

SOMMERVILLE

SOFTWARE ENGINEERING

9





SOFTWARE ENGINEERING

Ninth Edition

Ian Sommerville

Addison-Wesley

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
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
Managing Editor: Jeff Holcomb
Senior Production Project Manager: Marilyn Lloyd
Director of Marketing: Margaret Waples
Marketing Coordinator: Kathryn Ferranti
Senior Manufacturing Buyer: Carol Melville
Text Designer: Susan Raymond
Cover Art Director: Elena Sidorova
Front Cover Photograph: © Jacques Pavlovsky/Sygma/Corbis
Interior Chapter Opener: © graficart.net/Alamy
Full-Service Project Management: Andrea Stefanowicz, GGS Higher Education Resources,
a Division of PreMedia Global, Inc.
Composition and Illustrations: GGS Higher Education Resources, a Division of PreMedia Global, Inc.
Printer/Binder: Edwards Brothers
Cover Printer: Lehigh-Phoenix Color/Hagerstown

Copyright © 2011, 2006, 2005, 2001, 1996 Pearson Education, Inc., publishing as Addison-Wesley. All rights reserved. Manufactured in the United States of America. This publication is protected by copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, 501 Boylston Street, Suite 900, Boston, Massachusetts 02116.

Many of the designations by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Library of Congress Cataloging-in-Publication Data

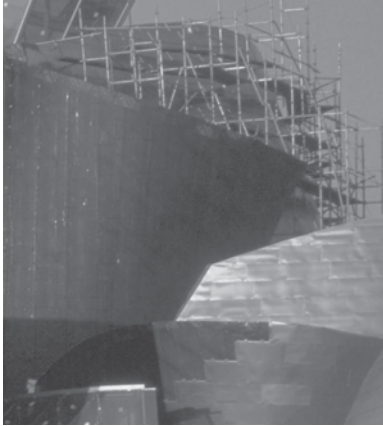
Sommerville, Ian
Software engineering / Ian Sommerville. — 9th ed.
p. cm.
Includes index.
ISBN-13: 978-0-13-703515-1
ISBN-10: 0-13-703515-2
1. Software engineering. I. Title.
QA76.758.S657 2011
005.1—dc22

2009053058

10 9 8 7 6 5 4 3 2 1—EB—14 13 12 11 10



ISBN 10: 0-13-703515-2
ISBN 13: 978-0-13-703515-1



PREFACE

As I was writing the final chapters in this book in the summer of 2009, I realized that software engineering was 40 years old. The name ‘software engineering’ was proposed in 1969 at a NATO conference to discuss software development problems—large software systems were late, did not deliver the functionality needed by their users, cost more than expected, and were unreliable. I did not attend that conference but, a year later, I wrote my first program and started my professional life in software.

Progress in software engineering has been remarkable over my professional lifetime. Our societies could not function without large, professional software systems. For building business systems, there is an alphabet soup of technologies—J2EE, .NET, SaaS, SAP, BPEL4WS, SOAP, CBSE, etc.—that support the development and deployment of large enterprise applications. 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. Humanity is now faced with a new 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 problems and, for sure, software will play a central role in these technologies.

Software engineering is, therefore, a critically important technology for the future of mankind. We must continue to educate software engineers and develop the discipline so that we can create more complex software systems. Of course, there are still problems with software projects. Software is still sometimes late and costs more than expected. 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.

Software engineering is now such a huge area that it is impossible to cover the whole subject in one book. My focus, therefore, is on key topics that are fundamental

to all development processes and topics concerned with the development of reliable, distributed systems. There is an increased emphasis on agile methods and software reuse. I strongly believe that agile methods have their place but so too does ‘traditional’ plan-driven software engineering. We need to combine the best of these approaches to build better software systems.

Books inevitably reflect the opinions and prejudices of their authors. Some readers will inevitably disagree with my opinions and with my choice of material. Such disagreement is a healthy reflection of the diversity of the discipline and is essential for its evolution. Nevertheless, I hope that all software engineers and software engineering students can find something of interest here.

Integration with the Web

There is an incredible amount of information on software engineering available on the Web and some people have questioned if textbooks like this one are still needed. However, the quality of available information is very patchy, information is sometimes presented badly and it can be hard to find the information that you need. Consequently, I believe that textbooks still have an important role to play in learning. They serve as a roadmap to the subject and allow information on method and techniques to be organized and presented in a coherent and readable way. They also provide a starting point for deeper exploration of the research literature and material available on the Web.

I strongly believe that textbooks have a future but only if they are integrated with and add value to material on the Web. This book has therefore been designed as a hybrid print/web text in which core information in the printed edition is linked to supplementary material on the Web. Almost all chapters include specially written ‘web sections’ that add to the information in that chapter. There are also four ‘web chapters’ on topics that I have not covered in the print version of the book.

The website that is associated with the book is:

<http://www.SoftwareEngineering-9.com>

The book’s web has four principal components:

1. *Web sections* These are extra sections that add to the content presented in each chapter. These web sections are linked from breakout boxes in each chapter.
2. *Web chapters* There are four web chapters covering formal methods, interaction design, documentation, and application architectures. I may add other chapters on new topics during the lifetime of the book.
3. *Material for instructors* The material in this section is intended to support people who are teaching software engineering. See the “Support Materials” section in this Preface.
4. *Case studies* These provide additional information about the case studies used in the book (insulin pump, mental health-care system, wilderness weather system)

as well as information about further case studies, such as the failure of the Ariane 5 launcher.

As well as these sections, there are also links to other sites with useful material on software engineering, further reading, blogs, newsletters, etc.

I welcome your constructive comments and suggestions about the book and the website. You can contact me at ian@SoftwareEngineering-9.com. Please include [SE9] in the subject of your message. Otherwise, my spam filters will probably reject your mail and you will not receive a reply. I do not have time to help students with their homework, so please don't ask.

Readership

The book is primarily aimed at university and college students taking introductory and advanced courses in software and systems engineering. Software engineers in the industry may find the book useful as general reading and as a means of updating their knowledge on topics such as software reuse, architectural design, dependability and security, and process improvement. I assume that readers have completed an introductory programming course and are familiar with programming terminology.

Changes from previous editions

This edition has retained the fundamental material on software engineering that was covered in previous editions but I have revised and updated all chapters and have included new material on many different topics. The most important changes are:

1. The move from a print-only book to a hybrid print/web book with the web material tightly integrated with the sections in the book. This has allowed me to reduce the number of chapters in the book and to focus on core material in each chapter.
2. Complete restructuring to make it easier to use the book in teaching software engineering. The book now has four rather than eight parts and each part may be used on its own or in combination with other parts as the basis of a software engineering course. The four parts are an introduction to software engineering, dependability and security, advanced software engineering, and software engineering management.
3. Several topics from previous editions are presented more concisely in a single chapter, with extra material moved onto the Web.
4. Additional web chapters, based on chapters from previous editions that I have not included here, are available on the Web.

5. I have updated and revised the content in all chapters. I estimate that between 30% and 40% of the text has been completely rewritten.
6. I have added new chapters on agile software development and embedded systems.
7. As well as these new chapters, there is new material on model-driven engineering, open source development, test-driven development, Reason's Swiss Cheese model, dependable systems architectures, static analysis and model checking, COTS reuse, software as a service, and agile planning.
8. A new case study on a patient record system for patients who are undergoing treatment for mental health problems has been used in several chapters.

Using the book for teaching

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

1. *General introductory courses in software engineering* The first part of the book has been designed explicitly to support a one-semester course in introductory software engineering.
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 engineering using the chapters in Part 2 plus chapters on quality management and configuration management.
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 16, 17, 18, and 19.

More information about using the book for teaching, including a comparison with previous editions, is available on the book's website.

Support materials

A wide range of support material is available to help people using the book for teaching software engineering courses. This includes:

- PowerPoint presentations for all of the chapters in the book.
- Figures in PowerPoint.

- An instructor's guide that gives advice on how to use the book in different courses and explains the relationship between the chapters in this edition and previous editions.
- Further information on the book's case studies.
- Additional case studies that may be used in software engineering courses.
- Additional PowerPoint presentations on systems engineering.
- Four web chapters covering formal methods, interaction design, application architectures, and documentation.

All of this material is available free to readers of the book from the book's website or from the Pearson support site below. Additional material for instructors is available on a restricted basis to accredited instructors only:

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

All support material, including restricted material, is available from:

<http://www.pearsonhighered.com/sommerville/>

Instructors using the book for teaching may obtain a password to access restricted material by registering at the Pearson website, by contacting their local Pearson representative, or by requesting a password by e-mail from computing@aw.com. Passwords are not available from the author.

Acknowledgments

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 help and support while the book was being written. A big thank-you especially to my daughter, Jane, who discovered a talent for proofreading and editing. She was tremendously helpful in reading the entire book and did a great job spotting and fixing a large number of typos and grammatical errors.

Ian Sommerville
October 2009

Contents at a glance

	Preface	iii
Part 1	Introduction to Software Engineering	1
	Chapter 1 Introduction	3
	Chapter 2 Software processes	27
	Chapter 3 Agile software development	56
	Chapter 4 Requirements engineering	82
	Chapter 5 System modeling	118
	Chapter 6 Architectural design	147
	Chapter 7 Design and implementation	176
	Chapter 8 Software testing	205
	Chapter 9 Software evolution	234
Part 2	Dependability and Security	261
	Chapter 10 Sociotechnical systems	263
	Chapter 11 Dependability and security	289
	Chapter 12 Dependability and security specification	309
	Chapter 13 Dependability engineering	341
	Chapter 14 Security engineering	366
	Chapter 15 Dependability and security assurance	393
Part 3	Advanced Software Engineering	423
	Chapter 16 Software reuse	425
	Chapter 17 Component-based software engineering	452
	Chapter 18 Distributed software engineering	479
	Chapter 19 Service-oriented architecture	508
	Chapter 20 Embedded software	537
	Chapter 21 Aspect-oriented software engineering	565
Part 4	Software Management	591
	Chapter 22 Project management	593
	Chapter 23 Project planning	618
	Chapter 24 Quality management	651
	Chapter 25 Configuration management	681
	Chapter 26 Process improvement	705
	Glossary	733
	Subject Index	749
	Author Index	767



CONTENTS

Preface

iii

Part 1 Introduction to Software Engineering **1**

Chapter 1 Introduction **3**

- 1.1 Professional software development 5
- 1.2 Software engineering ethics 14
- 1.3 Case studies 17

Chapter 2 Software processes **27**

- 2.1 Software process models 29
- 2.2 Process activities 36
- 2.3 Coping with change 43
- 2.4 The rational unified process 50

Chapter 3 Agile software development **56**

- 3.1 Agile methods 58
- 3.2 Plan-driven and agile development 62

3.3	Extreme programming	64
3.4	Agile project management	72
3.5	Scaling agile methods	74
Chapter 4	Requirements engineering	82
4.1	Functional and non-functional requirements	84
4.2	The software requirements document	91
4.3	Requirements specification	94
4.4	Requirements engineering processes	99
4.5	Requirements elicitation and analysis	100
4.6	Requirements validation	110
4.7	Requirements management	111
Chapter 5	System modeling	118
5.1	Context models	121
5.2	Interaction models	124
5.3	Structural models	129
5.4	Behavioral models	133
5.5	Model-driven engineering	138
Chapter 6	Architectural design	147
6.1	Architectural design decisions	151
6.2	Architectural views	153
6.3	Architectural patterns	155
6.4	Application architectures	164
Chapter 7	Design and implementation	176
7.1	Object-oriented design using the UML	178
7.2	Design patterns	189

7.3	Implementation issues	193
7.4	Open source development	198
Chapter 8	Software testing	205
8.1	Development testing	210
8.2	Test-driven development	221
8.3	Release testing	224
8.4	User testing	228
Chapter 9	Software evolution	234
9.1	Evolution processes	237
9.2	Program evolution dynamics	240
9.3	Software maintenance	242
9.4	Legacy system management	252
Part 2	Dependability and Security	261
Chapter 10	Sociotechnical systems	263
10.1	Complex systems	266
10.2	Systems engineering	273
10.3	System procurement	275
10.4	System development	278
10.5	System operation	281
Chapter 11	Dependability and security	289
11.1	Dependability properties	291
11.2	Availability and reliability	295
11.3	Safety	299
11.4	Security	302

Chapter 12	Dependability and security specification	309
12.1	Risk-driven requirements specification	311
12.2	Safety specification	313
12.3	Reliability specification	320
12.4	Security specification	329
12.5	Formal specification	333
Chapter 13	Dependability engineering	341
13.1	Redundancy and diversity	343
13.2	Dependable processes	345
13.3	Dependable system architectures	348
13.4	Dependable programming	355
Chapter 14	Security engineering	366
14.1	Security risk management	369
14.2	Design for security	375
14.3	System survivability	386
Chapter 15	Dependability and security assurance	393
15.1	Static analysis	395
15.2	Reliability testing	401
15.3	Security testing	404
15.4	Process assurance	406
15.5	Safety and dependability cases	410
Part 3	Advanced Software Engineering	423
Chapter 16	Software reuse	425
16.1	The reuse landscape	428
16.2	Application frameworks	431

16.3	Software product lines	434
16.4	COTS product reuse	440
Chapter 17	Component-based software engineering	452
17.1	Components and component models	455
17.2	CBSE processes	461
17.3	Component composition	468
Chapter 18	Distributed software engineering	479
18.1	Distributed systems issues	481
18.2	Client–server computing	488
18.3	Architectural patterns for distributed systems	490
18.4	Software as a service	501
Chapter 19	Service-oriented architecture	508
19.1	Services as reusable components	514
19.2	Service engineering	518
19.3	Software development with services	527
Chapter 20	Embedded software	537
20.1	Embedded systems design	540
20.2	Architectural patterns	547
20.3	Timing analysis	554
20.4	Real-time operating systems	558
Chapter 21	Aspect-oriented software engineering	565
21.1	The separation of concerns	567
21.2	Aspects, join points and pointcuts	571
21.3	Software engineering with aspects	576

Part 4	Software Management	591
Chapter 22	Project management	593
	22.1 Risk management	595
	22.2 Managing people	602
	22.3 Teamwork	607
Chapter 23	Project planning	618
	23.1 Software pricing	621
	23.2 Plan-driven development	623
	23.3 Project scheduling	626
	23.4 Agile planning	631
	23.5 Estimation techniques	633
Chapter 24	Quality management	651
	24.1 Software quality	655
	24.2 Software standards	657
	24.3 Reviews and inspections	663
	24.4 Software measurement and metrics	668
Chapter 25	Configuration management	681
	25.1 Change management	685
	25.2 Version management	690
	25.3 System building	693
	25.4 Release management	699
Chapter 26	Process improvement	705
	26.1 The process improvement process	708
	26.2 Process measurement	711

26.3 Process analysis	715
26.4 Process change	718
26.5 The CMMI process improvement framework	721
Glossary	733
Subject Index	749
Author Index	767

This page intentionally left blank



PART **1** Introduction to Software Engineering

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

Chapter 1 is a general introduction that introduces professional software engineering and defines some software engineering concepts. I have also written a brief discussion of ethical issues in software engineering. I think that it is important for software engineers to think about the wider implications of their work. This chapter also introduces three case studies that I use in the book, namely a system for managing records of patients undergoing treatment for mental health problems, a control system for a portable insulin pump and a wilderness weather system.

Chapters 2 and 3 cover software engineering processes and agile development. In Chapter 2, I introduce commonly used generic 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. I mostly use Extreme Programming as an example of an agile method but also briefly introduce Scrum in this chapter.

The remainder of the chapters in this part are extended descriptions of the software process activities that will be 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 introduces 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. Chapter 6 introduces architectural design and I discuss the importance of 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 systems may require different software engineering techniques;
- understand some ethical and professional issues that are important for software engineers;
- have been introduced to three systems, of different types, that will be used as examples throughout the book.

Contents

- 1.1** Professional software development
- 1.2** Software engineering ethics
- 1.3** Case studies

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. Therefore, software engineering is essential for the functioning of national and international societies.

Software systems are abstract and intangible. They are not constrained by the properties of materials, 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 systems, from simple embedded systems to complex, worldwide information systems. It is pointless to look for 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 techniques.

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

1. *Increasing demands* 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; systems have to have new capabilities that were previously thought to be impossible. Existing software engineering methods cannot cope and new software engineering techniques have to be developed to meet new these new demands.
2. *Low expectations* It is relatively 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, would not have the Internet or modern telecommunications. All forms of travel would be more dangerous and expensive. Software engineering has contributed a great deal and I am convinced that its contributions in the 21st century will be even greater.



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 and are now extensively used.

<http://www.SoftwareEngineering-9.com/Web/History/>

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, and hobbyists write programs for their own interest and enjoyment. However, the vast majority of software development is a professional activity where software is developed for specific business purposes, for inclusion in other devices, or as software products such as information systems, CAD systems, etc. Professional software, intended for use by someone apart from its developer, is usually developed by teams rather than individuals. 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 what software engineering is about, I have summarized some frequently asked questions 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 and configuration data that is required to make these programs operate correctly. A professionally developed software system is often more than a single program. The system usually consists of a number of 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 and you don't have to worry about writing program guides, documenting the program design, etc. 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.

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.
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. You can't, therefore, say that one method is better than another.
What differences has the Web made to software engineering?	The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Figure 1.1 Frequently asked questions about software

Software engineers are concerned with developing software products (i.e., software which can be sold to a customer). There are two kinds of software products:

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 software for PCs such as databases, word processors, drawing packages, and project-management tools. It also includes so-called vertical applications designed for some specific purpose such as library information systems, accounting systems, or systems for maintaining dental records.

2. *Customized (or bespoke) products* These are systems that are commissioned by a particular customer. A software contractor develops 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.

An important difference between these types of software is that, in generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually 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 the SAP system, 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 take into account 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 so-called quality or non-functional software 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, a banking system must be secure, 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 believe 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 characteristics	Description
Maintainability	Software should be written in such a way so 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.
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. Malicious users should not be able to 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, memory utilization, etc.
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.

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 to organizational and financial constraints so they 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 production.

Engineering is about getting results of the required quality within the 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 effective in some circumstances. Less formal development is particularly appropriate for the development of web-based systems, which requires 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 software systems rather than just write the programs as if it was a personal programming project. For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

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. There are four fundamental activities that are common to all software processes. These activities are:

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. 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. I describe software processes in more detail in Chapter 2.

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 cannot always be applied 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. They are less concerned with the engineering of the system components (hardware, software, etc.).

As I discuss in the next section, there are many different types of software. There is no universal software engineering method or technique that is applicable for all of these. However, there are three general 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. 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* Business 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 attack our software and that information security is maintained.

Of course, these are not independent issues. For example, it may be necessary to make rapid changes to a legacy system to provide it with a web service interface. 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. How this systematic approach is actually implemented varies dramatically depending 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 and techniques 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.

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

1. *Stand-alone applications* These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to

be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.

2. *Interactive transaction-based applications* These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. Obviously, these include web applications such as e-commerce applications where you can 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 anti-lock 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 include periodic billing systems, such as phone billing systems, and salary payment systems.
5. *Entertainment systems* These are systems that are primarily for personal use and which are intended to entertain the user. Most of these systems are games of one kind or another. 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 systems* These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing. The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location.
8. *Systems of systems* These are systems that are composed of a number of other software systems. Some of these may be generic software products, such as a spreadsheet program. 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 mobile (cell) 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 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.

You use different software engineering techniques for each type of system because the software has quite different characteristics. For example, an embedded control system in an automobile is safety-critical and is burned into ROM when installed in the vehicle. It is therefore very expensive to change. Such a system needs very 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 a web-based system, an approach based on iterative development and delivery may be appropriate, 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 system:

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, different processes are used for different types of software.
2. Dependability and performance are important for all types of systems. 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 as effective use as possible 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.

You should notice that these fundamentals do not cover implementation and programming. I don't cover specific programming techniques in this book because these vary dramatically from one type of system to another. For example, a scripting language such as Ruby is used for web-based system programming but would be completely inappropriate for embedded systems engineering.

1.1.3 Software engineering and the Web

The development of 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. Consequently, wherever it has been possible to do so, many businesses have moved to web-based interaction with company software systems.

The next stage in the development of web-based systems was the notion of web services. Web services are software components that deliver specific, useful functionality and which are accessed over the Web. Applications are constructed by integrating these web services, which may be provided by different companies. In principle, this linking can be dynamic so that an application may use different web services each time that it is executed. I cover this approach to software development in Chapter 19.

In the last few years, the notion of 'software as a service' has been developed. It has been proposed that software will not normally run on local computers but will run on 'computing clouds' that are accessed over the Internet. If you use a service such as web-based mail, you are using a cloud-based system. 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.

The advent of the web, therefore, has led to a significant 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 radical change in software organization has, obviously, led to changes in the ways that web-based systems are engineered. 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 pre-existing software components and systems.