



# 10

## Sociotechnical systems

### Objectives

The objectives of this chapter are to introduce the concept of a sociotechnical system—a system that includes people, software, and hardware—and to show that you need to take a systems perspective on security and dependability. When you have read this chapter, you will:

- know what is meant by a sociotechnical system and understand the difference between a technical, computer-based system and a sociotechnical system;
- have been introduced to the concept of emergent system properties, such as reliability, performance, safety, and security;
- know about the procurement, development, and operational activities that are involved in the systems engineering process;
- understand why software dependability and security should not be considered in isolation and how they are affected by systems issues, such as operator errors.

### Contents

- 10.1** Complex systems
- 10.2** Systems engineering
- 10.3** System procurement
- 10.4** System development
- 10.5** System operation

In a computer system, the software and the hardware are interdependent. Without hardware, a software system is an abstraction, which is simply a representation of some human knowledge and ideas. Without software, hardware is a set of inert electronic devices. However, if you put them together to form a system, you create a machine that can carry out complex computations and deliver the results of these computations to its environment.

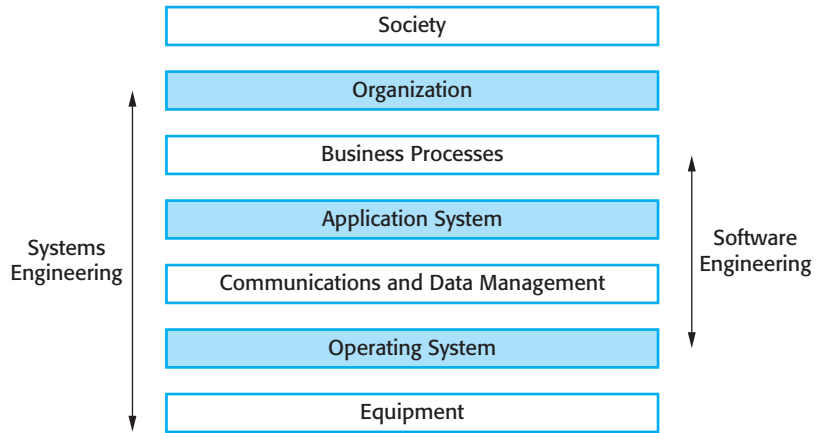
This illustrates one of the fundamental characteristics of a system—it is more than the sum of its parts. Systems have properties that only become apparent when their components are integrated and operate together. Therefore software engineering is not an isolated activity, but is an intrinsic part of more general systems engineering processes. Software systems are not isolated systems but rather essential components of more extensive systems that have some human, social, or organizational purpose.

For example, the wilderness weather system software controls the instruments in a weather station. It communicates with other software systems and is a part of wider national and international weather forecasting systems. As well as hardware and software, these systems include processes for forecasting the weather, people who operate the system and analyze its outputs. The system also includes the organizations that depend on the system to help them provide weather forecasts to individuals, government, industry, etc. These broader systems are sometimes called sociotechnical systems. They include nontechnical elements such as people, processes, regulations, etc., as well as technical components such as computers, software, and other equipment.

Sociotechnical systems are so complex that it is practically impossible to understand them as a whole. Rather, you have to view them as layers, as shown in Figure 10.1. These layers make up the sociotechnical systems stack:

1. *The equipment layer* This layer is composed of hardware devices, some of which may be computers.
2. *The operating system layer* This layer interacts with the hardware and provides a set of common facilities for higher software layers in the system.
3. *The communications and data management layer* This layer extends the operating system facilities and provides an interface that allows interaction with more extensive functionality, such as access to remote systems, access to a system database, etc. This is sometimes called middleware, as it is in between the application and the operating system.
4. *The application layer* This layer delivers the application-specific functionality that is required. There may be many different application programs in this layer.
5. *The business process layer* At this level, the organizational business processes, which make use of the software system, are defined and enacted.
6. *The organizational layer* This layer includes higher-level strategic processes as well as business rules, policies, and norms that should be followed when using the system.
7. *The social layer* At this layer, the laws and regulations of society that govern the operation of the system are defined.

**Figure 10.1** The sociotechnical systems stack



In principle, most interactions are between neighboring layers, with each layer hiding the detail of the layer below from the layer above. In practice, this is not always the case. There can be unexpected interactions between layers, which result in problems for the system as a whole. For example, say there is a change in the law governing access to personal information. This comes from the social layer. It leads to new organizational procedures and changes to the business processes. However, the application system may not be able to provide the required level of privacy so changes have to be implemented in the communications and data management layer.

Thinking holistically about systems, rather than simply considering software in isolation, it is essential when considering software security and dependability. Software failure, in itself, rarely has serious consequences because software is intangible and, even when damaged, is easily and cheaply restored. However, when these software failures ripple through other parts of the system, they affect the software's physical and human environment. Here, the consequences of failure are more significant. People may have to do extra work to contain or recover from the failure; for example, there may be physical damage to equipment, data may be lost or corrupted, or confidentiality may be breached with unknown consequences.

You must, therefore, take a system-level view when you are designing software that has to be secure and dependable. You need to understand the consequences of software failures for other elements in the system. You also need to understand how these other system elements may be the cause of software failures and how they can help to protect against and recover from software failures.

Therefore, it is a system rather than a software failure that is the real problem. This means that you need to examine how the software interacts with its immediate environment to ensure that:

1. Software failures are, as far as possible, contained within the enclosing layers of the system stack and do not seriously affect the operation of adjoining layers. In particular, software failures should not lead to system failures.

2. You understand how faults and failures in the non-software layers of the systems stack may affect the software. You may also consider how checks may be built into the software to help detect these failures, and how support can be provided for recovering from failure.

As software is inherently flexible, unexpected system problems are often left to software engineers to solve. Say a radar installation has been sited so that ghosting of the radar image occurs. It is impractical to move the radar to a site with less interference, so the systems engineers have to find another way of removing this ghosting. Their solution may be to enhance the image-processing capabilities of the software to remove the ghost images. This may slow down the software so that its performance becomes unacceptable. The problem may then be characterized as a ‘software failure’, whereas, in fact, it is a failure in the design process for the system as a whole.

This sort of situation, in which software engineers are left with the problem of enhancing software capabilities without increasing hardware costs, is very common. Many so-called software failures are not a consequence of inherent software problems but rather are the result of trying to change the software to accommodate modified system engineering requirements. A good example of this was the failure of the Denver airport baggage system (Swartz, 1996), where the controlling software was expected to deal with limitations of the equipment used.

