软件工程与方法丛书

影印版

成功的软件开发

Successful Software Development (2nd Edition)

（原书第2版）

- Practical and real-world tested approaches to software development
- Implement effective communication and risk reduction throughout a software project
- Easy-to-learn and easy-to-apply measurement technique in everyday language to measure product and process goodness

(美) Scott E. Donaldson
Stanley G. Siegel 编著

Pearson Education
科学出版社
www.sciencep.com

软件工程与方法丛书

成功的软件开发

（影印版）

Successful Software Development

（美） Scott E. Donaldson
Stanley G. Siegel 编著

科学出版社

北京

图字：01-2003-6694 号

内 容 简 介

本书围绕一种成熟的软件开发模型——SEE，以案例学习的方式讲述了软件开发全过程中涉及的一系列问题，内容包括：业务实例、项目规划、软件开发、变更控制、产品和过程评审、文化变更和过程改进规划等。

本书内容翔实，案例丰富，条理清晰，不仅可作为高等院校研究生或本科生软件工程类的教材，还适合在软件企业对开发及项目管理人员的培训中使用。

图书在版编目(CIP)数据

成功的软件开发= Successful Software Development /（美）唐纳森（Scott E. Donaldson）等编著.
—影印版. —北京：科学出版社，2004
（软件工程与方法丛书）
ISBN 7-03-012472-3
Ⅰ.成… Ⅱ.①唐… Ⅲ.软件开发—英文 Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2003）第 099954 号

策划编辑：李佩乾/责任编辑：袁永康
责任印制：吕春珉/封面设计：飞天创意

科 学 出 版 社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双 青 印 刷 厂 印刷

科学出版社发行 各地新华书店经销
*
2004 年 1 月第 一 版 开本：787×960 1/16
2004 年 1 月第一次印刷 印张：46 1/4
印数：1—3 000 字数：718 000

定价：78.00 元

（如有印装质量问题，我社负责调换<环伟>）

影印前言

“软件工程”是自20世纪60年代起针对所谓“软件危机”而发展起来的概念。它是指将工程化的方法应用到软件开发中，以求优质高效地生产软件产品。其中综合应用了计算机科学、数学、工程学和管理科学的原理和方法。自从这一概念提出以来，软件开发方法从60年代毫无工程性可言的手工作坊式开发，过渡到70年代的结构化分析设计方法、80年代初的实体关系方法，直到当今所流行的面向对象方法，经历了根本性的变革。随着时代的发展，软件项目管理人员越来越需要从系统和战略的角度来把握项目的方向，引导开发向更高层次发展。在这方面，国外的知名企业和研究机构已经总结了相对成熟的一套知识和方法体系，并在实践中获得了相当大的成功。这里我们就从著名的培生教育出版集团（Pearson Education Group）选取了一些软件工程与方法类的有代表性的教材影印出版，以期让国内的读者尽快了解国外此领域的发展动态，分享国外专家的知识和经验。以下对每本书的内容作一些简要的介绍，以便读者选择。

在Internet广泛普及的今天，软件承载着越来越重要的使命，工程化的开发必须能够提供健壮性和普适性更好的产品。Scott E. Donaldson和Stanley G. Siegel合作编著的《成功的软件开发》（*Successful Software Development*）一书从“软件系统开发无定式”这一事实出发，引入了一个灵活而成熟的开发过程模型——系统工程环境（Systems Engineering Environment，SEE）。该模型包含两个互相联系的基本要素：确定软件开发策略和规程，以及可用于实现目标的技术。围绕这一模型，书中对开发过程中关系项目成败的各种关键问题进行了透彻的论述，可作为软件开发团队的全程培训教材。

软件工程经过几十年的发展，其关键环节——需求分析和管理终于得到了真正的重视。伴随认识的深化和案例的丰富，一系列实用的工程化需求分析方法应运而生。Ralph R. Young的《有效需求分析》（*Effective Requirements Practices*）一书从管理和技术两个角度阐述了关系项目成败的各种问题，书中的案例分析让项目管理者能够对需求分析的框架体系和过程形成较为清晰的认识，在实践中准确了解客户的业务需求，正确地调配各种资源，从而更加准确地把握项目的方向，保证在整个项目周期中各种需求都能够得到应有的考虑和满足。

软件开发人员在系统组织方面通常会采用特定的模式，但就大多数而言，他们对体系结构的分析和判断在很大程度上都是出于自发而且并不规范。Mary Shaw和David Garlan的《软件体系结构》（*Software Architecture: Perspectives on an Emerging Discipline*）一书就对当前业界所公认的成功的系统设计思想进行了生动而全面的阐述。两位作者对软件体系结构的发展状况及其对设计的影响作用有独到的见解，相信各类读者都能从书

中获得有用的信息：专业开发人员可能对书中所介绍的设计模式比较熟悉，但从对这些模式的讨论和评价中也能够获得新的启发；学生们能够从书中学到一些有用的技巧，从较高的视角来了解系统的组织结构，而不是仅仅追逐业界的潮流或是过时的方法；对于教师而言，本书可以作为“软件体系结构”课程的教材，或者是“软件工程”或“软件设计”课程的补充教材。

对于现在已经或者将要从事软件项目管理的读者来说，Joel Henry 的《软件项目管理》（*Software Project Management: A Real-World Guide to Success*）应该说是一本难得的好教材。书中论述了软件项目的四个基本构成要素：人、过程、工具和评价体系，并向读者提供每一领域中可以选用的合适方法，使项目实施取得最满意的效果。作者本人在软件行业工作多年，积累了丰富的第一手材料，他在书中用案例对技术、管理和领导等多方面问题进行了透彻的讲解。尽管他对这些问题的结论对业内人士而言已经是耳熟能详，但不论是何种类型和水平的读者，相信都能从本书中得到指导和启发。

软件开发中出现错误在所难免，关键是要及早发现，而不要让其扩散，增加纠正的成本。软件同级评审制度是任何高质量软件开发过程的重要环节，然而受过这方面必要培训的专业人员却还是凤毛麟角。Karl E. Wiegers 的《软件同级评审》（*Peer Reviews in Software*）就是针对这方面需求而编写的。本书介绍了软件同级评审的全过程，内容涉及各种正规和非正规的评审方法和质量保证技巧。书中对大型项目及开发团队地域分散等情况下的同级评审进行了专门探讨。对于项目管理者而言，本书能够帮助他们以实用的资源启动同级评审计划，增进开发人员间的沟通，最终按期提交高质量的软件产品。

面向对象技术的应用已经越来越普遍，而与此同时越来越多的管理者在项目进行中却要面对许多原本隐藏着的成本和意外情况。对整个项目而言，管理者在规划阶段是否有远见、在进行过程中对各种情况反应是否得当，这些都影响着项目的成败。尽管市场上介绍对象技术的书很多，但对于项目实施中所需进行的规划和预测却还缺乏系统的知识归纳。Alistair Cockburn 的《对象软件项目求生法则》（*Surviving Object-Oriented Projects*）一书就以大量专家的知识和经验向读者提供成功管理对象软件项目的实用指导和建议，帮助读者应对项目中的意外挑战，使项目正常进行并最终获得成功。书中指出了对象软件项目所面临的潜在风险，并对时间安排、预算、人员安排以及成本合理化等重要问题进行了探讨，提供了可操作的解决方案。对于从事对象软件项目管理的读者而言，本书是一本相当合适的参考书，可以作为相关培训的教材。

以上就是目前这套影印版丛书的大致内容。需要指出的是，这套丛书并非一个封闭的体系。随着软件工程的深入发展，必将涌现出更多新的开发理念和方法，我们也将尽己所能将更多新的优秀图书吸纳进来。读者对于这套丛书目前的内容或未来的发展有何意见或建议，望不吝赐之。

编　者

2003 年 11 月

Preface

We must not promise what we ought not, lest we be called on to perform what we cannot.

—*Attributed to Abraham Lincoln, speech delivered before the first Republican convention of Illinois, May 29, 1856,* The Writings of Abraham Lincoln, *ed. Arthur B. Lapsley, vol. 2, p. 249 (1905).*

Successful Software Development means "the ability to produce 'good' software systems 'consistently'"

Customers want software systems to do what they are supposed to do, to be delivered on time, to be delivered for the agreed-upon cost, and to satisfy any other criteria they may specify. Sellers want the systems they develop to do what the customer wants, to be delivered ahead of schedule or on time, to earn them a reasonable profit, and to satisfy any other criteria that may govern the way they do business. Software systems satisfying both customer and seller criteria are "good." Customers and sellers also want their criteria satisfied "consistently." The software development business should not be a lottery.

This book is a practitioner's guide for achieving successful software development.

Making It Happen means "implementing a 'way' of successful software development"

There is no one "way" to develop software systems. If there were, software systems development would have been reduced to an assembly-line process long ago. People with diverse experiences and educational disciplines

contribute to advances in software development methodologies, processes, techniques, practices, and tools. This rich diversity brings about different "ways" to develop "good" software systems "consistently."

This book is a practitioner's guide for making successful software development happen in a way that makes sense for your environment.

Who Should Read This Book?

The software development business is centered on a relationship between a customer and a seller of software products and services. So, this book is for both software customers and software sellers. More specifically, the intended audience is anyone who performs one or more of the following activities:

- Develops software products and software-related products
- Directly manages people who do the above
- Manages the above managers
- Buys/uses products from the above
- Educates the people above

Individuals have used the first edition of this book to complement their particular expertise. Customers have used this book to refine their business dealings with sellers. Sellers have used this book to establish or refine their "way" of developing software systems for their customers. Sellers have also used the book to provide in-house training to their marketing personnel so the marketers better understand what their businesses are selling. Customers and sellers have used the book to train their human resource personnel so they better understand what skill sets are needed for the software development business. Universities have used the book in graduate schools to teach how to be successful in the software development business.

For the **software customer**, we explain and illustrate mechanisms to effectively communicate to the software seller (1) what you want, (2) when you want it, and (3) how much you want to pay for it.

For the **software seller**, we explain and illustrate the mechanisms to effectively communicate (1) to your customer your understanding of what the customer wants and (2) among your project team members how you are going to give the customer what the customer wants.

For the **educator**, we provide supplemental training materials for the classroom. This material is packaged in a separately available study guide that consists of the following items:

- Over 500 pages that recast the book's contents as presentation material. This material is organized by chapter and lays out the material in the order that it appears in the chapter. Most of the book's figures, or adaptations of these figures, appear in this material.
- Sample questions for homework assignments.
- Sample class project.
- Sample course syllabus.

Educators can use the above material in conjunction with the companion Web site at www.phptr.com/ptrbooks/ptr_0130868264.html to develop courses based on the book's material. These courses can be part of a corporate training program or a college or university curriculum. The study guide material is adapted from our teaching experience in both of these environments.

Regarding the presentation material in the study guide, we note that students can use this material without an instructor as a companion to the book. Example uses of this material as a companion to the book include:

- Before reading a chapter or part of a chapter from the book, the student can go to the corresponding study guide pages to get a quick look at the chapter or chapter part.
- While reading a chapter or chapter part, the student can, in parallel, look at the corresponding study guide pages. Sometimes a different look at the same material can facilitate learning.
- After reading a chapter or chapter part, the student can go to the corresponding study guide pages for review purposes and quickly recall key points, concepts, and book illustrations.

How Is This Software Development Book Different from Other Such Books?

Lots of things go into making successful software development happen. Among the important things, *effective communication, risk reduction,* and *an organizational "way" of successful software development* stand out and are threaded throughout this book.

Effective communication means "transmitting information, thought, or feeling so that it is *satisfactorily received or understood* [emphasis added]."[1] At the risk

[1]This definition is adapted from words used in one of the definitions for "communicate" given in *Merriam-Webster's Collegiate Dictionary,* Tenth Edition (Springfield, MA: Merriam-Webster, Inc., 2000). We note that some dictionaries include the notion of "effective" in the definition of "communicate" (see, for example, *Webster's II New College Dictionary* [Boston, MA: Houghton Mifflin Company, 1995]; this dictionary actually comments on the notion of "effectiveness"). We have chosen to risk redundancy in the eyes of some by coupling "effective" to "communication." Our rationale is that we want to stress the notion that the person who transmits information, thought, or feeling is obliged to carry out this transmission so that it is, in fact, satisfactorily received or understood.

of oversimplification, people understand the mechanics of creating software code, but both the customer and seller have trouble understanding each other. Customers have said, "We thought we told the developers what we thought we wanted, but what they delivered is not what we wanted." Sellers have said, "We thought we understood what the customer was trying to tell us, but come to find out, what we delivered is not what the customer wanted." Therefore, for us,

Successful software development is first and foremost an ongoing exercise in effective communication between the customer and the seller throughout a software project.

Risk reduction means "reducing the likelihood that software systems development products will (1) not be delivered on time, (2) not be delivered within budget, and (3) not do what the seller and customer mutually agreed that the products are supposed to do."

Simply stated, people understand there are risks with creating software code. However, many people do not assess risk, allocate appropriate resources to mitigate risk, monitor risk, and decide how to deal with risk. Customers have said, "We don't have documented requirements, but we expect the system to do what it is supposed to do." In response to such customer requests, sellers have said, "No problem! The software should do what you want and we will deliver it on time." Therefore, for us,

Successful software development is also an ongoing exercise in risk reduction.

An organizational "way" of successful software development means "a set of processes that an organization uses to develop and maintain software and software-related products." We proceed from the premise that, as we said earlier, there is no one way to build software systems. Again, at the risk of oversimplification, customers have said, "We don't have time to stop and plan the work to be done, just get started coding." Sellers have said, "We know what the users need, so let's get started." Therefore, for us,

A "way" of developing software systems consists of processes that (1) promote effective communication throughout software systems development and (2) continually reduce risk.

We present processes based on fundamental engineering and process principles that include (1) project planning, (2) change control, and (3) product and

process reviews. We present a language-based measurement technology[2] for evaluating software processes and the products they yield. We explain how to use such measurements to improve your processes and products. We explain how to plan process improvement to help bring about improvement in the way you develop software systems. We explain why the ideas presented work, give you suggestions on how you can make them work, and offer insights into what may not work.

An organizational "way" of doing business needs to incorporate such things as the lessons learned from people's experiences and previous software development projects. If the organizational "way" includes such experiences and lessons learned, known and/or anticipated risks are reduced, but not necessarily eliminated. Also, effective customer/seller communication reduces misunderstandings, thereby reducing the risk that a software product will not satisfy its criteria.

This book, therefore, presents you with techniques for effectively communicating and reducing risk. We explain and illustrate fundamental engineering and process principles for you to consider when **Making It Happen** in your environment.

We stress that these techniques have been successfully implemented on real-world software systems development projects and programs. The size of these projects and programs ranges from tens of thousands of dollars to hundreds of millions of dollars.

How Is the Book Organized?

Figure P–1 shows the title and purpose of each of the book's eight chapters. More specifically, these chapters address the following topics:

Chapter 1 The first chapter presents the business case for setting up a "consistent" way of doing software systems development. The chapter also presents some fundamental concepts and terms used throughout the book. These terms and concepts establish a working vocabulary to facilitate effective communication.

Chapter 2 The second chapter presents techniques for project planning and reducing risks. Many organizations develop project plans and then start working. For us, planning is just one part of an organization's "way" of developing software systems.

Chapter 3 The third chapter presents an organizational "way" (or process) for developing software systems—an organizational software

[2]Here, *language-based measurement technology* means "a measurement technology that associates language familiar to the intended audience with numbers arranged on value scales."

Purpose

Chapter 1 Business Case	Makes the business case for setting up a consistent way of doing software systems development and introduces fundamental concepts needed for the rest of the book.
Chapter 2 Project Planning Process	Provides practical guidance for effectively planning software systems development work by factoring in project risks to allocate project resources.
Chapter 3 Software Systems Development Process	Defines principles for putting together an organizational software systems development process that fosters success and illustrates these principles by defining a top-level process that you can use to formulate a process framework for your environment.
Chapter 4 Change Control Process	Defines change control board (CCB) mechanics and provides practical guidance for setting up a CCB for your software systems development process; the CCB is the most critical process element because it addresses the communications problems that plague *any* software project.
Chapter 5 Product and Process Reviews	Describes basic processes associated with the various reviews called out in Chapter 3 as a means for reducing software systems development risk, thereby enhancing the likelihood of success.
Chapter 6 Measurement	Provides practical guidance for measuring the "goodness" of products and the "goodness" of the software systems development process that produced the products. The focus is on how to use measurement to achieve consistent product and process "goodness"—that is, to achieve successful software systems development.
Chapter 7 Cultural Change	Addresses human issues bearing upon organizational cultural change during implementation of your systems engineering environment (SEE), where the SEE defines your desired way of engineering software systems.
Chapter 8 Process Improvement Planning	Provides practical guidance on how to write an SEE implementation plan to establish the framework for doing the things discussed in the preceding chapters.

Figure P–1 This eight-chapter book, organized as shown, gives you practical and proven guidance for answering the question, "How can you make successful software systems development happen?"

systems development process. In effect, this "way" of doing business helps to set the stage for the rest of the book. There are many "best practices" for software development. The question is, "How do these practices interface with one another?" The organizational "way" presented consists of a set of related processes that embody fundamental engineering and process principles that specifically address effective communication and risk reduction. The organizational "way" presented contains a project planning process, a change control process, product and process review processes, and a measurement process. We define and explain roles, responsibilities, activities, and communication linkages. We present this "way" of developing software systems for your consideration when defining your way of doing business. We stated above a key principle regarding software development—successful software development is an ongoing exercise in risk reduction. In the third chapter, when we present a "way" for developing software systems for your consideration, we stress the following corollary to this key principle:

> If you decide under certain particular circumstances that it may make better sense not to follow your organizational way of doing business, then you should keep in mind that you might be increasing software systems development risk.

Chapter 4 No matter how well a project is planned, it is almost axiomatic that things will change once the project gets underway. Therefore, the fourth chapter presents the fundamental process of change control. This chapter also addresses the communications problems that plague any software systems development project. Sometimes the customer believes that the requirements were effectively communicated to the developer. Sometimes the developer believes the customer requirements were understood. Subsequent to developer implementation of the requirements, the customer and developer may have vastly different perspectives regarding requirements satisfaction. This chapter offers guidance for reducing the likelihood of such disconnects arising in your environment.

Chapter 5 For us, "consistent" software development involves the systems disciplines of management, development, and product assurance. The fifth chapter presents product and process reviews from the perspectives of these three systems development disciplines. This chapter focuses on how reviews help customers and/or sellers gain visibility into project progress and risk so that intelligent, informed decisions can be made with respect to what needs to be done next.

Chapter 6 Measurement for the sake of measurement is a waste of time and resources. The sixth chapter presents practical guidance on how

to express meaningful measurement in everyday language that the intended audiences agree to and understand. Meaningful measurement contributes to (1) successful software systems development projects and (2) improvements in the "way" software systems are developed.

Chapter 7 Pressures such as competition generally push organizations to continue to improve their skill sets, processes, and products. The seventh chapter addresses people issues associated with maturing the organization's "way" of doing business. Getting software systems development processes on paper to document this "way" is a challenge. However, getting people in the organization to help build this "way" and then follow it is an even more imposing challenge. We present ideas for how to deal with this challenge.

Chapter 8 Finally, the eighth chapter presents guidance for planning improvements to your "way" of developing software systems. This chapter helps you select concepts from the preceding chapters to develop a process improvement approach. We discuss how to factor lessons learned from following your "way" of doing business or from not following your "way" into improvement activities. Also, we present other candidate practices for your consideration for improving your "way" of developing software systems.

Table P–1 highlights in more specific terms what you will learn from each chapter.

Table P–1 Chapter Highlights

Chapter	Title and Purpose	What You Will Learn
1	**Business Case**—(1) makes the business case for setting up a "consistent" way of doing software systems development and (2) introduces fundamental concepts needed for the rest of the book.	• What *successful software development* means. • Why investing in software process improvement to achieve consistently "good" products makes good business sense. • Business way refinement/transformation is first and foremost a cultural change exercise. • Successful software development must be a customer/seller partnership, where the "seller" is the software systems development enterprise and the "customer" is the buyer/user of what the seller provides. • The ideas in the book are scalable—they apply to customer/seller partnerships of almost any size. • The ideas in this book encompass customer/seller partnerships in any business environment (e.g., commercial, government). • Why the software development business does not have to be a lottery.

Table P–1 Continued

Chapter	Title and Purpose	What You Will Learn
1	**Business Case** (continued)	• Why successful software development is not inextricably tied to individual heroics to get the job done. • Why there is no one way to build software systems and how this viewpoint influences the way to achieve successful software development. • Why prescriptively applying an organization's business way makes good business sense. • What "prescriptive application of an organization's business way" means and why prescriptive application holds the key to institutionalizing the business way. • Definitions of key terms needed for the rest of the book (e.g., software, software process, software process capability, software process maturity, prescriptive application, product and process "goodness," software process improvement, life cycle, culture). • The role of organizational commitment in making successful software development happen. • Effective customer/seller communication is a key element of successful software development. • A key mechanism for effecting good customer/seller communication is the change control board (CCB). • People are the most important success factor—not automated tools. • Requisite software systems development disciplines for achieving success—management, development, product assurance. • Obstacles to effecting cultural change. • Making software development success happen extends far beyond (1) management edicts, (2) assembling a team of experienced and good people, and (3) a five-minute conversation with a customer and a three-week coding frenzy. • Alternative approaches to refining/transforming an organization's business way. • A systems engineering environment (SEE) provides a means for making successful software development happen—whether systems are developed sequentially or in parallel. • The SEE consists of a process component (application development process environment [ADPE]) and a technology component (ADTE).
2	**Project Planning Process**—provides practical guidance for effectively planning software systems development work.	• The project plan is a living contract that binds the customer/seller partnership by setting forth the work that the seller's management, development, and product assurance disciplines accomplish and the customer's management approves.

Table P–1 Continued

Chapter	Title and Purpose	What You Will Learn
2	**Project Planning Process** (continued)	• Life cycle's role in project planning. • Planning is an ongoing negotiation between the buyer/user and seller. • How to account for the interaction among the requisite disciplines—management, development, and product assurance—throughout the project life cycle. • How to plan for change. • Contrasting views of work accomplishment—ideal, real, and realistic—and their impact on project planning. • How to construct a simple, but powerful, risk assessment approach for project planning use. • How to incorporate risk reduction explicitly into a project plan budget. • How to construct a risk-reduced project plan. • How to develop an ADPE element defining an organization's project planning process.
3	**Software Systems Development Process**—(1) defines principles for putting together an organizationwide software systems development process framework that fosters success and (2) illustrates these principles by defining a top-level process that you can use as a starting point for defining a software development business way for your environment.	• Contractual agreements that can arise in the software systems development business. • How to write a "good" statement of work (SOW), where the SOW is a customer vehicle for communicating to the seller what he/she wants. • How the seller can constitute a project team and define associated responsibilities to accomplish project plan work. • How the customer can effectively interact with the seller project team. • How the seller can define a software systems development process that (1) explicitly includes the customer throughout the process and (2) can incorporate any product development life cycle. • How to plug the seller organization and the customer organization(s) into the software systems development process so that both sides know how business is to be transacted. • More about "prescriptive application" of the software systems development process. • How to address process issues in those environments where numerous software systems development projects are unfolding more or less in parallel. • How does a life cycle plug into the software systems development process. • How level of detail and organizational scope are two major considerations in defining an organizational software systems development process. • How the software systems development process can plug into a systems development process.

Table P–1 Continued

Chapter	Title and Purpose	What You Will Learn
3	**Software Systems Development Process** (continued)	• How to design a form that helps track a product as it winds its way through the software systems development process. • What are the responsibilities of the customer and the seller after the seller delivers the product to the customer. • How to develop an ADPE element defining an organization's software systems development process. • Why this element is a good place to begin setting up an ADPE.
4	**Change Control Process**—defines change control board (CCB) mechanics and provides practical guidance for setting up CCBs for your software projects.	• Why miscommunication can plague *any* software systems development project. • How the customer and seller can have dramatically different views of the state of a product and what to do to reduce the likelihood of such different views arising. • How to manage unplanned change as well as planned change. • Why management of all change is crucial to achieving successful software systems development. • How the need for managing all change mandates a CCB concept that extends far beyond the traditional configuration management control board concept. • Change control mechanics of the software systems development process. • How to establish seller and customer accountability through the CCB. • The three scenarios governing all of change control: 1. Do we want something new or different? 2. Is something wrong? 3. Should we baseline the product? • CCB mechanics (e.g., what to record at CCB meetings, CCB role in traversing a project life cycle, who should be the CCB chairperson, what should be the CCB voting mechanism, what is contained in a CCB charter, how is a CCB meeting conducted, how frequently should the CCB meet). • The information requirements for CCB minutes. • How to write CCB minutes. • When are CCB hierarchies appropriate and how they should be set up. • How to design change control forms that make sense for an organizational way of doing business. • How to develop an ADPE element defining the workings of CCBs in a software systems development process.

Table P–1 Continued

Chapter	Title and Purpose	What You Will Learn
5	**Product and Process Reviews**—describes the basic processes associated with the various reviews called out in Chapter 3 as a means for reducing software systems development risk and thereby achieving success.	• Principles pertaining to the purpose of reviews. • How to resolve key issues regarding the review process involving peers. • The mechanics of document reviews and acceptance testing that an independent product assurance organization conducts. • How to make software requirements testable so that the software systems development process can be brought to a successful conclusion. • What a software audit is and its relationship to reviews. • The key role that acceptance testing plays in harmonizing seller and customer understanding of what the delivered software system and supporting databases are to contain. • Senior management visibility needs and how reviews can help meet these needs. • How technical editing can be incorporated into the software systems development process to help prevent compromising good engineering work. • Technical editing suggestions that can be used as a starting point for constructing an organizational technical editing guide. • How to develop ADPE elements addressing (1) independent product assurance, (2) peer reviews, and (3) the acceptance testing cycle.
6	**Measurement**—provides practical guidance for measuring product "goodness" and the "goodness" of the software systems development process that produced the products. The focus is on how to use measurement to achieve *consistent* product and process "goodness."	• Knowing when it makes sense to try to improve the software systems development process. • How to avoid "measurement for the sake of measurement" activities. • How to use an easy-to-learn and easy-to-apply measurement technique, called **Object Measurement**, that you can use to measure almost anything, including product and process "goodness." • How to establish benchmarks to give meaning to product and process measurements. • How to measure customer satisfaction. • How to quantify the concept of *product integrity* as a means for assessing software product "goodness." • How to extend the quantified product integrity concept to the software systems development process domain to assess process "goodness." • How to use the product "goodness" metric to track product evolution through the software systems development process to head off product development problems. • How to set up value scales for measuring product and process "goodness" in any environment. • How to measure the "goodness" of the process described in Chapter 3.