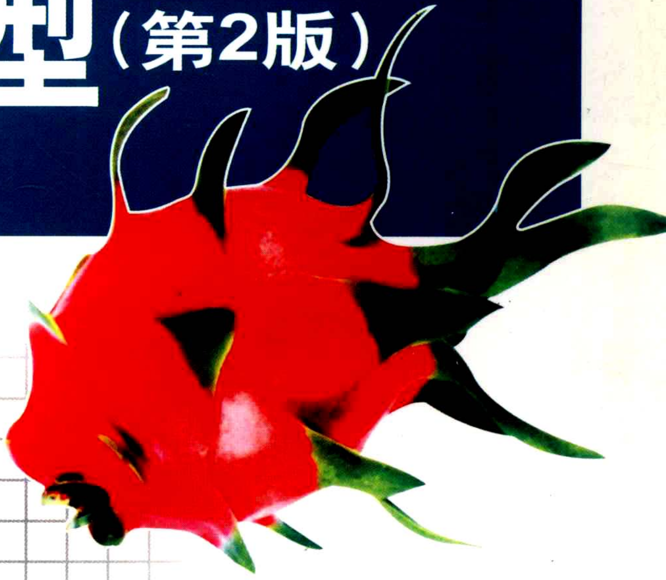
大学计算机教育国外著名教材系列（影印版）

PEARSON
Addison Wesley

METRICS AND MODELS IN SOFTWARE QUALITY ENGINEERING

SECOND EDITION

软件质量工程的度量与模型（第2版）



Stephen H. Kan 著

Pearson Education

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Metrics and Models in Software Quality Engineering

Second Edition

软件质量工程的度量与模型

（第 2 版）

Stephen H. Kan

清 华 大 学 出 版 社
北　京

图书在版编目(CIP)数据

软件质量工程的度量与模型(第2版)=Metrics and Models in Software Quality Engineering, Second Edition/卡恩(Kan, S. H.)著.—影印本.—北京:清华大学出版社,2004.7
(大学计算机教育国外著名教材系列)
ISBN 7-302-08839-X

Ⅰ.软…　Ⅱ.卡…　Ⅲ.软件质量—质量管理—高等学校—教材—英文　Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字(2004)第 056829 号

出 版 者: 清华大学出版社　　　地　　址: 北京清华大学学研大厦
　　　　　http://www.tup.com.cn　邮　　编: 100084
　　　　　社 总 机: 010-62770175　客户服务: 010-62776969
责任编辑: 龙啟铭
印 刷 者: 清华大学印刷厂
装 订 者: 三河市化甲屯小学装订二厂
发 行 者: 新华书店总店北京发行所
开　　本: 185×230　印张: 35
版　　次: 2004 年 7 月第 1 版　2004 年 7 月第1 次印刷
书　　号: ISBN 7-302-08839-X/TP・6271
印　　数: 1～3000
定　　价: 49.00 元

出版说明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书，受到国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高等本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列(影印版)”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好，更适合高校师生的需要。

清华大学出版社

影印版序

随着软件规模的日益增大，软件质量问题也日益突出。它不仅决定了软件交付后使用成本的增加和过早退役，而且也是软件开发延期交付、成本飙升，以至于开发失败的主要因素之一。事实上，软件科学和软件工程一直在寻求对软件本质更清晰的认识，试图以更加合理的方法组织和开发软件，在保证高质量的前提下，大量、快速开发软件。所以，各种软件书刊从分析、设计、构造、测试、维护，到管理、配置交付都涉及质量，而且各种构造方法、解决方案、实施规范层出不穷，吸引了从业者大量精力，而直观、系统地介绍软件质量最新研究成果和度量技术的书籍并不多见。本书正是这类较好的图书之一，作者是 IBM 公司资深研究员 Stephen H. Kan。

1995 年第 1 版出版后即引起业界广泛关注，第 2 版在原有 13 章的基础上增加了当今成熟的软件度量和质量保证技术，如软件测试过程中的度量，面向对象开发中的度量，可用性度量，过程中(in-process)和项目的评估方法，软件过程改进及其功能度量方法，总共达 19 章。

