Chapter 1
Introduction to systems engineering

'... it's thin at one end, much, much thicker in the middle, and then thin again at the far end.'

'The Brontosaurus Theory', Ann Elk, Monty Python sketch, 1974

1.1 Introduction

How do you solve a problem like systems engineering? Systems engineering is one of those disciplines that can be applied to just about anything by just about anyone. Also, people have been engineering systems since the dawn of mankind, so why is it a subject that is very often misunderstood?

Systems engineering is often seen as an emerging discipline, which may be true in terms of formal education, such as university programmes, but without systems engineering we wouldn't have Stonehenge or the pyramids – these are examples of systems that have clearly been systems-engineered. Also, consider political systems, financial systems, belief systems and so on. All of these have not happened by chance and must have been planned and developed in some way, according to an approach.

In the same way, many people will be badged with the term *systems engineer*, even though they are not. Many people will have evolved into a systems engineer from being a specialist in a particular field. For example, many people have evolved from being a software engineer into a systems engineer; many people who were systems analysts or designers are now systems engineers. One of the great misconceptions about systems engineering is that it is all about requirements engineering – sure, requirements are tremendously important, but systems engineering is more than a sum of its parts.

In order to understand how this has occurred, it is first necessary to try to define exactly what we mean by systems engineering.

1.2 Information sources

There is no single definition for the term *systems engineering*. This may be due to the vast scope of systems engineering as it can be applied to any type of application

in any domain and, hence, is very difficult to put a boundary on. It is one of those terms that mean different things to different people. However, as time goes on, there is a natural trend towards standardizing the terms and concepts used within the field. There are many places that one could visit to attempt to define systems engineering and in this book, for the sake of brevity, it is essential that a common source be used for the concepts and terminology used herein. The two main sources that were looked at were international standards and international industrial bodies.

There are many international standards that can be applied to systems engineering – see Chapter 6 for more examples of this. However, an obvious port of call is the International Standards Organization – ISO. The International Standards Organization is the world's largest and most recognized standards body, and produces many thousands of standards. The standard that is of particular interest for the purposes of this book is *IEC/ISO 15288 – systems engineering – systems life cycle processes* [1]. This standard identifies a high-level set of processes that may be used in any organization in order to understand and apply systems-engineering practices within it. Therefore, this standard will be used as one of the two key references throughout this book and, as much as practically possible, all terms and concepts in this standard will be adopted.

The other source of reference that is to be used for this book is from an industrial organization that produces a systems-engineering best-practice handbook – *The INCOSE Systems Engineering Handbook*. The International Council on Systems Engineering is a worldwide organization whose mission is to 'to advance the state of the art and practice of systems engineering in industry, academia, and government by promoting interdisciplinary, scalable approaches to produce technologically appropriate solutions that meet societal needs' [2].

The INCOSE handbook is based on what is perceived as being international best practice and, at the time of writing, this handbook is in version 3.0. Version 3.0 of the handbook, rather conveniently for the purposes of this book, is based directly on the processes that are defined in ISO 15288 [3].

Therefore, the two main references that are used in this book are ISO 15288 and the INCOSE handbook, which, for all intents and purposes, use the same core set of systems-engineering processes. The two sources are not 100 per cent consistent with one another but, where this becomes an issue, a discussion will be raised in the book.

1.3 Defining systems engineering

There are many definitions of systems engineering, all of which tend to differ depending on the context and the point of view or background of the author. This section presents a few of the more widely recognized definitions and then discusses each in turn.

Definition 1 'Systems engineering is a discipline that concentrates on the design and application of the whole (system) as distinct from the parts. It involves looking at a problem in its entirety, taking into account all the facets and all the variables relating the social to the technical aspect' [4].

Notice that in this definition there is an emphasis on looking at the bigger picture and this is brought up several times: 'whole [system]', 'problem in its entirety' and 'all the facets'. This is a key concept in systems engineering, where a system is looked at across its whole life cycle and not just one small part. Also notice here that non-technical facets of the system are mentioned as having an influence on the system.

Definition 2 'Systems engineering is an iterative process of top-down synthesis, development and operation of a real-world system that satisfies, in a near optimal manner, the full range of requirements for the system' [5].

There are a few concepts here that are introduced that are not in the previous definition. The first is the concept of an *iterative process*. Real-life systems are rarely developed in a linear fashion as, even with what may be perceived as a linear life-cycle model, for example, there will be much iteration involved inside each stage. The second interesting point here is that requirements have been mentioned for the first time in any of the definitions. Indeed, this definition talks about satisfying requirements, which must be one of the basic goals of any systems engineer. This is also qualified, however, by stating that this should be in a near-optimal manner. Obviously, in an ideal scenario, all requirements should be met in an optimum fashion but, in the real world, it is often the best that can be achieved given the resources and technology at one's disposal. This definition also includes a rather contentious statement in that the development is said to be 'top-down', which could be interpreted as being rather limited.

Definition 3 'Systems engineering is an inter-disciplinary approach and means to enable the realisation of successful systems' [6,7].

The INCOSE definition of systems engineering is rather more terse than the previous two, yet no less accurate. This statement simply states what must be the highest-level requirement for any systems engineer, which is to realize successful systems by using any appropriate means necessary.

Definition 4 'Systems engineering is the implementation of common sense' [8].

The final definition that is looked at here is definitely from the 'less is more' camp and makes a rather bold statement about systems engineering generally, in that it is mostly good common sense. But, of course, as any schoolchild will tell you, the strange thing about common sense is that it is not all that common!

