how to do that. It also describes the development-speed limits so that you'll have a firm foundation for distinguishing between realistic improvement programs and wishful-thinking fantasies.

Some of the practices this book describes are wholly technical. As a technical lead, you should have no problem implementing those. Other practices are more management oriented, and you might wonder why they are included here. In writing the book, I have made the simplifying assumption that you are Technical Super Lead—faster than a speeding hacker; more powerful than a loco-manager; able to leap both technical problems and management problems in a single bound. That is somewhat unrealistic, I know, but it saves both of us from the distraction of my constantly saying, "If you're a manager, do this, and if you're a developer, do that." Moreover, assuming that technical leads are responsible for both technical and management practices is not as far-fetched as it might sound. Technical leads are often called upon to make recommendations to upper management about technically oriented management issues, and this book will help prepare you to do that.

Individual Programmers

Many software projects are run by individual programmers or self-managed teams, and that puts individual technical participants into de facto technical-lead roles. If you're in that role, this book will help you improve your development speed for the same reasons that it will help bona fide technical leads.

此为试读，需要完整PDF请访问：www.ertongbook.com

Managers

Managers sometimes think that achieving rapid software development is primarily a technical job. If you're a manager, however, you can usually do as much to improve development speed as your developers can. This book describes many management-level rapid-development practices. Of course, you can also read the technically oriented practices to understand what your developers can do at their level.

Key Benefits of This Book

I conceived of this book as a *Common Sense* for software developers. Like Thomas Paine's original *Common Sense*, which laid out in pragmatic terms why America should secede from Mother England, this book lays out in pragmatic terms why many of our most common views about rapid development are fundamentally broken. These are the times that try developers' souls, and, for that reason, this book advocates its own small revolution in software-development practices.

My view of software development is that software projects can be optimized for any of several goals—lowest defect rate, fastest execution speed, greatest user acceptance, best maintainability, lowest cost, or shortest development schedule. Part of an engineering approach to software is to balance trade-offs: Can you optimize for development time by cutting quality? By cutting usability? By requiring developers to work overtime? When crunch time comes, how much schedule reduction can you ultimately achieve? This book helps answer such key trade-off questions as well as other questions.

Improved development speed. You can use the strategy and best practices described in this book to achieve the maximum possible development speed in your specific circumstances. Over time, most people can realize dramatic improvements in development speed by applying the strategies and practices described in this book. Some best practices won't work on some kinds of projects, but for virtually any kind of project, you'll find other best practices that will. Depending on your circumstances, "maximum development speed" might not be as fast as you'd like, but you'll never be completely out of luck just because you can't use a rapid-development language, are maintaining legacy code, or work in a noisy, unproductive environment.

Rapid-development slant on traditional topics. Some of the practices described in this book aren't typically thought of as rapid-development practices. Practices such as risk management, software-development fundamentals, and lifecycle planning are more commonly thought of as "good software-development practices" than as rapid-development methodologies.

These practices, however, have profound development-speed implications that in many cases dwarf those of the so-called rapid-development methods. This book puts the development-speed benefits of these practices into context with other practices.

Practical focus. To some people, "practical" means "code," and to those people I have to admit that this book might not seem very practical. I've avoided code-focused practices for two reasons. First, I've already written 800 pages about effective coding practices in *Code Complete* (Microsoft Press, 1993). I don't have much more to say about them. Second, it turns out that many of the critical insights about rapid development are not code-focused; they're strategic and philosophical. Sometimes, there is nothing more practical than a good theory.

Quick-reading organization. I've done all I can to present this book's rapid-development information in the most practical way possible. The first 400 pages of the book (Parts I and II) describe a strategy and philosophy of rapid development. About 50 pages of case studies are integrated into that discussion so that you can see how the strategy and philosophy play out in practice. If you don't like case studies, they've been formatted so that you can easily skip them. The rest of the book consists of a set of rapid-development *best practices*. The practices are described in quick-reference format so that you can skim to find the practices that will work best on your projects. The book describes how to use each practice, how much schedule reduction to expect, and what risks to watch out for.

The book also makes extensive use of marginal icons and text to help you quickly find additional information related to the topic you're reading about, avoid classic mistakes, zero in on best practices, and find quantitative support for many of the claims made in this book.

A new way to think about the topic of rapid development. In no other area of software development has there been as much disinformation as in the area of rapid development. Nearly useless development practices have been relentlessly hyped as "rapid-development practices," which has caused many developers to become cynical about claims made for any development practices whatsoever. Other practices are genuinely useful, but they have been hyped so far beyond their real capabilities that they too have contributed to developers' cynicism.

Each tool vendor and each methodology vendor want to convince you that their new silver bullet will be the answer to your development needs. In no other software area do you have to work as hard to separate the wheat from the chaff. This book provides guidelines for analyzing rapid-development information and finding the few grains of truth.

This book provides ready-made mental models that will allow you to assess what the silver-bullet vendors tell you and will also allow you to incorporate new ideas of your own. When someone comes into your office and says, "I just heard about a great new tool from the GigaCorp Silver Bullet Company that will cut our development time by 80 percent!" you will know how to react. It doesn't matter that I haven't said anything specifically about the GigaCorp Silver Bullet Company or their new tool. By the time you finish this book, you'll know what questions to ask, how seriously to take GigaCorp's claims, and how to incorporate their new tool into your development environment, if you decide to do that.

Unlike other books on rapid development, I'm not asking you to put all of your eggs into a single, one-size-fits-all basket. I recognize that different projects have different needs, and that one magic method is usually not enough to solve even one project's schedule problems. I have tried to be skeptical without being cynical—to be critical of practices' effectiveness but to stop short of *assuming* that they don't work. I revisit those old, overhyped practices and salvage some that are still genuinely useful—even if they aren't as useful as they were originally promised to be.

Why is this book about rapid development so big? Developers in the IS, shrink-wrap, military, and software-engineering fields have all discovered valuable rapid-development practices, but the people from these different fields rarely talk to one another. This book collects the most valuable practices from each field, bringing together rapid-development information from a wide variety of sources for the first time.

Does anyone who needs to know about rapid development really have time to read 650 pages about it? Possibly not, but a book half as long would have had to be oversimplified to the point of uselessness. To compensate, I've organized the book so that it can be read quickly and selectively—you can read short snippets while you're traveling or waiting. Chapters 1 and 2 contain the material that you *must* read to understand how to develop products more quickly. After you read those chapters, you can read whatever interests you most.

Why This Book Was Written

Clients' and managers' first response to the problem of slow development is usually to increase the amount of schedule pressure and overtime they heap on developers. Excessive schedule pressure occurs in about 75 percent of all large projects and in close to 100 percent of all very large projects (Jones 1994). Nearly 60 percent of developers report that the level of stress they feel is increasing (Glass 1994c). The average developer in the U.S. works from 48 to 50 hours per week (Krantz 1995). Many work considerably more.

In this environment, it isn't surprising that general job satisfaction of software developers has dropped significantly in the last 15 years (Zawacki 1993), and at a time when the industry desperately needs to be recruiting additional programmers to ease the schedule pressure, developers are spreading the word to their younger sisters, brothers, and children that our field is no fun anymore.

Clearly our field can be fun. Many of us got into it originally because we couldn't believe that people would actually pay us to write software. But something not-so-funny happened on the way to the forum, and that something is intimately bound up with the topic of rapid development.

It's time to start shoring up the dike that separates software developers from the sea of scheduling madness. This book is my attempt to stick a few fingers into that dike, holding the madness at bay long enough to get the job started.