本书有以下一些特点。

第一，面向工程实践、系统、完整。从软件质量的基本概念开始，介绍度量基本理论，软件开发过程中的各种度量，质量管理，七种基本的质量度量工具，直到上述最新度量方法成果。软件工程实践者可以从中得到直接的帮助。

第二，取材新颖，有一定的学术深度。传统的软件质量度量模型，因其过程编程背景压缩至相当小的篇幅，如 McCabe、Halstead 模型等，而引入了许多新颖的质量管理与度量模型，如可靠性增长模型，缺陷消除模型等。这些模型有较深厚的概率统计学理论基础，特别对面向对象软件，提供了一批新准则和经验公式，这对研究和开发当前基于构件、Web 服务软件的质量度量方法和规范的从业者，无疑是个很好的参照。

第三，本书符合 IEEE 和 ACM 2001 年发布的软件工程知识体(SWEBOK)指南。软件质量一章界定的内容，是计算机专业课程体系制定者很好的参考资料。事实上，本书很适合计算机软件、软件工程学科本科生和研究生的教材。

麦中凡

北京航空航天大学软件学院教授

Foreword to the Second Edition

For more than 50 years software has been a troublesome discipline. Software's problems are numerous and include cancelations, litigation, cost overruns, schedule overruns, high maintenance costs, and low levels of user satisfaction. The problems with software occur more often than not. My company's research indicates that more than half of large software projects will encounter some kind of delay, overrun, or failure to perform when deployed.

But software does not have to be as troublesome as it has been. Some complex software projects do achieve their delivery schedules and cost targets, and behave properly when used. Throughout my career in software I've been interested in what distinguishes successful software projects from failures and disasters.

It happens that the main factors leading to software success are easily identified when side-by-side comparisons of similar projects are performed, where one set was successful and the other set was troublesome. The successful software projects achieve excellence in software quality control, and they are able to do this because of excellence in software quality measurements.

Although it might be thought that excellent software quality control is expensive, it turns out to yield a very positive return on investment. When canceled software projects and disasters are studied by means of "autopsies," they all have similar patterns: Early phases of troubled projects are handled carelessly without adequate requirements analysis or design reviews. After rushing through the early phases and seeming to be ahead of schedule, problems begin to mount during coding and testing. When testing begins in earnest, serious problems are detected so that schedules and cost targets cannot be achieved. Indeed, some software projects have so many serious problems—termed bugs or defects—that they are canceled without completion.

By contrast, successful projects are more thorough at the start. The requirements are carefully analyzed and the designs are formally inspected. This takes some time and adds upfront costs, but once coding and testing begin, the value of careful quality control allows the projects to move rapidly to a successful conclusion.

Stephen Kan and I both learned the importance of software quality control and good software quality metrics at IBM. Even though Stephen and I worked in different IBM labs during different decades, we have some common viewpoints. We have both been convinced by empirical evidence that without excellent software quality control, large system development is hazardous and likely to fail. We also both know that effective software quality control is built on a foundation of careful measurements using accurate metrics.

Stephen and I both had access to IBM's internal quality data—one of the best and largest collections of software data in the world. But one company's internal information is not sufficient to be convincing unless other organizations replicate the findings. Stephen's excellent book goes beyond IBM; it offers valuable information from many companies and government agencies. He cites studies by Hewlett-Packard, Motorola, NASA, and other organizations that have solved the problems of how to build software successfully.

I first encountered Stephen Kan's *Metrics and Models in Software Quality Engineering* book in 1995 when the first edition was published. I found that the book contained solid research, and that it covered a broad range of topics and very clearly showed the relationships among them.

This new edition keeps all the virtues of the earlier work and adds substantial new information, such as a chapter on measuring quality within the object-oriented paradigm. Stephen Kan's new edition contains an excellent combination of solid scholarship and breadth of coverage. This is the single best book on software quality engineering and metrics that I've encountered.

