Software Engineering

TENTH EDITION

Ian Sommerville

ALWAYS LEARNING

PEARSON

SOFTWARE ENGINEERING

Tenth Edition

Ian Sommerville

PEARSON

Boston Columbus Indianapolis New York San Francisco Hoboken
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: Marcia Horton
Editor in Chief: Michael Hirsch
Acquisitions Editor: Matt Goldstein
Editorial Assistant: Chelsea Bell
Assistant Acquisitions Editor, Global Edition: Murchana Borthakur
Associate Project Editor, Global Edition: Binita Roy
Managing Editor: Jeff Holcomb
Senior Production Project Manager: Marilyn Lloyd
Director of Marketing: Margaret Waples
Marketing Coordinator: Kathryn Ferranti
Senior Manufacturing Buyer: Carol Melville
Senior Manufacturing Controller, Production, Global Edition: Trudy Kimber
Text Designer: Susan Raymond
Cover Art Designer: Lumina Datamatics
Cover Image: © Andrey Bayda/Shutterstock
Interior Chapter Opener: © graficart.net/Alamy
Full-Service Project Management: Rashmi Tickyani, Aptara®, Inc.
Composition and Illustrations: Aptara®, Inc.

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

Authorized adaptation from the United States edition, entitled Software Engineering, 10th edition, ISBN 978-0-13-394303-0, by Ian Sommerville, published by Pearson Education © 2016.

ISBN 10: 1-292-09613-6
ISBN 13: 978-1-292-09613-1

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

Typeset in 9 New Aster LT Std by Aptara®, Inc.

Printed and bound by Courier Westford in the United States of America.

PREFACE

Progress in software engineering over the last 50 years has been astonishing. Our societies could not function without large professional software systems. National utilities and infrastructure—energy, communications and transport—all rely on complex and mostly reliable computer systems. Software has allowed us to explore space and to create the World Wide Web—the most significant information system in the history of mankind. Smartphones and tablets are ubiquitous and an entire 'apps industry' developing software for these devices has emerged in the past few years.

Humanity is now facing a demanding set of challenges—climate change and extreme weather, declining natural resources, an increasing world population to be fed and housed, international terrorism, and the need to help elderly people lead satisfying and fulfilled lives. We need new technologies to help us address these challenges and, for sure, software will have a central role in these technologies. Software engineering is, therefore, critically important for our future on this planet. We have to continue to educate software engineers and develop the discipline so that we meet the demand for more software and create the increasingly complex future systems that we need.

Of course, there are still problems with software projects. Systems are still sometimes delivered late and cost more than expected. We are creating increasingly complex software systems of systems and we should not be surprised that we encounter difficulties along the way. However, we should not let these problems conceal the real successes in software engineering and the impressive software engineering methods and technologies that have been developed.

This book, in different editions, has now been around for over 30 years and this edition is based around the essential principles that were established in the first edition:

1. I write about software engineering as it is practiced in industry, without taking an evangelical position on particular approaches such as agile development or formal methods. In reality, industry mixes techniques such as agile and plan-based development and this is reflected in the book.

2. I write about what I know and understand. I have had many suggestions for additional topics that might be covered in more detail such as open source development, the use of the UML and mobile software engineering. But I don't really know enough about these areas. My own work has been in system dependability and in systems engineering and this is reflected in my selection of advanced topics for the book.

I believe that the key issues for modern software engineering are managing complexity, integrating agility with other methods and ensuring that our systems are secure and resilient. These issues have been the driver for the changes and additions in this new edition of my book.

Changes from the 9th edition

In summary, the major updates and additions in this book from the 9th edition are:

- I have extensively updated the chapter on agile software engineering, with new material on Scrum. I have updated other chapters as required to reflect the increasing use of agile methods of software engineering.
- I have added new chapters on resilience engineering, systems engineering, and systems of systems.
- I have completely reorganized three chapters covering reliability, safety, and security.
- I have added new material on RESTful services to the chapter covering service-oriented software engineering.
- I have revised and updated the chapter on configuration management with new material on distributed version control systems.
- I have moved chapters on aspect-oriented software engineering and process improvement from the print version of the book to the web site.
- New supplementary material has been added to the web site, including a set of supporting videos. I have explained key topics on video and recommended related YouTube videos.

The 4-part structure of the book, introduced in earlier editions, has been retained but I have made significant changes in each part of the book.