So, there have been four definitions presented here, each of which is correct, yet each of which is very different. This is, perhaps, symptomatic of a discipline that includes all other disciplines, that cannot be bounded and that can be applied to any system in any domain!

1.4 The need for systems engineering

It has been established that it is difficult to pin down an exact definition for systems engineering. However, it is not so difficult to pin down why we *need* systems engineering. Many systems end in failure or disaster. The term *failure* here refers to a system where the project never reached delivery and where time and money were wasted. The term *disaster* here refers to a system where people were hurt or the environment was damaged as a result of the system failing.

One of the problems with systems is that the inherent complexity of modern-day systems is increasing all the time. This complexity does not just apply in the actual system, or system of interest as it will be referred to in this book, but also in the interactions with other systems.

There are many examples of disasters and failures that can be found in the literature and these are often grouped by their application domain. In all cases the impact of other systems is enormous and is difficult to calculate. In the following tables, just a few of the more famous disasters and failures are listed and a very brief summary of the end result is provided. In order to appreciate the full scale of the impact of such failures and disasters, readers are encouraged to look into some of these in more detail to gain a fuller insight into the horrors of what can happen when things go wrong. Perhaps the most frightening thing about all these examples is not the end result, but how easily things might have turned out differently if a more rigorous systems-engineering approach had been applied throughout the project and product life cycles.

The examples in Table 1.1 identify some of the better-known failures and disasters, and one of the first questions that enter many people's minds is that, surely, these were all examples of 'risky' systems.

Table 1.1 Examples of failures and disasters

Name	Application domain	Location	End result
Bhopal	Chemical	India	Massive loss of human life, damage to the environment
Flixborough	Chemical	UK	Environmental damage
Therac 25	Medical	USA	Loss of life
Challenger	Aerospace	USA	
DC-10 doors	Aerospace		Loss of human life
Chernobyl	Nuclear	USSR	Massive loss of human life, damage to environment
Windscale	Nuclear	UK	Massive environmental damage
Three Mile Island	Nuclear	UK	Massive environmental damage

One of the requirements for good systems engineering is to minimize the risk inherent in a system. This risk may take on a number of different contexts, such as business risk, technical risk, financial risk, environmental risk. The context of the risk is very important here, as it is possible to have risk manifest itself at different levels in different contexts. For example, a system may have a very low financial risk, as it may be worth little money or be a system with adequate resources. However, consider the situation where a project associated with the system is in a new area where no domain knowledge exists. In this scenario, it may be that there is a very low financial risk, but a very high technical risk. In such cases, which occur all too often, it is necessary to be able to trade off between these different contexts.

Risk is not a new concept, however, and as time goes on the nature of risk is changing. Historically, the greatest risks to mankind were always natural, such as earthquakes and floods. However, there are now new – manmade – risks, such as war and pollution. To compound this even further, these two types of risk do not exist in isolation, and will impact one another. Manmade systems influence natural ones, such as deforestation projects causing landslides. Also, natural systems will influence manmade ones, such as a strong wind spreading the effects of radiation. Again, this comes back to the concept of different contexts, where one context will influence or affect another context.

It is essential, therefore, that, when we consider our systems, we also consider other systems that we interact with, whether they are physical systems, people or, indeed, the environment that we are working in. These relationships are very complex and it is important that we can understand them in some way and relate them effectively to our system model.

In an ideal world, it would be nice to eliminate risk altogether. However, in the real world, this is simply not possible. Also, progress demands taking risk whether it is making a first flight, developing new power technologies or crossing a busy road.

One important aspect of risk that needs to be understood and considered is the nature of responsibility. It is possible that, as time moves on, the nature of responsibility changes. For example, if a person took up smoking in the 1940s, it could be argued that the tobacco companies would be responsible for any smoking-related illnesses, as the risks involved with smoking were not understood at the time. However, if someone in the 2000s starts to smoke and as a consequence has a smoking-related illness, then it is that individual's fault, since the health risks relating to smoking are now well understood and are emblazoned across the front of all tobacco products.

Therefore, it is essential that both the risk and the responsibility for the risk be well understood.

1.5 The three evils of engineering

Projects fail and disasters occur for many reasons. However, there are three underlying reasons why things go wrong in life, which are known as the 'three evils' of systems engineering: complexity, a lack of understanding and communication issues.

1.5.1 Complexity

The concept of complexity will be illustrated in two ways – one that emphasizes the importance of relationships, and one that uses a brontosaurus to visualize the nature of the evolution of complexity.

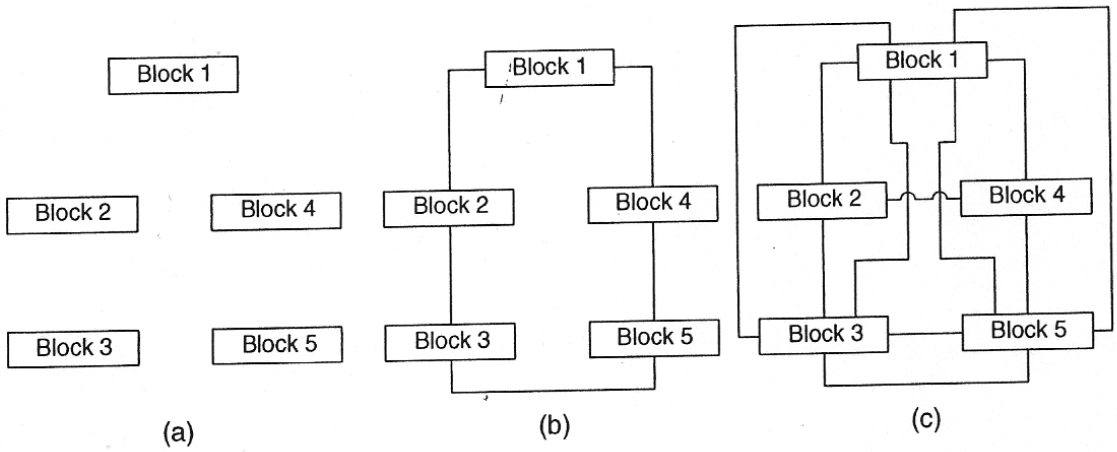

Figure 1.1 Complexity manifesting through relationships

For the first example, consider three boxes that may represent three elements within a system, as shown in Figure 1.1(a). Each element may represent almost anything, ranging from a text sentence that represents a requirement, to an assembly that makes up a system, to users that relate to a system. Each of these elements may very well be understood by whoever is reading the diagram, but this does not necessarily mean that the system itself is understood.

Consider now Figure 1.1(b), and it is quite clear that this diagram is more complex than the previous one, although nothing has changed in the elements themselves, only the relationships between them.

Consider now Figure 1.1(c) where it is, again, obvious that this diagram is more complex than its predecessor and far more complex than the first.

In fact, the more relationships that are added between the system elements, the higher the complexity of the overall system. More and more lines could be drawn onto this diagram and the complexity will increase dramatically, despite the fact that the complexity of each of the three elements has not actually increased as such.

The point that is being communicated here is that, just because someone understands each element within a system, this does not, by any means, mean that the system itself is understood. The complexity of a system manifests itself by relationships between things – in this case the system elements. It should be borne in mind, however, that these elements may exist at any level of abstraction of the systems, depending on what they represent. Therefore, the complexity may manifest itself at any point in the system. The complexity of the whole of the system is certainly higher than the complexity of the sum of its parts.

This may be thought of as being able to see both the woods *and* the trees.

The second way that complexity is illustrated is through the concept of the 'brontosaurus of complexity'. In this slightly bizarre analogy, complexity is visualized as a brontosaurus, in that the complexity of a system at the outset is represented by the dinosaur's head and, as the project life cycle progresses, this complexity increases (travelling down the neck); it increases even further (through the belly) before reducing and finally ending up at the tail of the brontosaurus.

This fits with the shape of the brontosaurus, that is 'thin at one end, much, much thicker in the middle, and then thin again at the far end' [9].

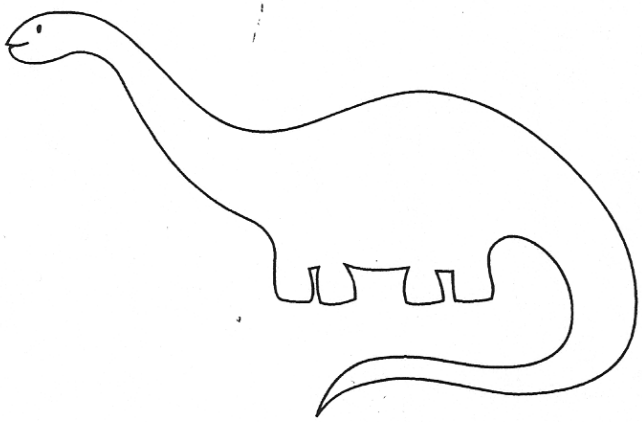

Figure 1.2 A brontosaurus

The perceived complexity of a project is almost always low to begin with, but balloons during the analysis of a project, as the understanding of the full impact of the requirements and the constraints is fully understood. By the time the problem is fully understood, a project is well and truly in the 'belly of the brontosaurus', whereas, when the design begins and becomes optimized, the project should be heading towards the 'tail of the brontosaurus'. By applying the brontosaurus-of-complexity analogy, it is shown that one must go from the head (initial ideas and requirements) to the tail (system), but that it is impossible to do this without going through the belly of the beast.

Consider the situation when a project is at the head of the brontosaurus, then this may be visualized as the illustration in Figure 1.1. As the complexity of the project increases and we move down the neck of the brontosaurus, so the complexity increases, as shown in Figure 1.1b. In fact, the more relationships that are added between the system elements (and, hence, the more interactions between them) the closer to the belly we actually get.

Many projects will fail because the project never leaves the belly or, in some cases, is left even higher up in the neck. If a project stays in the head or neck, there is a large danger of the system being oversimplified and the complexity inherent in the system is never uncovered until it is too late. If the project remains in the belly, however, the complexity has been realized, but it has not been managed effectively. Unfortunately, when a project is in the belly of the brontosaurus, it may seem to the project personnel that the world is at an end and that there is no understanding of the project as a whole.

This can be quite soul-destroying and is a situation that many systems engineers would have found themselves in at some point in their careers. Successful systems engineering is about being able to see the brontosaurus as a whole and that there is life after the belly.

In a final twist to this analogy, there is an ironic major difference between complexity and the brontosaurus. Complexity is difficult to visualize, but definitely exists in any system. A brontosaurus, on the other hand, is easy to visualize (see Figure 1.2) but never actually existed (it was demonstrated in 1974 that the brontosaurus was actually the apatosaurus).

1.5.2 Lack of understanding

A lack of understanding may occur at any stage of the system life cycle and also may occur during any process. Consider the following examples of a lack of understanding affecting a project.

- A lack of understanding may occur during the conception stage of a project, during a requirement-related process. If the requirements are not stated in a concise and unambiguous fashion (or, in reality, as unambiguously as possible) then this lack of understanding will cascade throughout the whole project. It is widely accepted that mistakes made during early stages of the life cycle cost many times more to fix during later stages, so it makes sense to get things right as early as possible [10].
- A lack of understanding may occur during the development stage of a project, during an analysis-related process. For example, there may be a lack of domain knowledge during analysis that may lead someone to state false assumptions, or actually to get something completely wrong due to insufficient knowledge. Again, uncorrected mistakes here will lead to larger problems further down the development life cycle.
- A lack of understanding may occur during the operational stage of a project, during an operation-related process. Incorrect usage of a system may lead to a system failure or disaster. For example, people not following safety procedures, people not using the correct tools, etc.

Of course, these examples are merely a representation of some ways that a lack of understanding can manifest itself in a system – there are many other places in the life cycle where problems may occur.

1.5.3 Communication problems

The third of the three evils is the problem of communication or, rather, *ineffective* communication. The richness and complexity of human communication is what separates humans from other species. One of the earliest recorded examples of project failure is that of the Tower of Babel, as described wonderfully by Fred Brookes [11]. The Tower of Babel started life as a very successful project and, indeed, the first few stages went off without a hitch and the project was running on schedule, within budget and was meeting all the project requirements. However, one of the key stakeholder's requirements was not considered properly, which was to cause the downfall of the

project. When the stakeholder intervened in a divine fashion, the communication between project personnel was effectively destroyed.

Communication problems may occur at any level of the organization or project.

- *Person-to-person level.* If individuals cannot communicate on a personal level, then there is little hope for the project's success. This may be because people have different spoken languages, technical backgrounds or even a clash of personalities.
- *Group-to-group level.* Groups, or organizational units, within an organization must be able to communicate effectively with one another. It has already been pointed out that the world of systems engineering is populated by people with different backgrounds and from different disciplines and this often manifests itself with groups of like-minded people, none of whom can communicate with anyone outside their circle of interest. These groups may be from different technical areas, such as hardware and software, or may span boundaries, such as management and technical, or marketing and engineering.
- *Organization-to-organization level.* Different organizations speak different languages – each will have its own specific terms for different concepts, as well as having an industry-specific terminology. When two organizations are working in a customer–supplier relationship, the onus is often on the supplier to speak the customer's language so that communication can be effective, rather than the customer having to speak the supplier's language. After all, if the supplier won't make an effort to speak the customer's language, it is quite likely that they will not remain customers for very long.
- *System-to-system level.* Even nonhuman systems must be able to communicate with one another. Technical systems must be able to communicate with technical systems, but also with financial systems, accountancy systems, environmental systems, etc.
- *Any combination of the above.* Just to make matters even more confusing, just about any combination of the above communication types is also possible.

These problems, therefore, lead to ambiguities in interpreting any sort of communication, whether it is a spoken language or an application-specific or technical language.

1.5.4 The vicious triangle of evil

Having established that these three evils of engineering exist, matters become even worse. Each of these three evils does not exist in isolation, but they will actually feed into one another. Therefore, unmanaged complexity will lead to a lack of understanding and communication problems. Communication problems will lead to unidentified complexity and a lack of understanding. Finally, a lack of understanding will lead to communication problems and complexity.

The three evils, therefore, form a triangle of evil that it is impossible to eliminate. In fact, the best that one may hope for is to address each of these evils in its entirety and try to minimize it. It is important always to consider these three evils at all stages of a project and when looking at any view of a system.

1.6 Systems-engineering concepts

There are several key systems-engineering concepts and it is important to have these identified and understood before any applications of systems engineering can be discussed. Therefore, based on the information in *The INCOSE Systems Engineering Handbook* and, hence, ISO 15288, the following generic systems-engineering meta-model has been generated. This meta-model is simply a defined set of concepts and terminology along with their relationships to one another. This diagram by no means represents every concept in systems engineering, but it does represent a set of the key concepts when considering systems engineering.

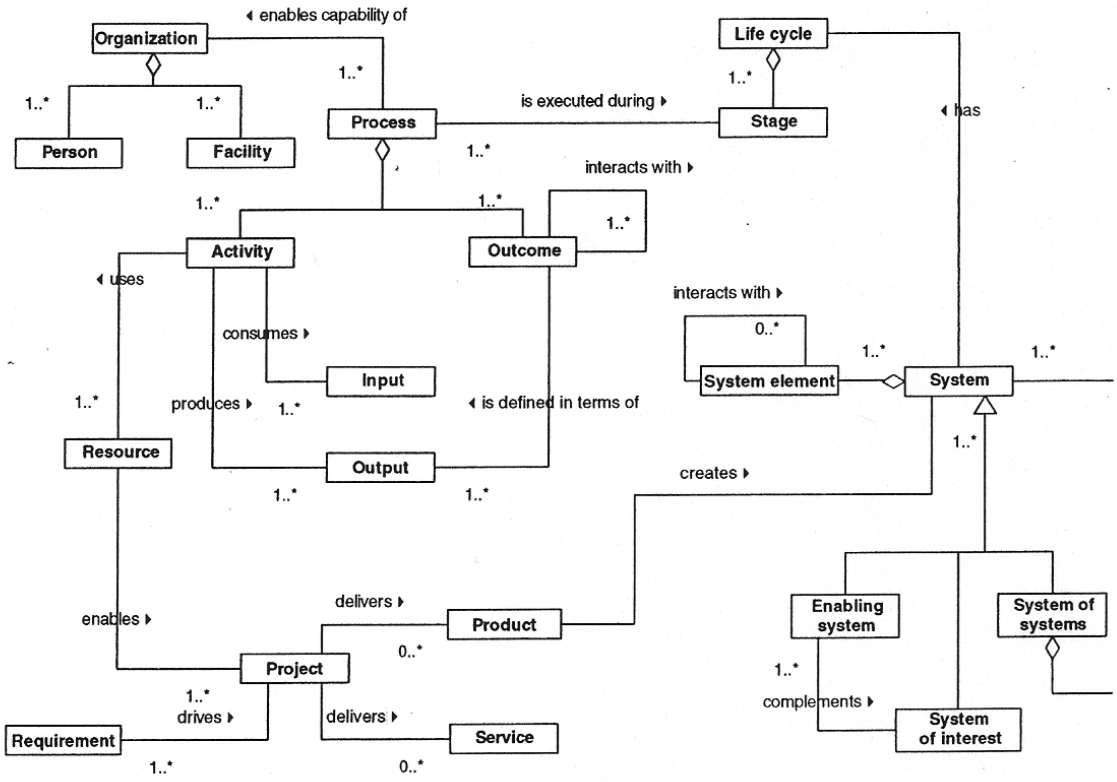

Figure 1.3 Systems engineering generic meta-model

The diagram in Figure 1.3 shows the key concepts that must be defined in order successfully to apply systems-engineering techniques. Rather than simply list each of these concepts and provide a description, it is useful to think of several of these concepts at the same time, as they are naturally related. This diagram may be augmented by grouping several of these concepts together into logical groupings, as shown in Figure 1.4, which is identical to the one in Figure 1.3 except that certain concepts have been grouped into packages.

These groups are processes, life cycles, projects and systems. Each of these groups will be discussed in more detail in subsequent sections, and it is essential that a common high-level understanding be established at the outset of this book.

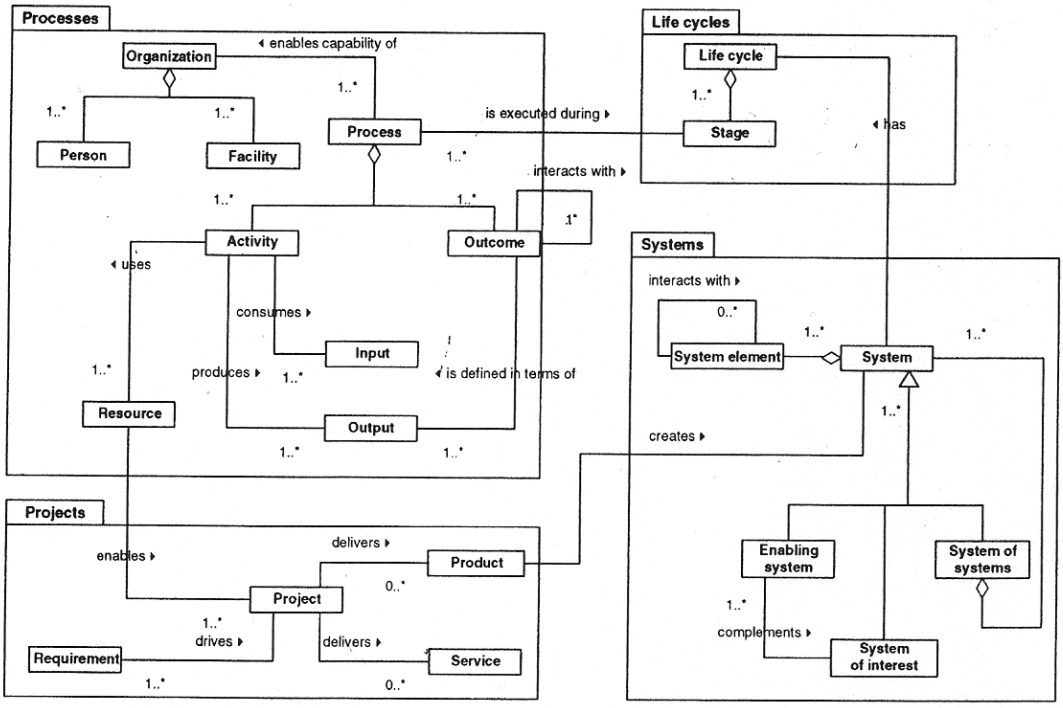

Figure 1.4 Systems engineering meta-model with groups

Each group will be introduced briefly and the main elements of the diagram defined and described. The terms and concepts that are described in this chapter will set the terminology that will be used throughout the rest of this book, so it is important that each concept and term can be understood. Finally, the relevance and importance of the concepts in each group are discussed, with other examples introduced wherever necessary.

1.6.1 Processes

Processes are an integral part of any organization. In real life, everything follows a process, be it a person, an organization or a system – be it technical, natural, financial, social or whatever. The importance of process modelling will de discussed in more detail in Chapter 6 and only the key elements will be introduced here.

It is essential to establish a good understanding of the processes that exist and that are needed to realize the requirements of a system or project effectively.

The diagram in Figure 1.5 shows the systems engineering meta-model, with the area of particular interest – that of processes, shaded in for emphasis. The key concept here is that of the 'process'. A process is defined as 'an interrelated set of activities etc.'. A process, however, simply describes an approach to doing something that, in the context of this book, refers to our approach to systems engineering. A number of processes are executed at each stage of the life cycle. It is important here to note that processes are not the same as stages, a misunderstanding that often occurs in the

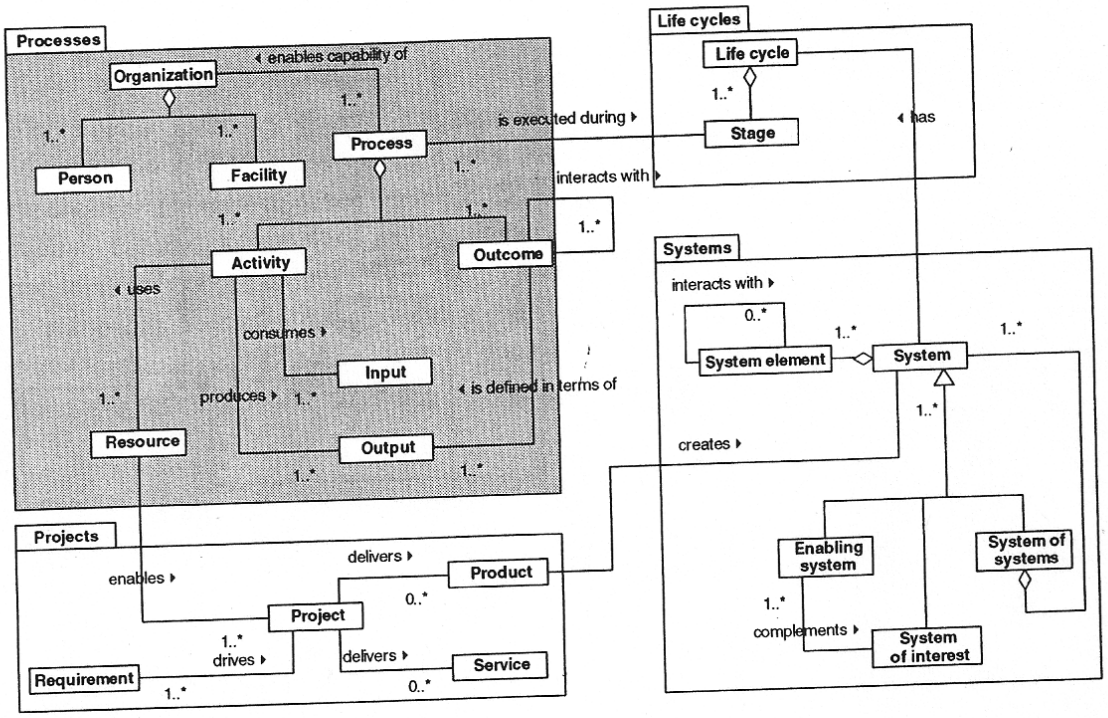

Figure 1.5 Systems-engineering concepts with a focus on 'processes'

world of systems engineering. A process is made up of two main elements:

- *Activity*. An activity may be thought of as a unit of behaviour in a process. These are often referred to by different terms in different standards, such as practice, step, etc. Both ISO 15288 and *The INCOSE Systems Engineering Handbook* use this term in the same way. A process is defined by a set of interrelated activities. The artefacts of a process are defined in terms of one or more 'Input' or 'Output'. An activity also consumes one or more 'Resources', which may be human, systems, money, assets, etc.
- *Outcome*. An outcome describes something that is generated or that occurs as the result of executing a number of activities in a process. This is one of those concepts where the definitions in ISO 15288 and the INCOSE handbook actually differ somewhat. In ISO 15288, an outcome occurs as the result of an activity, but is not the same, conceptually, as an artefact related to a process – not necessarily a produced thing, such as a document, specification or subsystem. In the INCOSE handbook, the artefacts of a process are defined as inputs and outputs, but these are not necessarily the same as the concept of an outcome in ISO 15288. This inconsistency of terms has been addressed by relating the process outputs to the process outcomes and by stating that a process 'Outcome' is defined in terms of one or more 'Output'.

Processes are executed within the bounds of an 'Organization' that is defined as being made up of the following.

- *Person*. This represents the staff at the organization, each of whom will hold a number of roles and responsibilities and have a set of competencies defined.

- *Facility*. This represents the role of any systems or services that are available to the organization.

Processes are described in more detail in Chapter 6.

1.6.2 Systems

Perhaps the most obvious concept that needs to be looked at is that of the system itself. The INCOSE systems-engineering handbook and ISO 15288 identify a number of key concepts that need to be understood, which are presented in Figure 1.6. However, there are a few more concepts that will be introduced for the purposes of this book that will be expanded upon later.

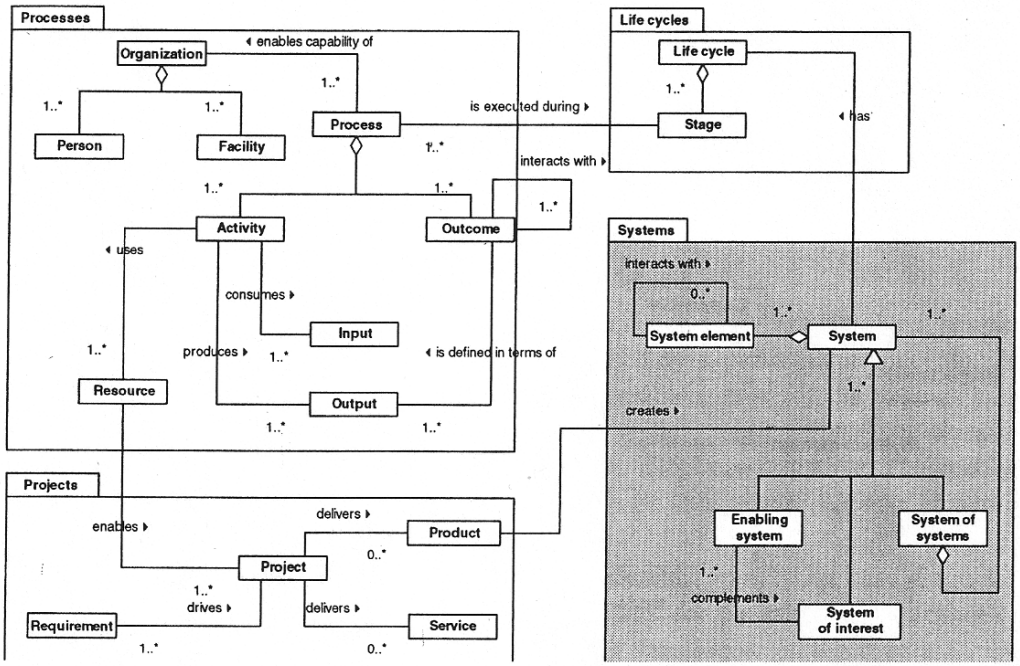

Figure 1.6 Systems engineering concepts with a focus on 'systems'

The key concept when thinking of systems is, unsurprisingly, the system. A system may be represented as some sort of hierarchy, with a system being made up of one or more system elements, each of which interacts with one or more other system elements. The diagram here represents a system at its highest level, with only a single level of decomposition. In reality, it is usual to break a system down into any number of levels, for example: subsystems, assemblies, subassemblies.

There are three main types of system that must be considered.

- *System of interest*. This is the system that is being developed by the project at hand. This is also what many people think of when considering systems engineering – a single system with interfaces to the outside world. However, in order to ensure that

the system of interest is realized successfully and optimally, one must consider the bigger picture.

- *Enabling systems*. These are systems that are outside the boundary of the system of interest and that complement the system of interest. It should also be borne in mind that the concepts of a system of interest and an enabling system are entirely dependent on the context of the system. For example, consider two systems: a robot and a human who controls the robot. From one point of view (context), the human is the system of interest and the robot is the enabling system, whereas from another point of view (context) the robot is the system of interest and the human is the enabling system.
- *System of systems*. Any system may be considered as being part of a larger system of systems. Therefore, in the above example, both systems described may be considered a single system that consists of (in this case) two systems. This may also be abstracted up to any higher level, such as the country, the world, the universe.

It is essential that any systems modelling can cope with being able to represent any of these types of system. Indeed, it will be seen later in this book that there are several ways to model these different types of system.

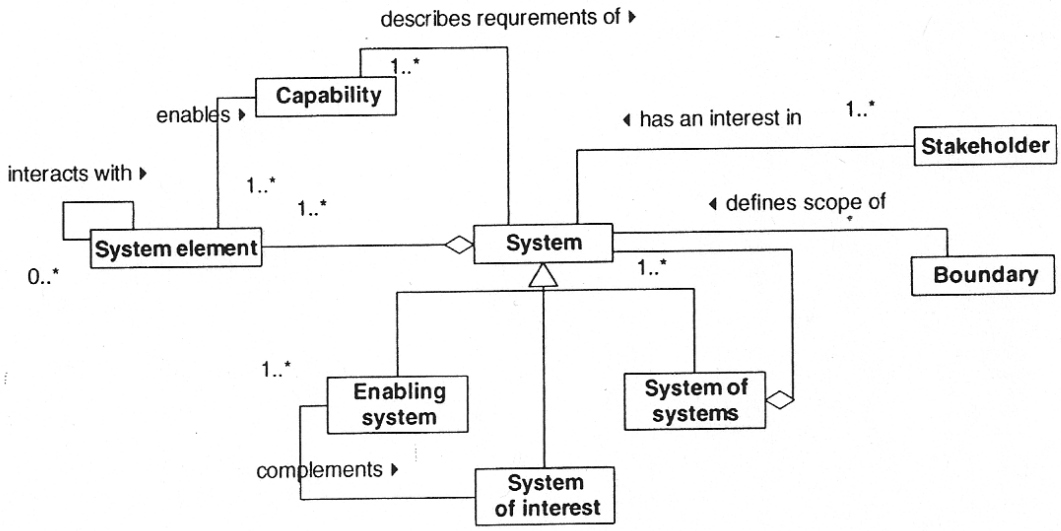

Figure 1.7 Expanded systems-engineering terms and concepts

The diagram in Figure 1.7 shows the same concepts as in the previous diagram, but three more key concepts have been added here.

- *Capability*. Like many terms associated with systems engineering, 'capability' will have different meaning for different people, depending on the application domain, industry or educational background. For the purposes of this book, a capability describes the requirements of a system at a high level. When a system is expressed in terms of its capability, no particular solution is held in mind but the emphasis is on what the system should be able to do, rather than how it should do it, or what technologies it should use.

