国外经典计算机科学教材

Fundamentals of Software Engineering

(Second Edition)

软件工程基础

(第二版·影印版)

[意] Carlo Ghezzi
Mehdi Jazayeri 著
Dino Mandrioli

Fundamentals of Software Engineering

(Second Edition)

软件工程基础

Carlo Ghezzi
[意] Mehdi Jazayeri
Dino Mandrioli

中国电力出版社
www.infopower.com.cn

Fundamentals of Software Engineering, 2th edition (ISBN 0-13-305699-6)
Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli

北京市版权局著作合同登记号 图字：01-2005-5626

图书在版编目（CIP）数据
软件工程基础：第 2 版/（意）盖伊（Ghezzi,C.），（意）查耶（Jazayeri,M.），（意）曼德若利（Mandrioli,D.）著. —影印本. —北京：中国电力出版社，2006
（国外经典计算机科学教材系列）
ISBN 7-5083-3876-6
Ⅰ.软... Ⅱ.①盖...②查...③曼... Ⅲ.软件工程－高等学校－教材－英文 Ⅳ.TP311.5
中国版本图书馆 CIP 数据核字（2005）第 134046 号

从 书 名：国外经典计算机科学教材系列
书　　名：软件工程基础（第二版·影印版）
编　　著：（意）Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli
责任编辑：牛贵华
出版发行：中国电力出版社
地址：北京市三里河路 6 号　邮政编码：100044
电话：（010）88515918　传　真：（010）88518169
印　刷：北京同江印刷厂
开本尺寸：185×233　印　张：38.25
书　号：ISBN 7-5083-3876-6
版　次：2006 年 1 月北京第 1 版　2006 年 1 月第 1 次印刷
定　价：58.00 元

出 版 说 明

新世纪的朝阳刚刚露出丝抹微红，如火如荼的全球信息化浪潮便汹涌而至，让人无时无刻不感受到新一轮产业革命的气息。如何在这场变革中占尽先机，既是对民族信息业的挑战，也是机遇。从而，作为民族信息产业发展基石的高等教育事业就被赋予了比以往更重的责任，对培养和造就我国21世纪的一代新人提出了更高的要求。但在计算机科学突飞猛进的同时，专业教材的发展却严重滞后，越来越成为人才培养的瓶颈。同时，以美国为代表的西方国家计算机科学教育经历了充分的发展，产生了一批有着巨大影响力的经典教材，因此，以批判、借鉴的态度有选择地引进这些国外经典计算机教材，将促进国内教学体系和国外接轨，大大推动我国计算机教育事业的发展。

中国电力出版社进入计算机图书市场已有近6个年头，通过坚持“高端、精品、经典”战略，致力于与国外著名出版机构合作，出版了大批博得计算机业界和教育界赞誉的作品。通过与信息技术教育界人士的广泛沟通，同时依托丰富的出版资源，中国电力出版社适时推出了“国外经典计算机科学教材”的出版计划。本次教材出版计划是和美国最大的计算机教育出版机构——Pearson教育集团（Addison-Wesley、Prentice-Hall等皆为其下属子公司）合作，依托其数十年积累的大批经典教材资源，确保了教材选题的权威经典。

为保证这套教材的含金量，并做到有的放矢，我们在国内组织了由中国科学院、北京大学等一流院校教师组成的专家指导委员会，对高校课程教学体系做了系统、详细的调查，听取了众多教育专家、行业专家的意见，对教育部的教育规划进行了认真研究，并深入了解国外大学实际教学选用的教材状况，对国外教材做了理性的分析，确立了依托国家教育计划、传播先进教学理念、为培养符合社会需要的高素质创新型人才服务，来作为本次“国外经典计算机科学教材”出版计划的宗旨。