1. In Part 1, Introduction to software engineering, I have completely rewritten Chapter 3 (agile methods) and updated this to reflect the increasing use of Scrum. A new case study on a digital learning environment has been added to Chapter 1 and is used in a number of chapters. Legacy systems are covered in more detail in Chapter 9. Minor changes and updates have been made to all other chapters.

2. Part 2, which covers dependable systems, has been revised and restructured. Rather than an activity-oriented approach where information on safety, security and reliability is spread over several chapters, I have reorganized this so that each topic has a chapter in its own right. This makes it easier to cover a single topic, such as security, as part of a more general course. I have added a completely new chapter on resilience engineering which covers cybersecurity, organizational resilience, and resilient systems design.

3. In Part 3, I have added new chapters on systems engineering and systems of systems and have extensively revised the material on service-oriented systems engineering to reflect the increasing use of RESTful services. The chapter on aspect-oriented software engineering has been deleted from the print version but remains available as a web chapter.

4. In Part 4, I have updated the material on configuration management to reflect the increasing use of distributed version control tools such as Git. The chapter on process improvement has been deleted from the print version but remains available as a web chapter.

An important change in the supplementary material for the book is the addition of video recommendations in all chapters. I have made over 40 videos on a range of topics that are available on my YouTube channel and linked from the book's web pages. In cases where I have not made videos, I have recommended YouTube videos that may be useful.

I explain the rationale behind the changes that I've made in this short video:

http://software-engineering-book/videos/10th-edition-changes

Readership

The book is primarily aimed at university and college students taking introductory and advanced courses in software and systems engineering. I assume that readers understand the basics of programming and fundamental data structures.

Software engineers in industry may find the book useful as general reading and to update their knowledge on topics such as software reuse, architectural design, dependability and security and systems engineering.

Using the book in software engineering courses

I have designed the book so that it can be used in three different types of software engineering course:

1. *General introductory courses in software engineering.* The first part of the book has been designed to support a 1-semester course in introductory software engineering. There are 9 chapters that cover fundamental topics in software engineering.

If your course has a practical component, management chapters in Part 4 may be substituted for some of these.

2. *Introductory or intermediate courses on specific software engineering topics.* You can create a range of more advanced courses using the chapters in parts 2–4. For example, I have taught a course in critical systems using the chapters in Part 2 plus chapters on systems engineering and quality management. In a course covering software-intensive systems engineering, I used chapters on systems engineering, requirements engineering, systems of systems, distributed software engineering, embedded software, project management and project planning.
3. *More advanced courses in specific software engineering topics.* In this case, the chapters in the book form a foundation for the course. These are then supplemented with further reading that explores the topic in more detail. For example, a course on software reuse could be based around Chapters 15–18.

Instructors may access additional teaching support material from Pearson's website. Some of this is password-protected and instructors using the book for teaching can obtain a password by registering at the Pearson website. The material available includes:

- Model answers to selected end of chapter exercises.
- Quiz questions and answers for each chapter.

You can access this material at:

www.pearsonglobaleditions.com/Sommerville

Book website

This book has been designed as a hybrid print/web text in which core information in the printed edition is linked to supplementary material on the web. Several chapters include specially written 'web sections' that add to the information in that chapter. There are also six 'web chapters' on topics that I have not covered in the print version of the book.

You can download a wide range of supporting material from the book's website (software-engineering-book.com) including:

- A set of videos where I cover a range of software engineering topics. I also recommend other YouTube videos that can support learning.
- An instructor's guide that gives advice on how to use the book in teaching different courses.
- Further information on the book's case studies (insulin pump, mental health care system, wilderness weather system, digital learning system), as well other case studies, such as the failure of the Ariane 5 launcher.

- Six web chapters covering process improvement, formal methods, interaction design, application architectures, documentation and aspect-oriented development.
- Web sections that add to the content presented in each chapter. These web sections are linked from breakout boxes in each chapter.
- PowerPoint presentations for all of the chapters in the book and additional PowerPoint presentations covering a range of systems engineering topics are available at pearsonglobaleditions.com/Sommerville.

In response to requests from users of the book, I have published a complete requirements specification for one of the system case studies on the book's web site. It is difficult for students to get access to such documents and so understand their structure and complexity. To avoid confidentiality issues, I have re-engineered the requirements document from a real system so there are no restrictions on its use.

Contact details

Website: software-engineering-book.com
Email: name: software.engineering.book; domain: gmail.com
Blog: iansommerville.com/systems-software-and-technology
YouTube: youtube.com/user/SoftwareEngBook
Facebook: facebook.com/sommerville.software.engineering
Twitter: @SoftwareEngBook or @iansommerville (for more general tweets)

Follow me on Twitter or Facebook to get updates on new material and comments on software and systems engineering.

Acknowledgements

A large number of people have contributed over the years to the evolution of this book and I'd like to thank everyone (reviewers, students and book users) who have commented on previous editions and made constructive suggestions for change. I'd particularly like to thank my family, Anne, Ali, and Jane, for their love, help and support while I was working on this book (and all of the previous editions).

Ian Sommerville,
September 2014

Contents at a glance

	Preface	3
Part 1	Introduction to Software Engineering	15
Chapter 1	Introduction	17
Chapter 2	Software processes	43
Chapter 3	Agile software development	72
Chapter 4	Requirements engineering	101
Chapter 5	System modeling	138
Chapter 6	Architectural design	167
Chapter 7	Design and implementation	196
Chapter 8	Software testing	226
Chapter 9	Software evolution	255
Part 2	System Dependability and Security	283
Chapter 10	Dependable systems	285
Chapter 11	Reliability engineering	306
Chapter 12	Safety engineering	339
Chapter 13	Security engineering	373
Chapter 14	Resilience engineering	408
Part 3	Advanced Software Engineering	435
Chapter 15	Software reuse	437
Chapter 16	Component-based software engineering	464
Chapter 17	Distributed software engineering	490
Chapter 18	Service-oriented software engineering	520
Chapter 19	Systems engineering	551
Chapter 20	Systems of systems	580
Chapter 21	Real-time software engineering	610
Part 4	Software Management	639
Chapter 22	Project management	641
Chapter 23	Project planning	667
Chapter 24	Quality management	700
Chapter 25	Configuration management	730
	Glossary	757
	Subject index	777
	Author index	803

Pearson wishes to thank and acknowledge the following people for their work on the Global Edition:

Contributor

Sherif G. Aly, The American University in Cairo
Muthuraj M., Android developer

Reviewers

Mohit P. Tahiliani, National Institute of Technology Karnataka, Surathkal
Chitra Dhawale, P. R. Patil Group of Educational Institutes, Amravati
Sanjeevni Shantaiya, Disha Institute of Management & Technology

CONTENTS

Preface 3

Part 1 Introduction to Software Engineering 15

Chapter 1 Introduction 17

1.1 Professional software development 19
1.2 Software engineering ethics 28
1.3 Case studies 31

Chapter 2 Software processes 43

2.1 Software process models 45
2.2 Process activities 54
2.3 Coping with change 61
2.4 Process improvement 65

Chapter 3 Agile software development 72

3.1 Agile methods 75
3.2 Agile development techniques 77
3.3 Agile project management 84
3.4 Scaling agile methods 88

Chapter 4 Requirements engineering 101

4.1 Functional and non-functional requirements 105
4.2 Requirements engineering processes 111
4.3 Requirements elicitation 112
4.4 Requirements specification 120
4.5 Requirements validation 129
4.6 Requirements change 130

Chapter 5 System modeling 138

5.1 Context models 141
5.2 Interaction models 144
5.3 Structural models 149
5.4 Behavioral models 154
5.5 Model-driven architecture 159

Chapter 6 Architectural design 167

6.1 Architectural design decisions 171
6.2 Architectural views 173
6.3 Architectural patterns 175
6.4 Application architectures 184

Chapter 7 Design and implementation 196

7.1 Object-oriented design using the UML 198
7.2 Design patterns 209
7.3 Implementation issues 212
7.4 Open-source development 219

Chapter 8 Software testing 226

8.1 Development testing 231
8.2 Test-driven development 242

8.3 Release testing 245
8.4 User testing 249

Chapter 9 Software evolution 255

9.1 Evolution processes 258
9.2 Legacy systems 261
9.3 Software maintenance 270

Part 2 System Dependability and Security 283

Chapter 10 Dependable systems 285

10.1 Dependability properties 288
10.2 Sociotechnical systems 291
10.3 Redundancy and diversity 295
10.4 Dependable processes 297
10.5 Formal methods and dependability 299

Chapter 11 Reliability engineering 306

11.1 Availability and reliability 309
11.2 Reliability requirements 312
11.3 Fault-tolerant architectures 318
11.4 Programming for reliability 325
11.5 Reliability measurement 331

Chapter 12 Safety engineering 339

12.1 Safety-critical systems 341
12.2 Safety requirements 344
12.3 Safety engineering processes 352
12.4 Safety cases 361

Chapter 13 Security engineering 373

13.1 Security and dependability 376
13.2 Security and organizations 380
13.3 Security requirements 382
13.4 Secure systems design 388
13.5 Security testing and assurance 402

Chapter 14 Resilience engineering 408

14.1 Cybersecurity 412
14.2 Sociotechnical resilience 416
14.3 Resilient systems design 424

Part 3 Advanced Software Engineering 435

Chapter 15 Software reuse 437

15.1 The reuse landscape 440
15.2 Application frameworks 443
15.3 Software product lines 446
15.4 Application system reuse 453

Chapter 16 Component-based software engineering 464

16.1 Components and component models 467
16.2 CBSE processes 473
16.3 Component composition 480

Chapter 17 Distributed software engineering 490

17.1 Distributed systems 492
17.2 Client–server computing 499

17.3 Architectural patterns for distributed systems 501
17.4 Software as a service 512

Chapter 18 Service-oriented software engineering 520

18.1 Service-oriented architecture 524
18.2 RESTful services 529
18.3 Service engineering 533
18.4 Service composition 541

Chapter 19 Systems engineering 551

19.1 Sociotechnical systems 556
19.2 Conceptual design 563
19.3 System procurement 566
19.4 System development 570
19.5 System operation and evolution 574

Chapter 20 Systems of systems 580

20.1 System complexity 584
20.2 Systems of systems classification 587
20.3 Reductionism and complex systems 590
20.4 Systems of systems engineering 593
20.5 Systems of systems architecture 599

Chapter 21 Real-time software engineering 610

21.1 Embedded system design 613
21.2 Architectural patterns for real-time software 620
21.3 Timing analysis 626
21.4 Real-time operating systems 631

Part 4 Software Management 639

Chapter 22 Project management 641

22.1 Risk management 644
22.2 Managing people 652
22.3 Teamwork 656

Chapter 23 Project planning 667

23.1 Software pricing 670
23.2 Plan-driven development 672
23.3 Project scheduling 675
23.4 Agile planning 680
23.5 Estimation techniques 682
23.6 COCOMO cost modeling 686

Chapter 24 Quality management 700

24.1 Software quality 703
24.2 Software standards 706
24.3 Reviews and inspections 710
24.4 Quality management and agile development 714
24.5 Software measurement 716

Chapter 25 Configuration management 730

25.1 Version management 735
25.2 System building 740
25.3 Change management 745
25.4 Release management 750

Glossary 757
Subject index 777
Author index 803

PART 1

Introduction to Software Engineering

My aim in this part of the book is to provide a general introduction to software engineering. The chapters in this part have been designed to support a one-semester first course in software engineering. I introduce important concepts such as software processes and agile methods, and describe essential software development activities, from requirements specification through to system evolution.

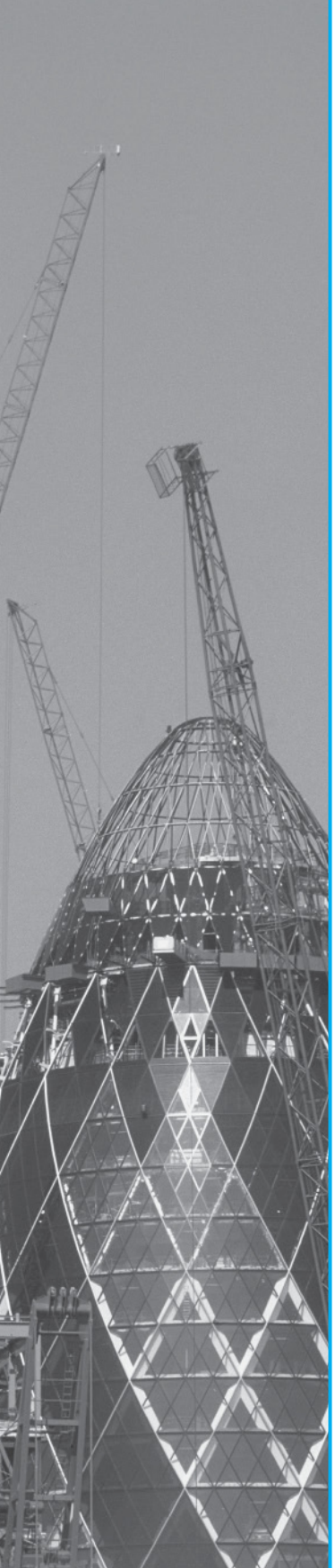
Chapter 1 is a general introduction that introduces professional software engineering and defines some software engineering concepts. I have also included a brief discussion of ethical issues in software engineering. It is important for software engineers to think about the wider implications of their work. This chapter also introduces four case studies that I use in the book. These are an information system for managing records of patients undergoing treatment for mental health problems (Mentcare), a control system for a portable insulin pump, an embedded system for a wilderness weather station and a digital learning environment (iLearn).

Chapters 2 and 3 cover software engineering processes and agile development. In Chapter 2, I introduce software process models, such as the waterfall model, and I discuss the basic activities that are part of these processes. Chapter 3 supplements this with a discussion of agile development methods for software engineering. This chapter had been

extensively changed from previous editions with a focus on agile development using Scrum and a discussion of agile practices such as stories for requirements definition and test-driven development.

The remaining chapters in this part are extended descriptions of the software process activities that are introduced in Chapter 2. Chapter 4 covers the critically important topic of requirements engineering, where the requirements for what a system should do are defined. Chapter 5 explains system modeling using the UML, where I focus on the use of use case diagrams, class diagrams, sequence diagrams and state diagrams for modeling a software system. In Chapter 6, I discuss the importance of software architecture and the use of architectural patterns in software design.

Chapter 7 introduces object oriented design and the use of design patterns. I also introduce important implementation issues here—reuse, configuration management and host-target development and discuss open source development. Chapter 8 focuses on software testing from unit testing during system development to the testing of software releases. I also discuss the use of test-driven development—an approach pioneered in agile methods but which has wide applicability. Finally, Chapter 9 presents an overview of software evolution issues. I cover evolution processes, software maintenance and legacy system management.



1 Introduction

Objectives

The objectives of this chapter are to introduce software engineering and to provide a framework for understanding the rest of the book. When you have read this chapter, you will:

- understand what software engineering is and why it is important;
- understand that the development of different types of software system may require different software engineering techniques;
- understand ethical and professional issues that are important for software engineers;
- have been introduced to four systems, of different types, which are used as examples throughout the book.

Contents

1.1 Professional software development
1.2 Software engineering ethics
1.3 Case studies

Software engineering is essential for the functioning of government, society, and national and international businesses and institutions. We can't run the modern world without software. National infrastructures and utilities are controlled by computer-based systems, and most electrical products include a computer and controlling software. Industrial manufacturing and distribution is completely computerized, as is the financial system. Entertainment, including the music industry, computer games, and film and television, is software-intensive. More than 75% of the world's population have a software-controlled mobile phone, and, by 2016, almost all of these will be Internet-enabled.

Software systems are abstract and intangible. They are not constrained by the properties of materials, nor are they governed by physical laws or by manufacturing processes. This simplifies software engineering, as there are no natural limits to the potential of software. However, because of the lack of physical constraints, software systems can quickly become extremely complex, difficult to understand, and expensive to change.

There are many different types of software system, ranging from simple embedded systems to complex, worldwide information systems. There are no universal notations, methods, or techniques for software engineering because different types of software require different approaches. Developing an organizational information system is completely different from developing a controller for a scientific instrument. Neither of these systems has much in common with a graphics-intensive computer game. All of these applications need software engineering; they do not all need the same software engineering methods and techniques.

There are still many reports of software projects going wrong and of "software failures." Software engineering is criticized as inadequate for modern software development. However, in my opinion, many of these so-called software failures are a consequence of two factors:

1. *Increasing system complexity* As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly; larger, even more complex systems are required; and systems have to have new capabilities that were previously thought to be impossible. New software engineering techniques have to be developed to meet new the challenges of delivering more complex software.
2. *Failure to use software engineering methods* It is fairly easy to write computer programs without using software engineering methods and techniques. Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be. We need better software engineering education and training to address this problem.

Software engineers can be rightly proud of their achievements. Of course, we still have problems developing complex software, but without software engineering we would not have explored space and we would not have the Internet or modern telecommunications. All forms of travel would be more dangerous and expensive. Challenges for humanity in the 21st century are climate change, fewer natural

History of software engineering

The notion of software engineering was first proposed in 1968 at a conference held to discuss what was then called the software crisis (Naur and Randell 1969). It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late.

Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding, and object-oriented development. Tools and standard notations were developed which are the basis of today's software engineering.

http://software-engineering-book.com/web/history/

resources, changing demographics, and an expanding world population. We will rely on software engineering to develop the systems that we need to cope with these issues.

1.1 Professional software development

Lots of people write programs. People in business write spreadsheet programs to simplify their jobs; scientists and engineers write programs to process their experimental data; hobbyists write programs for their own interest and enjoyment. However, most software development is a professional activity in which software is developed for business purposes, for inclusion in other devices, or as software products such as information systems and computer-aided design systems. The key distinctions are that professional software is intended for use by someone apart from its developer and that teams rather than individuals usually develop the software. It is maintained and changed throughout its life.

Software engineering is intended to support professional software development rather than individual programming. It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development. To help you to get a broad view of software engineering, I have summarized frequently asked questions about the subject in Figure 1.1.

Many people think that software is simply another word for computer programs. However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful. A professionally developed software system is often more than a single program. A system may consist of several separate programs and configuration files that are used to set up these programs. It may include system documentation, which describes the structure of the system, user documentation, which explains how to use the system, and websites for users to download recent product information.

This is one of the important differences between professional and amateur software development. If you are writing a program for yourself, no one else will use it

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. There are no methods and techniques that are good for everything.
What differences has the Internet made to software engineering?	Not only has the Internet led to the development of massive, highly distributed, service-based systems, it has also supported the creation of an "app" industry for mobile devices which has changed the economics of software.

Figure 1.1 Frequently asked questions about software engineering

and you don't have to worry about writing program guides, documenting the program design, and so on. However, if you are writing software that other people will use and other engineers will change, then you usually have to provide additional information as well as the code of the program.

Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:

1. *Generic products* These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include apps for mobile devices, software for PCs such as databases, word processors, drawing packages, and project management tools. This kind of software also includes "vertical"

applications designed for a specific market such as library information systems, accounting systems, or systems for maintaining dental records.

2. *Customized (or bespoke) software* These are systems that are commissioned by and developed for a particular customer. A software contractor designs and implements the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

The critical distinction between these types of software is that, in generic products, the organization that develops the software controls the software specification. This means that if they run into development problems, they can rethink what is to be developed. For custom products, the specification is developed and controlled by the organization that is buying the software. The software developers must work to that specification.

However, the distinction between these system product types is becoming increasingly blurred. More and more systems are now being built with a generic product as a base, which is then adapted to suit the requirements of a customer. Enterprise Resource Planning (ERP) systems, such as systems from SAP and Oracle, are the best examples of this approach. Here, a large and complex system is adapted for a company by incorporating information about business rules and processes, reports required, and so on.

When we talk about the quality of professional software, we have to consider that the software is used and changed by people apart from its developers. Quality is therefore not just concerned with what the software does. Rather, it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation. This is reflected in the software's quality or non-functional attributes. Examples of these attributes are the software's response time to a user query and the understandability of the program code.

The specific set of attributes that you might expect from a software system obviously depends on its application. Therefore, an aircraft control system must be safe, an interactive game must be responsive, a telephone switching system must be reliable, and so on. These can be generalized into the set of attributes shown in Figure 1.2, which I think are the essential characteristics of a professional software system.

1.1.1 Software engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use. In this definition, there are two key phrases:

1. *Engineering discipline* Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively

Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

Figure 1.2 Essential attributes of good software

and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work within organizational and financial constraints, and they must look for solutions within these constraints.

2. *All aspects of software production* Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

Engineering is about getting results of the required quality within schedule and budget. This often involves making compromises—engineers cannot be perfectionists. People writing programs for themselves, however, can spend as much time as they wish on the program development.

In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software. However, engineering is all about selecting the most appropriate method for a set of circumstances, so a more creative, less formal approach to development may be the right one for some kinds of software. A more flexible software process that accommodates rapid change is particularly appropriate for the development of interactive web-based systems and mobile apps, which require a blend of software and graphical design skills.

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as

a personal programming project. Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance.

The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product. Four fundamental activities are common to all software processes.