Capers Jones
Chief Scientist Emeritus
Software Productivity Research, Inc.
(an Artemis company)

Foreword to the First Edition

Quality management and engineering have enjoyed a diversity of applications over the past few years. For example:

- ☐ A system of teaching hospitals conservatively estimates $17.8 million saved on an investment of $2.5 million in quality management over a five-year time period.
- ☐ The U.S. Air Force Military Airlift Command improved capacity so much through quality improvement problem solving during the Gulf War that they avoided having to deploy civilian aircraft (thus avoiding the suspension of next-day mail delivery, among other conveniences).
- ☐ The U.S. Bureau of Labor Statistics reduced the time needed to produce the monthly Consumer Price Index (CPI), compiled by 650 people in five departments, by 33 percent with no loss in accuracy.
- ☐ The University of Pennsylvania saved more than $60,000 a year from one project focused on reducing mailing costs.

The examples go on and on, from industries such as telecommunications, health care, law, hospitals, government, pharmaceuticals, railways, and schools. The variety of terrains where the seeds of TQM successfully take hold is almost baffling.

As the rest of the world moves headlong into quality improvement at revolutionary rates, the software developers and engineers—consumed in debates over metrics and process models, over methodologies and CASE tools—often lag far behind. Some of the reasons include unique challenges in defining user requirements, the "guru" mentality prevalent in many software organizations, and the relative immaturity of the application of software engineering. Whereas the first two are fascinating topics in their own right, it is the third challenge at which Stephen Kan's book is squarely aimed.

Imagine designing an airplane using a small fraction of the aeronautical engineering knowledge available. Imagine designing an automobile while ignoring mechanical engineering. Imagine running a refinery with no knowledge of chemical process engineering. It is not surprising that the first recommendations from consultants offering credible software quality solutions would be to apply proven software engineering methods first. Just as these methods only slowly find acceptance in software development communities, so also have the methods of quality engineering.

One reason for this slow adoption lies with the relative lack of literature that gives clear descriptions of the fundamentals in these fields while illustrating actual use within leading-edge software development organizations. Stephen Kan's book, *Metrics and Models in Software Quality Engineering,* represents a laudable move to fill this need.

Dr. Kan provides a uniquely comprehensive reference to the field of software quality engineering. He has managed a delightful balance between the technical details needed and practical applications of the models and techniques. This book is peppered with industry examples, not only from Kan's own employer, the Malcolm Baldrige National Quality Award-winning IBM Rochester, but from NEC's Switching Systems Division, Hewlett-Packard, Motorola, NASA Software Engineering Laboratory, and IBM Federal Systems Division. Concepts and theory are illustrated by software industry examples, which make the reading that much richer.

Dr. Joseph Juran, one of the key founders of modern quality management and engineering, describes "life behind the quality dikes." As society becomes more reliant on technology, failures of that technology have increasing adverse impacts. Quality helps to insulate society from these dangers. This role of quality in software development certainly rivals that of business competitiveness, and gives another compelling reason to read, understand, and apply the ideas within this book.

Brian Thomas Eck, Ph.D.
Vice President
Juran Institute, Inc.
Wilton, Connecticut

Preface

Looking at software engineering from a historical perspective, the 1960s and earlier could be viewed as the functional era, the 1970s the schedule era, the 1980s the cost era, and the 1990s and beyond the quality and efficiency era. In the 1960s we learned how to exploit information technology to meet institutional needs and began to link software with the daily operations of institutions. In the 1970s, as the industry was characterized by massive schedule delays and cost overruns, the focus was on planning and control of software projects. Phase-based life-cycle models were introduced, and analysis, like the mythical man-month, emerged. In the 1980s hardware costs continued to decline, and information technology permeated every facet of our institutions and became available to individuals. As competition in the industry became keen and low-cost applications became widely implemented, the importance of productivity in software development increased significantly. Various software engineering cost models were developed and used. In the late 1980s, the importance of quality was also recognized.