我们从2002年的下半年开始着手这套教材的策划工作，并多次组织了专家研讨会、座谈会等，分析现有教材的优点与不足，采其精华，并力争体现本套教材的质量和特色。

1. 深入理解国内的教学体系结构，并比照国外相同专业的课程设置，既具有现实的适用性，又立足发展眼光，具备一定的前瞻性。

2. 以计算机专业的核心课程为基础，同时配合专业教学计划，争取覆盖专业选修课程和专业任选课程。

3. 选取国外的最新教材版本，同时对照国内同专业课程的学时要求，对不适用的版本进行剔除，充分满足国内教学要求。

4. 根据专业对口和必须具备同课程教学经验的要求，严格挑选译者，并严把质量关，确保教材翻译的高质量。

5. 通过从原出版社网站下载勘误表及与原书作者进行沟通的方式，对原书中的错误一一做了修改。

6. 对教材出版的后期工作，如审校、编辑、排版、印刷进行了严格的质量把关。

经过专家指导委员会的集体讨论，并广泛听取广大高等院校师生的意见，反复比较，从数百种国外教材中遴选出数十种，列入第一阶段的出版计划。这些教材的作者无一不是学富五车的大师，如 Stallings，Date，Ullman，Aho，Bryant，Sedgewick 等，他们的作品均是一版再版，并被众多国外一流大学如 Stanford University，MIT，UC Bekerley，Carnegie Mellon Univeristy，University of Michigan 等采用为教材。拟订的第一阶段出版计划包括 30 种图书，内容覆盖程序设计、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、离散数学等计算机专业核心基础课程，基本满足国内计算机专业的教学要求。

此外，为了帮助广大任课教师加深对本系列教材的理解，减轻他们的备课难度，我们从国外出版机构引进了大批的课程教学辅助资料，并积极延请国内优秀教师，根据其使用该系列教材中的教学经验，着手编写更加适合国内应用状况的教辅材料。

由于我们对国内高校计算机教育存在认识深度上的不足，在选题、翻译、编辑加工出版等方面的工作中还有许多有待提高之处，恳请广大师生和读者提出批评和建议,并期待有更多的人加入到我们的工作中来。我们的联系方式是：

电子邮件：csbook@cepp.com.cn

联系电话：010-88515918-300

联系地址：北京市西城区三里河路 6 号中国电力出版社

邮政编码：100044

Preface to the Second Edition

The first edition of this book was published in 1991. Since then, there has been a lot of progress in computing technology and also in software engineering. Certainly the proliferation of the Internet has had a profound influence on education, research, development, business, and commerce. We decided to produce this second edition in order to bring the book up to date with respect to the advances in software engineering in the last 10 years.

We were pleased to find that the basic premise of the book—the durability and importance of principles—has been borne out by the passage of time: Even though the technology has improved, principles of software engineering have remained the same. We have therefore been able to update every chapter without changing the original structure of the book. The following is still the structure:

Introduction: Chapters 1 –3;

The product: Chapters 4 –6;

Process and management: Chapters 7 –8;

Tools and environments: Chapter 9.

The product-related chapters follow the sequence consisting of design (4), specification (5), and verification (6). This is different from the approach taken by other books, which cover specification before design. The reason for our choice follows from the principles-based approach of the book. All of these activities—design, specification, and verification—are basic activities that must be learned and applied throughout the software life cycle. For example, design is something we do not only with software architecture, but also with software specifications. The modular design approach helps us structure software and also the specification documents. Other books present specification first and then design, allegedly because—according to the traditional software processes—first we specify a software and then we design it. By contrast, we believe that learning about the design activity and approaches first, creates the needed motivation for the study of specification and provides the skills and techniques for structuring those specifications.

While all areas of software engineering have evolved since the first edition of the book was written, the area of tools and environments has changed substantially. Chapter 9, therefore, is revised considerably. Our approach in this chapter also is to present primarily principles rather than specific tools. We have seen over the years that tools change as technology evolves, and the choice of what particular tools to study depends on the student's environment and focus. We therefore cover a framework for studying and evaluating software tools without a detailed look at any particular tools.

Besides many minor improvements and changes, we have made the following major additions:

In Chapter 3, we have added two new case studies, one of a simple compiler and the other of the elevator system that we use throughout much of the book. The two case studies are complementary in that they deal with different application areas and pose different design challenges. They are presented in this chapter in a simple and intuitive way to get the student oriented towards thinking of system issues. They are intended to illustrate the use of general principles with concrete examples.

In Chapter 4, we have extended the treatment of object orientation, software architecture, components, and distributed systems.

In Chapter 5, we have added a treatment of Z and UML. A new section gives a more systematic treatment of requirements engineering.

In Chapter 6, we have added model checking and GQM as evaluation and verification techniques.

In Chapter 7, we have included a treatment of the unified process, the open-source process, and the synchronize-and-stabilize process. We have also added a new case study on requirements engineering.

In Chapter 8, we have added the capability maturity model and a description of the Nokia software factories.

In Chapter 9, we have added a treatment of the concurrent versioning system (CVS).

In Chapter 10, we have provided coverage of the Software Engineering Code of Ethics.

In the appendix, we have added a new case study on the use of formal methods in industry.

THE ROLE OF OBJECT ORIENTATION

The book covers the principles of object orientation in a balanced way, rather than as the only way to do software engineering. Object-oriented analysis, design, and programming have certainly evolved and become a dominant approach to software engineering. We believe, however, that the principles underlying software engineering are deeper than objects. What the student should learn are principles and methods that can be used in different approaches. The student should learn how to choose between approaches and should be able to apply object orientation when it is the right choice. For example, the student should learn about information hiding before learning about objects and inheritance.

THE PURPOSE OF CASE STUDIES

The case studies presented throughout the book and also in the appendix have two purposes. One is to present the issues discussed in a larger context, in order to give the student a broader view of why the principles or techniques are important. The second reason is to give those students who have not seen real projects a picture of realistic projects. The case studies are necessarily simplified to focus on important issues, but we have found that they are useful especially to less experienced students. The study of software engineering poses a challenge in a university setting because the typical student has not been exposed to the problems that software engineers face daily. These case studies attempt to overcome this challenge.

INSTRUCTOR RESOURCES

A companion CD, including solutions and sample course sylllabi is available to instructors. A companion Web site is available through the publisher to both students and instructors. You may contact the authors through the Web site. We welcome your feedback, comments, and suggestions.

CARLO GHEZZI
Milan, Italy

MEHDI JAZAYERI
Palo Alto, California

DINO MANDRIOLI
Lugano, Switzerland

Preface to the First Edition

This is a textbook on *software engineering*. The theme underlying the book is the importance of rigor in the practice of software engineering. Traditional textbooks on the subject are based on the lifecycle model of software development—that is, requirements, specification, design, coding, maintenance—examining each phase in turn. In contrast, our presentation is based on important principles that can be applied independently of the lifecycle model and often in several phases of the lifecycle. Our emphasis is on identifying and applying fundamental principles that are applicable throughout the software lifecycle.

The general characteristics of the book are the following:

- *It deals with software engineering as opposed to programming.* Thus, we do not discuss any programming issues. For example, we omit any discussion of programming language constructs such as **goto**, loops, etc. We believe that the student of software engineering should have prior familiarity with these issues, which are more properly covered in textbooks on programming languages. On the other hand, we do discuss the issue of mapping software design constructs into specific programming languages. We concentrate on intermodule issues and assume as prerequisite the ability to program individual modules.
- *It emphasizes principles and techniques as opposed to specific tools (which may be used in examples).* Many companies are actively developing software engineering tools and environments today and we expect that better and more sophisticated tools will be invented as our knowledge of software engineering increases. Once the student understands the principles and techniques that the tool is based on, mastery of the tool will be easy. The principles and techniques are applicable across tools while mastering the use of any particular tool does not better prepare the student for the use of other tools. Further, use of tools without understanding their underlying principles is dangerous.
- *It presents engineering principles; it is not an engineering handbook.* Principles are general and are likely to remain applicable for many years while particular techniques will change due to technology, increased knowledge, etc. An engineering handbook may be consulted to learn *how* to apply a particular technique: it contains a set of prescriptions. This book, on the other hand, aims to enable the reader to understand *why* a particular technique should be used and, just as important, why it should *not* be. Even though we do show how a particular technique can be used to implement a given principle, our primary emphasis is on the understanding of the *why* question.

