**1**

**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

**4** Chapter 1 \_ Introduction

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.

1.1 \_ Professional software development **5**

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.

**6** Chapter 1 \_ Introduction

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.

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

Figure 1.1 Frequently

asked questions about

software

1.1 \_ Professional software development **7**

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

**8** Chapter 1 \_ Introduction

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.

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

1.1 \_ Professional software development **9**

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.).

**10** Chapter 1 \_ Introduction

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

1.1 \_ Professional software development **11**

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,

**12** Chapter 1 \_ Introduction

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 \_ Professional software development **13**

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.

**14** Chapter 1 \_ Introduction

2. It is now generally recognized that it is impractical to specify all the requirements

for such systems in advance. Web-based systems should be developed

and delivered incrementally.

3. User interfaces are constrained by the capabilities of web browsers. Although

technologies such as AJAX (Holdener, 2008) mean that rich interfaces can be

created within a web browser, these technologies are still difficult to use. Web

forms with local scripting are more commonly used. Application interfaces on

web-based systems are often poorer than the specially designed user interfaces

on PC system products.

The fundamental ideas of software engineering, discussed in the previous section,

apply to web-based software in the same way that they apply to other types of software

system. Experience gained with large system development in the 20th century

is still relevant to web-based software.

**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 irrespective 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.

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, say) to extremely serious (dissemination of viruses or

other malware).

1.2 \_ Software engineering ethics **15**

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 et al., 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:

*Computers have a central and growing role in commerce, industry, government,*

*medicine, education, entertainment and society at large. Software engineers are*

*those who contribute by direct participation or by teaching, to the analysis, specification,*

*design, development, certification, maintenance and testing of software*

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 (© IEEE/ACM

1999)

**16** Chapter 1 \_ Introduction

*systems. Because of their roles in developing software systems, software engineers*

*have significant opportunities to do good or cause harm, to enable others to*

*do good or cause harm, or to influence others to do good or cause harm. To*

*ensure, as much as possible, that their efforts will be used for good, software engineers*

*must commit themselves to making software engineering a beneficial and*

*respected profession. In accordance with that commitment, software engineers*

*shall adhere to the following Code of Ethics and Professional Practice.*

*The Code contains eight Principles related to the behaviour of and decisions*

*made by professional software engineers, including practitioners, educators,*

*managers, supervisors and policy makers, as well as trainees and students of*

*the profession. The Principles identify the ethically responsible relationships*

*in which individuals, groups, and organizations participate and the primary*

*obligations within these relationships. The Clauses of each Principle are illustrations*

*of some of the obligations included in these relationships. These obligations*

*are founded in the software engineer’s humanity, in special care owed*

*to people affected by the work of software engineers, and the unique elements*

*of the practice of software engineering. The Code prescribes these as obligations*

*of anyone claiming to be or aspiring to be a software engineer.*

In any situation where different people have different views and objectives you

are likely to be faced with ethical dilemmas. For example, if you disagree, in principle,

with the policies of more senior management in the company, how should you

react? Clearly, this depends on the particular individuals and the nature of the disagreement.

Is it best to argue a case for your position from within the organization or

to resign in principle? If you feel that there are problems with a software project,

when do you reveal these to management? If you discuss these while they are just a

suspicion, you may be overreacting to a situation; if you leave it too late, it may be

impossible to resolve the difficulties.

Such ethical dilemmas face all of us in our professional lives and, fortunately, in

most cases they are either relatively minor or can be resolved without too much difficulty.

Where they cannot be resolved, the engineer is faced with, perhaps, another

problem. The principled action may be to resign from their job but this may well

affect others such as their partner or their children.

A particularly difficult situation for professional engineers arises when their

employer acts in an unethical way. Say a company is responsible for developing a

safety-critical system and, because of time pressure, falsifies the safety validation

records. Is the engineer’s responsibility to maintain confidentiality or to alert the

customer or publicize, in some way, that the delivered system may be unsafe?

The problem here is that there are no absolutes when it comes to safety. Although

the system may not have been validated according to predefined criteria, these criteria

may be too strict. The system may actually operate safely throughout its lifetime.

It is also the case that, even when properly validated, the system may fail and cause

an accident. Early disclosure of problems may result in damage to the employer and

other employees; failure to disclose problems may result in damage to others.

1.3 \_ Case studies **17**

You must make up your own mind in these matters. The appropriate ethical position

here depends entirely on the views of the individuals who are involved. In this

case, the potential for damage, the extent of the damage, and the people affected by

the damage should influence the decision. If the situation is very dangerous, it may

be justified to publicize it using the national press (say). However, you should

always try to resolve the situation while respecting the rights of your employer.

Another ethical issue is participation in the development of military and nuclear

systems. Some people feel strongly about these issues and do not wish to participate in

any systems development associated with military systems. Others will work on military

systems but not on weapons systems. Yet others feel that national security is an

overriding principle and have no ethical objections to working on weapons systems.

In this situation, it is important that both employers and employees should make

their views known to each other in advance. Where an organization is involved in

military or nuclear work, they should be able to specify that employees must be willing

to accept any work assignment. Equally, if an employee is taken on and makes

clear that they do not wish to work on such systems, employers should not put pressure

on them to do so at some later date.

The general area of ethics and professional responsibility is becoming more

important as software-intensive systems pervade every aspect of work and everyday

life. It can be considered from a philosophical standpoint where the basic principles

of ethics are considered and software engineering ethics are discussed with reference

to these basic principles. This is the approach taken by Laudon (1995) and to a lesser

extent by Huff and Martin (1995). Johnson’s text on computer ethics (2001) also

approaches the topic from a philosophical perspective.

However, I find that this philosophical approach is too abstract and difficult to

relate to everyday experience. I prefer the more concrete approach embodied in codes

of conduct and practice. I think that ethics are best discussed in a software engineering

context and not as a subject in their own right. In this book, therefore, I do not

include abstract ethical discussions but, where appropriate, include examples in the

exercises that can be the starting point for a group discussion on ethical issues.

**1.3 Case studies**

To illustrate software engineering concepts, I use examples from three different

types of systems throughout the book. The reason why I have not used a single case

study is that one of the key messages in this book is that software engineering practice

depends on the type of systems being produced. I therefore choose an appropriate

example when discussing concepts such as safety and dependability, system

modeling, reuse, etc.

The three types of systems that I use as case studies are:

1. *An embedded system* This is a system where the software controls a hardware

device and is embedded in that device. Issues in embedded systems typically

**18** Chapter 1 \_ Introduction

include physical size, responsiveness, power management, etc. The example of an

embedded system that I use is a software system to control a medical device.

2. *An information system* This is a system whose primary purpose is to manage

and provide access to a database of information. Issues in information systems

include security, usability, privacy, and maintaining data integrity. The example

of an information system that I use is a medical records system.

3. *A sensor-based data collection system* This is a system whose primary purpose

is to collect data from a set of sensors and process that data in some way. The

key requirements of such systems are reliability, even in hostile environmental

conditions, and maintainability. The example of a data collection system that

I use is a wilderness weather station.

I introduce each of these systems in this chapter, with more information about

each of them available on the Web.

1.3.1 An insulin pump control system

An insulin pump is a medical system that simulates the operation of the pancreas (an

internal organ). The software controlling this system is an embedded system, which

collects information from a sensor and controls a pump that delivers a controlled

dose of insulin to a user.

People who suffer from diabetes use the system. Diabetes is a relatively common

condition where the human pancreas is unable to produce sufficient quantities of a

hormone called insulin. Insulin metabolises glucose (sugar) in the blood. The conventional

treatment of diabetes involves regular injections of genetically engineered

insulin. Diabetics measure their blood sugar levels using an external meter and then

calculate the dose of insulin that they should inject.

The problem with this treatment is that the level of insulin required does not just

depend on the blood glucose level but also on the time of the last insulin injection.

This can lead to very low levels of blood glucose (if there is too much insulin) or very

high levels of blood sugar (if there is too little insulin). Low blood glucose is, in the

short term, a more serious condition as it can result in temporary brain malfunctioning

and, ultimately, unconsciousness and death. In the long term, however, continual high

levels of blood glucose can lead to eye damage, kidney damage, and heart problems.

Current advances in developing miniaturized sensors have meant that it is now possible

to develop automated insulin delivery systems. These systems monitor blood sugar

levels and deliver an appropriate dose of insulin when required. Insulin delivery systems

like this already exist for the treatment of hospital patients. In the future, it may be possible

for many diabetics to have such systems permanently attached to their bodies.

A software-controlled insulin delivery system might work by using a microsensor

embedded in the patient to measure some blood parameter that is proportional

to the sugar level. This is then sent to the pump controller. This controller computes

the sugar level and the amount of insulin that is needed. It then sends signals to a

miniaturized pump to deliver the insulin via a permanently attached needle.

1.3 \_ Case studies **19**

Figure 1.4 shows the hardware components and organization of the insulin

pump. To understand the examples in this book, all you need to know is that the

blood sensor measures the electrical conductivity of the blood under different

conditions and that these values can be related to the blood sugar level. The

insulin pump delivers one unit of insulin in response to a single pulse from a controller.

Therefore, to deliver 10 units of insulin, the controller sends 10 pulses to

the pump. Figure 1.5 is a UML activity model that illustrates how the software

transforms an input blood sugar level to a sequence of commands that drive the

insulin pump.

Clearly, this is a safety-critical system. If the pump fails to operate or does not

operate correctly, then the user’s health may be damaged or they may fall into a

coma because their blood sugar levels are too high or too low. There are, therefore,

two essential high-level requirements that this system must meet:

1. The system shall be available to deliver insulin when required.

2. The system shall perform reliably and deliver the correct amount of insulin to

counteract the current level of blood sugar.

Needle

Assembly

Sensor

Display1 Display2

Alarm

Pump Clock

Controller

Power Supply

Insulin Reservoir

Figure 1.4 Insulin

pump hardware

Blood

Sensor

Insulin

Pump

Blood

Sugar

Analyze Sensor

Reading

Compute

Insulin

Insulin

Dose

Insulin

Log

Log Dose

Compute Pump

Commands

Pump

Data

Control Insulin

Pump

Figure 1.5 Activity

model of the insulin

pump

**20** Chapter 1 \_ Introduction

The system must therefore be designed and implemented to ensure that the system

always meets these requirements. More detailed requirements and discussions

of how to ensure that the system is safe are discussed in later chapters.

1.3.2 A patient information system for mental health care

A patient information system to support mental health care is a medical information

system that maintains information about patients suffering from mental

health problems and the treatments that they have received. Most mental health

patients do not require dedicated hospital treatment but need to attend specialist

clinics regularly where they can meet a doctor who has detailed knowledge of

their problems. To make it easier for patients to attend, these clinics are not just

run in hospitals. They may also be held in local medical practices or community

centers.

The MHC-PMS (Mental Health Care-Patient Management System) is an information

system that is intended for use in clinics. It makes use of a centralized database of

patient information but has also been designed to run on a PC, so that it may be accessed

and used from sites that do not have secure network connectivity. When the local systems

have secure network access, they use patient information in the database but they

can download and use local copies of patient records when they are disconnected. The

system is not a complete medical records system so does not maintain information

about other medical conditions. However, it may interact and exchange data with other

clinical information systems. Figure 1.6 illustrates the organization of the MHC-PMS.

The MHC-PMS has two overall goals:

1. To generate management information that allows health service managers to

assess performance against local and government targets.

2. To provide medical staff with timely information to support the treatment of

patients.

MHC-PMS Server

Patient Database

MHC-PMS

Local

MHC-PMS

Local

MHC-PMS

Local

Figure 1.6 The

organization of

the MHC-PMS

1.3 \_ Case studies **21**

The nature of mental health problems is such that patients are often disorganized

so may miss appointments, deliberately or accidentally lose prescriptions and medication,

forget instructions, and make unreasonable demands on medical staff. They

may drop in on clinics unexpectedly. In a minority of cases, they may be a danger to

themselves or to other people. They may regularly change address or may be homeless

on a long-term or short-term basis. Where patients are dangerous, they may need

to be ‘sectioned’—confined to a secure hospital for treatment and observation.

Users of the system include clinical staff such as doctors, nurses, and health visitors

(nurses who visit people at home to check on their treatment). Nonmedical users

include receptionists who make appointments, medical records staff who maintain

the records system, and administrative staff who generate reports.

The system is used to record information about patients (name, address, age, next

of kin, etc.), consultations (date, doctor seen, subjective impressions of the patient,

etc.), conditions, and treatments. Reports are generated at regular intervals for medical

staff and health authority managers. Typically, reports for medical staff focus on

information about individual patients whereas management reports are anonymized

and are concerned with conditions, costs of treatment, etc.

The key features of the system are:

1. *Individual care management* Clinicians can create records for patients, edit the

information in the system, view patient history, etc. The system supports data

summaries so that doctors who have not previously met a patient can quickly

learn about the key problems and treatments that have been prescribed.

2. *Patient monitoring* The system regularly monitors the records of patients that

are involved in treatment and issues warnings if possible problems are detected.

Therefore, if a patient has not seen a doctor for some time, a warning may be

issued. One of the most important elements of the monitoring system is to keep

track of patients who have been sectioned and to ensure that the legally required

checks are carried out at the right time.

3. *Administrative reporting* The system generates monthly management reports

showing the number of patients treated at each clinic, the number of patients

who have entered and left the care system, number of patients sectioned, the

drugs prescribed and their costs, etc.

Two different laws affect the system. These are laws on data protection that govern

the confidentiality of personal information and mental health laws that govern the compulsory

detention of patients deemed to be a danger to themselves or others. Mental

health is unique in this respect as it is the only medical speciality that can recommend

the detention of patients against their will. This is subject to very strict legislative safeguards.

One of the aims of the MHC-PMS is to ensure that staff always act in accordance

with the law and that their decisions are recorded for judicial review if necessary.

As in all medical systems, privacy is a critical system requirement. It is essential that

patient information is confidential and is never disclosed to anyone apart from authorized

medical staff and the patient themselves. The MHC-PMS is also a safety-critical

**22** Chapter 1 \_ Introduction

system. Some mental illnesses cause patients to become suicidal or a danger to other

people. Wherever possible, the system should warn medical staff about potentially suicidal

or dangerous patients.

The overall design of the system has to take into account privacy and safety

requirements. The system must be available when needed otherwise safety may be

compromised and it may be impossible to prescribe the correct medication to patients.

There is a potential conflict here—privacy is easiest to maintain when there is only a

single copy of the system data. However, to ensure availability in the event of server

failure or when disconnected from a network, multiple copies of the data should be

maintained. I discuss the trade-offs between these requirements in later chapters.

1.3.3 A wilderness weather station

To help monitor climate change and to improve the accuracy of weather forecasts in

remote areas, the government of a country with large areas of wilderness decides to

deploy several hundred weather stations in remote areas. These weather stations collect

data from a set of instruments that measure temperature and pressure, sunshine,

rainfall, wind speed, and wind direction.

Wilderness weather stations are part of a larger system (Figure 1.7), which is a

weather information system that collects data from weather stations and makes it

available to other systems for processing. The systems in Figure 1.7 are:

1. *The weather station system* This is responsible for collecting weather data,

carrying out some initial data processing, and transmitting it to the data management

system.

2. *The data management and archiving system* This system collects the data from

all of the wilderness weather stations, carries out data processing and analysis,

and archives the data in a form that can be retrieved by other systems, such as

weather forecasting systems.

3. *The station maintenance system* This system can communicate by satellite

with all wilderness weather stations to monitor the health of these systems and

provide reports of problems. It can update the embedded software in these

systems. In the event of system problems, this system can also be used to

remotely control a wilderness weather system.

«system»

Data Management

and Archiving

«system»

Station Maintenance

«system»

Weather Station

Figure 1.7 The weather

station’s environment

1.3 \_ Case studies **23**

In Figure 1.7, I have used the UML package symbol to indicate that each system

is a collection of components and have identified the separate systems, using the

UML stereotype «system». The associations between the packages indicate there is

an exchange of information but, at this stage, there is no need to define them in any

more detail.

Each weather station includes a number of instruments that measure weather

parameters such as the wind speed and direction, the ground and air temperatures,

the barometric pressure, and the rainfall over a 24-hour period. Each of these instruments

is controlled by a software system that takes parameter readings periodically

and manages the data collected from the instruments.

The weather station system operates by collecting weather observations at frequent

intervals—for example, temperatures are measured every minute. However,

because the bandwidth to the satellite is relatively narrow, the weather station carries

out some local processing and aggregation of the data. It then transmits this aggregated

data when requested by the data collection system. If, for whatever reason, it is

impossible to make a connection, then the weather station maintains the data locally

until communication can be resumed.

Each weather station is battery-powered and must be entirely self-contained—there

are no external power or network cables available. All communications are through a relatively

slow-speed satellite link and the weather station must include some mechanism

(solar or wind power) to charge its batteries. As they are deployed in wilderness areas,

they are exposed to severe environmental conditions and may be damaged by animals.

The station software is therefore not just concerned with data collection. It must also:

1. Monitor the instruments, power, and communication hardware and report faults

to the management system.

2. Manage the system power, ensuring that batteries are charged whenever the

environmental conditions permit but also that generators are shut down in

potentially damaging weather conditions, such as high wind.

3. Allow for dynamic reconfiguration where parts of the software are replaced

with new versions and where backup instruments are switched into the system

in the event of system failure.

Because weather stations have to be self-contained and unattended, this means

that the software installed is complex, even though the data collection functionality

is fairly simple.

**24** Chapter 1 \_ Introduction

KEY POINTS

\_ Software engineering is an engineering discipline that is concerned with all aspects of software

production.

\_ Software is not just a program or programs but also includes documentation. Essential software

product attributes are maintainability, dependability, security, efficiency, and acceptability.

\_ The software process includes all of the activities involved in software development. The highlevel

activities of specification, development, validation, and evolution are part of all software

processes.

\_ The fundamental notions of software engineering are universally applicable to all types of

system development. These fundamentals include software processes, dependability, security,

requirements, and reuse.

\_ There are many different types of systems and each requires appropriate software engineering

tools and techniques for their development. There are few, if any, specific design and

implementation techniques that are applicable to all kinds of systems.

\_ The fundamental ideas of software engineering are applicable to all types of software systems.

These fundamentals include managed software processes, software dependability and security,

requirements engineering, and software reuse.

\_ Software engineers have responsibilities to the engineering profession and society. They should

not simply be concerned with technical issues.

\_ Professional societies publish codes of conduct that set out the standards of behavior expected

of their members.

**FURTHER RE ADING**

‘No silver bullet: Essence and accidents of software engineering’. In spite of its age, this paper is a

good general introduction to the problems of software engineering. The essential message of the

paper still hasn’t changed. (F. P. Brooks, *IEEE Computer,* **20** (4), April 1987.)

http://doi.ieeecomputersociety.org/10.1109/MC.1987.1663532.

‘Software engineering code of ethics is approved’. An article that discusses the background to the

development of the ACM/IEEE Code of Ethics and that includes both the short and long form of the

code. (*Comm. ACM*, D. Gotterbarn, K. Miller, and S. Rogerson, October 1999.)

http://portal.acm.org/citation.cfm?doid=317665.317682.

*Professional Issues in Software Engineering*. This is an excellent book discussing legal and

professional issues as well as ethics. I prefer its practical approach to more theoretical texts on

ethics. (F. Bott, A. Coleman, J. Eaton and D. Rowland, 3rd edition, 2000, Taylor and Francis.)

Chapter 1 \_ Exercises **25**

*IEEE Software, March/April 2002*. This is a special issue of the magazine devoted to the

development of Web-based software. This area has changed very quickly so some articles are a little

dated but most are still relevant. (*IEEE Software,* **19** (2), 2002.)

http://www2.computer.org/portal/web/software.

‘A View of 20th and 21st Century Software Engineering’. A backward and forward look at software

engineering from one of the first and most distinguished software engineers. Barry Boehm identifies

timeless software engineering principles but also suggests that some commonly used practices are

obsolete. (B. Boehm, *Proc. 28th Software Engineering Conf.,* Shanghai. 2006.)

http://doi.ieeecomputersociety.org/10.1145/1134285.1134288.

‘Software Engineering Ethics’. Special issue of IEEE Computer, with a number of papers on the topic.

(*IEEE Computer,* **42** (6), June 2009.)

**E XERCISES**

**1.1.** Explain why professional software is not just the programs that are developed for a customer.

**1.2.** What is the most important difference between generic software product development and

custom software development? What might this mean in practice for users of generic software

products?

**1.3.** What are the four important attributes that all professional software should have? Suggest

four other attributes that may sometimes be significant.

**1.4.** Apart from the challenges of heterogeneity, business and social change, and trust and

security, identify other problems and challenges that software engineering is likely to face in

the 21st century (Hint: think about the environment).

**1.5.** Based on your own knowledge of some of the application types discussed in section 1.1.2,

explain, with examples, why different application types require specialized software

engineering techniques to support their design and development.

**1.6.** Explain why there are fundamental ideas of software engineering that apply to all types of

software systems.

**1.7.** Explain how the universal use of the Web has changed software systems.

**1.8.** Discuss whether professional engineers should be certified in the same way as doctors or

lawyers.

**1.9.** For each of the clauses in the ACM/IEEE Code of Ethics shown in Figure 1.3, suggest an

appropriate example that illustrates that clause.

**1.10.** To help counter terrorism, many countries are planning or have developed computer systems

that track large numbers of their citizens and their actions. Clearly this has privacy

implications. Discuss the ethics of working on the development of this type of system.

**26** Chapter 1 \_ Introduction

**RE FERENCES**

Gotterbarn, D., Miller, K. and Rogerson, S. (1999). Software Engineering Code of Ethics is Approved.

Comm. ACM, **42** (10), 102–7.

Holdener, A. T. (2008). *Ajax: The Definitive Guide*. Sebastopol, Ca.: O’Reilly and Associates.

Huff, C. and Martin, C. D. (1995). Computing Consequences: A Framework for Teaching Ethical

Computing. Comm. ACM, **38** (12), 75–84.

Johnson, D. G. (2001). *Computer Ethics*. Englewood Cliffs, NJ: Prentice Hall.

Laudon, K. (1995). Ethical Concepts and Information Technology. Comm. ACM, **38** (12), 33–9.

Naur, P. and Randell, B. (1969). Software Engineering: Report on a Conference sponsored by the

NATO Science Committee, Garmisch, Germany. 7th to 11th October