The 1990s and beyond is certainly the quality era. With state-of-the-art technology now able to provide abundant functionality, customers demand high quality. Demand for quality is further intensified by the ever-increasing dependence of society on software. Billing errors, large-scale disrupted telephone services, and even missile failures during recent wars can all be traced to the issue of software quality. In this era, quality has been brought to the center of the software development process. From the standpoint of software vendors, quality is no longer an advantage factor in the marketplace; it has become a necessary condition if a company is to compete successfully.

Starting in the mid 1990s two major factors emerged that have proved to have an unprecedented impact on not only software engineering but also on global business environments: business reengineering for efficiency and the Internet. Software development has to be more efficient and the quality level of the delivered products has to be high to meet requirements and to be successful. This is especially the case for mission-critical applications. The adverse impact of poor quality is much more significant and at a much wider scale; the quality "dikes" that software is supposed to provide are never more important. These factors will continue to affect software engineering for many years to come during this new millennium.

Measurement plays a critical role in effective and efficient software development, as well as provides the scientific basis for software engineering that makes it a true engineering discipline. This book describes the software quality engineering metrics and models: quality planning, process improvement and quality control, in-process quality management, product engineering (design and code complexity), reliability estimation and projection, and analysis of customer satisfaction data. Many measurement books take an encyclopedic approach, in which every possible software measurement is included, but this book confines its scope to the metrics and models of software quality. Areas such as cost estimation, productivity, staffing, and performance measurement, for which numerous publications exist, are not covered.

In this edition, seven new chapters have been added, covering in-process metrics for software testing, object-oriented metrics, availability metrics, in-process quality assessment, software project assessment, process improvement dos and don'ts, and measuring software process improvement. The chapter that described the AS/400 software quality management system has been eliminated. Updates and revisions have been made throughout the original chapters, and new sections, figures, and tables have been added.

Two of the new chapters are special contributions from two experts. This is a key feature of the new edition. The chapter on the dos and don'ts of software process improvement is contributed by Patrick O'Toole. A highly regarded process improvement expert and with over 20 years of experience, Patrick brings to this book a perspective on process improvement that I share as a practitioner. That perspective is based on practical experience, is project-centric, and is aligned with the strategic business imperative of the organization. Patrick also brings humor to this otherwise serious subject, making the reading of the chapter so enjoyable. The chapter on measuring software process improvement is a special contribution by Capers Jones. A pioneer in software metrics, productivity research, software quality control, and software assessments, Capers's work is well known nationally and internationally. His data-based and fact-based approach in software assessments and benchmarking studies is unparalleled. Based on experience and data from more than 10,000 projects, he brings to the readers a practical approach to software process improvement and the major quantitative findings related to software process improvement. The value of function point metrics is demonstrated via the analyses and findings. The chapter is a must read for software process professionals who are interested in measuring software process improvement.

Another new feature in this edition is a set of recommendations for small teams and organizations that are starting to implement a metrics program, with minimum resources. These recommendations are shown in the form of box inserts in nine of the chapters. A number of examples in the book are based on small team projects, and many methods and techniques are appropriate for large projects as well as small ones. This set of recommendations is from the perspective of small organizations or teams using a small number of metrics, with the intent to effect improvement in their software development effort.

This book is intended for use by software quality professionals; software project managers; software product managers; software development managers; software engineers; software product assurance personnel; and students in software engineering, management information systems, systems engineering, and quality engineering and management. For teachers, it is intended to provide a basis for a course at the upper-division undergraduate or graduate level. A number of software engineering, computer science, and quality engineering programs in the United States and overseas have used the first edition of this book as a text.

