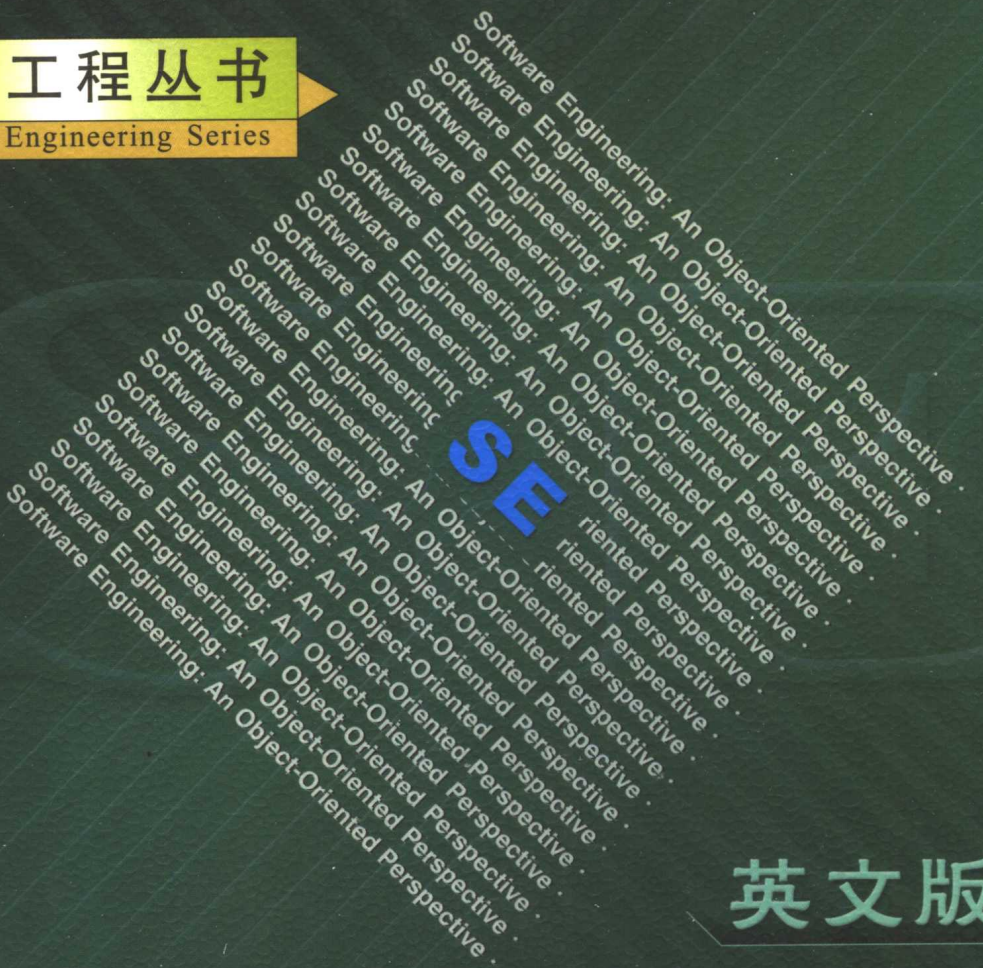
软件工程丛书
Software Engineering Series



英文版

面向对象的软件工程

Software Engineering
An Object-Oriented Perspective

[美] Eric J. Braude 著

WILEY
電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

软件工程丛书

面向对象的软件工程

（英文版）

Software Engineering

An Object-Oriented Perspective

[美] Eric J. Braude 著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书结合面向对象的分析和设计方法，深入浅出地讲解了软件工程原理，并详细描述了实施这些软件工程活动的方法和标准。书中包括了项目管理、需求分析、软件体系结构、单元实现和测试、系统整合、验证以及维护等内容。同时，作者通过一个精彩的连续案例分析，帮助读者深刻理解先进的项目管理技术、各种质量因素、全面细致的需求文档和现代设计方法为软件开发所带来的种种益处。