Acknowledgments

Heartfelt thanks go first to Jack Litewka, the project editor, for making the creation of this book a thoroughly constructive and enjoyable experience. Thanks also go to Peggy Herman and Kim Eggleston for the book's design, to Michael Victor for the diagrams, and to Mark Monlux for the terrific illustrations. Sally Brunsman, David Clark, Susanne Freet, Dean Holmes, Wendy Maier, and Heidi Saastamoinen also helped this project go smoothly. Literally dozens of other people contributed to this book in one way or another; I didn't have personal contact with any of them, but I appreciate their contributions, too. (Chief among these, I am told, are layout artist Jeannie McGivern and production manager Jean Trenary of ArtSource and Microsoft Press's proof/copy-edit platoon supervised by Brenda Morris: Richard Carey, Roger LeBlanc, Patrick Forgette, Ron Drummond, Patricia Masserman, Paula Thurman, Jocelyn Elliott, Deborah Long, and Devon Musgrave.)

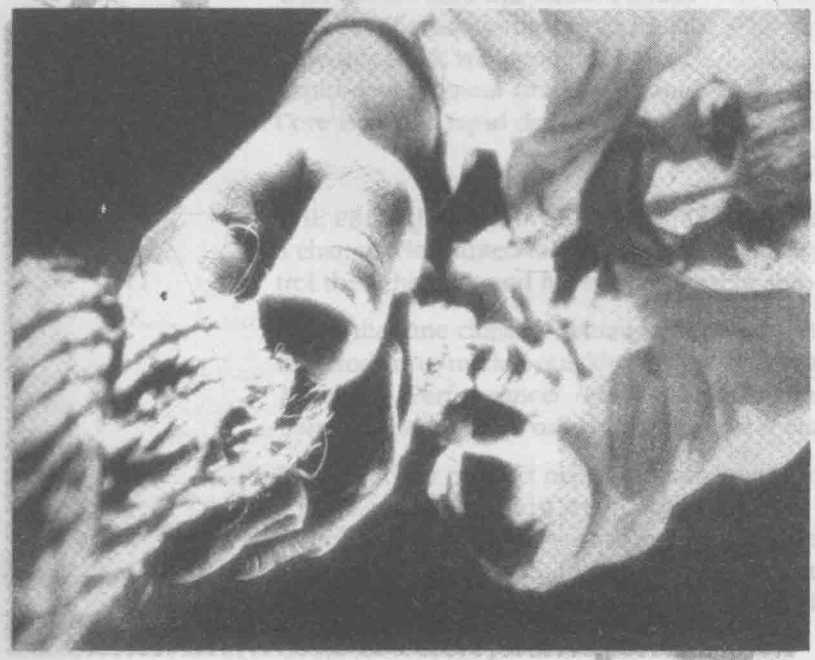
Microsoft Corporation's technical library provided invaluable aid in digging up the hundreds of books and articles that laid the foundation for this book. Keith Barland spearheaded that effort, making my research efforts much less arduous and time-consuming than they otherwise might have been. Other people at the library who helped included Janelle Jones, Christine Shannon, Linda Shaw, Amy Victor, Kyle Wagner, Amy Westfall, and Eilidh Zuvich.

I expound on the virtue of reviews in several places in this book, and this book has benefited greatly from extensive peer reviews. Al Corwin, Pat Forman, Tony Garland, Hank Meuret, and Matt Peloquin stuck with the project from beginning to end. Thanks to them for seeing that the book you

hold in your hands doesn't look very much like the book I originally set out to write! I also received valuable comments from Wayne Beardsley, Duane Bedard, Ray Bernard, Bob Glass, Sharon Graham, Greg Hitchcock, Dave Moore, Tony Pisculli, Steve Rinn, and Bob Stacy—constructive critics, all. David Sommer (age 11) came up with the idea for the last panel of Figure 14-3. Thanks, David. And, finally, I'd like to thank my wife, Tammy, for her moral support and good humor. I have to start working on my third book immediately so that she will stop elbowing me in the ribs and calling me a Two-Time Author!

Bellevue, Washington
June 1996

EFFICIENT DEVELOPMENT



PART I

此为试读,需要完整PDF请访问: www.ertongbook.com