Themes of This Book

This book has several themes. First, balancing theory, techniques, and real-life examples, it provides practical guidelines in the practice of quality engineering in software development. Although equations and formulas are involved, the focus is on the understanding and applications of the metrics and models rather than mathematical derivations. Throughout the book, numerous real-life examples are used from the software development laboratory at IBM Rochester, Minnesota, home of the AS/400 and the IBM eServer iSeries computer systems, and from other companies in the software industry. IBM Rochester won the Malcolm Baldrige National Quality Award in 1990. A number of metrics described in this book were being used at that time, and many have been developed and refined since then. All metrics are substantiated by ample implementation experience. IBM Rochester develops and delivers numerous projects of different sizes and types every year, including very large and complex as well as small ones; and they range from firmware, to operating systems, to middleware, to applications.

Second, I attempt to provide a good coverage of the various types of metrics and models in the emerging field of software quality engineering. In addition to general discussions about metrics and techniques, this book categorizes and covers four types of metrics and models: (1) quality management models; (2) software reliability and projection models; (3) complexity metrics and models; and (4) customer-view metrics, measurements, and models. These metrics and models cover the entire software development process from high-level design to testing and maintenance, as well as all phases of reliability. Furthermore, although this book is not on total quality management (TQM), it is a major consideration in the coverage of metrics. The philosophy of TQM is the linking of product quality and customer satisfaction for the purpose of achieving long-term success. TQM is the reason for including two chapters on customer-view metrics and measurements—availability metrics and customer satisfaction—in addition to the many chapters on product and process metrics. In other discussions in the book, the customer's perspective is included where appropriate.

Third, by linking metrics and models to quality improvement strategies and improvement actions, we attempt to focus on using, not just describing, metrics. A framework for interpreting in-process metrics and assessing in-process quality status—the effort/outcome model—is presented. The direct link between a recommended quality strategy during development and the defect-removal model is shown. Examples of actions tied to specific metrics and analysis are given. Furthermore, to illustrate the metrics, many figures and graphs are used. This is a reflection of the fact that in real-life project and quality management, a clear visual presentation often improves understanding and increases the effectiveness of the metrics.

Fourth, following up on quality and process improvement at a more general level than specific metric discussions, the book continues with chapters that discuss the in-process quality assessment process, a method for conducting software project assessments, practical advice on process improvement dos and don'ts, and quantitative analysis of software process improvement. The common thread underlying these chapters, as with other chapters on metrics and models, is practical experience with industry projects.

Organization of This Book

The following list details the focus of each chapter.

- *Chapter 1, What Is Software Quality?,* discusses the definition of quality and software quality. The customer's role in the definition is highlighted. Quality attributes and their relationships are discussed. The second part of the chapter covers the definition and framework of TQM and the customer's view of quality, a key focus in this book.
- *Chapter 2, Software Development Process Models,* reviews various development process models that are used in the software industry. It briefly describes two methods of software process maturity assessment—the SEI process capability maturity model (CMM) (by the Software Engineering Institute) and the SPR assessment method (by the Software Productivity Research, Inc.). It summarizes two bodies of quality management standards—the Malcolm Baldrige National Quality Award assessment discipline and ISO 9000.

- ☐ *Chapter 3, Fundamentals of Measurement Theory,* examines measurement theory fundamentals, which are very important for the practice of software measurement. The concept of operational definition and its importance in measurement are illustrated with an example. The level of measurement, some basic measures, and the concept of six sigma are discussed. The two key criteria of measurement quality, reliability and validity, and the related issue of measurement errors are examined and their importance is articulated. This chapter also provides a discussion on correlation and addresses the criteria necessary to establish causality based on observational data.
- ☐ *Chapter 4, Software Quality Metrics Overview,* presents examples of quality metrics for the three categories of metrics associated with the software life cycle: end-product, in-process, and maintenance. It describes the metrics programs of several large software companies and discusses collection of software engineering data.
- ☐ *Chapter 5, Applying the Seven Basic Quality Tools in Software Development,* describes the application of the basic statistical tools for quality control, known as Ishikawa's seven basic tools, in software development. The potentials and challenges of applying the control chart in software environments are discussed. In addition, a qualitative tool for brainstorming and for displaying complex cause-and-effect relationships—the relations diagram—is discussed.
- ☐ *Chapter 6, Defect Removal Effectiveness,* is the first of five chapters about the models and metrics that describe the quality dynamics of software development. Through two types of models, quality management models and software reliability and projection models, the quality of software development can be planned, engineered, managed, and projected. This chapter examines the central concept of defect removal effectiveness, its measurements, and its role in quality planning.
- ☐ *Chapter 7, The Rayleigh Model,* describes the model and its implementation as a reliability and projection model. The Rayleigh Model's use as a quality management model is discussed in Chapter 9.
- ☐ *Chapter 8, Exponential Distribution and Reliability Growth Models,* discusses the exponential distribution and the major software reliability growth models. These models, like the Rayleigh Model, are used for quality projection before the software is shipped to customers, just before development is complete. The models are also used for maintenance planning, to model the failure pattern or the defect arrival patterns in the field.
- ☐ *Chapter 9, Quality Management Models,* describes several quality management models that cover the entire development cycle. In-process metrics and reports that support the models are shown and discussed. A framework for interpreting in-process metrics and assessing in-process quality status—the effort/outcome model—is presented.

