2 Computer-based system engineering

Objectives

The objective of this chapter is to introduce the concept of computer-based system engineering and to explain why knowledge of system engineering is important for software engineers. When you have read this chapter, you will:

- know why the software in a system is affected by broader system engineering issues;
- have been introduced to the concept of emergent system properties such as reliability, performance, safety and security;
- understand why the system's environment must be considered during the system design process;
- understand system engineering and system procurement processes.

Contents

System engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining systems *as a whole*. Systems engineers are not just concerned with software but with software, hardware and the system 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 interactions of the system with its environment. Software engineers need an understanding of system engineering because problems of software engineering are often a result of system engineering decisions.

There are many possible definitions of a system from the very abstract to the concrete but a useful working definition is:

A system is a purposeful collection of interrelated components that work together to achieve some objective.

This general definition embraces a vast range of systems. For example, a very simple system such as a pen may only include three or four hardware components. By contrast, an air traffic control system is made up of thousands of hardware and software components plus human users who make decisions based on system information.

A characteristic of systems is that the properties and the behaviour of the system components are inextricably intermingled. The successful functioning of each system component depends on the functioning of some 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.

Systems are often hierarchical in that they 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 other systems are called *sub-systems*. A characteristic of sub-systems is that they can operate as independent systems in their own right. Therefore, the same geographical information system may be used in different systems. Its behaviour in a particular system, however, depends on its relationships with other sub-systems.

The complex relationships between the components in a system mean that the 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 emerge only when the system as a whole is considered. Some of these properties can be derived directly from the comparable properties of sub-systems but, more often, they result from complex sub-system interrelationships which cannot, in practice, be understood by analysing the individual system components.

Some examples of these emergent properties are:

1. *The overall weight of the system* This is an example of an emergent property that can be computed from individual component properties.

2. *The reliability of the system* This depends on the reliability of system components and the relationships between the components.
3. *The usability of a system* This is a very complex property which is not simply dependent on the system hardware and software but also depends on the system operators and the environment where it is used.

In this book, I am concerned with computer-based systems which include hardware and software and which offer an interface, implemented in software, to human users. Software engineers should have some knowledge of computer-based system engineering (CBSE) (White *et al.*, 1993) because of the importance of software in these systems. For example, there were less than 10 megabytes of software in the US Apollo space programme to put a man on the moon in 1969 but there are about 100 megabytes of software in the US space station programme. Software engineering is therefore critical for the successful development of complex computer-based systems.

2.1 Emergent system properties

As discussed in the introduction to this chapter, the emergent properties of a system are attributes of the system as a whole. It is often difficult to predict the values of these emergent properties in advance. They can only be measured once the subsystems have been integrated to form the complete system.

There are two types of emergent properties:

1. Functional properties that appear when all the parts of a system work together to achieve some objective. 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 such as reliability, performance, safety and security. These relate to the behaviour of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable. Some system functions may not be needed by all users 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.

To illustrate the complexity of emergent properties, consider system reliability. Reliability is a complex concept which must always be considered at the system rather than the individual component level. The components in a system are interdependent, so failures in one component can be propagated through the system and

affect the operation of other components. System designers often cannot anticipate how the consequences of failures propagate through the system, so cannot make reliable estimates of reliability from data about the reliability of system components.

There are three closely related influences on the overall reliability of a system:

1. *Hardware reliability* What is the probability of a hardware component failing and how long does it take to repair that component?
2. *Software reliability* How likely is it that a software component will produce an incorrect output? Software failure is usually distinct from hardware failure in that software does not wear out. It can continue in operation even after an incorrect result has been produced. Software reliability is discussed in Chapters 16 and 17.
3. *Operator reliability* How likely is it that the operator of a system will make an error?

All of these are closely linked. Hardware failure can cause spurious signals to be generated which are outside the range of inputs expected by software. The software can then behave unpredictably. Operator error is most likely in conditions of stress. These conditions arise when system failures are occurring. Operator errors may further stress the hardware, causing more failures and so on. Therefore, a situation can occur where a single sub-system failure that is recoverable can rapidly develop into a serious problem requiring a complete system shutdown.

Observed reliability depends on the context in which the system is used. As discussed already, the system environment cannot be specified in advance nor can the system designers place restrictions on that environment for operational systems. Different systems in an environment may react to problems in unpredictable ways, thus affecting the reliability of all of these systems. Therefore, even when the system has been integrated, it may be difficult to make accurate measurements of its reliability.

For example, say a system is designed to operate at normal room temperatures. To allow for variations and exceptional conditions, the electronic components of a system should be designed to operate within a certain range of temperature, say from 0 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 these components and hence the whole system may then fail.

If this system had been installed elsewhere in that environment there would have been no problems. When the air conditioner worked normally there were no problems. However, because of the physical closeness of these machines, an unanticipated relationship existed between them that caused system failure.

Like reliability, other emergent properties such as performance or usability are difficult to assess but can be measured after the system is operational. Properties such as safety and security, however, pose different problems. Here, you are not simply concerned with an attribute that is related to the overall behaviour of the

system but are concerned with behaviour that the system should *not* exhibit. A secure system is one which does not allow unauthorised access to its data but it is clearly impossible to predict all possible modes of access and explicitly forbid them. Therefore, it may only be possible to assess these properties by default. That is, you only know that a system is insecure when someone breaks into it.

2.2 Systems and their environment

Systems are not independent entities but exist in an environment. This environment affects the functioning and the performance of the system. Sometimes, the environment may be thought of as a system in its own right but, more generally, it consists of a number of other systems which interact with each other.

Figure 2.1 shows some of the systems that might be incorporated in an office building. The heating system, the power system, the lighting system, the plumbing system, the waste system and the security system are all sub-systems within the building which is itself a system. The building is located in a street that is in a town etc. The local environment of a system is the systems at the same level. The overall environment is composed of the local environment plus the environment of the containing system.

Consider the security system shown in the bottom left corner of Figure 2.1. The local environment of that security system is the other systems within the building. The overall environment includes all other systems outside the building in the street and the town as well as natural systems such as the weather system.

There are two main reasons why the environment of a system must be understood by systems engineers:

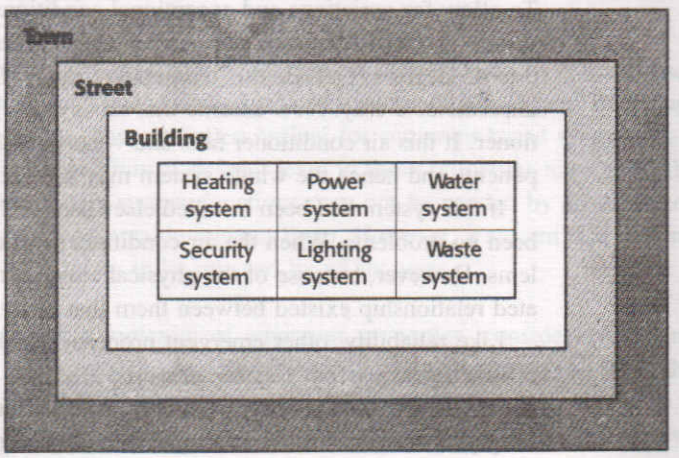

Figure 2.1 System hierarchies

1. In many cases, the system is intended to make some changes in its environment. Therefore, a heating system changes its environment by increasing or decreasing its temperature. The correct functioning of the system can therefore only be assessed by the effects on the environment.
2. The functioning of a system can be affected by changes in its environment in ways which can be difficult to predict. For example, the electrical system in a building may be affected by environmental changes outside the building. Works in the street outside may cut a power cable and the electrical system is thus disabled. More subtly, a lightning storm can induce currents in the electrical system which affect its normal functioning.

As well as the physical environment shown in Figure 2.1, systems are also situated in an organisational environment. This includes policies and procedures that are themselves governed by wider political, economic, social and environmental issues. If the organisational environment is not properly understood, systems may not meet business needs and may be rejected by users and organisational managers.

Human and organisational factors deriving from the system's environment that affect the system design include:

1. *Process changes* Does the system 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 system will be resisted by the users.
2. *Job changes* Does the system de-skill the users in an environment or cause them to change the way they work? If so, they may actively resist the introduction of the system into the organisation. Designs that involve managers having to change their way of working to fit the computer system are often resented. The managers may feel that their status in the organisation is being reduced by the system.
3. *Organisational changes* Does the system change the political power structure in an organisation? For example, if an organisation is dependent on a complex system, those who know how to operate the system have a great deal of political power.

These human, social and organisational factors are often critical in determining whether or not a system successfully meets its objectives. Unfortunately, predicting their effects on systems is very difficult for engineers who have little experience of social or cultural studies. To help understand the effects of systems on organisations, various methodologies have been developed such as Mumford's sociotechnics (Mumford, 1989) and Checkland's Soft Systems Methodology (Checkland, 1981; Checkland and Scholes, 1990). There have also been extensive sociological studies of the effects of computer-based systems on work (Ackroyd *et al.*, 1992).

Ideally, all relevant environmental knowledge should be included in the system specification so that it may be taken into account by the system designers. In reality, this is impossible. System designers must make environmental assumptions based on other comparable systems and on common sense. If they get these wrong, the system may malfunction in unpredictable ways. For example, if the designers of a system do not understand the notion of electromagnetic compatibility, then the system may malfunction when it is installed alongside other systems which emit electromagnetic radiation.

2.3 System modelling

As part of the system requirements and design activity, the system has to be modelled as a set of components and relationships between these components. These are normally illustrated graphically in a system architecture model that gives the reader an overview of the system organisation.

The system architecture is usually depicted as a block diagram showing the major sub-systems and the interconnections between these sub-systems. Each sub-system is represented by a rectangle in the block diagram and the existence of relationships between sub-systems is indicated by arrows joining these rectangles. The relationships indicated may include data flow, a 'uses'/'used by' relationship or some other type of dependency relationship.

This is illustrated in Figure 2.2 which shows the decomposition of an intruder alarm system into its principal components. The block diagram should be supplemented by brief descriptions of each sub-system as shown in Figure 2.3.