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

Different types of systems need different development processes, as I explain in Chapter 2. For example, real-time software in an aircraft has to be completely specified before development begins. In e-commerce systems, the specification and the program are usually developed together. Consequently, these generic activities may be organized in different ways and described at different levels of detail, depending on the type of software being developed.

Software engineering is related to both computer science and systems engineering.

1. Computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Computer science theory, however, is often most applicable to relatively small programs. Elegant theories of computer science are rarely relevant to large, complex problems that require a software solution.
2. System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design, and system deployment, as well as software engineering. System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system.

As I discuss in the next section, there are many different types of software. There are no universal software engineering methods or techniques that may be used. However, there are four related issues that affect many different types of software:

1. *Heterogeneity* Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. As well as running on general-purpose computers, software may also have to execute on mobile phones and tablets. You often have to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope with this heterogeneity.
2. *Business and social change* Businesses and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software. Many traditional software engineering techniques are time consuming, and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
3. *Security and trust* As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot successfully attack our software and that information security is maintained.
4. *Scale* Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

To address these challenges, we will need new tools and techniques as well as innovative ways of combining and using existing software engineering methods.

1.1.2 Software engineering diversity

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers. The specific methods, tools, and techniques used depend on the organization developing the software, the type of software, and the people involved in the development process. There are no universal software engineering methods that are suitable for all systems and all companies. Rather, a diverse set of software engineering methods and tools has evolved over the past 50 years. However, the SEMAT initiative (Jacobson et al. 2013) proposes that there can be a fundamental meta-process that can be instantiated to create different kinds of process. This is at an early stage of development and may be a basis for improving our current software engineering methods.

Perhaps the most significant factor in determining which software engineering methods and techniques are most important is the type of application being developed. There are many different types of application, including:

1. *Stand-alone applications* These are application systems that run on a personal computer or apps that run on a mobile device. They include all necessary functionality and may not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, travel apps, productivity apps, and so on.

2. *Interactive transaction-based applications* These are applications that execute on a remote computer and that are accessed by users from their own computers, phones, or tablets. Obviously, these include web applications such as e-commerce applications where you interact with a remote system to buy goods and services. This class of application also includes business systems, where a business provides access to its systems through a web browser or special-purpose client program and cloud-based services, such as mail and photo sharing. Interactive applications often incorporate a large data store that is accessed and updated in each transaction.

3. *Embedded control systems* These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system. Examples of embedded systems include the software in a mobile (cell) phone, software that controls antilock braking in a car, and software in a microwave oven to control the cooking process.

4. *Batch processing systems* These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems are periodic billing systems, such as phone billing systems, and salary payment systems.

5. *Entertainment systems* These are systems for personal use that are intended to entertain the user. Most of these systems are games of one kind or another, which may run on special-purpose console hardware. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.

6. *Systems for modeling and simulation* These are systems that are developed by scientists and engineers to model physical processes or situations, which include many separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.

7. *Data collection and analysis systems* Data collection systems are systems that collect data from their environment and send that data to other systems for processing. The software may have to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location. “Big data” analysis may involve cloud-based systems carrying out statistical analysis and looking for relationships in the collected data.

8. *Systems of systems* These are systems, used in enterprises and other large organizations, that are composed of a number of other software systems. Some of these may be generic software products, such as an ERP system. Other systems in the assembly may be specially written for that environment.

Of course, the boundaries between these system types are blurred. If you develop a game for a phone, you have to take into account the same constraints (power, hardware interaction) as the developers of the phone software. Batch processing systems are often used in conjunction with web-based transaction systems. For example, in a company, travel expense claims may be submitted through a web application but processed in a batch application for monthly payment.

Each type of system requires specialized software engineering techniques because the software has different characteristics. For example, an embedded control system in an automobile is safety-critical and is burned into ROM (read-only memory) when installed in the vehicle. It is therefore very expensive to change. Such a system needs extensive verification and validation so that the chances of having to recall cars after sale to fix software problems are minimized. User interaction is minimal (or perhaps nonexistent), so there is no need to use a development process that relies on user interface prototyping.

For an interactive web-based system or app, iterative development and delivery is the best approach, with the system being composed of reusable components. However, such an approach may be impractical for a system of systems, where detailed specifications of the system interactions have to be specified in advance so that each system can be separately developed.