- ☐ *Chapter 10, In-Process Metrics for Software Testing,* is a continuation of Chapter 9; it focuses on the metrics for software testing. The effort/outcome model, as it applies to metrics during the testing phase, is elaborated. Candidate metrics for acceptance testing to evaluate vendor-developed software, and the central question of how to know your product is good enough to ship, are also discussed.
- ☐ *Chapter 11, Complexity Metrics and Models,* discusses the third type of metrics and models in software engineering. While quality management models and reliability and projection models are for project management and quality management, the objective of the complexity metrics and models is for software engineers to be able to improve their design and implementation of software development.
- ☐ *Chapter 12, Metrics and Lessons Learned for Object-Oriented Projects,* covers design and complexity metrics, productivity metrics, quality and quality management metrics for object-oriented development, and lessons learned from the deployment and implementation of OO projects. The first section can be viewed as a continuation of the discussion on complexity metrics and models; the other sections fall within the framework of quality and project management.
- ☐ *Chapter 13, Availability Metrics,* discusses system availability and outage metrics, and explores the relationships among availability, reliability, and the traditional defect-rate measurement. Availability metrics and customer satisfaction measurements are the fourth type of metrics and models—customer-oriented metrics.
- ☐ *Chapter 14, Measuring and Analyzing Customer Satisfaction,* discusses data collection and measurements of customer satisfaction, and techniques and models for the analysis of customer satisfaction data. From Chapter 3 to this chapter, the entire spectrum of metrics and models is covered.
- ☐ *Chapter 15, Conducting In-Process Quality Assessments,* describes in-process quality assessments as an integrated element of good project quality management. Quality assessments are based on both quantitative indicators, such as those discussed in previous chapters, and qualitative information.
- ☐ *Chapter 16, Conducting Software Project Assessments,* takes the discussion to yet another level; this chapter proposes a software project assessment method. The focus is at the project level and the discussion is from a practitioner's perspective.
- ☐ *Chapter 17, Dos and Don'ts of Software Process Improvement* by Patrick O'Toole, offers practical advice for software process improvement professionals. It provides a link to the process maturity discussions in Chapter 2.
- ☐ *Chapter 18, Using Function Point Metrics to Measure Software Process Improvement* by Capers Jones, discusses the six stages of software process improvement. Based on a large body of empirical data, it examines the costs and effects of process improvement. It shows the results of quantitative analy-

ses with regard to costs, time, schedule, productivity, and quality. It articulates the value of Function Point metrics. It provides a link to the process maturity discussions in Chapter 2.

- ☐ *Chapter 19, Concluding Remarks,* provides several observations with regard to software measurement in general and software quality metrics and models in particular, and it offers a perspective on the future of software engineering measurement.
- ☐ In the *Appendix,* a real-life example of a project assessment questionnaire is shown. Per the methods and techniques discussed in Chapter 16, readers can customize the questionnaire for their project assessment efforts.