At this level of detail, the system is decomposed into sub-systems. Each sub-system can be represented in a similar way until the system is decomposed into

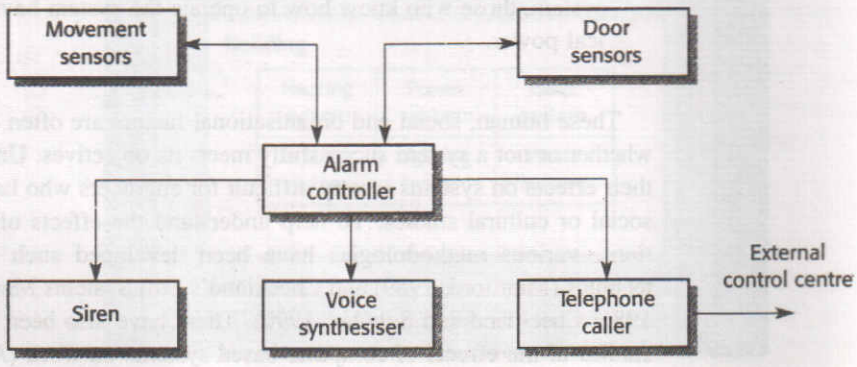

Figure 2.2 A simple intruder alarm system

Figure 2.3 Sub-system functionality in the intruder alarm system

Subsystem	Description
Movement sensors	Detects movement in the rooms monitored by the system
Door sensors	Detects door opening in the external doors of the building
Alarm controller	Controls the operation of the system
Siren	Emits an audible warning when an intruder is suspected
Voice synthesiser	Synthesises a voice message giving the location of the suspected intruder
Telephone caller	Makes external calls to notify security, the police, etc.

functional components. Functional components are components that, when viewed from the perspective of the sub-system, provide a single function. By contrast, a sub-system usually is multi-functional. Of course, when viewed from another perspective (say that of the component manufacturer), a functional component may itself be a system in its own right.

Historically, the system architecture model was used to identify hardware and software components which could be developed in parallel. However, this hardware/software distinction is becoming irrelevant. Almost all components now include some embedded computing capabilities. For example, a network linking machines will consist of physical cables plus repeaters and network gateways. The repeaters and the gateways include processors and software to drive these processors as well as specialised electronic components.

At the architectural level, it is now more appropriate to classify sub-systems according to their function before making decisions about hardware/software trade-offs. The decision whether a function should be provided in hardware or software may be governed by non-technical factors such as the availability of COTS components or the time available to develop the component.

Block diagrams may be used for all sizes of system. Figure 2.4 illustrates the architecture of a much larger system for air traffic control. There are several major sub-systems which are themselves large systems. Information flow between these sub-systems is shown by the arrowed lines connecting these systems.

2.3.1 Functional system components

As discussed in the previous section, a system architecture should be designed in terms of functional sub-systems without regard to whether these are hardware or software sub-systems. Rather, the functional components in a system may be classified under a number of different headings:

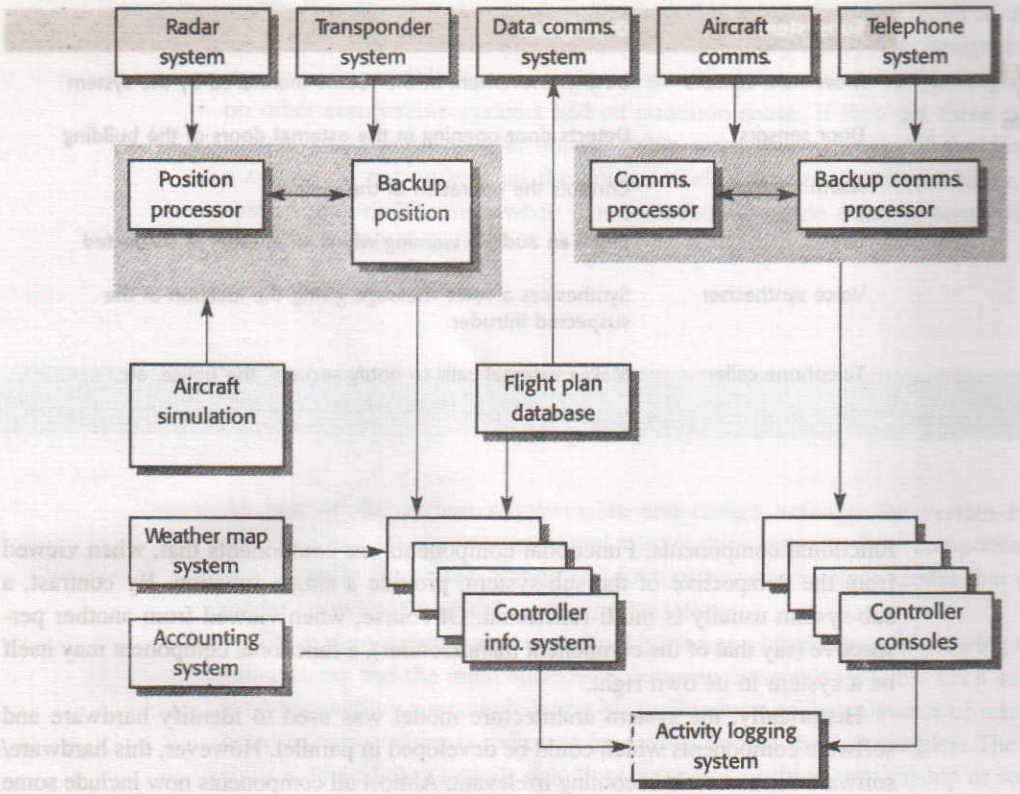

Figure 2.4 The architecture of an air traffic control system

1. *Sensor components* collect information from the system's environment. Examples of sensor components are radars in an air traffic control system, paper position sensors in a laser printer and a thermocouple in a furnace.
2. *Actuator components* cause some change in the systems environment. Examples of actuators are valves which open and close to increase or decrease the flow rate of liquid in a pipe, the flight surfaces on an aircraft which control the angle of flight and the paper feed mechanism on a laser printer which moves the paper across the scanning beam.
3. *Computation components* are components which, given some input, carry out some computations on that input and produce some output. An example of a computation component is a floating-point processor which carries out computations on real numbers.
4. *Communication components* are components whose function is to allow other system components to communicate with each other. An example of a communication component is an Ethernet linking different computers in a building.
5. *Coordination components* are system components whose function is to coordinate the operation of other components. An example of a coordination

Figure 2.5 Component types in the intruder alarm system

Component type	Components	Function
Sensor	Movement sensor, Door sensor	Detect movement in a protected space, detect protected door opening
Actuator	Siren	Audible warning of intrusion
Communication	Telephone caller	Call external control centre to issue warning of intrusion. Receive commands from control centre
Coordination	Alarm controller	Coordinate all system components. Act on commands from control panel and control centre
Interface	Voice synthesiser	Synthesise message giving location of intrusion

component is a scheduler in a real-time system. This decides when the different processes should be scheduled to run on a processor.

6. *Interface components* are components that transform the representation used by one system component into the representation used by another component. An example is a human interface component that takes some system model and displays it for the human operator. Another example is an analog–digital converter that converts an analog input into a digital output.

Figure 2.5 explains which of these different functional component types are used in the alarm system architecture illustrated in Figure 2.2.

Of course, there is no hard and fast division between the different component types. In the types of systems in which software engineers are likely to be involved, most components will include embedded software. Software will usually be used to control the overall system.

These component classifications are helpful when designing a system. Most systems include all of these types of component and you should explicitly identify each type of component from the requirements. If one or more component types is missing, this suggests that you may have left something out of your design.

2.4 The system engineering process

The phases of the system engineering process are shown in Figure 2.6. This process was an important influence on the 'waterfall' model of the software process which we discuss in Chapter 3.

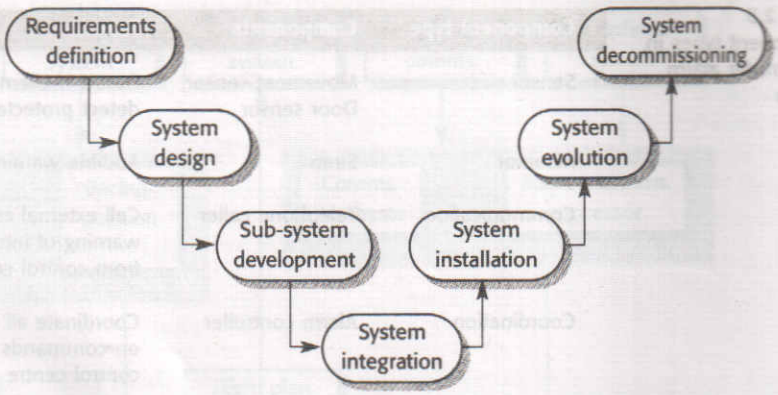

Figure 2.6 The system engineering process

There are important distinctions between the system engineering process and the software development process:

1. *Interdisciplinary involvement* Many different engineering disciplines may be involved in system engineering. There is immense scope for misunderstanding because of the different terminology used by different engineers.
2. *Reduced scope for rework during system development* Once some system engineering decisions, such as the siting of radars in an ATC system, have been made they are very expensive to change. Reworking the system design to solve these problems is rarely possible. One reason why software has become so important in systems is that it allows for flexibility as changes can be made during the system development in response to new requirements.

System engineering is an interdisciplinary activity involving teams drawn from different backgrounds. System engineering teams are needed because of the wide knowledge required to consider all the implications of system design decisions. Consider an air traffic control (ATC) system that uses radars and other sensors to determine aircraft position (see Figure 2.4). Figure 2.7 shows some of the different disciplines that may be involved in the system engineering team.

For many systems, there are almost infinite possibilities for trade-offs between different types of sub-system. Different engineering disciplines must negotiate to decide how functionality should be provided. Often there is no 'correct' decision on how a system should be decomposed. Rather, there is a range of possible alternatives. The selection of one of these alternatives need not necessarily be made for technical reasons. Say one alternative in an air traffic control system is to build new radars rather than refit existing installations. If the civil engineers involved in this process do not have much other work, they may favour this alternative because it allows them to keep their jobs. They may then rationalise this choice using technical arguments.

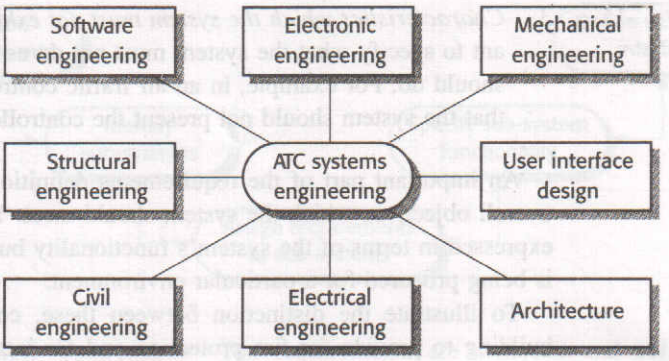

Figure 2.7 Interdisciplinary involvement in system engineering

Because software is inherently flexible, many unexpected problems are left to software engineers to solve. Say the site of a radar is such that some image ghosting occurs. It is impractical to move the radar so some other way of removing this ghosting is required. The solution may be to enhance the image processing capabilities of the software to remove the ghost images. This may then require increased processor power in the system which may be difficult to provide.

Software engineers are often left with the problem of enhancing the software capabilities without increasing the hardware cost. Many so-called 'software failures' were not a consequence of inherent software problems. They were 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).

2.4.1 System requirements definition

The system requirements definition activity is intended to discover the requirements for the system as a whole. As with software requirements analysis, the process involves consultations with system customers and end-users. This requirements definition phase usually concentrates on deriving three types of requirement:

1. *Abstract functional requirements* The basic functions that the system must provide are defined at an abstract level. Detailed functional requirements specification takes place at the sub-system level. For example, in the air traffic control system, this requirements activity would probably identify the need for a flight-plan database to store the flight plans of all aircraft entering the controlled airspace. However, the details of the database would not be specified unless they affected the requirements of other sub-systems.
2. *System properties* These are non-functional emergent system properties as discussed above. These may include properties such as availability, performance, safety, etc. These non-functional system properties affect the requirements for all sub-systems.

3. *Characteristics which the system must not exhibit* It is sometimes as important to specify what the system must not do as it is to specify what the system should do. For example, in an air traffic control system, it might be specified that the system should not present the controller with too much information.

An important part of the requirements definition phase is to establish a set of overall objectives which the system should meet. These should not necessarily be expressed in terms of the system's functionality but should define why the system is being procured for a particular environment.

To illustrate the distinction between these, consider a system for an office building to provide for fire protection and for intruder detection. A statement of objectives which is based around the system functionality might be:

To provide a fire and intruder alarm system for the building that will provide internal and external warning of fire or unauthorised intrusion.

This objective states explicitly that there needs to be an alarm system which provides some warnings of undesired events. Such a statement might be appropriate if there were already an existing alarm system which was to be replaced. By contrast, a broader statement of objectives might be:

To ensure that the normal functioning of the work carried out in the building is not seriously disrupted by events such as fire and unauthorised intrusion.

Stating the objective in this way both broadens and limits some design choices. It allows for intruder protection using sophisticated locking technology without any internal alarms. It may exclude the use of sprinklers for fire protection. These may affect the electrical systems in the building and seriously disrupt work which is going on.

A fundamental difficulty in establishing system requirements is that the problems which complex systems are usually built to help tackle are usually 'wicked problems' (Rittel and Webber, 1973). A 'wicked problem' is a problem which is so complex and where there are so many related entities that there is no definitive problem specification. The true nature of the problem only emerges as a solution is developed. An extreme example of a 'wicked problem' is earthquake planning. No one can accurately predict where the epicentre of an earthquake will be, what time it will occur, what effect it will have on the local environment, etc. We cannot therefore completely specify how to deal with a major earthquake. The problem can only be tackled after it has happened.

2.4.2 System design

System design (Figure 2.8) is concerned with how the system functionality is to be provided by the different components of the system. The activities involved in this process are:

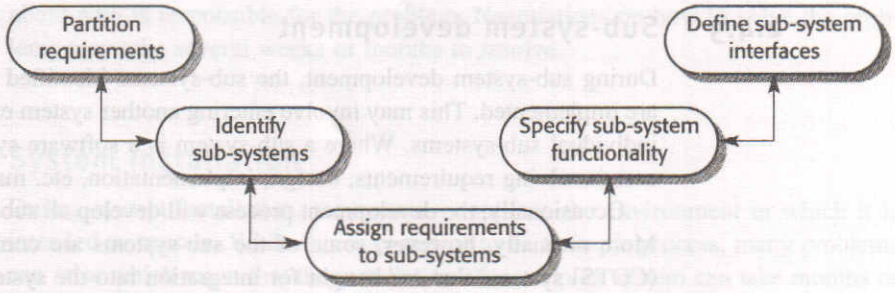

Figure 2.8 The system design process

1. *Partition requirements* The requirements are analysed and collected into related groups. There are usually several possible partitioning options and a number of alternatives may be produced at this stage of the process.
2. *Identify sub-systems* Different sub-systems that can individually or collectively meet the requirements are identified. Groups of requirements are usually related to sub-systems, so this activity and requirements partitioning may be amalgamated. However, the sub-system identification may also be influenced by other organisational or environmental factors.
3. *Assign requirements to sub-systems* The requirements are assigned to sub-systems. In principle, this should be straightforward if the requirements partitioning is used to drive the sub-system identification. In practice, there is never a clean match between requirements partitions and identified sub-systems. Limitations of externally purchased sub-systems (COTS, see section 2.4.3) may mean that requirements have to be modified.
4. *Specify sub-system functionality* The specific functions provided by each sub-system are specified. This may be seen as part of the system design phase or, if the sub-system is a software system, part of the requirements specification activity for that system. Relationships between sub-systems should also be identified at this stage.
5. *Define sub-system interfaces* This involves defining the interfaces that are provided and required by each sub-system. Once these interfaces have been agreed, parallel development of the sub-systems becomes possible.

As the double-ended arrows in Figure 2.8 imply, there is a great deal of feedback and iteration from one stage to another in this design process. As problems and questions arise, rework of earlier stages is often necessary.

For almost all systems, there are many possible designs which may be developed. These cover a range of solutions with different combinations of hardware, software and human operations. The solution chosen for further development may be the most appropriate technical solution which meets the requirements. However, in many cases wider organisational and political influences influence the choice of solution. For example, if the system is a government system, it may prefer national rather than foreign suppliers even if the national product is technically inferior.

2.4.3 Sub-system development

During sub-system development, the sub-systems identified during system design are implemented. This may involve entering another system engineering process for individual sub-systems. Where a sub-system is a software system, a software process involving requirements, design, implementation, etc. may be started.

Occasionally, the development process will develop all sub-systems from scratch. More normally, however, some of the sub-systems are commercial, off-the-shelf (COTS) systems that are bought for integration into the system. It is usually much cheaper to buy existing products rather than develop special-purpose components. At this stage, the design activity may have to be re-entered to accommodate a bought-in component. COTS systems may not meet the requirements exactly but, if off-the-shelf products are available, it is usually worth the expense of rethinking the design.

Different sub-systems are usually developed in parallel. When problems are encountered which cut across sub-system boundaries, 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' which compensate for the problem must be found. These 'work-arounds' usually involve software changes because of the software's inherent flexibility. This leads to changes in the software requirements so, as I have discussed in Chapter 1, it is important to design software for change.

2.4.4 System integration

System integration involves taking independently developed sub-systems and putting them together to make up a complete system. Integration can be done using a 'big bang' approach where all the sub-systems are integrated at the same time. However, for both technical and managerial reasons, an incremental integration process where sub-systems are integrated one at a time is the best approach to adopt.

This incremental process is the most appropriate approach for two reasons:

1. It is usually impossible to schedule all the different sub-system developments so that all development is completed at the same time.
2. Incremental integration reduces the cost of error location. If many sub-systems are simultaneously integrated, an error that arises during testing may be located in any of these sub-systems. When a single sub-system is integrated with an already working system, errors which occur are probably in the newly integrated sub-system or in the interactions between the existing sub-systems and the new sub-system.

Sub-system faults that are a consequence of invalid assumptions about other sub-systems are often revealed during system integration. These may lead to disputes between the various contractors responsible for the different sub-systems. When problems are discovered in sub-system interaction, the different contractors may argue

about who is responsible for the problem. Negotiations on how to solve the problems may take several weeks or months to resolve.

2.4.5 System installation

During system installation, the system is put into the environment in which it is intended to operate. While this may appear to be a simple process, many problems can arise which mean that the installation of a complex system can take months or even years.

Examples of these problems are:

1. The environment in which the system is to be installed is not the same as the environment assumed by the developers of the system. This is a common problem when software systems are installed. For example, the system may use functions provided by a specific version of the operating system. These may not be identical in the operating system version in the installation environment. When the system is installed, it may not work at all or may operate in a way that was not anticipated by its developers.
2. Potential users of the system may be hostile to its introduction. It may reduce their responsibility or the number of jobs in an organisation. People may therefore deliberately refuse to cooperate with the system installers. For example, they may refuse to participate in operator training or may deny access to information that is essential for system installation.
3. A new system may have to coexist with an existing system until the organisation is satisfied that the new system works properly. This causes particular installation problems if the systems are not completely independent but share some components. It may be impossible to install the new system without de-installing the old system. Trials of the new system can therefore only take place at times when the existing system is not used.
4. There may be physical installation problems. There may be difficulties fitting a new system into an existing building as there may not be enough room in existing ducts for network cables, air conditioning may be required, the furniture may not be large enough, etc. If the installation is to take place in a historic building, building modifications may be completely forbidden.

2.4.6 System operation

Once the system has been installed, it is put into operation. Operating the system may involve organising training sessions for operators and changing the normal work process to make effective use of the new system. Undetected problems may arise at this stage because the system specification may contain errors or omissions. While the system may perform to specification, its functions may not meet real operational

needs. Consequently, the mode of use of the system may not be as anticipated by the system designers.

A problem that may emerge only after the system goes into operation is the problem of operating the new system with existing systems. There may be physical problems of incompatibility. It may be difficult to transfer data from one system to another. More subtle problems might be radically different user interfaces offered by different systems. Introducing the new system may increase the operator error rate for existing systems as operators mix up user interface commands.

2.4.7 System evolution

Large and complex systems have a very long lifetime. During their life, they have to evolve to correct errors in the original system requirements and meet new requirements which have emerged. The system's computers are likely to be replaced with new, faster machines. The organisation which uses the system may reorganise itself and hence use the system in a different way. The external environment of the system may change, thus forcing changes to the system.

System evolution, like software evolution (discussed in Part 7), is inherently costly for a number of reasons:

1. Proposed changes have to be analysed very carefully both from a business and a technical perspective. They must be approved by a range of people before being put into effect.
2. Because sub-systems are never completely independent, changes to one sub-system may adversely affect the performance or behaviour of other sub-systems. Consequent changes to these sub-systems 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.

As society becomes increasingly dependent on systems of various types, the amount of effort devoted to evolution rather than new system development is increasing. These existing systems that must be retained are now sometimes called *legacy systems*. I discuss legacy systems in Chapter 26.

2.4.8 System decommissioning

System decommissioning means taking the system out of service after the end of its useful operational lifetime. Sometimes this is straightforward but some systems may contain materials which are potentially damaging to the environment. The system engineering activity should anticipate decommissioning and take the problems

of disposing of the materials into account during the design phase. For example, the use of toxic chemicals might be confined to sealed modules which can be removed as a single unit and reprocessed.

As far as software is concerned there are, of course, no physical decommissioning problems. However, some software functionality may be incorporated in a system to assist with the decommissioning process. For example, software may be used to monitor the state of other system components. When the system is decommissioned, components which are not worn can therefore be identified and reused in other systems.

If the data in the system that is being decommissioned must be retained by the organisation, it must be converted for use by some other system. This can often involve significant costs as the data structure may be implicitly defined in the software itself. I cover some of these problems of data re-engineering in Chapter 28.

2.5 System procurement

The customers for complex computer-based systems are usually large organisations such as the military, government and emergency services. The system may be bought as a whole, may be bought as separate parts which are then integrated or may be specially designed and developed. For large systems, deciding which of these options to choose can take several months or years. The process of system procurement is concerned with making decisions about the best way for an organisation to acquire a system and deciding on the best suppliers of that system.

The procurement process is closely related to the systems engineering process. Some system specification and architectural design is done before these procurement decisions are made. There are two main reasons for this:

1. To buy or let a contract to design and build a system, a high-level specification of what that system should do must be completed.
2. It is almost always cheaper to buy a system than to design, manufacture and build it as a separate project. Some architectural design is necessary to identify those sub-systems that can be bought rather than specially designed and manufactured.

Large complex systems usually consist of a mixture of off-the-shelf components (COTS) 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 a 'glue' to make these different pieces of hardware 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. I discuss the use of COTS systems in Chapter 14.

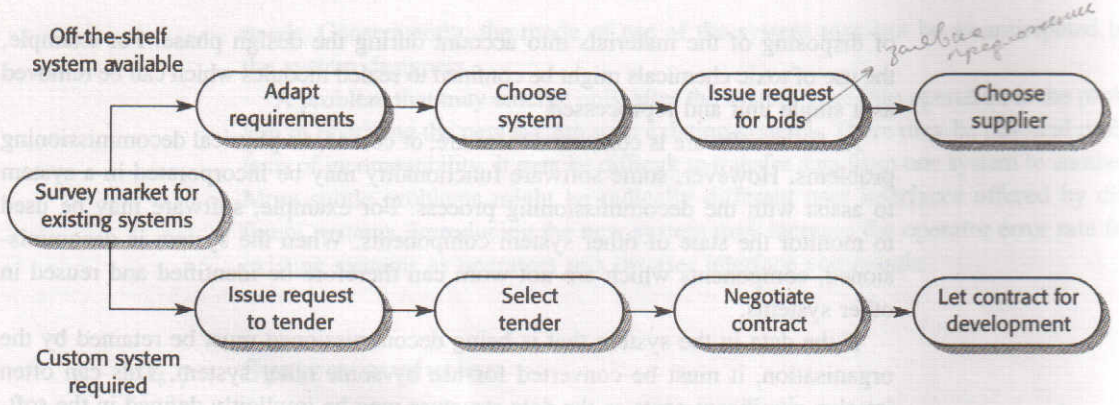

Figure 2.9 The system procurement process

Figure 2.9 shows the procurement process for both existing systems and systems which have to be specially designed. Some 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 finding the closest match between the system requirements and the facilities offered by off-the-shelf systems. The requirements then have to be modified. This can have knock-on effects on other sub-systems.
2. When a system is to be built specially, the specification of requirements acts as the basis of a contract for the system procurement. It is therefore a legal as well as a technical document.
3. After a contractor to build a system has been selected, there is a further contract negotiation period where further changes to the requirements may be agreed and issues such as the cost of change discussed.

Most hardware sub-systems and many software sub-systems such as database management systems are not developed specially when they are included in some larger system. Rather, existing sub-systems are used as they stand or are adapted for use in that system.

Very few single organisations have the capabilities to design, manufacture and test all the components of a large complex system. This supplier, who is usually called the principal contractor, may contract out the development of different sub-systems to a number of subcontractors (Figure 2.10). 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 so may include computer hardware suppliers, software developers, peripheral suppliers and suppliers of specialist equipment such as radars.

This contractor/subcontractor model minimises the number of organisations with which the procurer must deal. The subcontractors design and build parts of the system to a specification produced by the principal contractor. Once completed, these

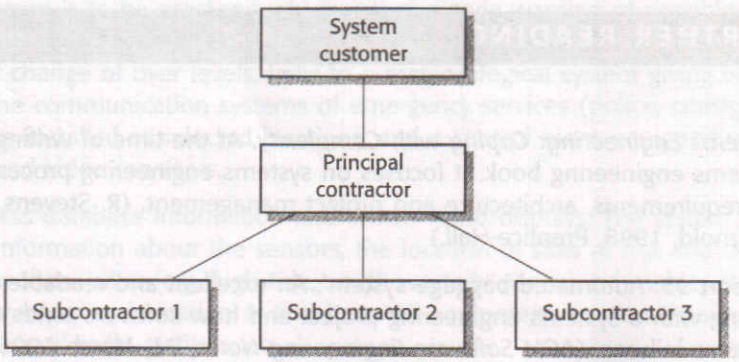

Figure 2.10
The contractor/ subcontractor model

different parts are integrated by the principal contractor. They are then delivered to the customer buying the system. 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.

KEY POINTS

- System engineering is a complex and difficult process which requires input from a range of engineering disciplines.
- The emergent properties of a system are properties that are characteristic 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.
- System architectures are usually described using block diagrams showing the major sub-systems and their relationships.
- Types of functional system component include sensor components, actuator components, computation components, coordination components, communication components and interface components.
- The system engineering process includes specification, design, development, integration and testing. System integration, where sub-systems from different suppliers must be made to work together, is particularly critical.
- The system procurement process involves specifying the system, issuing a request for proposals, choosing a supplier and then letting a contract for the system. Usually, some parts of large computer-based systems are procured as commercial off-the-shelf (COTS) components.

FURTHER READING

Systems Engineering: Coping with Complexity. At the time of writing, this is the best available systems engineering book. It focuses on systems engineering processes with good chapters on requirements, architecture and project management. (R. Stevens, P. Brook, K. Jackson and S. Arnold, 1998, Prentice-Hall.)

'Airport 95: Automated baggage system'. An excellent and 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.)

'System engineering of computer-based systems'. This paper is a good overview of computer-based system engineering and calls for the establishment of a discipline of ECBS, i.e. the Engineering of Computer-based Systems. (S. White *et al., IEEE Computer*, **26**(11), November 1993.)

Systems Engineering: Principles and Practice of Computer-based Systems Engineering. Covers various aspects of CBSE, including the development process, project management and system design methods. (B. Thomé (ed.), 1993, John Wiley and Sons.)

EXERCISES

2.1 Explain why other systems within a system's environment can have unanticipated effects on the functioning of a system.

2.2 Modify Figure 2.8 to incorporate an explicit procurement activity once the sub-systems have been identified. Show, in your diagram, the feedback that results from the incorporation of this activity.

2.3 Explain why specifying a system to be used by emergency services for disaster management is an inherently wicked problem.

2.4 Suggest how the software systems used in a car can help with the decommissioning (scrapping) of the overall system.

2.5 Explain why it is important to produce an overall description of a system architecture at an early stage in the system specification process.

2.6 Figure 2.1 shows a range of systems in a building. The security system is an extended version of the system shown in Figure 2.2 that is intended to protect against intrusion and to detect fire. It incorporates smoke sensors, movement sensors and door sensors, video cameras, under computer control, located at various places in the building, an operator console where the system status is reported, and external communication facilities to call the appropriate services such as police, fire, etc. Draw a block diagram of a possible design for such a system.

2.7 A flood warning system is to be procured which will give early warning of possible flood dangers to sites that are threatened by floods. The system will include a set of sensors to monitor the rate of change of river levels, links to a meteorological system giving weather forecasts, links to the communication systems of emergency services (police, coastguard, etc.), video monitors installed at selected locations, and a control room equipped with operator consoles and video monitors.

Controllers can access database information and switch video displays. The system database includes information about the sensors, the location of sites at risk and the threat conditions for these sites (e.g. high tide, south-westerly winds), tide tables for coastal sites, the inventory and location of flood control equipment, contact details for emergency services, local radio stations, etc.

Draw a block diagram of a possible architecture for such a system. You should identify the principal sub-systems and the links between them.

2.8 Assuming that some system which has been ordered meets its specification, describe, using associated examples, three problems which might arise when it is installed in an organisation.

2.9 What are the arguments for and against considering system engineering as a profession in its own right such as electrical engineering or software engineering?

2.10 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? Is it your professional responsibility to complete the installation as contracted? Should you simply abandon the work until the procuring organisation has sorted out the problem?