Nevertheless, there are software engineering fundamentals that apply to all types of software systems:

1. They should be developed using a managed and understood development process. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, the specific process that you should use depends on the type of software that you are developing.
2. Dependability and performance are important for all types of system. Software should behave as expected, without failures, and should be available for use when it is required. It should be safe in its operation and, as far as possible, should be secure against external attack. The system should perform efficiently and should not waste resources.
3. Understanding and managing the software specification and requirements (what the software should do) are important. You have to know what different customers and users of the system expect from it, and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.
4. You should make effective use of existing resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

These fundamental notions of process, dependability, requirements, management, and reuse are important themes of this book. Different methods reflect them in different ways, but they underlie all professional software development.

These fundamentals are independent of the program language used for software development. I don't cover specific programming techniques in this book because these vary dramatically from one type of system to another. For example, a dynamic language, such as Ruby, is the right type of language for interactive system development but is inappropriate for embedded systems engineering.

1.1.3 Internet software engineering

The development of the Internet and the World Wide Web has had a profound effect on all of our lives. Initially, the web was primarily a universally accessible information store, and it had little effect on software systems. These systems ran on local computers and were only accessible from within an organization. Around 2000, the web started to evolve, and more and more functionality was added to browsers. This meant that web-based systems could be developed where, instead of a special-purpose user interface, these systems could be accessed using a web browser. This led to the development of a vast range of new system products that delivered innovative services, accessed over the web. These are often funded by adverts that are displayed on the user's screen and do not involve direct payment from users.

As well as these system products, the development of web browsers that could run small programs and do some local processing led to an evolution in business and organizational software. Instead of writing software and deploying it on users' PCs, the software was deployed on a web server. This made it much cheaper to change and upgrade the software, as there was no need to install the software on every PC. It also reduced costs, as user interface development is particularly expensive. Wherever it has been possible to do so, businesses have moved to web-based interaction with company software systems.

The notion of software as a service (Chapter 17) was proposed early in the 21st century This has now become the standard approach to the delivery of web-based system products such as Google Apps, Microsoft Office 365, and Adobe Creative Suite. More and more software runs on remote "clouds" instead of local servers and is accessed over the Internet. A computing cloud is a huge number of linked computer systems that is shared by many users. Users do not buy software but pay according to how much the software is used or are given free access in return for watching adverts that are displayed on their screen. If you use services such as web-based mail, storage, or video, you are using a cloud-based system.

The advent of the web has led to a dramatic change in the way that business software is organized. Before the web, business applications were mostly monolithic, single programs running on single computers or computer clusters. Communications were local, within an organization. Now, software is highly distributed, sometimes across the world. Business applications are not programmed from scratch but involve extensive reuse of components and programs.

This change in software organization has had a major effect on software engineering for web-based systems. For example:

1. Software reuse has become the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from preexisting software components and systems, often bundled together in a framework.
2. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance. Web-based systems are always developed and delivered incrementally.
3. Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services. I discuss this approach to software engineering in Chapter 18.
4. Interface development technology such as AJAX (Holdener 2008) and HTML5 (Freeman 2011) have emerged that support the creation of rich interfaces within a web browser.

The fundamental ideas of software engineering, discussed in the previous section, apply to web-based software, as they do to other types of software. Web-based systems are getting larger and larger, so software engineering techniques that deal with scale and complexity are relevant for these systems.

1.2 Software engineering ethics

Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area. As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills. You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer.

It goes without saying that you should uphold normal standards of honesty and integrity. You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession. However, there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility. Some of these are:

1. *Confidentiality* You should normally respect the confidentiality of your employers or clients regardless of whether or not a formal confidentiality agreement has been signed.
2. *Competence* You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. *Intellectual property rights* You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC – Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.
8. SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Figure 1.3 The ACM/IEEE Code of Ethics (ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices, short version. http://www.acm.org/about/se-code)

4. *Computer misuse* You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).

Professional societies and institutions have an important role to play in setting ethical standards. Organizations such as the ACM, the IEEE (Institute of Electrical and Electronic Engineers), and the British Computer Society publish a code of professional conduct or code of ethics. Members of these organizations undertake to follow that code when they sign up for membership. These codes of conduct are generally concerned with fundamental ethical behavior.

Professional associations, notably the ACM and the IEEE, have cooperated to produce a joint code of ethics and professional practice. This code exists in both a short form, shown in Figure 1.3, and a longer form (Gotterbarn, Miller, and Rogerson 1999) that adds detail and substance to the shorter version. The rationale behind this code is summarized in the first two paragraphs of the longer form: