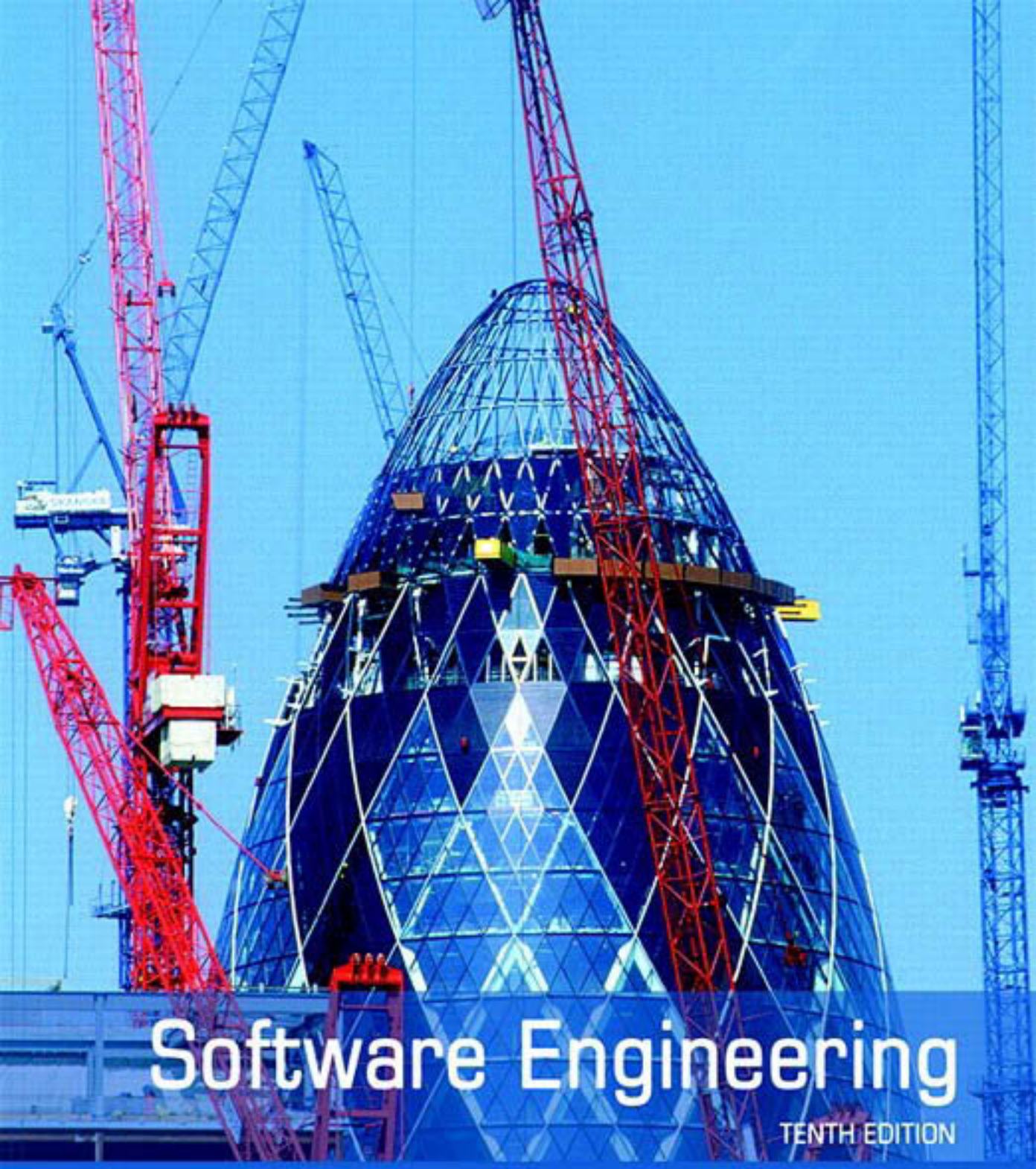


# Software Engineering

TENTH EDITION

Ian Sommerville

# 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

**Vice President and Editorial Director, ECS:** Marcia J. Horton
**Acquisitions Editor:** Matt Goldstein
**Editorial Assistant:** Kelsey Loanes
**Product Marketing Manager:** Bram Van Kempen
**Marketing Assistant:** Jon Bryant
**Senior Managing Editor:** Scott Disanno
**Production Project Manager:** Rose Kernan
**Program Manager:** Carole Snyder
**Global HE Director of Vendor Sourcing and Procurement:** Diane Hynes
**Director of Operations**: Nick Sklitsis
**Operations Specialist:** Maura Zaldivar-Garcia
**Cover Designer:** Black Horse Designs
**Cover Image:** Construction of the Gherkin, London, UK./Corbis
**Manager, Rights and Permissions:** Rachel Youdelman
**Associate Project Manager, Rights and Permissions:** Timothy Nicholls
**Full-Service Project Management:** Rashmi Tickyani, iEnergizer Aptara®, Ltd.
**Composition:** iEnergizer Aptara®, Ltd.
**Printer/Binder:** Edwards Brothers Malloy
**Cover Printer:** Phoenix Color/Hagerstown
**Typeface:** 10/12.5 Times LT Std

---



Pearson Education Ltd., *London*
Pearson Education Singapore, Pte. Ltd
Pearson Education Canada, Inc.
Pearson Education—Japan
Pearson Education Australia PTY, Limited
Pearson Education North Asia, Ltd., *Hong Kong*
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education Malaysia, Pte.Ltd.
Pearson Education, Inc., *Hoboken, New Jersey*

**Library of Congress Cataloging-in-Publication Data on File**

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-394303-8
ISBN 13: 978-0-13-394303-0

# 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:

**http://www.pearsonhighered.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:

- PowerPoint presentations for all of the chapters in the book.
- 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.
- Additional PowerPoint presentations covering a range of systems engineering topics.

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

# CONTENTS

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.

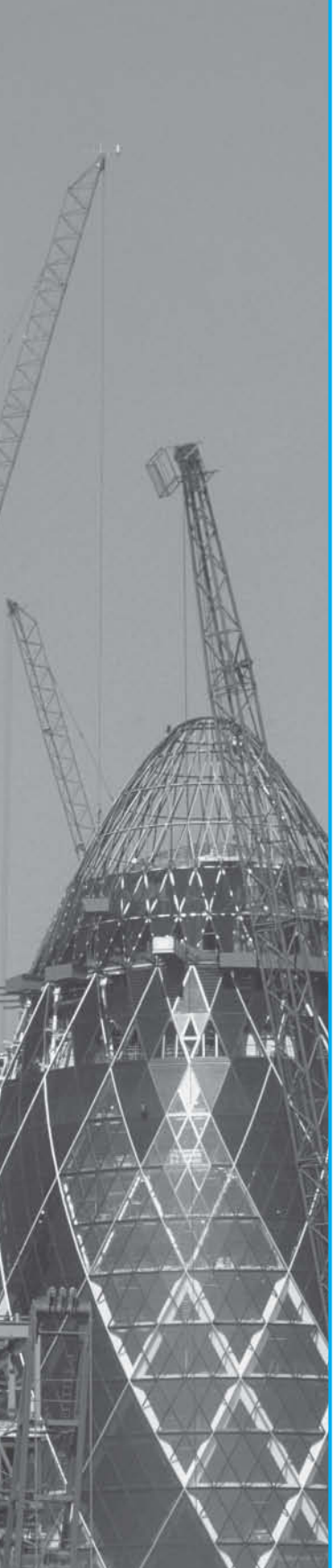
Chapter 1 is a general introduction that introduces professional software engineering and defines some software engineering concepts. I have also included a brief discussion of ethical issues in software engineering. It is important for software engineers to think about the wider implications of their 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

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: