19 Systems Engineering

Objectives

The objectives of this chapter are to explain why software engineers should understand systems engineering and to introduce the most important systems engineering processes. When you have read this chapter, you will:

- know what is meant by a sociotechnical system and understand why human, social and organizational issues affect the requirements and design of software systems;
- understand the idea of conceptual design and why it is an essential first stage in the systems engineering process;
- know what is meant by system procurement and understand why different system procurement processes are used for different types of system;
- know about the key systems engineering development processes and their relationships.

Contents

19.1 Sociotechnical systems
19.2 Conceptual design
19.3 System procurement
19.4 System development
19.5 System operation and evolution
A computer only becomes useful when it includes both software and hardware. Without hardware, a software system is an abstraction – simply a representation of some human knowledge and ideas. Without software, a hardware system is a set of inert electronic devices. However, if you put them together to form a computer 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. Furthermore, systems are developed to support human activities – work, entertainment, communication, protection of people and the environment and so on. They interact with people and their design is influenced by human and organizational concerns. Hardware, human, social and organizational factors have to be taken into account when developing all professional software systems.

Systems that include software fall into two categories:

1. **Technical computer-based systems** 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. Applications for PCs, computer games, etc. are also technical systems. Individuals and organizations use technical systems for a particular purpose but knowledge of this purpose is not part of the technical system. For example, the word processor I am using (Microsoft Word) is not aware that is it being used to write a book.

2. **Sociotechnical systems** 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 (creation, editing, layout, etc.) and technical systems (Microsoft Word and Excel, Adobe Illustrator, Quark Xpress, etc.).

Systems engineering (White et al. 1993; Stevens et al. 1998; Thayer 2002) is the activity of designing entire systems, taking into account the characteristics of hardware, software and human elements of these systems. Systems engineering includes everything to do with procuring, specifying, developing, deploying, operating and maintaining both technical and socio-technical systems. Systems engineers have to consider the capabilities of hardware and software and the system’s interactions with users and its environment. They must think about the system’s services, the constraints under which the system must be built and operated and the ways in which the system is used.

In this chapter, my focus is on the engineering of large and complex software-intensive systems. These are ‘enterprise systems’ i.e. systems that are used to support the goals of a large organization. Enterprise systems are used by government and the military services as well as large companies and other public
bodies. They are sociotechnical systems that are influenced by the ways that the organization works and by national and international rules and regulations. They may be made up of a number of separate systems and are distributed systems with large-scale databases. They have a long lifetime and are critical for the operation of the enterprise.

I believe that it is important for software engineers to know about systems engineering and to be active participants in systems engineering processes for two reasons:

1. Software is now the dominant element in all enterprise systems yet many senior decision makers in organizations have a limited understanding of software. Software engineers have to play a more active part in high-level systems decision making if the system software is to be dependable and developed on time and to budget.

2. As a software engineer, it helps if you have a broader awareness of how software interacts with other hardware and software systems, and the human, social and organizational factors that affect the ways in which software is used. This knowledge helps you understand the limits of software and to design better software systems.

There are four overlapping stages (Figure 19.1) in the lifetime of large, complex systems:

1. **Conceptual design** This initial systems engineering activity develops the concept of the type of system that is required. It sets out, in non-technical language, the purpose of the system, why it is needed and the high-level features that users might expect to see in the system. It may also describe broad constraints, such as the need for interoperability with other systems. These limit the freedom of systems engineers in designing and developing the system.
2. **Procurement or acquisition** During this stage, the conceptual design is further developed so that information is available to make decisions about the contract for the system development. This may involve making decisions about the distribution of functionality across hardware, software and operational processes. You also make decisions about which hardware and software has to be acquired, which suppliers should develop the system and the terms and conditions of the supply contract.

2. **Development** During this stage, the system is developed. Development processes include 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.

Figure 19.1 shows the interactions between these stages. The conceptual design activity is a basis for the system procurement and development but is also used to provide information to users about the system. Development and procurement overlap and further procurement during development and operation may be needed as new equipment and software becomes available. Once the system is operational, requirements changes are inevitable and implementing these changes requires further development and, perhaps, software and hardware procurement.

Decisions made at any one of these stages may have a profound influence on the other stages. Design options may be restricted by procurement decisions on the scope of the system and on its hardware and software. Human errors made during the specification, design and development stages may mean that faults are introduced into the system. A decision to limit testing for budget reasons may mean that faults are not discovered before a system is put into use. During operation, errors in configuring the system for deployment may lead to problems in using the system. Decisions made during the original procurement may be forgotten when system changes are proposed. This may lead to unforeseen consequences arising from the implementation of the changes.

An important difference between systems and software engineering is the involvement of a range of professional disciplines throughout the lifetime of the system. These include engineers who may be involved in hardware and software design, system end-users, managers who are concerned with organizational issues and experts in the system’s application domain. For example, engineering the insulin pump system introduced in Chapter 1 requires experts in electronics, mechanical engineering, software and medicine.
For very large systems, an even wider range of expertise may be required. Figure 19.2 illustrates the technical disciplines that may be involved in the procurement and development of a new system for air traffic management. 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 of the different types of components in complex systems. However, differences and misunderstandings between disciplines can lead to inappropriate design decisions being made. These can delay the system’s development or make it less suitable for its intended purpose. There are three reasons why there may be misunderstandings or other differences between engineers with different backgrounds:

1. Different professional disciplines often use the same words but these do not always mean the same thing. Consequently, 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 may know a bit about C programming but may not understand that a method in Java is like a function in C.

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 possible. For example, an electronic engineer may decide that all signal processing (a computationally intensive task) should be done by software to simplify the hardware design. However, this may mean significantly greater software effort to ensure that the system processor can cope with the amount of computation that is resolved.
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.

My experience is that inter-disciplinary working can only be successful if enough time is available for these issues to be discussed and resolved. This requires regular face-to-face discussions and a flexible approach from everyone involved in the systems engineering process.

19.1 Sociotechnical 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 essential 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. I focus here on systems that include computers and software and that have some specific purpose such as to enable communication, support navigation, or maintain medical records. A useful working definition of these types of system is as follows:

A system is a purposeful collection of interrelated components of different kinds that work together to deliver a set of services to the system owner and its users.

This general definition can cover a very wide range of systems. For example, a simple system, such as laser pointer, delivers an indication service. It may include a few hardware components plus a tiny control program in ROM. 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. It delivers a range of services including providing information to pilots, maintaining safe separation of planes, airspace utilisation and so on.

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

Large-scale systems are often ‘systems of systems’. That is, they are made up of several separate systems. For example, a police command and control system may include a geographical information system to provide details of the location of incidents. The same geographical information system may be used in systems for transport logistics and emergency command and control. Engineering ‘systems of
systems’ is an increasingly important topic in software engineering that I cover in Chapter 20.

Large-scale systems are, with a few exceptions, sociotechnical systems, which I explained in Chapter 10. That is, they do not just include software and hardware but also people, processes and organizational policies. Sociotechnical systems are enterprise systems that are intended to help deliver a business purpose. 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.

The close relationships between sociotechnical systems and the organizations that use these systems means that it is often difficult to establish system boundaries. Different people within the organization will see the boundaries of the system in different ways. This is significant because establishing what is and what is not in the scope of the system is important when defining the system requirements.

Figure 19.3 illustrates this problem. The diagram shows a sociotechnical system as a set of layers, where each layer contributes, in some way, to the functioning of the system. At the core is a software-intensive technical system and its operational processes (shaded in Figure 19.3). Most people would agree that these are both parts of the system. However, the system’s behaviour is influenced by a range of sociotechnical factors outside of the core. Should the system boundary simply be drawn around the core or should it include other organizational levels?

Whether or not these broader sociotechnical considerations should be considered to be part of a system depends on the organization and its policies and rules. If organizational rules and policies can be changed, then some people might
argue they should be part of the system. However, it is more difficult to change organizational culture and even more challenging to change strategy and goals. Only governments can change laws to accommodate a system. Moreover, different stakeholders may have different opinions on where the system boundaries should be drawn. There are no simple answers to these questions but they have to be discussed and negotiated during the system design process.

Generally, large sociotechnical systems are used in organizations. When you are designing and developing sociotechnical systems, you need to understand, as far as possible, the organizational environment in which they will be used. If you don’t, the systems may not meet business needs. Users and their managers may reject the system or fail to use it to its full potential.

Figure 19.4 shows the key elements in an organization that may affect the requirements, design and operation of a sociotechnical system. A new system is likely may lead to changes in some or all of these elements:

1. **Process changes** A new system may mean that people have to change the way that they work. 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 deskill 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. Professional staff, such as doctors or teachers may resist system designs that require them to change their normal way of working. The people involved may feel that their professional expertise is being eroded and that their status in the organization is being reduced by the system.

3. **Organizational policies** The proposed system may not be completely consistent with organizational policies (e.g. on privacy). This may require system changes, policy changes or process changes to bring the system and policies into line.

4. **Organizational politics** 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. Alternatively, if an organization reorganizes itself into a different structure, this may affect the requirements and use of the system.

Sociotechnical systems are complex systems, which means that it is practically impossible to have a complete understanding, in advance, of their behaviour. This complexity leads to three important characteristics of sociotechnical systems:

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. Some of these relationships only come into existence when the system is integrated from its components so the emergent properties can only be evaluated at that time. Security and dependability are examples of important emergent system properties.

2. They are non-deterministic. This means that when presented with a specific input, they may not always produce the same output. The system’s behaviour 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 behaviour.

3. The system’s success criteria are subjective rather than objective. 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 re-interpret the organizational objectives that a system was designed to support so that a ‘successful’ system may then be seen as no longer fit for its intended purpose.

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. To help understand the effects of systems on organizations, various sociotechnical systems methodologies have been proposed. My paper on sociotechnical systems design discusses the advantages and disadvantages of these sociotechnical design methodologies (Baxter and Sommerville 2011).

19.1.1 Emergent 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 emergent properties, such as weight, can be derived directly from the subsystem properties. More often, however, they emerge from a combination of subsystem properties and subsystem
relationships. The system property cannot be calculated directly from the properties of the individual system components. Examples of emergent properties are shown in Figure 19.5.

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 behaviour of the system in its operational environment. Reliability, performance, safety and security are examples of these properties. These system characteristics are critical for computer-based systems, as failure to achieve a minimum defined level in these properties usually makes 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 properties, such as reliability, depend on both the properties of individual components and their interactions or relationships. For example, the reliability of a sociotechnical system is influenced by three things:

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 unlike 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 but affect each other in unpredictable ways. Figure 19.6 shows how failures at one level can be propagated to other levels in the system. Say a hardware component in a system starts to go wrong. Hardware failure can sometimes 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 cause stress in the system operator.

We know that people are more likely to make mistakes when they feel stressed. So a hardware failure may be the trigger for operator errors. These mistakes can, in turn, lead to unexpected software behaviour, resulting in additional demands on the processor. This could overload the hardware, causing more failures and so on. Thus, an initial, relatively minor, failure, can rapidly develop into a serious problem that could lead to 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 and it is often impossible for the system designers to limit the 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 40 degrees Celsius. Outside this temperature range, the components will behave in an unpredictable way. Now assume that this
A 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 directly measurable. Here, you are not simply concerned with attributes that relate to the behaviour of the system but also with unwanted or unacceptable behaviour.

A secure system is one that does not allow unauthorized access to its data. Unfortunately, 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 after the system is operational. That is, you only know that a system is insecure when someone manages to penetrate the system.

19.1.2 Non-determinism

A deterministic system is one that is absolutely predictable. If we ignore issues of concurrency, software systems that run on reliable hardware are deterministic. When they 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 non-deterministic. 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; sometimes they will perform a task well and sometimes they will do it badly.

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 behaviour of the system is unpredictable. Users do not know when and why changes have been made so they see the system as non-deterministic.

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 behaviour? You can only find this out by looking at the results in more depth and making judgements about the way the system has handled each input.

Non-determinism is often seen as a bad thing and that designers should try to avoid non-deterministic behaviour wherever this is possible. In fact, in sociotechnical systems, non-determinism has important benefits. It means that the
behaviour of a system is not fixed for all time but can change depending on the systems environment. For example, operators may observe that a system is showing signs of failure. Instead of using the system normally, they can change their behaviour to diagnose and recover from the detected problems.

19.1.3 Success criteria

Generally, complex sociotechnical systems are developed to tackle ‘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 emergency planning to deal with the aftermath of an earthquake. No one can accurately predict where the epicentre 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 the problem. System designers have to make assumptions but understanding what is really required only emerges after the earthquake has happened.

This makes it difficult to define the success criteria for a system. How do you decide if a new system contributes 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 Mentcare system is based was designed to support two separate business goals:

1. Improve the quality of care for sufferers from mental illness.
2. Improve the cost-effectiveness of treatments by providing managers with detailed reports of care provided and the costs of that care.

Unfortunately, these proved to be conflicting goals because the information that was needed to satisfy the reporting goal meant that doctors and nurses had to provide additional information, over and above the health records that they 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.

What this means is that any success criteria that are established in the early stages of the systems engineering process have to be regularly reconsidered during system development and use. You cannot evaluate these criteria objectively as they depend on the affect that the system has on its environment and its users.
may apparently meet its requirements as originally specified but be practically useless because of changes in the environment where it is used.

19.2 Conceptual design

Once an idea for a system has been suggested, conceptual design is the very first thing that you do in the systems engineering process. In the conceptual design phase, you take that initial idea, investigate its feasibility and develop it to create an overall vision of a system that could be developed. You then have to describe the envisaged system so that non-experts, such as system users, senior company decision makers or politicians, can understand what you are proposing.

There is an obvious overlap between conceptual design and requirements engineering. As part of the conceptual design process you have to imagine how the proposed system will be used. This may involve discussions with potential users and other stakeholders, focus groups and observations of how existing systems are used. The goal of these activities is to understand how users work, what is important to them and what practical constraints on the system there might be.

The importance of establishing a vision of a proposed system is rarely mentioned in the software design and requirements literature. However, this has been part of the systems engineering process for military systems for many years. Fairley et al. (Fairley, Thayer, and Bjorke 1994) discuss the idea of ‘concept analysis’ and the documentation of the results of concept analysis in a ‘Concept of Operations’ (ConOps) document. This idea of developing a ConOps document is now widely used for large-scale systems and you can find many examples of ConOps documents on the web.

Unfortunately, as is so often the case with military and government systems, good ideas can become mired in bureaucracy and inflexible standards. This is exactly what happened with ConOps and a ConOps document standard was proposed (IEEE, 2007). As Mostashari et al. say (Mostashari et al. 2012), this tends to lead to long and unreadable documents, which do not serve their intended purpose. They propose a more agile approach to the development of a ConOps document with a shorter and more flexible document as the output of the process.

I don’t like the term ‘Concept of Operations’, partly because of its military connotations and partly because I think that a conceptual design document is not just about system operation. It should also present the system engineer’s understanding of why the system is being developed, an explanation of why the design proposals are appropriate and, sometimes, an initial organization for the system. As Fairley says, ‘It should be organized to tell a story’ that is, written so that people without a technical background can understand the proposals that are being made.
Figure 19.7 shows activities that may be part of the conceptual design process. Conceptual design should always be a team process that involves people from different backgrounds. I was part of the conceptual design team for the digital learning environment, introduced in Chapter 1. For the digital learning environment, the design team included teachers, education researchers, software engineers, system administrators and system managers.

Concept formulation is the first stage of the process where you try to refine an initial statement of needs and work out what type of system would be best to meet the needs of system stakeholders. Initially, we were tasked with proposing an intranet for information sharing across schools that was easier to use than the current system. However, after discussions with teachers we discovered that this was not really what was required. The existing system was awkward to use but people had found workarounds to this. What was really required was a flexible digital learning environment that could be adapted by adding subject and age specific tools and content that is freely available on the Internet.

We discovered this because the concept formulation activity overlapped with the activity of problem understanding. To understand a problem you need to discuss with users and other stakeholders how they do their work. You need to find out what is important to them, what are the barriers that stop them doing what they want to do and their ideas of what changes are required. You need to be open-minded (it is their problem, not yours) and to be prepared to change your ideas when the reality does not match your initial vision.

In the system proposal development stage, the conceptual design team set out their ideas for alternative systems and these are the basis for a feasibility study to decide which of the ideas are worth further development. In a feasibility study you should look at whether or not comparable systems have been developed elsewhere, technological issues (e.g. use of mobile devices) that may affect the use
I have found that an additional useful activity is to develop an outline structure or architecture for the system. This is helpful both for feasibility assessment and to provide a basis for more detailed requirements engineering and architectural design. Furthermore, as the majority of systems are now assembled from existing systems and components, an initial architecture means that the key parts of the system have been identified and can be procured separately. This is often a better approach than procuring a system as a monolithic unit from a single supplier.

For the digital learning environment, we decided on a layered service architecture (shown in Figure 1.8). All components in the system should be considered to be replaceable services. This allows users to replace a standard service with their preferred alternative and so adapt the system to the ages and interests of the students learning with the system.

All of these activities generate information that is used to develop the system vision document. This is a critical document that is used by senior decision makers to decide whether or not further development of the system should go ahead. It is also used to develop further documents such as a risk analysis and budget estimate that are also important inputs to the decision making process.

The system vision document is used by managers to understand the system, by a procurement team to define a tender document and by requirements engineers as a basis for refining the system requirements. These different people need
different levels of detail so I suggest that the document should be structured into two parts:

1. A short summary for senior decision makers that presents the key points of the problem and the proposed system. It should be written so that readers can immediately see how the system will be used and the benefits that it will provide.

2. A number of appendices that develop the ideas in more detail and which can be used in the system procurement and requirements engineering activities.

It is challenging to write a summary of the system vision as the readers are busy people who are unlikely to have a technical background. I have found that using user stories is very effective as they provide a tangible vision of system use that non-technical people can relate to. Stories should be short, personalized and should be a feasible description of the use of the system, as shown in Figure 19.8. There is another example of a user story from the same system in Chapter 4 (Figure 4.9).

User stories are effective because readers can relate to them and they can show the capabilities of the proposed system in an easily accessible way. Of course, these are only part of a system vision and the summary must also include a high-level description of the basic assumptions made and the ways in which the system will deliver value to the organization.

19.3 System procurement

System procurement or system acquisition is a process whose outcome is a decision to buy one or more systems from system suppliers. 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 replacement of other organizational systems* If the organization has a mixture of systems that cannot work together or that are expensive to maintain, then procuring a replacement system, with additional capabilities, 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 USA). This may require the replacement of non-compliant 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, managers may decide to buy new systems.
to improve business efficiency or effectiveness. For military systems, the need to improve capability in the face of new threats is an important reason for procuring new systems.

4. **Business re-organization** Businesses and other organizations frequently restructure with the intention of improving efficiency and/or customer service. Re-organizations 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.

   Large complex systems are usually engineered using a mixture of off-the-shelf and specially built components. They are often integrated with existing legacy systems and organizational databases. When legacy systems and off-the-shelf systems are used, new custom software may be needed to integrate these components. The new software manages the component interfaces so that these components can inter-operate. 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.

   There are three types of systems or system components that may have to be procured:

   1. Off-the-shelf applications that may be used without change and which need only minimal configuration for use.

   2. Configurable application or ERP systems that have to be modified or adapted for use either by modifying the code or by using inbuilt configuration features, such as process definitions and rules.

   3. Custom systems that have to be designed and implemented specially for use.

   Each of these tends to follow a different procurement process. Figure 19.9 illustrates the main features of the procurement process for these types of system. Key issues that affect procurement processes are:

   1. Organizations often have an approved and recommended set of application software that has been checked by the IT department. It is usually possible to buy or acquire open source software from this set directly without the need for detailed justification. For example, in the iLearn system, we recommended that Wordpress should be made available for student and staff blogs. If microphones are needed, off-the-shelf hardware can be bought. There are no detailed requirements and the users adapt to the features of the chosen application.
2. 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. ERP and other large-scale application systems usually fall into this category. You may then have to modify the requirements to fit in with the system assumptions. This can have knock-on effects on other subsystems. You also usually have an extensive configuration process to tailor and adapt the application or ERP system to the buyer’s working environment.

3. 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. The requirements document is critical and procurement processes of this type usually take a considerable amount of time.

4. For public sector systems especially, there are detailed rules and regulations that affect the procurement of systems. For example, in the European Union, all public sector systems over a certain price must be open to tender by any supplier in Europe. This requires detailed tender documents to be drawn up and the tender to be advertised across Europe for a fixed period of time. Not only does this slow down the procurement process, it also tends to inhibit agile development. It forces the system buyer to develop requirements so that all companies have enough information to bid for the system contract.

5. For application systems that require change or for custom systems there is usually a contract negotiation period where the customer and supplier negotiate the terms and conditions for the development of the system. Once a system has been selected, you may negotiate with the supplier on costs, licence conditions, possible changes to the system, etc. For custom systems,
negotiations are likely to involve payment schedules, reporting, acceptance criteria, requirements change requests and costs of system changes. During this process, requirements changes may be agreed to reduce the overall costs and avoid some development problems.

Complex sociotechnical systems are rarely developed ‘in house’ by the buyer of the system. Rather, external systems companies are invited to bid for the systems engineering contract. The customer’s business is not systems engineering so its employees do not have the skills needed to develop the systems themselves. For complex hardware/software systems, it may be necessary to use a group of suppliers, each with a different type of expertise.

For large systems, such as an air traffic management system, a group of suppliers may form a consortium to bid for a contract. The consortium should include all of the capabilities required for this type of system. For an ATC system, this would include computer hardware suppliers, software companies, peripheral suppliers and suppliers of specialist equipment such as radar systems.

Customers do not usually wish to negotiate with multiple suppliers so the contract is usually awarded to a principal contractor, who coordinates the project. The principal contractor, coordinates the development of different subsystems by number of sub-contractors. The sub-contractors design and build parts of the system to a specification that is agreed with the principal contractor and the customer. Once completed, the principal contractor integrates these components and delivers them to the customer.

Decisions made at the procurement stage of the systems engineering process are critical for later stages in that process. Poor procurement decisions often lead to problems such as late delivery of a system and the development of systems that are unsuited to their operational environment. If the wrong system or the wrong supplier is chosen then the technical processes of system and software engineering become more complex.

For example, I studied a system ‘failure’ where a decision was made to choose an ERP system because this would ‘standardize’ operations across the organization. These operations were very diverse and it turned out there were good reasons for this. Standardization was practically impossible. The ERP system could not be adapted to cope with this diversity. It was ultimately abandoned after incurring costs of around £10 million.

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 no influence over the security and dependability requirements of this system. System security depends 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 significant source of security problems.

On the other hand, a decision to procure a custom system means that a lot of 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
Many bad procurement decisions stem from political rather than technical causes. Senior management may wish to have more control so demand that a single system is used across an organization. Suppliers may be chosen because they have a long-standing relationship with a company rather than because they offer the best technology. Managers may wish to maintain compatibility with existing systems because they feel threatened by new technologies. As I discuss in Chapter 20, people who do not understand the required system are often responsible for procurement decisions. Engineering issues do not necessarily play a major part in their decision making process.

19.4 System development

System development is a complex process where the elements that are part of the system are developed or purchased and then integrated to create the final system. The system requirements are the bridge between the conceptual design and the development processes. During conceptual design, 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 19.1. Once contracts for the system elements have been agreed, more detailed requirements engineering then takes place.

Figure 19.10 is a model of the systems development process. Systems engineering processes usually follow a ‘waterfall’ process model similar to the one that I discussed in Chapter 2. Although the waterfall model is inappropriate for most types of software development, higher-level systems engineering processes are plan-driven processes that still follow this model.

Plan-driven processes are used in systems engineering because different elements of the system are independently developed. Different contractors are working concurrently on separate subsystems. Therefore, the interfaces to these
elements have to be designed before development begins. 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.

One of the most confusing aspects of systems engineering is that companies use different terminology for each stage of the process. Sometimes, requirements engineering is part of the development process and sometimes it is a separate activity. However, after conceptual design, there are seven fundamental development activities:

1. **Requirements engineering** is the process of refining, analysing and documenting the high-level and business requirements identified in the conceptual design. I have covered the most important requirements engineering activities in Chapter 4.

2. **Architectural design** overlaps significantly with the requirements engineering process. The process involves establishing the overall architecture of the system, identifying the different system components and understanding the relationships between them.

3. **Requirements partitioning** is concerned with deciding which subsystems (identified in the system architecture) are responsible for implementing the system requirements. Requirements may have to be allocated to hardware, software or processes and prioritized for implementation. Ideally, you should allocate requirements to individual subsystems so that the implementation of a critical requirement does not need subsystem collaboration. However, this is not always possible. At this stage you also decide on the operational processes and how these are used in the requirements implementation.

4. **Subsystem engineering** 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 re-designing essential business processes.

5. **System integration** is the process of putting together system elements to create a new system. Only then do the emergent system properties become apparent.

6. **System testing** is an extended activity where the whole system is tested and problems are exposed. The subsystem engineering and system integration phases are re-entered 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.
7. *System deployment* 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 19.11.

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 centre, each round of the spiral may add detail to the requirements and the design. As subsystems are identified in the architecture, decisions are made on the responsibilities of these subsystems for providing the system requirements. Some rounds of the spiral may focus on requirements, 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 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.

Subsystem engineering involves designing and building the system’s hardware and software components. 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 bought rather than developed. It is usually much cheaper to buy existing products than to develop special-purpose components. However, if you buy large off-the-shelf systems, such as ERP systems, there is a significant cost in configuring these systems for use in their operational environment.

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 do implement new requirements.

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 all 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 an increasing number of systems are built by integrating off-the-shelf hardware and software application systems, the distinction between implementation and integration is becoming blurred. In some cases, there is no need to develop new hardware or software. Essentially, systems integration is 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 behaviour of the system as a whole. Inevitably, this also reveals problems with individual
subsystems that have to be repaired. Testing takes a long time and a common problem in system development is that the testing team run out of either budget or time. This can lead to the delivery of error-prone systems that need be repaired after they have been deployed.

Subsystem faults that are a consequence of invalid assumptions about other subsystems are often exposed 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 inter-operates with them.

Although straightforward in principle, system deployment may also be problematic. The user environment may be different from that anticipated by the system developers. Adapting the system to make it work in an unexpected environment can be difficult. The existing system data may require extensive cleanup and parts of it may involve more effort than expected. The interfaces to other systems may not be properly documented. You may find that the planned operational processes have to be changed because they are not compatible with the operational processes for other systems. User training is often difficult to arrange with the consequences that, initially at least, users are unable to access the capabilities of the system. These factors mean that system deployment can take much longer and cost much more than anticipated.

### 19.5 System operation and evolution

Operational processes are the processes that are involved in using the system as intended by its designers. 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. While 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.

Although the designers of operational processes may have based their process designs on extensive user studies, there is always a period of ‘domestication’ (Stewart and Williams 2005) where users adapt to the new system
and work out practical processes of how to use it. While user interface design is important, studies have shown that, given time, users can adapt to complex interfaces. As they become experienced, they prefer ways of using the system quickly rather than easily. This means that when designing systems, you should not simply cater for inexperienced users but you should design the user interface to be adaptable for experienced users.

Some people think that system operators are a source of problems in a system and that we should move towards automated systems where operator involvement is minimized. In my opinion, there are two problems with this approach:

1. It is likely to increase the technical complexity of the system because it has to be designed to cope with all anticipated failure modes. This increases the costs and time required to build the system. Provision also has to be made to bring in people to deal with unanticipated failures.

2. People are adaptable and can cope with problems and unexpected situations. This means that you do not have to anticipate everything that could possibly go wrong when you are specifying and designing the system.

   People have a unique capability of being able to respond effectively to the unexpected, even when they have never had direct experience of these unexpected events or system states. Therefore, when things go wrong, the system operators can often recover the situation by finding workarounds and using the system in non-standard ways. 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.

19.5.1 System evolution

Large, complex systems usually have a long lifetime. Complex hardware/software systems may remain in use for more than 20 years, even although both the original hardware and software technologies used are obsolete. There are a variety of reasons for this longevity as shown in Figure 19.12.
Over their lifetime, large complex systems change and evolve 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 re-organize 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 is a process that runs alongside normal system operational processes. System evolution involves re-entering the development process to make changes and extensions to the system’s hardware, software and operational processes.

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 have side-effects that adversely affect the performance or behaviour of other subsystems. Consequent changes to these subsystems may therefore be needed.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Investment cost</td>
<td>The costs of a systems engineering project may be tens or even hundreds of millions of dollars. These costs can only be justified if the system can deliver value to an organization for many years.</td>
</tr>
<tr>
<td>Loss of expertise</td>
<td>As businesses change and restructure to focus on their core activities, they often lose engineering expertise. This may mean that they lack the ability to specify the requirements for a new system.</td>
</tr>
<tr>
<td>Replacement cost</td>
<td>The cost of replacing a large system is very high. Replacing an existing system can only be justified if this leads to significant cost savings over the existing system.</td>
</tr>
<tr>
<td>Return on investment</td>
<td>If a fixed budget is available for systems engineering, spending this on new systems in some other area of the business may lead to a higher return on investment than replacing an existing system.</td>
</tr>
<tr>
<td>Risks of change</td>
<td>Systems are an inherent part of business operations and the risks of replacing existing systems with new systems cannot be justified. The danger with a new system is that things can go wrong in the hardware, software and operational processes. The potential costs of these problems for the business may be so high that they cannot take the risk of system replacement.</td>
</tr>
<tr>
<td>System dependencies</td>
<td>Other systems may depend on a system and making changes to these other systems to accommodate a replacement system may be impractical.</td>
</tr>
</tbody>
</table>

Figure 19.12
Factors that influence system lifetimes

©Ian Sommerville 2014
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 becomes corrupted by change so the costs of making further changes increases.

   Systems that have been in use for many years are often reliant on obsolete hardware and software technology. These ‘legacy systems’ (discussed in Chapter 9) are socio-technical computer-based systems that have been developed using technology that is now obsolete. However, they don’t just include legacy hardware and software. They also rely on 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.

   Changes made to a system during system evolution 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 taken 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 re-testing may be impossible after every system change. Consequently, testing may not discover adverse side-effects of changes that introduce or expose faults in other system components.

**Key Points**

Systems engineering is concerned with all aspects of specifying, buying, designing and testing complex sociotechnical systems.

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.

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 fundamental systems engineering processes are conceptual systems design, system procurement, system development and system operation.

Conceptual systems design is a key activity where high level system requirements and a vision of the operational system is developed.

System procurement covers all of the activities involved in deciding what system to buy and who should supply that system. Different procurement processes are used for off-the-shelf application systems, configurable COTS systems and custom systems.
System development processes include requirements specification, design, construction, integration and testing.

When a system is put into use, the operational processes and the system itself inevitably change to reflect changes to the business requirements and the system’s environment.

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

‘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 socio-technical issues to be considered. Dependability is obviously critically important. *(In NASA Systems Engineering Handbook, NASA-SP2007-6105, 2007)*

http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080008301_2008008500.pdf

The LSCITS Socio-technical Systems Handbook. This handbook introduces sociotechnical systems in an accessible way and provides access to more detailed papers on socio-technical topics. *(Various authors, 2012)*


*Architecting systems: A Primer on Purpose, Principles and Concepts*. This is a refreshingly different book on systems engineering that does not have the hardware focus of many ‘traditional’ systems engineering books. The author, who is an experienced systems engineer, draws on examples from a wide range of systems and recognizes the importance of sociotechnical as well as technical issues. *(H. Sillitto, College Publications, 2015)*

WEBSITE

PowerPoint slides for this chapter:


Links to supporting videos:


EXERCISES
19.1 Give two examples of government functions that are supported by complex socio-technical systems and explain why, in the foreseeable future, these functions cannot be completely automated.

19.2 Explain why the environment in which a computer-based system is installed may have unanticipated effects on the system that lead to system failure.

19.3 Why is it impossible to infer the emergent properties of a complex system from the properties of the system components?

19.4 What is a 'wicked problem'? Explain why the development of a national medical records system should be considered a 'wicked problem'.

19.5 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. Develop a conceptual design for such a system, highlighting its key characteristics and essential high-level requirements.

19.6 Explain why you need to be flexible and adapt system requirements when procuring large off-the-shelf software systems, such as ERP systems. Search the web for discussions of the failures of such systems and explain, from a sociotechnical perspective, why these failures occurred. A possible starting point is: http://blog.360cloudsolutions.com/blog/bid/94028/Top-Six-ERP-Implementation-Failures

19.7 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.

19.8 Explain why legacy systems are often critical to the operation of a business.

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

19.10 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 have sorted out the problem?

REFERENCES