本书适于计算机相关专业高年级本科生和低年级研究生阅读，也可以作为软件开发人员参与团队开发的学习资料。

Eric J. Braude: **Software Engineering: An Object-Oriented Perspective.**
ISBN 0-471-32208-3

版权贸易合同登记号：图字：01-2003-1040

图书在版编目（CIP）数据

面向对象的软件工程 = Software Engineering: An Object-Oriented Perspective /（美）布劳德（Braude, E.J.）著. -北京：电子工业出版社，2003.4
（软件工程丛书）
ISBN 7-5053-8622-0

Ⅰ.面... Ⅱ.布... Ⅲ.软件工程-英文 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2003）第 022574 号

责任编辑：赵宏英
印 刷 者：北京东光印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路 173 信箱 邮编：100036
经 销：各地新华书店
开 本：787 × 980 1/16 印张：32.5 字数：728 千字
版 次：2003 年 4 月第 1 版 2003 年 4 月第 1 次印刷
定 价：49.00 元

PREFACE

This book is not just *about* software engineering, but also how to do software engineering. Such a book has not been fully attempted in the past because no technical approach has enjoyed sufficiently broad acceptance. The object-oriented approach, in particular, has merely been included as a section in software engineering textbooks, despite the fact that a very large proportion of contemporary projects utilize object-oriented languages.

During the 1990's, the Object-Oriented Analysis and Design community fashioned an approach to application design, together with a corresponding notation: the Unified Modeling Language. The broad acceptance of this approach and language makes the new millennium's beginning an appropriate time to teach the doing of software engineering, and not merely talk about software engineering. Thus, although this book necessarily includes aspects of software engineering that are not object-oriented, it is designed to support instruction in the application of frameworks, use cases, and design patterns: It also relates object-orientation to techniques for requirements analysis and testing. As a result, instructors can spend less time trying to cover numerous approaches, and more time promoting depth and practice.

Any book showing how to do software engineering must include a case study. In addition, since software engineering deals largely with complexity, a software engineering textbook needs a substantial case study, rather than a token one. Finally, the case study should be interesting enough to students so that they can envision building upon it just for fun. For these reasons, this book shows throughout how software engineering principles are applied to the construction of a particular role-playing video game. Video games afford a rich opportunity to demonstrate frameworks, design patterns, state behavior, concurrency, and nontrivial graphical user interfaces. Scientific and business examples complement the case study.

The typical software product is built by a team of software engineers, rather than by an individual working alone. To satisfy the corresponding educational need, this book provides extensive support for student teams. Watts Humphrey's pioneering work on the Personal Software Process [Hu] and the Team Software Process [Hu7] inspired much of this support.

Audience

This book is intended for senior-level undergraduates and first year graduate students. Since the goal of the book is to produce good practice in developing software, the book will also be useful for practicing professionals who wish to improve their knowledge and performance. The text assumes familiarity with programming using classes and objects, preferably in Java.

Organization

Each of Chapters 1, 2, 4, 5, and 6 is divided into two parts. This is designed to help those readers and instructors who wish to progress rapidly through the fundamentals of requirements analysis and design. They can do this by first covering Part I of these chapters, returning later for the Part II's.

- The introduction provides a brief overview of software engineering, together with suggestions for student teams. This section also provides a summary of the case study so that students can be reassured that technical challenges do indeed await them beyond the "process" and project management concepts that they must understand.
- Chapter 1 provides an extensive introduction to software engineering. Section 6, on process, is the heart of this chapter.
- Chapter 2 is concerned with how software projects are organized. Technical people sometimes try to avoid this subject, but they will be much happier in their team work if they understand organizational and management issues. In particular, Section 1, an introduction, and Section 4 on risk, discuss knowledge which is indispensable for software engineers.

Chapters 3 through 10 follow the logical order in which software is produced during each iteration.

- Chapters 3 and 4 concern "requirements analysis": the process of understanding what is to be produced.
- Chapters 5 and 6 indicate how products are designed and how designs are expressed.
- Chapter 7 discusses programming in the context of software engineering.
- Chapters 8 and 9 focus on the testing process.
- Chapter 10 discusses the activities required after the product has been released.
- References are found at the end of the Book. Acronyms are summarized on page 475.

Ways to use this book

Although the sequence of the book's chapters is logical, it does not entirely parallel the way in which applications are actually produced. The *requirements analysis / design / program / test* sequence is usually repeated at least once. Chapters 1 and 2 discuss the ways in which this repetition can be organized.

There are several basic ways in which this book can be used, each motivated by different priorities, and these are discussed next.

The linear way to use this book: Read the chapters in this order: Introduction / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10 The author has taught from this material by reviewing this preface, followed by Chapters 1 through 10 in sequence. He has relied on chapters 1 and 2 for an overview, and encouraged class teams to first go through elements of the *requirements analysis/design/program/test* sequence with a *trivial* set of requirements. This accustoms the group to the idea of "process", exercises group interaction, and exposes the members to the problems to be faced. It requires little exposure to the substance of the material beyond chapter 2.

The "Two-pass" way to use this book: Read the chapters in this order: Introduction / 1 Part I / 2 Part I / 3 / 4 Part I / 5 Part I / 6 part I / 1 Part II / 2 Part II / 4 Part II / 5 Part II / 6 Part II / 7 / 8 / 9 / 10 The first pass covers the introductory sections of each chapter, and the second pass extends to the entire chapters. This sequence has the advantage of enabling teams to build a small prototype while reviewing the Part I sections, then embarking on the full development process afterwards. The author recommends that this prototype be extremely modest. Its main purpose should be to get the team working together, and to exercise aspects of the software engineering process. These activities take a long time for new teams. Useful features should not be expected.

The "career ladder" way to use this book: Read the chapters in this order:Introduction / 1/ 7 / 8 / 6 / 5 / 2 / 3 / 4 / 9 / 10 After introducing software engineering using Chapter 1, other instructors may prefer

to order the topics according to the typical career of a software engineer within a company. Careers start with the role of *programmer* (this starts with Chapters 7 and 8, using the case study design of Chapter 6 as the basis for an example). Programmers are eventually given the responsibility of a *designer* (Chapter 6, using the case study architecture in Chapter 4). Designers typically transition into the role of *architect* (Chapter 5 using the requirements in the case study of Chapter 6). The final career level for which this book is relevant is *project leader* (Chapters 2, 3, 4, 9 and 10).

The website for this book

Among the features of the book's web site at http://www.wiley.com/college/braude, are the following.

- Slides of all the figures and bullet sets in the book, in color, and in original PowerPoint form. This allows instructors to modify and customize the slides, and to selectively integrate them with other slides.
- Answers to exercises for faculty (in a password-protected mode)
- Java source code for the book's case study

There are many ongoing plans for the website, and the reader is referred there for its list of current features.

Exercises

There are three kinds of exercises in each chapter. "Review" exercises have short answers, and a solution or a hint to each is provided in the same chapter as the question. "Team" exercises provide specific goals and evaluation criteria for teams performing term projects. Solutions to the third exercise type, "general" exercises, are not provided in the book, but are available to instructors at the book's web site.

Acknowledgment

In my career as a software engineer and manager in industry, and in my present practice as a professor and consultant, I have been struck by how widespread a hunger there is for learning how to do software engineering "right", within the relentless pressures of business. I am grateful to my colleagues and students for taking the time to articulate this need when I have asked them about it.

For assisting me at every step in writing this book, my gratitude to Dick Bostwick is boundless. I am indebted to Tom van Court for his extensive and painstaking assistance. To my students at the Metropolitan College of Boston University: thank you all for your feedback. To the reviewers: your comments and reviews have made a significant, positive difference to this book. The reviewers included the following: Henry A. Etlinger, Rochester Institute of Technology; Michael Godfrey, University of Waterloo; David A. Gustafson, Kansas State University; Peter Hitchcock, DalTech, Dalhousie University; Floyd Lecureux, California State University—Sacramento; Steven P. Reiss, Brown University; and Laurie Werth, The University of Texas at Austin. I want to thank my colleagues and administrators at Boston University's Metropolitan College for their interest and encouragement. I am most grateful to Paul Crockett, Jenny Welter, and Bill Zobrist at John Wiley & Sons, for working so diligently with me on this project.

Finally, this brief space allows me, however inadequately, to record my deep appreciation to my wife, Judy, and my son, Michael, for supporting my passion for writing this book.

Eric J. Braude
Boston University
Metropolitan College
Boston, Massachusetts; April 2000

CONTENTS

INTRODUCTION

". . . enterprises of great pith and moment . . ."
— Hamlet

This introduction describes what Software Engineering is, and how this book is organized.

The creation of large software applications is one of the most important engineering challenges of modern times.

- Sections 1 through 9
- Exercises

1. The Context of Software Engineering

Software engineering is by definition a kind of engineering, and it therefore has the same set of social responsibilities as all of the other kinds of engineering.

During the history of computing, much of the work of software people has been regarded as "development," that uses programming language skills but little *engineering* discipline. The Accreditation Board for Engineering and Technology defines engineering as shown in Figure 1. Much thought had been given to engineering as a human endeavor long before the birth of software. As of the early 2000s, software engineering is beginning to command the same degree of discipline from its practitioners as other branches of engineering such as electrical, mechanical, and civil. The nature of that discipline is the theme of this book.

How is software engineering different from, and how is it the same as, other kinds of engineering? One property that software engineering shares with the others is the necessity for a thorough description of what is to be

produced, a process called "requirements analysis." On the other hand, software projects are subject to particularly frequent changes, including those imposed while the product is under development.

> The *profession* in which
> a knowledge of the *mathematical* and
> *natural sciences* gained by study, experience, and practice
> is *applied with judgment*
> to develop ways to *utilize*, economically, the
> *materials and forces of nature for the benefit of*
> *mankind*
>
> —Accreditation Board for Engineering and Technology, 1996

Figure 1 A Definition of "Engineering"

2. The Activities of Software Engineering

Two trends dominated software engineering in the 1980s and 1990s. One was the explosive growth of applications, including those associated with the Web. The other trend was a flowering of new tools and paradigms (ways of thinking, such as object-orientation).

Despite the advent of new trends, however, the basic activities required for the construction of software have remained stable. These activities include those listed in Figure 2 and Figure 3.

- *defining* the software development *process* to be used
  - Chapter 1
- *managing* the development *project*
  - introduced in Chapter 2; also referenced in the remaining chapters
- *describing* the intended software *product*
  - Chapters 3 and 4
- *designing* the *product*
  - Chapters 5 and 6

Figure 2 Basic Activities of Software Engineering, 1 of 2

Development teams vary both the sequence and the frequency of these activities, as explained in Chapter 2. Software development in the real world is usually driven by a demanding list of features, as well as tight, market-driven deadlines. As a result, only well-organized groups of engineers, educated in the methods of software engineering, are capable of carrying out these activities appropriately. The alternative is often chaos, and sometimes disaster.

Software engineering involves *people*, *process*, *project*, and *product*, as suggested by Figure 4. The symbols used are from the Unified Software Development Process (USDP) of Jacobson, Booch, and Rumbaugh [Ja1],

one of the processes for software development explained in this book. The icon shown in the "process" part of Figure 4 is explained in Chapter 1. The "project" diagram shows engineers doing various kinds of work, according to their role, and then passing the results to other engineers who then perform their roles. The "products" of a software development effort consist of much more than the object code and the source code. For example, they also include documentation, test results, and productivity measurements. In conformance with the USDP, we will call these products *artifacts*. This book describes what a complete set of artifacts contains.

- *implementing* the *product*
  i.e. programming it
  - Chapter 7
- *testing* the *parts* of the product
  - Chapter 8
- *integrating* the parts and testing them as a whole
  - Chapter 9
- *maintaining* the product
  - Chapter 10

Figure 3 Basic Activities of Software Engineering, 2 of 2

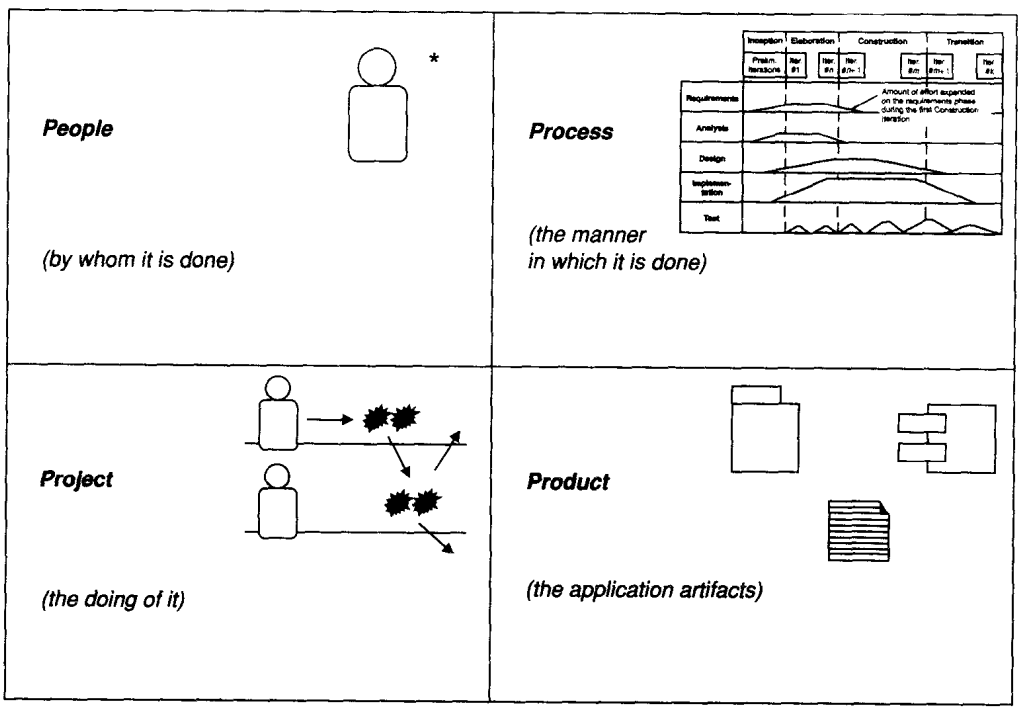


Figure 4 The Four "P"'s of Software Engineering

3. Process

This section is summarized in Figure 5. The "waterfall" process begins with the specification of the requirements for the application, then proceeds to the design phase, then the implementation phase and, finally, the testing phase. The maintenance phase, described in Chapter 10, is sometimes included in the waterfall process. In this book, we have divided "design" into "architecture" (Chapter 5) and "detailed design" (Chapter 6), and "testing" into "unit testing" (the parts: Chapter 8), and system testing (the whole: Chapter 9). Software development rarely occurs in the strict "waterfall" sequence. Web development, for example, tends to skip back and forth among specification, design, integration, and testing. In practice, then, we often use *iterative* processes for software development, in which the waterfall is repeated several times in whole or in part. This is explained in Chapter 1. When performed in a disciplined manner, iterative styles can be highly beneficial.

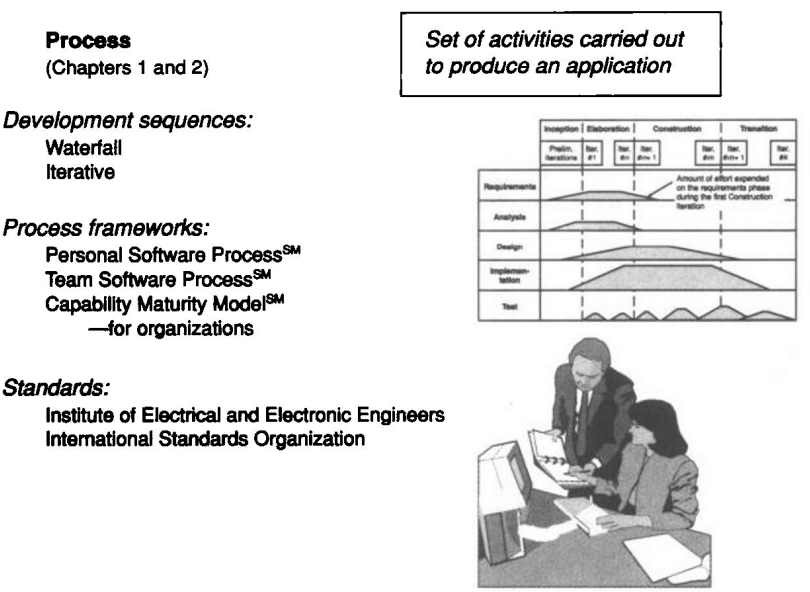


Figure 5 "Process" (graphics reproduced with permission from Corel)

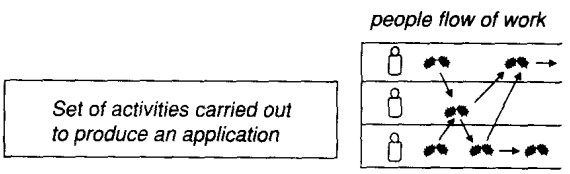
Decisions about software process often take place at an organizational level (company, department, group, etc.) and so it becomes important to measure the software development capabilities of organizations. Several chapters discuss such a measure, the *Capability Maturity Model℠* (CMM). The CMM was developed by Watts Humphrey and the Software Engineering Institute (SEI). The SEI is described in Chapter 1.

The software engineering capability of individual engineers can be developed and measured by the *Personal Software Process℠* (PSP) created by Humphrey [Hu]. The highlights of CMM and PSP are woven through several chapters of this book. A third level of software organization is Humphrey's *Team Software Process℠* (TSP) [Hu7] which describes the process by which teams of software engineers get their work done. The author believes that disciplined frameworks such as the CMM, PSP, and TSP will form a basis for the professional software engineer in the twenty-first century.

Well thought out documentation standards make it much easier to produce useful, reliable artifacts. Several standards are available. For the most part, this book applies the IEEE (Institute of Electrical, and Electronics Engineers) software engineering standards, many of which are also sanctioned by ANSI (American National Standards Institute). Many companies provide in-house standards. Although standards have to be modified over time to reflect new issues, the core of good standards has remained stable for a number of years. Unless they work from standards and, if possible, case studies applying them, teams typically waste a great deal of time dreaming up the structure (as opposed to the substance) of documents. Standards focus the process by providing a baseline for engineer, instructor, and students. In practice, they are modified and tailored to specific projects.

4. Project

A *project* is the set of activities needed to produce the required artifacts. It includes contact with the customer, writing the documentation, developing the design, writing the code, and testing the product. Selected aspects of projects are summarized in Figure 6.



- *Object Orientation*: very useful paradigm
- *Unified Modeling Language*: design notation
- *Legacy system*: common starting point
  - enhancement or usage of existing system

Figure 6 "Project"

The object-oriented paradigm (way of thinking) can be very useful for project development. It is particularly helpful in facilitating continual change because it can be used to organize designs and code in parts (classes and packages) that match the real-world problem.

The Unified Modeling Language (UML: see [Ra]) is an industry standard for describing designs, and is used throughout this book. Note that the UML is not a methodology in itself, but a notation. The UML is summarized on the book's inside covers.

Chapter 5 explores the ways in which the architecture of an application can be developed. The approach borrows from the exciting field of Design Patterns, and from research classifying software architecture. Chapter 6 completes the discussion of designs, demonstrating how complete details can be specified. Chapters 7 through 9 cover the integration and test of the application. Chapter 10 discusses maintenance, the last— and ongoing— process phase.

The overwhelming proportion of real world development work is not the building of brand new systems at all, but the enhancement or usage of existing ("legacy") systems. Even applications which are apparently new, usu-