This book embodies our beliefs in the use of fundamental principles and the importance of theory in the practice of engineering. We have used the material in this book in both university and professional courses on various aspects of software engineering.

AUDIENCE

This book is designed to be used as a textbook by students of software engineering either in a classroom or for self-study. Professional engineers and managers will find material here to convince them of the usefulness of modern practices of software engineering and the need to adopt them. It may be used by professionals who are willing to invest the time for serious study; it is not really appropriate for a cursory reading. In particular, wherever necessary, we have sacrificed breadth for depth. For the professional, the notes on further references will be especially helpful. An Instructor's Manual is available with ideas for course organizations and solutions to some of the exercises.

PREREQUISITES

The book is designed for junior, senior, or beginning-graduate level students in computer science. The reader must have had a course in data structures and should be fluent in one or more programming languages. We assume that the reader is already proficient in programming. Analytical reasoning, although not strictly necessary, will greatly enhance the ability of the reader to appreciate the deeper concepts of the book. This skill is developed by mathematics courses such as calculus, discrete mathematics, or-even better-theoretical computer science. "Mathematical maturity" is necessary for the student of any engineering discipline.

ORGANIZATION AND CONTENT

Software engineering is a large, multi-dimensional discipline. Organizing a textbook on the subject poses a challenge because a textbook should present material sequentially, but the many facets of software engineering are so interrelated that there is no optimal sequence of topics. We have organized this textbook based on the view that in software engineering:

We are building a *product*: the software;
We use a *process* to build that product; and
We use *tools* in support of that process.

The book thus has three technical sections dealing in turn with the software product (Chapters 4 through 6), the software engineering process and management (Chapters 7 and 8), and the software engineering environment (Chapter 9). Chapters 1 through 3 form a general introduction to the field and the subsequent more technical sections of the book.

In Chapter 2, we discuss the many facets of software and common desirable characteristics for software. These characteristics impose constraints on the software builder and the process to be used. In Chapter 3, we present principles for building high-quality software. By studying principles rather than specific tools, the student gains knowledge that is independent of a particular technology and application environment. Because technology changes and environments evolve, the student should be armed with principles and techniques that can be utilized in different application areas. Chapters 4 through 8 present and discuss techniques for applying the principles of Chapter 3 to, respectively, design, specification, verification, engineering process, and engineering

management. In Chapter 9, we discuss the use of computers themselves to help in the building of software. We postpone the discussion of any specific tools to this chapter.

While the material in the first two sections should withstand the passage of time, it is likely that the material in the third section will become outdated (we hope) because newer and better tools are being developed. Since programming languages are a fundamental tool of the software engineer, we use Chapter 9 as a bridge between the design issues of Chapter 4 and specific programming language constructs.

EXERCISES

The book contains many exercises of three types:

- short, paper exercises, aimed at extending the knowledge gained from the book or applying the knowledge more deeply; these exercises are interspersed throughout the chapters.
- longer paper exercises at the end of each chapter, requiring integration of the material in the chapter.
- term-projects requiring the development of some substantial software system by a small team.

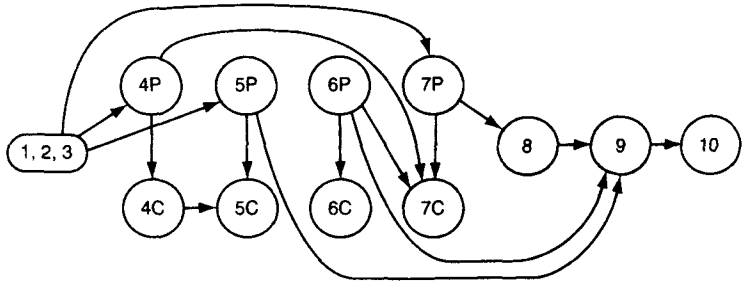
Solutions to some of the exercises are provided at the end of each chapter. More exercise solutions are given in the Instructor's Manual.

CASE STUDIES

Several case studies are used in the text to demonstrate the integration of different concepts and to contrast different approaches in realistic situations. In addition, three case studies of real-life software engineering projects and their analyses are presented at the end of the book. These case studies may be read and studied at different times and for different purposes. From these case studies, the new student with little industrial experience can gain a quick view of the diversity of problems faced in industrial practice. The student with some experience perhaps will identify with certain aspects of these case studies and learn from others. The case studies may be read concurrently with the main text. Several exercises in the book refer to these case studies.

LABORATORY COURSE

Many software engineering courses combine lectures and a laboratory project. To do this in a single semester is rather difficult. The teacher will find himself discussing organizational issues while the students are concentrating on their daily forays into debugging. We believe that software engineering must be taught as all other engineering disciplines by first providing the student with a solid foundation in the "theory." Only after this has been achieved will laboratory experience enhance the student's knowledge. This implies that the student project must start closer to the middle of the semester rather than at the beginning. In our view, a better approach is to spend one semester on the theory and a second semester on the laboratory. The Instructor's Manual offers several ideas for organizing a laboratory course based on this text.



READING GRAPH

The book may be read in different sequences and at different levels. Each of Chapters 4 through 7 contains material that may be skipped on the first reading or for a less detailed study. Chapters 1 through 3 are required reading for the subsequent chapters. The graph shows the dependencies among the chapters and the various paths through the book. The notation *n*P refers to a partial reading of Chapter *n*, skipping some sections; *n*C stands for a complete reading.

The Instructor's Manual discusses different course organizations based on the book. The conventional one-semester project software engineering course may follow the sequence: 1, 2, 3, 7P, 5P, 4P, 6P, 8, 9, 10. We ourselves prefer the sequence 1, 2, 3, 4P, 5P, 6P, 7P, 8, 9, 10. In either case, the students should start on the project after 5P.

ACKNOWLEDGMENTS

We gratefully acknowledge reviews of earlier drafts provided by Reda A. Ammar of the University of Connecticut, Larry C. Christensen of Brigham Young University, William F. Decker of the University of Iowa, David A. Gustafson of Kansas State University, Richard A. Kemmerer of the University of California at Santa Barbara, John C. Knight of the University of Virginia, Seymour V. Pollack of Washington University, and K. C. Tai of North Carolina State University.

We would also like to thank the following people who have provided valuable feedback on various drafts of the manuscript: Vincenzo Ambriola, Paola Bertaina, David Jacobson, and Milon Mackey.

Hewlett-Packard Laboratories and Politecnico di Milano made it possible to conceive this book by supporting a course offered by Mehdi Jazayeri at the Politecnico di Milano during the spring of 1988. Alfredo Scarfone and HP Italiana provided us with support in Italy. We would like to acknowledge the support of management at Hewlett-Packard Laboratories, especially John Wilkes, Dick Lampman, Bob Ritchie, and Frank Carrubba in Palo Alto, and Peter Porzer in Pisa. We would like to thank Bart Sears for his help with various systems, and John Wilkes for the use of his data base for managing references. We have also received support from Consiglio Nazionale delle Ricerche.

CARLO GHEZZI
Milan, Italy

MEHDI JAZAYERI
Palo Alto, California

DINO MANDRIOLI
Pisa, Italy

Contents