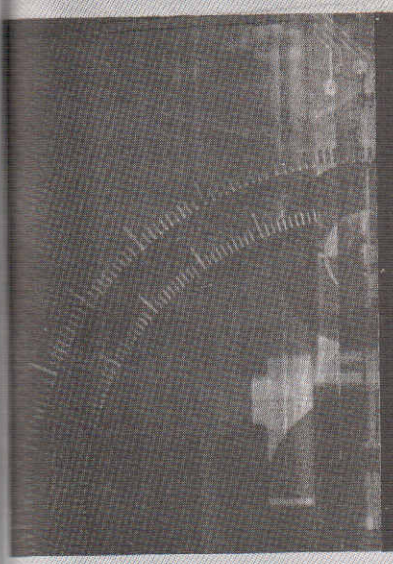


PART ONE

Overview

1 Introduction

Objectives

The objective of this chapter is to introduce the subject of software engineering. When you have read this chapter you will:

- understand what software engineering is and why it is important;
- know the answers to key questions which provide an introduction to software engineering;
- understand ethical and professional issues which are important for software engineers.

Contents

Virtually all countries now depend on complex computer-based systems. More and more products incorporate computers and controlling software in some form. The software in these systems represents a large and increasing proportion of the total system costs. Therefore, producing software in a cost-effective way is essential for the functioning of national and international economies.

Software engineering is an engineering discipline whose goal is the cost-effective development of software systems. Software is abstract and intangible. It is not constrained by materials, governed by physical laws or by manufacturing processes. In some ways, this simplifies software engineering as there are no physical limitations on the potential of software. In other ways, however, this lack of natural constraints means that software can easily become extremely complex and hence very difficult to understand.

Software engineering is still a relatively young discipline. The notion of 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called the 'software crisis'. This software crisis resulted directly from the introduction of (at that time) powerful, third-generation computer hardware. Their power made hitherto unrealisable computer applications a feasible proposition. The resulting software was orders of magnitude larger and more complex than previous software systems.

Early experience in building these systems showed that an informal approach to software development was not good enough. Major projects were sometimes years late. They cost much more than originally predicted, were unreliable, difficult to maintain and performed poorly. Software development was in crisis. Hardware costs were tumbling whilst software costs were rising rapidly. New techniques and methods were needed to control the complexity inherent in large software systems.

These techniques have become part of software engineering and are now widely although not universally used. However, there are still problems in producing complex software which meets user expectations, is delivered on time and to budget. Many software projects still have problems and this has led to some commentators (Pressman, 1997) suggesting that software engineering is in a state of chronic affliction.

As our ability to produce software has increased so too has the complexity of the software systems required. New technologies resulting from the convergence of computers and communication systems place new demands on software engineers. For this reason and because many companies do not apply software engineering techniques effectively, we still have problems. Things are not as bad as the doomsayers suggest but there is clearly room for improvement.

I think that we have made tremendous progress since 1968 and that the development of software engineering has markedly improved our software. We have a much better understanding of the activities involved in software development. We have developed effective methods of software specification, design and implementation. New notations and tools reduce the effort required to produce large and complex systems.

Software engineers can be rightly proud of their achievements. Without complex software we would not have explored space, would not have the Internet and modern telecommunications, and all forms of travel would be more dangerous and

expensive. Software engineering has contributed a great deal in its short lifetime and I am convinced that, as the discipline matures, its contributions in the 21st century will be even greater.

1.1 FAQs about software engineering

This section is designed to answer some fundamental questions about software engineering and also to give you some impression of my views of the discipline. The format that I have used here is the 'FAQ (Frequently Asked Questions) list'. This approach is commonly used in Internet newsgroups to provide newcomers with answers to frequently asked questions. I believe that it is a very effective way to give a succinct introduction to the subject of software engineering.

The questions which are answered in this section are shown in Figure 1.1.

1.1.1 What is software?

Many people equate the term *software* with computer programs. In fact, this is too restrictive a view. Software is not just the programs but also all associated documentation and configuration data which is needed to make these programs operate correctly. A software system usually consists of a number of separate programs, configuration files which are used to set up these programs, system documentation which describes the structure of the system and user documentation which explains how to use the system and, for software products, web sites for users to download recent product information.

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

1. *Generic products* These are stand-alone systems which are produced by a development organisation and sold on the open market to any customer who is able to buy them. Sometimes they are referred to as shrink-wrapped software. Examples of this type of product include databases, word processors, drawing packages and project management tools.
2. *Bespoke (or customised) products* These are systems which are commissioned by a particular customer. The software is developed specially for that customer by a software contractor. 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 different types of software is that, in generic products, the organisation which develops the software controls the software

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 is software engineering?	Software engineering is an engineering discipline which is concerned with all aspects of software production.
What is the difference between software engineering and computer science?	Computer science is concerned with 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 process.
What is a software process?	A set of activities whose goal is the development or evolution of software.
What is a software process model?	A simplified representation of a software process, presented from a specific perspective.
What are the costs of software engineering?	Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are software engineering methods?	Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
What is CASE (Computer-Aided Software Engineering)?	Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support.
What are the attributes of good software?	The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What are the key challenges facing software engineering?	Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times.

Figure 1.1 Frequently asked questions about software engineering

specification. For custom products, the specification is usually developed and controlled by the organisation that is buying the software. The software developers must work to that specification.

1.1.2 What is software engineering?

Software engineering is an engineering discipline which 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 but they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods to support them. Engineers also recognise that they must work to organisational 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 but also with activities such as software project management and with the development of tools, methods and theories to support software production.

In general, software engineers adopt a systematic and organised 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 and a more creative, informal approach to development may be effective in some circumstances. Informal development is particularly appropriate for the development of web-based e-commerce systems which requires a blend of software and graphical design skills.

1.1.3 What is the difference between software engineering and computer science?

Essentially, computer science is concerned with the theories and methods which 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.

Ideally, all of software engineering should be underpinned by theories of computer science but in reality this is not the case. Software engineers must often use *ad hoc* approaches to develop the software. Elegant theories of computer science cannot always be applied to real, complex problems which require a software solution.

1.1.4 What is the difference between software engineering and system engineering?

System engineering or, more precisely, computer-based 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.).

System engineering is an older discipline than software engineering. People have been specifying and assembling complex industrial systems such as trains and chemical plants for more than 100 years. However, as the percentage of software in systems has increased, software engineering techniques such as use-case modelling, configuration management, etc. are being used in the systems engineering process. I discuss system engineering in more detail in Chapter 2.

1.1.5 What is a software process?

A software process is a set of activities and associated results which produce a software product. These activities are mostly carried out by software engineers. There are four fundamental process activities (covered later in the book) which are common to all software processes. These activities are:

1. *Software specification* The functionality of the software and constraints on its operation must be defined.
2. *Software development* The software to meet the specification must be produced.
3. *Software validation* The software must be validated to ensure that it does what the customer wants.
4. *Software evolution* The software must evolve to meet changing customer needs.

Different software processes organise these activities in different ways and are described at different levels of detail. The timing of the activities varies, as does the results of each activity. Different organisations may use different processes to produce the same type of product. However, some processes are more suitable than others for some types of application. If an inappropriate process is used, this will probably reduce the quality or the usefulness of the software product to be developed.

Software processes are discussed in more detail in Chapter 3 and the important topic of software process improvement is covered in Chapter 25.

1.1.6 What is a software process model?

A software process model is a simplified description of a software process which is presented from a particular perspective. Models, by their very nature, are simplifications, so a software process model is an abstraction of the actual process which is being described. Process models may include activities which are part of the software process, software products and the roles of people involved in software engineering. Some examples of the types of software process model which may be produced are:

1. *A workflow model* This shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model represent human actions.

2. *A data-flow or activity model* This represents the process as a set of activities each of which carries out some data transformation. It shows how the input to the process such as a specification is transformed to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may represent transformations carried out by people or by computers.
3. *A role/action model* This represents the roles of the people involved in the software process and the activities for which they are responsible.

There are a number of different general models or paradigms of software development:

1. *The waterfall approach* This takes the above activities and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on. After each stage is defined it is 'signed off' and development goes on to the following stage.
2. *Evolutionary development* This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system which satisfies the customer's needs. The system may then be delivered. Alternatively, it may be reimplemented using a more structured approach to produce a more robust and maintainable system.
3. *Formal transformation* This approach is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods, to a program. These transformations are 'correctness-preserving'. This means that you can be sure that the developed program meets its specification.
4. *System assembly from reusable components* This technique assumes that parts of the system already exist. The system development process focuses on integrating these parts rather than developing them from scratch. I discuss software reuse in Chapter 14.

I return to these generic process models in Chapter 3.

1.1.7 What are the costs of software engineering?

There is no simple answer to this question as the precise distribution of costs across the software process depends on the process used and the type of software which is being developed. If we take the total cost of developing a complex software system as 100 cost units, the distribution of these cost units is likely to be something like that shown in Figure 1.2.

This cost distribution holds where the costs of specification, design, implementation and integration are measured separately. Notice that system integration and testing is the most expensive development activity. Figure 1.2 suggests that this

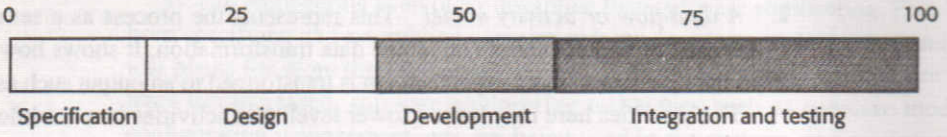
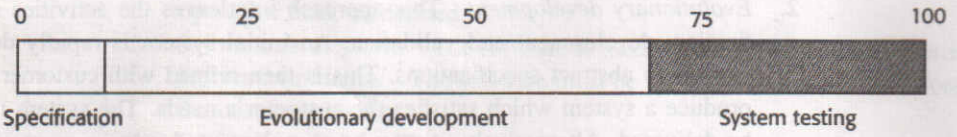


Figure 1.2 Development cost distribution

is about 40 per cent of the total development costs but for some critical systems it is likely to be nearer 50 per cent of the total system costs.

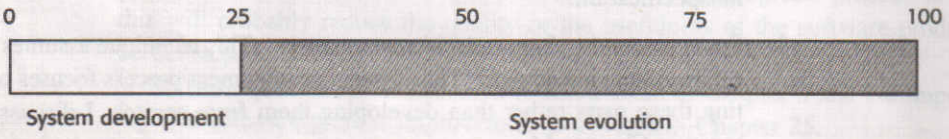
If the software is developed using an evolutionary approach, there is no hard line between specification, design and development. Figure 1.2 would have to be modified for this type of development as shown in Figure 1.3. Specification costs are reduced because only a high-level specification is produced before development in this approach. Specification, design, implementation, integration and testing are carried out in parallel within a development activity. However, there is still a need for a separate system testing activity once the initial implementation is complete.

Figure 1.3 Costs of evolutionary development



On top of development costs, costs are also incurred in changing the software after it has gone into use. For many software systems which have a long lifetime, these costs are likely to exceed the development costs by a factor of 3 or 4 (Figure 1.4).

Figure 1.4 Evolution costs



The above cost distribution holds for customised software which is specified by a customer and developed by a contractor. For software products which are (mostly) sold for PCs, the cost profile is likely to be different. These products are usually developed from an outline specification using an evolutionary development approach. Specification costs are relatively low. However, because they are intended for use on a range of different configurations, they must be extensively tested. Figure 1.5 shows the type of cost profile that might be expected for these products.

Figure 1.5 Product development costs

The evolution costs for generic software products are particularly hard to estimate. In many cases, there is little formal evolution of a product. Once a version of the product has been released, work starts on the next release and, for marketing reasons, this is likely to be presented as a new (but compatible) product rather than a modified version of a product which the user has already bought. Therefore, the evolution costs are not assessed separately as they are in customised software but are simply the development costs for the next version of the system.

The cost model for e-commerce web-based systems is likely to be different from both of these. These systems usually use off-the-shelf software for information management and have high user interface development costs. At the time of writing, these systems have only just come into use. I don't have any reliable figures on their development costs.

1.1.8 What are software engineering methods?

A software engineering method is a structured approach to software development whose aim is to facilitate the production of high-quality software in a cost-effective way. Methods such as Structured Analysis (DeMarco, 1978) and JSD (Jackson, 1983) were first developed in the 1970s. These methods attempted to identify the basic functional components of a system and function-oriented methods are still widely used. In the 1980s and 1990s, these function-oriented methods were supplemented by object-oriented methods such as those proposed by Booch (1994) and Rumbaugh (Rumbaugh *et al.*, 1991). These different approaches have now been integrated into a single unified approach built around the Unified Modeling Language (UML) (Fowler and Scott, 1997; Booch *et al.*, 1999; Rumbaugh *et al.*, 1999a, 1999b).

All methods are based on the idea of developing models of a system which may be represented graphically and using these models as a system specification or design. Methods should include a number of different components (Figure 1.6).

Figure 1.6 Method components

Component	Description	Example
System model descriptions	Descriptions of the system models which should be developed and the notation used to define these models.	Object models, data-flow models, state machine models, etc.
Rules	Constraints which always apply to system models.	Every entity in a system model must have a unique name.
Recommendations	Heuristics which characterise good design practice in this method. Following these recommendations should lead to a well-organised system model.	No object should have more than seven sub-objects associated with it.
Process guidance	Descriptions of the activities which may be followed to develop the system models and the organisation of these activities.	Object attributes should be documented before defining the operations associated with an object.

There is no ideal method and different methods have different areas where they are applicable. For example, object-oriented methods are often appropriate for interactive systems but not for systems with stringent real-time requirements.

1.1.9 What is CASE?

The acronym CASE stands for Computer-Aided Software Engineering. It covers a wide range of different types of program which are used to support software process activities such as requirements analysis, system modelling, debugging and testing. All methods now come with associated CASE technology such as editors for the notations used in the method, analysis modules which check the system model according to the method rules and report generators to help create system documentation. The CASE tools may also include a code generator which automatically generates source code from the system model and some process guidance which gives advice to the software engineer on what to do next.

This type of CASE tool, aimed at supporting analysis and design, is sometimes called an upper-CASE tool because it supports early phases of the software process. By contrast, CASE tools which are designed to support implementation and testing such as debuggers, program analysis systems, test case generators and program editors are sometimes called lower-CASE tools.

1.1.10 What are the attributes of good software?

As well as the services which they provide, software products have a number of other associated attributes which reflect the quality of that software. These attributes are not directly concerned with what the software does. Rather, they reflect its behaviour while it is executing and the structure and organisation of the source program and associated documentation. Examples of these attributes (sometimes called non-functional attributes) are the software's response time to a user query and the understandability of the program code.

The specific set of attributes which 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, etc. These can be generalised into the set of attributes shown in Figure 1.7 which I believe are the essential characteristics of a well-designed software system.

The techniques discussed in this book focus on two of these attributes, namely maintainability and dependability. The majority of software engineering methods, tools and techniques are intended to help produce software with these characteristics. Software performance improvement is usually dependent on very specific domain knowledge and usability is a major, separate topic in its own right. However, I do discuss usability in Chapter 15.

Figure 1.7 Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.
Dependability	Software dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Usability	Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

1.1.11 What are the key challenges facing software engineering?

Software engineering in the 21st century faces three key challenges:

1. *The legacy challenge* The majority of large software systems which are in use today were developed many years ago yet they perform critical business functions. The legacy challenge is the challenge of maintaining and updating this software in such a way that excessive costs are avoided and essential business services continue to be delivered.
2. *The heterogeneity challenge* Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and with different kinds of support systems. The heterogeneity challenge is the challenge of developing techniques to build dependable software which is flexible enough to cope with this heterogeneity.
3. *The delivery challenge* Many traditional software engineering techniques are time-consuming. The time they take is required to achieve software quality. However, businesses today must be responsive and change very rapidly. Their supporting software must change equally rapidly. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

Of course, these are not independent. For example, it may be necessary to make rapid changes to a legacy system to make it accessible across a network. To address these challenges we will need new tools and techniques as well as innovative ways of combining and using existing software engineering methods.

1.2 Professional and ethical responsibility

Like other engineers, software engineers must accept that their job involves wider responsibilities than simply the application of technical skills. Their work is carried out within a legal and social framework. Software engineering is obviously bounded by local, national and international laws. Software engineers must behave in an ethical and morally responsible way if they are to be respected as professionals.

It goes without saying that engineers should uphold normal standards of honesty and integrity. They should not use their 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 behaviour are not bounded by laws but by the more tenuous notion of professional responsibility. Some of these are:

1. *Confidentiality* Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

2. *Competence* Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.

3. *Intellectual property rights* Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

4. *Computer misuse* Software engineers should not use their 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).

In this respect, professional societies and institutions have an important role to play. Organisations 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 organisations undertake to follow that code when they sign up for membership. These codes of conduct are generally concerned with fundamental ethical behaviour.

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 1.8, and a longer form (Gotterbarn *et al.*, 1999) which adds detail and substance to the shorter version. The rationale behind this code is summarised 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

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 1.8 ACM/IEEE Code of Ethics (©IEEE/ACM 1999)

those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software 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 organisation or to resign on 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 over-reacting 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, engineers are 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 publicise, 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.

You must make up your own mind in these matters. 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 publicise 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 defence is an overriding principle and have no ethical objections to working on weapons systems. The appropriate ethical position here depends entirely on the views of the individuals who are involved.

In this situation it is important that both employers and employees should make their views known to each other in advance. Where an organisation 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 employees are taken on and make 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 one which has received increasing attention over the past few years. 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).

However, I find this approach rather abstract and difficult to relate to my everyday experience. I much 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 which can be the basis of an ethical discussion.

KEY POINTS

- Software engineering is an engineering discipline which is concerned with all aspects of software production.
- Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
- The software process consists of activities which are involved in developing software products. Basic activities are software specification, development, validation and evolution.
- Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.
- CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.
- 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 which set out the standards of behaviour expected of their members.

FURTHER READING

Software Engineering: An Engineering Approach. A general text that includes a number of useful case studies. (J. F. Peters and W. Pedrycz, 2000, John Wiley and Sons.)

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

Software Engineering: A Practitioner's Approach. A general textbook that surveys a wide variety of software engineering topics. (R. S. Pressman, 1997, McGraw Hill.)

Ethics and Computing: Living Responsibly in a Computerized World. A good overview of the topic along with a number of more specialised papers. (K. W. Bowyer, 1996, IEEE Computer Society Press.)

Professional Issues in Software Engineering. This is an excellent book discussing legal and professional issues as well as ethics. (F. Bott, A. Coleman, J. Eaton and D. Rowland, 1995, UCL Press.)

'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 hasn't changed in the past 13 years. (F. P. Brooks, *IEEE Computer*, 20(4), April 1987.)

EXERCISES

1.1 By making reference to the distribution of software costs discussed in section 1.1.7, explain why it is appropriate to consider software to be more than the programs which can be executed by end-users of a system.

1.2 What are the four important attributes which all software products should have? Suggest four other attributes which may be significant.

1.3 What is the difference between a software process model and a software process? Suggest two ways in which a software process model might be helpful in identifying possible process improvements.

1.4 Explain why system testing costs are particularly high for generic software products which are sold to a very wide market.

1.5 Software engineering methods only became widely used when CASE technology became available to support them. Suggest five types of method support which can be provided by CASE tools.

1.6 Apart from the challenges of legacy systems, heterogeneity and rapid delivery, identify other problems and challenges that software engineering is likely to face in the 21st century.

1.7 Discuss whether professional engineers should be certified in the same way as doctors or lawyers.

1.8 For each of the clauses in the ACM/IEEE Code of Ethics shown in Figure 1.8, suggest an appropriate example that illustrates that clause.