Systems engineering (Stevens et al., 1998; Thayer, 2002; Thomé, 1993; White et al., 1993) is the process of designing entire systems—not just the software in these systems. Software is the controlling and integrating element in these systems and software engineering costs are often the main cost component in the overall system costs. As a software engineer, it helps if you have a broader awareness of how software interacts with other hardware and software systems, and how it is supposed to be used. This knowledge helps you understand the limits of software, to design better software, and to participate in a systems engineering group.

## 10.1 Complex systems

The term ‘system’ is one that is universally used. We talk about computer systems, operating systems, payment systems, the education system, the system of government, and so on. These are all obviously quite different uses of the word ‘system’, although they share the characteristic that, somehow, the system is more than simply the sum of its parts.

Abstract systems, such as the system of government, are outside the scope of this book. Rather, I focus on systems that include computers and that have some specific purpose such as to enable communication, support navigation, or compute salaries. A useful working definition of these types of systems is as follows:

*A system is a purposeful collection of interrelated components, of different kinds, which work together to achieve some objective.*

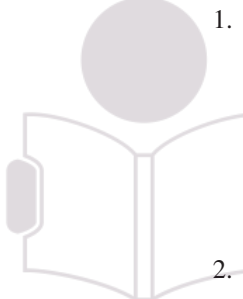
This general definition embraces a vast range of systems. For example, a simple system, such as laser pointer, may include a few hardware components plus a small

amount of control software. By contrast, an air traffic control system includes thousands of hardware and software components plus human users who make decisions based on information from that computer system.

A characteristic of all complex systems is that the properties and behavior of the system components are inextricably intermingled. The successful functioning of each system component depends on the functioning of other components. Thus, software can only operate if the processor is operational. The processor can only carry out computations if the software system defining these computations has been successfully installed.

Complex systems are usually hierarchical and so include other systems. For example, a police command and control system may include a geographical information system to provide details of the location of incidents. These included systems are called ‘subsystems’. Subsystems can operate as independent systems in their own right. For example, the same geographical information system may be used in systems for transport logistics and emergency command and control.

Systems that include software fall into two categories:

- 
1. *Technical computer-based systems* These are systems that include hardware and software components but not procedures and processes. Examples of technical systems include televisions, mobile phones, and other equipment with embedded software. Most software for PCs, computer games, etc., also falls into this category. Individuals and organizations use technical systems for a particular purpose but knowledge of this purpose is not part of the system. For example, the word processor I am using is not aware that it is being used to write a book.
  2. *Sociotechnical systems* These include one or more technical systems but, crucially, also include people who understand the purpose of the system within the system itself. Sociotechnical systems have defined operational processes and people (the operators) are inherent parts of the system. They are governed by organizational policies and rules and may be affected by external constraints such as national laws and regulatory policies. For example, this book was created through a sociotechnical publishing system that includes various processes and technical systems.

Sociotechnical systems are enterprise systems that are intended to help deliver a business goal. This might be to increase sales, reduce material used in manufacturing, collect taxes, maintain a safe airspace, etc. Because they are embedded in an organizational environment, the procurement, development, and use of these systems are influenced by the organization’s policies and procedures, and by its working culture. The users of the system are people who are influenced by the way the organization is managed and by their interactions with other people inside and outside of the organization.

When you are trying to develop sociotechnical systems, you need to understand the organizational environment in which they are used. If you don’t, the systems may not meet business needs and users and their managers may reject the system.

Organizational factors from the system's environment that may affect the requirements, design, and operation of a sociotechnical system include:

1. *Process changes* The system may require changes to the work processes in the environment. If so, training will certainly be required. If changes are significant, or if they involve people losing their jobs, there is a danger that the users will resist the introduction of the system.
2. *Job changes* New systems may de-skill the users in an environment or cause them to change the way they work. If so, users may actively resist the introduction of the system into the organization. Designs that involve managers having to change their way of working to fit a new computer system are often resented. The managers may feel that their status in the organization is being reduced by the system.
3. *Organizational changes* The system may change the political power structure in an organization. For example, if an organization is dependent on a complex system, those who control access to that system have a great deal of political power.

Sociotechnical systems have three characteristics that are particularly important when considering security and dependability:

1. They have emergent properties that are properties of the system as a whole, rather than associated with individual parts of the system. Emergent properties depend on both the system components and the relationships between them. Given this complexity, the emergent properties can only be evaluated once the system has been assembled. Security and dependability are emergent system properties.
2. They are often nondeterministic. This means that when presented with a specific input, they may not always produce the same output. The system's behavior depends on the human operators and people do not always react in the same way. Furthermore, use of the system may create new relationships between the system components and hence change its emergent behavior. System faults and failures may therefore be transient, and people may disagree about whether or not a failure has actually occurred.
3. The extent to which the system supports organizational objectives does not just depend on the system itself. It also depends on the stability of these objectives, the relationships, and conflicts between organizational objectives and how people in the organization interpret these objectives. New management may reinterpret the organizational objectives that a system was designed to support so that a 'successful' system may then be seen as a 'failure'.

Sociotechnical considerations are often critical in determining whether or not a system has successfully met its objectives. Unfortunately, taking these into account is very difficult for engineers who have little experience of social or cultural studies.



Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failures and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty, and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators, and its operating environment.

**Figure 10.2**  
Examples of  
emergent properties

To help understand the effects of systems on organizations, various methodologies have been developed, such as Mumford's sociotechnics (1989) and Checkland's Soft Systems Methodology (1981; Checkland and Scholes, 1990). There have also been sociological studies of the effects of computer-based systems on work (Ackroyd et al., 1992; Anderson et al., 1989; Suchman, 1987).

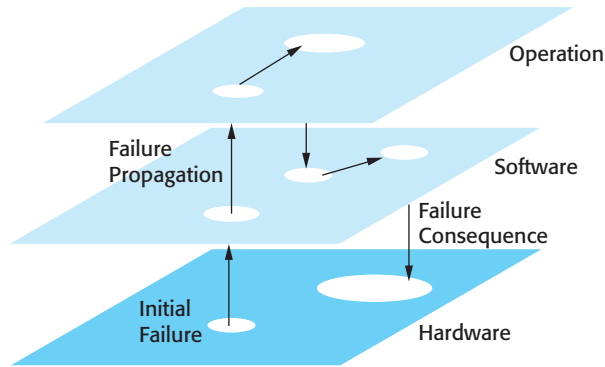
### 10.1.1 Emergent system properties

The complex relationships between the components in a system mean that a system is more than simply the sum of its parts. It has properties that are properties of the system as a whole. These 'emergent properties' (Checkland, 1981) cannot be attributed to any specific part of the system. Rather, they only emerge once the system components have been integrated. Some of these properties, such as weight, can be derived directly from the comparable properties of subsystems. More often, however, they result from complex subsystem interrelationships. The system property cannot be calculated directly from the properties of the individual system components. Examples of some emergent properties are shown in Figure 10.2.

There are two types of emergent properties:

1. Functional emergent properties when the purpose of a system only emerges after its components are integrated. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
2. Non-functional emergent properties, which relate to the behavior of the system in its operational environment. Reliability, performance, safety, and security are examples of emergent properties. These are critical for computer-based systems, as failure to achieve a minimum defined level in these properties usually makes

**Figure 10.3** Failure propagation



the system unusable. Some users may not need some of the system functions, so the system may be acceptable without them. However, a system that is unreliable or too slow is likely to be rejected by all its users.

Emergent dependability properties, such as reliability, depend on both the properties of individual components and their interactions. The components in a system are interdependent. Failures in one component can be propagated through the system and affect the operation of other components. However, it is often difficult to anticipate how these component failures will affect other components. It is, therefore, practically impossible to estimate overall system reliability from data about the reliability of system components.

In a sociotechnical system, you need to consider reliability from three perspectives:

1. *Hardware reliability* What is the probability of hardware components failing and how long does it take to repair a failed component?
2. *Software reliability* How likely is it that a software component will produce an incorrect output? Software failure is distinct from hardware failure in that software does not wear out. Failures are often transient. The system carries on working after an incorrect result has been produced.
3. *Operator reliability* How likely is it that the operator of a system will make an error and provide an incorrect input? How likely is it that the software will fail to detect this error and propagate the mistake?

Hardware, software, and operator reliability are not independent. Figure 10.3 shows how failures at one level can be propagated to other levels in the system. Hardware failure can generate spurious signals that are outside the range of inputs expected by the software. The software can then behave unpredictably and produce unexpected outputs. These may confuse and consequently stress the system operator.

Operator error is most likely when the operator is feeling stressed. So a hardware failure may then mean that the system operator makes mistakes which, in turn, could lead to further software problems or additional processing. This could overload the



hardware, causing more failures and so on. Thus, the initial failure, which might be recoverable, can rapidly develop into a serious problem that may result in a complete shutdown of the system.

The reliability of a system depends on the context in which that system is used. However, the system's environment cannot be completely specified, nor can the system designers place restrictions on that environment for operational systems. Different systems operating within an environment may react to problems in unpredictable ways, thus affecting the reliability of all of these systems.

For example, say a system is designed to operate at normal room temperature. To allow for variations and exceptional conditions, the electronic components of a system are designed to operate within a certain range of temperatures, say from 0 degrees to 45 degrees. Outside this temperature range, the components will behave in an unpredictable way. Now assume that this system is installed close to an air conditioner. If this air conditioner fails and vents hot gas over the electronics, then the system may overheat. The components, and hence the whole system, may then fail.

If this system had been installed elsewhere in that environment, this problem would not have occurred. When the air conditioner worked properly there were no problems. However, because of the physical closeness of these machines, an unanticipated relationship existed between them that led to system failure.

Like reliability, emergent properties such as performance or usability are hard to assess but can be measured after the system is operational. Properties, such as safety and security, however, are not measurable. Here, you are not simply concerned with attributes that relate to the behavior of the system but also with unwanted or unacceptable behavior. A secure system is one that does not allow unauthorized access to its data. However, it is clearly impossible to predict all possible modes of access and explicitly forbid them. Therefore, it may only be possible to assess these 'shall not' properties by default. That is, you only know that a system is not secure when someone manages to penetrate the system.



### 10.1.2 Non-determinism

A deterministic system is one that is completely predictable. If we ignore timing issues, software systems that run on completely reliable hardware and that are presented with a sequence of inputs will always produce the same sequence of outputs. Of course, there is no such thing as completely reliable hardware, but hardware is usually reliable enough to think of hardware systems as deterministic.

People, on the other hand, are nondeterministic. When presented with exactly the same input (say a request to complete a task), their responses will depend on their emotional and physical state, the person making the request, other people in the environment, and whatever else they are doing. Sometimes they will be happy to do the work and, at other times, they will refuse.

Sociotechnical systems are non-deterministic partly because they include people and partly because changes to the hardware, software, and data in these systems are so frequent. The interactions between these changes are complex and so the behavior

of the system is unpredictable. This is not a problem in itself but, from a dependability perspective, it can make it difficult to decide whether or not a system failure has occurred, and to estimate the frequency of system failures.

For example, say a system is presented with a set of 20 test inputs. It processes these inputs and the results are recorded. At some later time, the same 20 test inputs are processed and the results compared to the previous stored results. Five of them are different. Does this mean that there have been five failures? Or are the differences simply reasonable variations in the system's behavior? You can only find this out by looking at the results in more depth and making judgments about the way the system has handled each input.

### 10.1.3 Success criteria

Generally, complex sociotechnical systems are developed to tackle what are sometimes called 'wicked problems' (Rittel and Webber, 1973). A wicked problem is a problem that is so complex and which involves so many related entities that there is no definitive problem specification. Different stakeholders see the problem in different ways and no one has a full understanding of the problem as a whole. The true nature of the problem may only emerge as a solution is developed. An extreme example of a wicked problem is earthquake planning. No one can accurately predict where the epicenter of an earthquake will be, what time it will occur, or what effect it will have on the local environment. It is impossible to specify in detail how to deal with a major earthquake.

This makes it difficult to define the success criteria for a system. How do you decide if a new system contributes, as planned, to the business goals of the company that paid for the system? The judgment of success is not usually made against the original reasons for procuring and developing the system. Rather, it is based on whether or not the system is effective at the time it is deployed. As the business environment can change very quickly, the business goals may have changed significantly during the development of the system.

The situation is even more complex when there are multiple conflicting goals that are interpreted differently by different stakeholders. For instance, the system on which the MHC-PMS (discussed in Chapter 1) is based was designed to support two distinct business goals:

1. Improve the quality of care for sufferers from mental illness.
2. Increase income by providing detailed reports of care provided and the costs of that care.

Unfortunately, these proved to be conflicting goals because the information required to satisfy the reporting goal meant that doctors and nurses had to provide additional information, over and above the health records that are normally maintained. This reduced the quality of care for patients as it meant that clinical staff had

less time to talk with them. From a doctor's perspective, this system was not an improvement on the previous manual system; from a manager's perspective, it was.

The nature of security and dependability attributes sometimes makes it even more difficult to decide if a system is successful. The intention of a new system may be to improve security by replacing an existing system with a more secure data environment. Say, after installation, the system is attacked, a security breach occurs, and some data is corrupted. Does this mean that the system is a failure? We cannot tell, because we don't know the extent of the losses that would have occurred with the old system, given the same attacks.

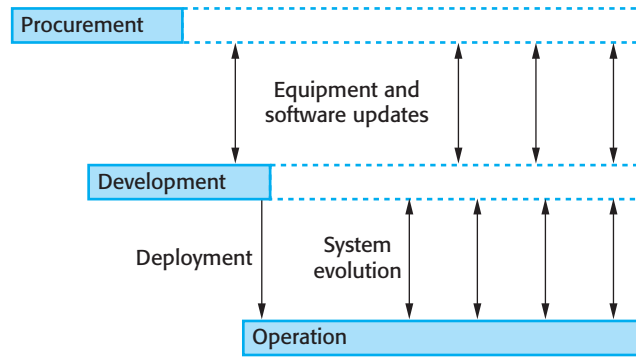
## 10.2 Systems engineering

Systems engineering encompasses all of the activities involved in procuring, specifying, designing, implementing, validating, deploying, operating, and maintaining sociotechnical systems. Systems engineers are not just concerned with software but also with hardware and the system's interactions with users and its environment. They must think about the services that the system provides, the constraints under which the system must be built and operated, and the ways in which the system is used to fulfill its purpose or purposes.

There are three overlapping stages (Figure 10.4) in the lifetime of large and complex sociotechnical systems:

1. *Procurement or acquisition* During this stage, the purpose of a system is decided; high-level system requirements are established; decisions are made on how functionality will be distributed across hardware, software, and people; and the components that will make up the system are purchased.
2. *Development* During this stage, the system is developed. Development processes include all of the activities involved in system development such as requirements definition, system design, hardware and software engineering, system integration, and testing. Operational processes are defined and the training courses for system users are designed.
3. *Operation* At this stage, the system is deployed, users are trained, and the system is brought into use. The planned operational processes usually then have to change to reflect the real working environment where the system is used. Over time, the system evolves as new requirements are identified. Eventually, the system declines in value and it is decommissioned and replaced.

These stages are not independent. Once the system is operational, new equipment and software may have to be procured to replace obsolete system components, to provide new functionality, or to cope with increased demand. Similarly, requests for changes during operation require further system development.



**Figure 10.4** Stages of systems engineering

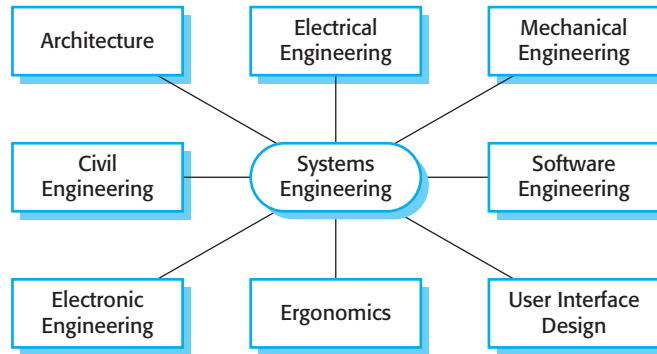
The overall security and dependability of a system is influenced by activities at all of these stages. Design options may be restricted by procurement decisions on the scope of the system and on its hardware and software. It may be impossible to implement some kinds of system safeguards. They may introduce vulnerabilities that could lead to future system failures. Human errors made during the specification, design, and development stages may mean that faults are introduced into the system. Inadequate testing may mean that faults are not discovered before a system is deployed. During operation, errors in configuring the system for deployment may lead to further vulnerabilities. System operators may make mistakes in using the system. Assumptions made during the original procurement may be forgotten when system changes are made and, again, vulnerabilities can be introduced into the system.

An important difference between systems and software engineering is the involvement of a range of professional disciplines throughout the lifetime of the system. For example, the technical disciplines that may be involved in the procurement and development of a new system for air traffic management are shown in Figure 10.5. Architects and civil engineers are involved because new air traffic management systems usually have to be installed in a new building. Electrical and mechanical engineers are involved to specify and maintain the power and air conditioning. Electronic engineers are concerned with computers, radars, and other equipment. Ergonomists design the controller workstations and software engineers and user interface designers are responsible for the software in the system.

The involvement of a range of professional disciplines is essential because there are so many different aspects of complex sociotechnical systems. However, differences between disciplines can introduce vulnerabilities into systems and so compromise the security and dependability of the system being developed:

1. Different disciplines use the same words to mean different things. Misunderstandings are common in discussions between engineers from different backgrounds. If these are not discovered and resolved during system development, they can lead to errors in delivered systems. For example, an electronic engineer who may know a little bit about C# programming may not understand that a method in Java is comparable to a function in C.

**Figure 10.5**  
Professional disciplines  
involved in systems  
engineering



2. Each discipline makes assumptions about what can or can't be done by other disciplines. These are often based on an inadequate understanding of what is actually possible. For example, a user interface designer may propose a graphical UI for an embedded system that requires a great deal of processing and so overloads the processor in the system.
3. Disciplines try to protect their professional boundaries and may argue for certain design decisions because these decisions will call for their professional expertise. Therefore, a software engineer may argue for a software-based door locking system in a building, although a mechanical, key-based system may be more reliable.

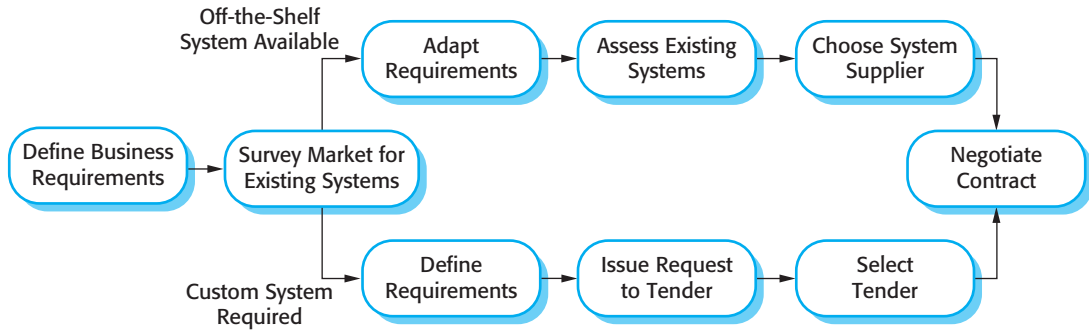


THE NEXT LEVEL OF EDUCATION

## 10.3 System procurement

The initial phase of systems engineering is system procurement (sometimes called system acquisition). At this stage, decisions are made on the scope of a system that is to be purchased, system budgets and timescales, and the high-level system requirements. Using this information, further decisions are then made on whether to procure a system, the type of system required, and the supplier or suppliers of the system. The drivers for these decisions are:

1. *The state of other organizational systems* If the organization has a mixture of systems that cannot easily communicate or that are expensive to maintain, then procuring a replacement system may lead to significant business benefits.
2. *The need to comply with external regulations* Increasingly, businesses are regulated and have to demonstrate compliance with externally defined regulations (e.g., Sarbanes-Oxley accounting regulations in the United States). This may require the replacement of noncompliant systems or the provision of new systems specifically to monitor compliance.
3. *External competition* If a business needs to compete more effectively or maintain a competitive position, investment in new systems that improve the efficiency of



**Figure 10.6** System procurement processes

business processes may be advisable. For military systems, the need to improve capability in the face of new threats is an important reason for procuring new systems.

4. *Business reorganization* Businesses and other organizations frequently restructure with the intention of improving efficiency and/or customer service. Reorganizations lead to changes in business processes that require new systems support.
5. *Available budget* The budget available is an obvious factor in determining the scope of new systems that can be procured.

In addition, new government systems are often procured to reflect political changes and political policies. For example, politicians may decide to buy new surveillance systems, which they claim will counter terrorism. Buying such systems shows voters that they are taking action. However, such systems are often procured without a cost-benefit analysis, where the benefits that result from different spending options are compared.

Large, complex systems usually consist of a mixture of off-the-shelf and specially built components. One reason why more and more software is included in systems is that it allows more use of existing hardware components, with the software acting as ‘glue’ to make these hardware components work together effectively. The need to develop this ‘glueware’ is one reason why the savings from using off-the-shelf components are sometimes not as great as anticipated.

Figure 10.6 shows a simplified model of the procurement process for both COTS system components and system components that have to be specially designed and developed. Important points about the process shown in this diagram are:

1. Off-the-shelf components do not usually match requirements exactly, unless the requirements have been written with these components in mind. Therefore, choosing a system means that you have to find the closest match between the system requirements and the facilities offered by off-the-shelf systems. You may then have to modify the requirements. This can have knock-on effects on other subsystems.



2. When a system is to be built specially, the specification of requirements is part of the contract for the system being acquired. It is therefore a legal as well as a technical document.
3. After a contractor has been selected, to build a system, there is a contract negotiation period where you may have to negotiate further changes to the requirements and discuss issues such as the cost of changes to the system. Similarly, once a COTS system has been selected, you may negotiate with the supplier on costs, licence conditions, possible changes to the system, etc.

The software and hardware in sociotechnical systems are usually developed by a different organization (the supplier) from the organization that is procuring the overall sociotechnical system. The reason for this is that the customer's business is rarely software development so its employees do not have the skills needed to develop the systems themselves. In fact, very few companies have the capabilities to design, manufacture, and test all the components of a large, complex sociotechnical system.

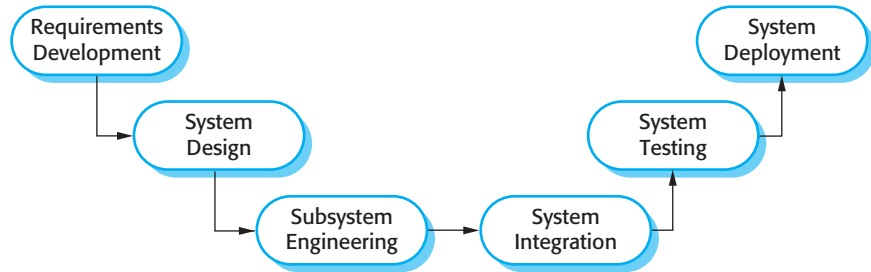
Consequently, the system supplier, who is usually called the principal contractor, often contracts out the development of different subsystems to a number of subcontractors. For large systems, such as air traffic control systems, a group of suppliers may form a consortium to bid for the contract. The consortium should include all of the capabilities required for this type of system. This includes computer hardware suppliers, software developers, peripheral suppliers, and suppliers of specialist equipment such as radar systems.

The procurer deals with the contractor rather than the subcontractors so that there is a single procurer/supplier interface. The subcontractors design and build parts of the system to a specification that is produced by the principal contractor. Once completed, the principal contractor integrates these different components and delivers them to the customer. Depending on the contract, the procurer may allow the principal contractor a free choice of subcontractors or may require the principal contractor to choose subcontractors from an approved list.

Decisions and choices made during system procurement have a profound effect on the security and dependability of a system. For example, if a decision is made to procure an off-the-shelf system, then the organization has to accept that they have very limited influence over the security and dependability requirements of this system. These largely depend on decisions made by system vendors. In addition, off-the-shelf systems may have known security weaknesses or require complex configuration. Configuration errors, where entry points to the system are not properly secured, are a major source of security problems.

On the other hand, a decision to procure a custom system means that significant effort must be devoted to understanding and defining security and dependability requirements. If a company has limited experience in this area, this is quite a difficult thing to do. If the required level of dependability as well as acceptable system performance is to be achieved, then the development time may have to be extended and the budget increased.





**Figure 10.7** Systems development

## 10.4 System development

The goals of the system development process are to develop or acquire all of the components of a system and then to integrate these components to create the final system. The requirements are the bridge between the procurement and the development processes. During procurement, business and high-level functional and non-functional system requirements are defined. You can think of this as the start of development, hence the overlapping processes shown in Figure 10.4. Once contracts for the system components have been agreed, more detailed requirements engineering then takes place.

Figure 10.7 is a model of the systems development process. This systems engineering process was an important influence on the ‘waterfall’ model of the software process that I discussed in Chapter 2. Although it is now accepted that the ‘waterfall’ model is not usually appropriate for software development, most systems development processes are plan-driven processes that still follow this model.

Plan-driven processes are used in systems engineering because different parts of the system are being developed at the same time. For systems that include hardware and other equipment, changes during development can be very expensive or, sometimes, practically impossible. It is essential therefore, that the system requirements are fully understood before hardware development or building work begins. Reworking the system design to solve hardware problems is rarely possible. For this reason, more and more system functionality is being assigned to the system software. This allows some changes to be made during system development, in response to new system requirements that inevitably arise.

One of the most confusing aspects of systems engineering is that companies use different terminology for each stage of the process. The process structure also varies. Sometimes, requirements engineering is part of the development process and sometimes it is a separate activity. However, there are essentially six fundamental activities in systems development:

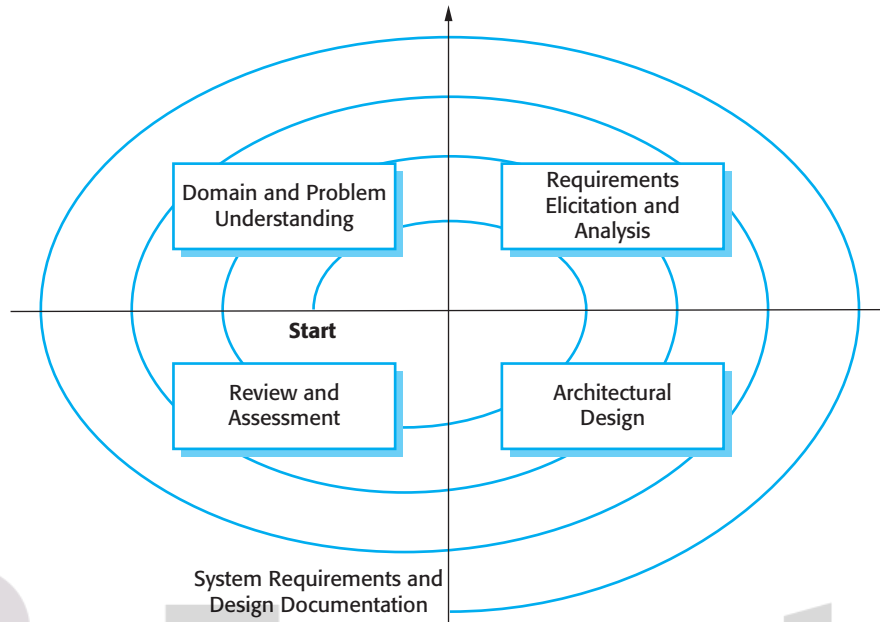
1. *Requirements development* The high-level and business requirements identified during the procurement process have to be developed in more detail. Requirements may have to be allocated to hardware, software, or processes and prioritized for implementation.

2. *System design* This process overlaps significantly with the requirements development process. It involves establishing the overall architecture of the system, identifying the different system components and understanding the relationships between them.
3. *Subsystem engineering* This stage involves developing the software components of the system; configuring off-the-shelf hardware and software, designing, if necessary, special-purpose hardware; defining the operational processes for the system; and redesigning essential business processes.
4. *System integration* During this stage, the components are put together to create a new system. Only then do the emergent system properties become apparent.
5. *System testing* This is usually an extensive, prolonged activity where problems are discovered. The subsystem engineering and system integration phases are reentered to repair these problems, tune the performance of the system, and implement new requirements. System testing may involve both testing by the system developer and acceptance/user testing by the organization that has procured the system.
6. *System deployment* This is the process of making the system available to its users, transferring data from existing systems, and establishing communications with other systems in the environment. The process culminates with a 'go live' after which users start to use the system to support their work.

Although the overall process is plan driven, the processes of requirements development and system design are inextricably linked. The requirements and the high-level design are developed concurrently. Constraints posed by existing systems may limit design choices and these choices may be specified in the requirements. You may have to do some initial design to structure and organize the requirements engineering process. As the design process continues, you may discover problems with existing requirements and new requirements may emerge. Consequently, you can think of these linked processes as a spiral, as shown in Figure 10.8.

The spiral reflects the reality that requirements affect design decisions and vice versa, and so it makes sense to interleave these processes. Starting in the center, each round of the spiral may add detail to the requirements and the design. Some rounds may focus on requirements, and some on design. Sometimes new knowledge collected during the requirements and design process means that the problem statement itself has to be changed.

For almost all systems, there are many possible designs that meet the requirements. These cover a range of solutions that combine hardware, software, and human operations. The solution that you choose for further development may be the most appropriate technical solution that meets the requirements. However, wider organizational and political considerations may influence the choice of solution. For example, a government client may prefer to use national rather than foreign suppliers for its system, even if national products are technically inferior. These influences usually take effect in the review and assessment phase of the spiral model where



**Figure 10.8**  
Requirements and  
design spiral

designs and requirements may be accepted or rejected. The process ends when a review decides that the requirements and high-level design are sufficiently detailed for subsystems to be specified and designed.

In the subsystem engineering phase, the hardware and software components of the system are implemented. For some types of system, such as spacecraft, all hardware and software components may be designed and built during the development process. However, in most systems, some components are commercial off-the-shelf (COTS) systems. It is usually much cheaper to buy existing products than to develop special-purpose components.

Subsystems are usually developed in parallel. When problems that cut across subsystem boundaries are encountered, a system modification request must be made. Where systems involve extensive hardware engineering, making modifications after manufacturing has started is usually very expensive. Often ‘work-arounds’ that compensate for the problem must be found. These ‘work-arounds’ usually involve software changes because of the software’s inherent flexibility.

During systems integration, you take the independently developed subsystems and put them together to make up a complete system. This integration can be done using a ‘big-bang’ approach, where all the subsystems are integrated at the same time. However, for technical and managerial reasons, an incremental integration process where subsystems are integrated one at a time is the best approach:

1. It is usually impossible to schedule the development of the subsystems so that they are all finished at the same time.
2. Incremental integration reduces the cost of error location. If many subsystems are simultaneously integrated, an error that arises during testing may be in any of

these subsystems. When a single subsystem is integrated with an already working system, errors that occur are probably in the newly integrated subsystem or in the interactions between the existing subsystems and the new subsystem.

As more and more systems are built by integrating COTS hardware and software components, the distinction between implementation and integration is increasingly blurred. In some cases, there is no need to develop new hardware or software and the integration is, essentially, the implementation phase of the system.

During and after the integration process, the system is tested. This testing should focus on testing the interfaces between components and the behavior of the system as a whole. Inevitably, this will also reveal problems with individual subsystems that have to be repaired.

Subsystem faults that are a consequence of invalid assumptions about other subsystems are often revealed during system integration. This may lead to disputes between the contractors responsible for implementing different subsystems. When problems are discovered in subsystem interaction, the contractors may argue about which subsystem is faulty. Negotiations on how to solve the problems can take weeks or months.

The final stage of the system development process is system delivery and deployment. The software is installed on the hardware and is readied for operation. This may involve more system configuration to reflect the local environment where it is used, the transfer of data from existing systems, and the preparation of user documentation and training. At this stage, you may also have to reconfigure other systems in the environment to ensure that the new system interoperates with them.

Although straightforward in principle, many difficulties can arise during deployment. The user environment may be different from that anticipated by the system developers and adapting the system to cope with diverse user environments can be difficult. The existing data may require extensive cleanup and parts of it may be missing. The interfaces to other systems may not be properly documented.

The influence of system development processes on dependability and security is obvious. It is during these processes that decisions are made on dependability and security requirements and on trade-offs between costs, schedule, performance, and dependability. Human errors at all stages of the development process may lead to the introduction of faults into the system which, in operation, can lead to system failure. Testing and validation processes are inevitably constrained by the costs and time available. As a result, the system may not be properly tested. Users are left to test the system as it is being used. Finally, problems in system deployment may mean that there is a mismatch between the system and its operational environment. These can lead to human errors when using the system.



## 10.5 System operation

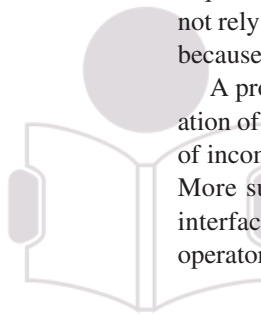
Operational processes are the processes that are involved in using the system for its defined purpose. For example, operators of an air traffic control system follow specific processes when aircraft enter and leave airspace, when they have to change

height or speed, when an emergency occurs, and so on. For new systems, these operational processes have to be defined and documented during the system development process. Operators may have to be trained and other work processes adapted to make effective use of the new system. Undetected problems may arise at this stage because the system specification may contain errors or omissions. Although the system may perform to specification, its functions may not meet the real operational needs. Consequently, the operators may not use the system as its designers intended.

The key benefit of having system operators is that people have a unique capability of being able to respond effectively to unexpected situations, even when they have never had direct experience of these situations. Therefore, when things go wrong, the operators can often recover the situation although this may sometimes mean that the defined process is violated. Operators also use their local knowledge to adapt and improve processes. Normally, the actual operational processes are different from those anticipated by the system designers.

Consequently, you should design operational processes to be flexible and adaptable. The operational processes should not be too constraining, they should not require operations to be done in a particular order, and the system software should not rely on a specific process being followed. Operators usually improve the process because they know what does and does not work in a real situation.

A problem that may only emerge after the system goes into operation is the operation of the new system alongside existing systems. There may be physical problems of incompatibility or it may be difficult to transfer data from one system to another. More subtle problems might arise because different systems have different user interfaces. Introducing a new system may increase the operator error rate, as the operators use user interface commands for the wrong system.



### 10.5.1 Human error

I suggested earlier in the chapter that non-determinism was an important issue in sociotechnical systems and that one reason for this is that the people in the system do not always behave in the same way. Sometimes they make mistakes in using the system and this has the potential to cause system failure. For example, an operator may forget to record that some action has been taken so that another operator (erroneously) repeats that action. If the action is to debit or credit a bank account, say, then a system failure occurs as the amount in the account is then incorrect.

As Reason discusses (2000) human errors will always occur and there are two ways to view the problem of human error:

1. *The person approach.* Errors are considered to be the responsibility of the individual and 'unsafe acts' (such as an operator failing to engage a safety barrier) are a consequence of individual carelessness or reckless behavior. People who adopt this approach believe that human errors can be reduced by threats of disciplinary action, more stringent procedures, retraining, etc. Their view is that the error is the fault of the individual responsible for making the mistake.



2. *The systems approach.* The basic assumption is that people are fallible and will make mistakes. The errors that people make are often a consequence of system design decisions that lead to erroneous ways of working, or of organizational factors, which affect the system operators. Good systems should recognize the possibility of human error and include barriers and safeguards that detect human errors and allow the system to recover before failure occurs. When a failure does occur, the issue is not finding an individual to blame but to understand how and why the system defences did not trap the error.

I believe that the systems approach is the right one and that systems engineers should assume that human errors will occur during system operation. Therefore, to improve the security and dependability of a system, designers have to think about the defenses and barriers to human error that should be included in a system. They should also think about whether these barriers should be built into the technical components of the system. If not, they could be part of the processes and procedures for using the system or could be operator guidelines that are reliant on human checking and judgment.

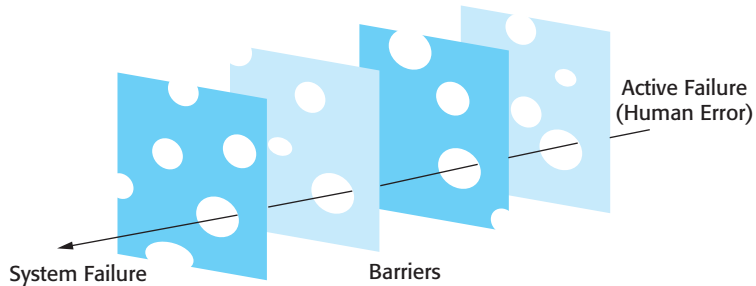
Examples of defenses that may be included in a system are:

1. An air traffic control system may include an automated conflict alert system. When a controller instructs an aircraft to change its speed or altitude, the system extrapolates its trajectory to see if it could intersect with any other aircraft. If so, it sounds an alarm.
2. The same system may have a clearly defined procedure to record the control instructions that have been issued. These procedures help the controller check if they have issued the instruction correctly and make the information available to others for checking.
3. Air traffic control usually involves a team of controllers who constantly monitor each other's work. Therefore, when a mistake is made, it is likely that it will be detected and corrected before an incident occurs.

Inevitably, all barriers have weaknesses of some kind. Reason calls these 'latent conditions' as they usually only contribute to system failure when some other problem occurs. For example, in the above defenses, a weakness of a conflict alert system is that it may lead to many false alarms. Controllers may therefore ignore warnings from the system. A weakness of a procedural system may be that unusual but essential information can't be easily recorded. Human checking may fail when all of the people involved are under stress and make the same mistake.

Latent conditions lead to system failure when the defenses built into the system do not trap an active failure by a system operator. The human error is a trigger for the failure but should not be considered to be the sole cause of the failure. Reason explains this using his well-known 'Swiss cheese' model of system failure (Figure 10.9).

**Figure 10.9**  
Reason's Swiss cheese model of system failure



In this model, the defenses built into a system are compared to slices of Swiss cheese. Some types of Swiss cheese, such as Emmental, have holes and so the analogy is that the latent conditions are comparable to the holes in cheese slices. The position of these holes is not static but changes depending on the state of the overall sociotechnical system. If each slice represents a barrier, failures can occur when the holes line up at the same time as a human operational error. An active failure of system operation gets through the holes and leads to an overall system failure.

Normally, of course, the holes should not be aligned so operational failures are trapped by the system. To reduce the probability that system failure will result from human error, designers should:

1. Design a system so that different types of barriers are included. This means that the 'holes' will probably be in different places and so there is less chance of the holes lining up and failing to trap an error.
2. Minimize the number of latent conditions in a system. Effectively, this means reducing the number and size of system 'holes'.

Of course, the design of the system as a whole should also attempt to avoid the active failures that can trigger a system failure. This may involve designing the operational processes and the system to ensure that operators are not overworked, distracted, or presented with excessive amounts of information.

### 10.5.2 System evolution

Large, complex systems have a very long lifetime. During their life, they are changed to correct errors in the original system requirements and to implement new requirements that have emerged. The system's computers are likely to be replaced with new, faster machines. The organization that uses the system may reorganize itself and hence use the system in a different way. The external environment of the system may change, forcing changes to the system. Hence evolution, where the system changes to accommodate environmental change, is a process that runs alongside normal system operational processes. System evolution involves reentering the development process to make changes and extensions to the system's hardware, software, and operational processes.



### Legacy systems

Legacy systems are sociotechnical computer-based systems that have been developed in the past, often using older or obsolete technology. These systems include not only hardware and software but also legacy processes and procedures—old ways of doing things that are difficult to change because they rely on legacy software. Changes to one part of the system inevitably involve changes to other components. Legacy systems are often business-critical systems. They are maintained because it is too risky to replace them.

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

System evolution, like software evolution (discussed in Chapter 9), is inherently costly for several reasons:

1. Proposed changes have to be analyzed very carefully from a business and a technical perspective. Changes have to contribute to the goals of the system and should not simply be technically motivated.
2. Because subsystems are never completely independent, changes to one subsystem may adversely affect the performance or behavior of other subsystems. Consequent changes to these subsystems may therefore be needed.
3. The reasons for original design decisions are often unrecorded. Those responsible for the system evolution have to work out why particular design decisions were made.
4. As systems age, their structure typically becomes corrupted by change so the costs of making further changes increases.

Systems that have evolved over time are often reliant on obsolete hardware and software technology. If they have a critical role in an organization, they are known as ‘legacy systems’. These are usually systems that the organization would like to replace but don’t do so as the risks or costs of replacement cannot be justified.

From a dependability and security perspective, changes to a system are often a source of problems and vulnerabilities. If the people implementing the change are different from those who developed the system, they may be unaware that a design decision was made for dependability and security reasons. Therefore, they may change the system and lose some safeguards that were deliberately implemented when the system was built. Furthermore, as testing is so expensive, complete retesting may be impossible after every system change. Adverse side effects of changes that introduce or expose faults in other system components may not then be discovered.

## KEY POINTS

- Sociotechnical systems include computer hardware, software, and people, and are situated within an organization. They are designed to support organizational or business goals and objectives.
- Human and organizational factors such as organizational structure and politics have a significant effect on the operation of sociotechnical systems.
- The emergent properties of a system are characteristics of the system as a whole rather than of its component parts. They include properties such as performance, reliability, usability, safety, and security. The success or failure of a system is often dependent on these emergent properties.
- The fundamental systems engineering processes are system procurement, system development, and system operation.
- System procurement covers all of the activities involved in deciding what system to buy and who should supply that system. High-level requirements are developed as part of the procurement process.
- System development includes requirements specification, design, construction, integration, and testing. System integration, where subsystems from more than one supplier must be made to work together, is particularly critical.
- When a system is put into use, the operational processes and the system itself have to change to reflect changing business requirements.
- Human errors are inevitable and systems should include barriers to detect these errors before they lead to system failure. Reason's Swiss cheese model explains how human error plus latent defects in the barriers can lead to system failure.

## FURTHER READING

'Airport 95: Automated baggage system'. An excellent, readable case study of what can go wrong with a systems engineering project and how software tends to get the blame for wider systems failures. (*ACM Software Engineering Notes*, 21, March 1996.)

<http://doi.acm.org/10.1145/227531.227544>.

'Software system engineering: A tutorial'. A good general overview of systems engineering, although Thayer focuses exclusively on computer-based systems and does not discuss sociotechnical issues. (R. H. Thayer. *IEEE Computer*, April 2002.)

<http://dx.doi.org/10.1109/MC.2002.993773>.

*Trust in Technology: A Socio-technical Perspective*. This book is a set of papers that are all concerned, in some way, with the dependability of sociotechnical systems. (K. Clarke, G. Hardstone, M. Rouncefield and I. Sommerville (eds.), Springer, 2006.)

'Fundamentals of Systems Engineering'. This is the introductory chapter in NASA's systems engineering handbook. It presents an overview of the systems engineering process for space

systems. Although these are mostly technical systems, there are sociotechnical issues to be considered. Dependability is obviously critically important. (In *NASA Systems Engineering Handbook*, NASA-SP2007-6105, 2007.) <http://education.ksc.nasa.gov/esmdspacegrant/Documents/NASA%20SP-2007-6105%20Rev%201%20Final%2031Dec2007.pdf>.

## EXERCISES

- 10.1. Give two examples of government functions that are supported by complex sociotechnical systems and explain why, in the foreseeable future, these functions cannot be completely automated.
- 10.2. Explain why the environment in which a computer-based system is installed may have unanticipated effects on the system that lead to system failure. Illustrate your answer with a different example from that used in this chapter.
- 10.3. Why is it impossible to infer the emergent properties of a complex system from the properties of the system components?
- 10.4. Why is it sometimes difficult to decide whether or not there has been a failure in a sociotechnical system? Illustrate your answer by using examples from the MHC-PMS that has been discussed in earlier chapters.
- 10.5. What is a ‘wicked problem’? Explain why the development of a national medical records system should be considered a ‘wicked problem’.
- 10.6. A multimedia virtual museum system offering virtual experiences of ancient Greece is to be developed for a consortium of European museums. The system should provide users with the facility to view 3-D models of ancient Greece through a standard web browser and should also support an immersive virtual reality experience. What political and organizational difficulties might arise when the system is installed in the museums that make up the consortium?
- 10.7. Why is system integration a particularly critical part of the systems development process? Suggest three sociotechnical issues that may cause difficulties in the system integration process.
- 10.8. Explain why legacy systems may be critical to the operation of a business.
- 10.9. What are the arguments for and against considering system engineering as a profession in its own right, like electrical engineering or software engineering?
- 10.10. You are an engineer involved in the development of a financial system. During installation, you discover that this system will make a significant number of people redundant. The people in the environment deny you access to essential information to complete the system installation. To what extent should you, as a systems engineer, become involved in this situation? Is it your professional responsibility to complete the installation as contracted? Should you simply abandon the work until the procuring organization has sorted out the problem?