Suggested Ways to Read This Book

The chapters of this book are organized for reading from beginning to end. Later chapters refer to concepts and discussions in earlier chapters. At the same time, each chapter addresses a separate topic and chapters in some groups are more closely coupled than others. Some readers may choose to read specific topics or decide on different starting points. For example, those who are not interested in quality definitions, process models, and measurement fundamentals discussions can start with Chapter 4, *Software Quality Metrics Overview.* Those who intend to immediately get to the central topics of defect removals, metrics and models for quality planning, and management and projection can start with Chapter 6, *Defect Removal Effectiveness.* In general, I recommend that the chapters be read in groups, as follows.

- ☐ Chapters 1 through 3
- ☐ Chapter 4
- ☐ Chapter 5
- ☐ Chapters 6 through 10
- ☐ Chapters 11 and 12
- ☐ Chapters 13 and 14
- ☐ Chapters 15 through 18
- ☐ Chapter 19

Acknowledgments

I would like to thank Terry Larson, Steve Braddish, and Mike Tomashek for their support of this project. I wish to thank the entire iSeries software development team at IBM Rochester, especially the release and project management team and the test teams, who made the many metrics and models described in this book a state of practice instead of just a theoretical discussion. A special thanks is due Diane Manlove and Jerry Parrish for their leadership in the implementation of many of the in-process metrics, and Bob Gintowt for his leadership work, knowledge, and insights on system availability and outage metrics.

Appreciation is due all of my former and present colleagues in software and system quality at IBM Rochester, other IBM divisions, IBM Center for Software Engineering, and IBM Corporate Quality, for the numerous discussions, debates, and insights on the subjects of measurement, quality and process improvement. There are too many individuals and teams to name them all. Among them are: Lionel Craddock, John E. Peterson, Dave Lind, Dr. Sam Huang, Don Mitchell, Judy Wasser, Marijeanne Swift, Duane Miller, Dr. Dave Jacobson, Dick Bhend, Brad Talmo, Brock Peterson, Vern Peterson, Charlie Gilmore, Jim Vlazny, Mary Ann Donovan, Jerry Miller, Mike Tappon, Phil Einspahr, Tami Mena, Peter Bradford, Max Maurer, Roger McKnight, Jack Hickley, Marilyn Moncol, Darrell Moore, Dusan Gasich, Eileen Gottschall, Carl Chamberlin, Paul Hutchings, Gary Davidson, George Stark, Kathleen Coyle, Ram Chillarege, Peter Santhanam, Kathryn Bassin, Beng Chiu, Wanda Sarti, Brent Hodges, Bill Woodworth, and Mike Jesrani. I am grateful to Dave Amundson, Ben Borgen, Dick Sulack, Rod Morlock, Brian Truskowski, Jeff VerHeul, Judy Tenney, Mike Tomashek, and Paul Loftus, from whom I learned a great deal not only about quality and quality management, but also about prudent decision making with regard to business objectives and quality.

A special gratitude is due Capers Jones for his review of this new edition, many excellent suggestions, and his chapter on measuring process improvement. In my early career at IBM I benefitted a great deal from reading Capers' pioneer work on programming productivity, software quality control, and software assessments. Then in 1995, I had the chance to meet him in person in Salt Lake City, where he gave a keynote speech on software measurements at a software technology conference. Now many years later, I continue to learn and benefit from reading his work and from his reviews and suggestions.

My sincere thanks are due Patrick O'Toole for his special contribution of the chapter on the dos and don'ts of software process improvement, amid his very busy schedule. A special thanks is due Steve Hoisington for his review and helpful suggestions of the new materials, and as a former colleague, his passion in quality, his professionalism in quality management, and his long-time influence on the customer view of quality.

Much appreciation is due Dick Hedger, for his review of and help with both the first and second editions, his passion for and expertise in software process improvement, and his continual support over many years. I am thankful to the reviewers for the first edition, Dr. Brian Eck, Dr. Alan Yaung, Professor Wei-Tsek Tsai, and others. They contributed many constructive suggestions.