- *Stakeholder*. A stakeholder, for the purposes of this book, represents the role of a person, place or thing that has an interest in the project. A role is *not* necessarily a person, which is a very common mistake to make, but may be, for example, a system, an organization or even a standard. Also, people often think that there is a one-to-one mapping between the stakeholder and the name of the person, place or thing that is realizing it. There are many instances where a stakeholder may be realized by many different names, or where a single name takes on a number of different roles. The words *stakeholder*, *role*, *actor*, *person* and so on are often used interchangeably, which can lead to a great deal of confusion between people working with different definitions and different organizations.
- *Boundary*. The system boundary draws the line between what is inside and what is outside the system. Effectively, the system boundary defines the scope and context of the system.

These new terms are very important, as they allow the concept of a 'context' to be introduced, which is a very useful tool when it comes to trying to understand the three basic types of system that were introduced in this section: systems of interest, enabling systems and systems of systems.

1.6.3 The 'context'

In order to illustrate the idea of a context, it is first necessary to consider a simple example. Imagine a system that must be developed by a particular project that is to be some sort of robot system. There are few requirements for the sake of this example. The aim of the project is to provide a vehicle that can navigate terrain, both automatically and through manual operation, and that can stay alive. The system must have a self-contained power supply and must be able to avoid obstacles while navigating the terrain.

This may be visualized by drawing up a 'use case diagram'. Use case diagrams will be described in more detail later in the book, but, for the sake of this example, all that one needs to be able to understand is that the ellipses in the diagram represent requirements, the stick people represent stakeholders and the large rectangle represents the system boundary.

The diagram in Figure 1.8 shows the context of the system – what the system is intended to do, where the boundary lies and what lies outside the boundary of the system. In fact, the last sentence as written here is not true. The diagram does not show *the* context of the system, but shows *a* context of the system. This point is critical to understanding systems. A context simply represents a system *from a particular stakeholder's point of view*. This diagram shows the context from the point of view of the robot product.

It is now useful to tie this diagram back to the concepts that were introduced previously.

Everything inside the system boundary represents the requirements of the 'system of interest'. The stakeholders (represented by stick people) outside the boundary of the system may be considered to be the 'enabling systems'.

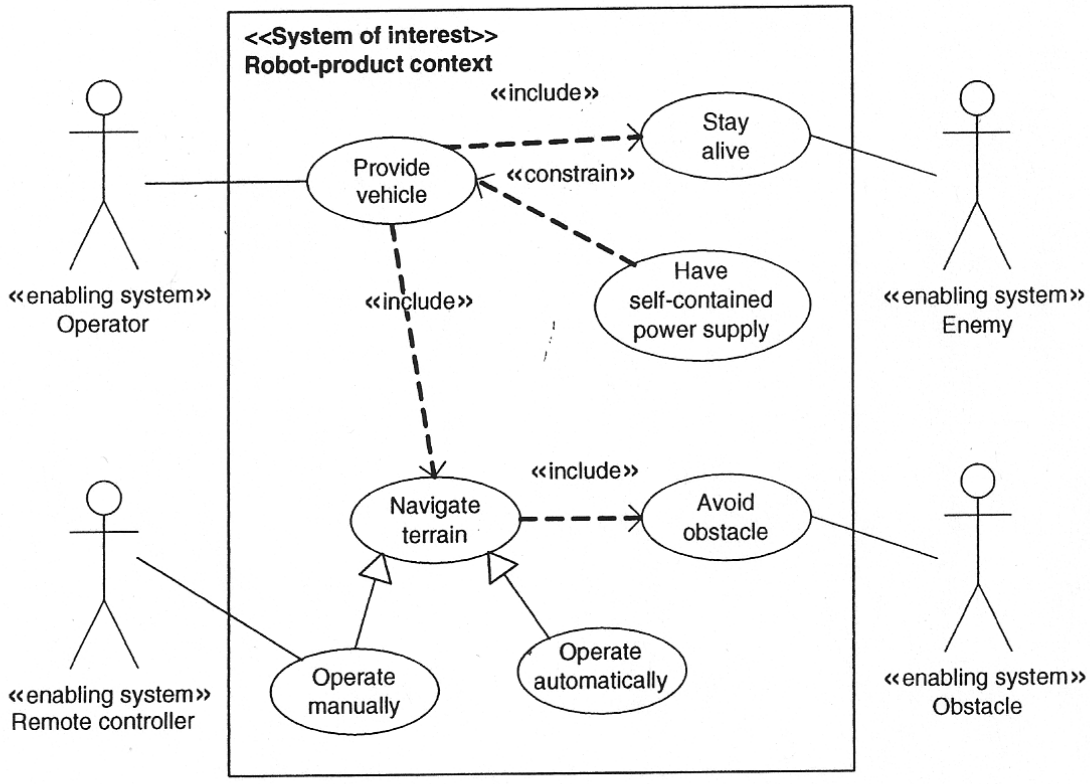

Figure 1.8 Example context

Consider now, however, the same system – the robot – but this time from the point of view of the enemy. This is the same system, but from a different point of view – hence it represents a different context.

Consider the diagram in Figure 1.9, where the same system is looked at from several different points of view, based on the stakeholders from the previous diagram.

The diagram in Figure 1.9 shows the same system from six different stakeholders' points of view, all of which are quite different. Note that, as stated previously, one stakeholder's system of interest is another stakeholder's enabling system, and vice versa. Also, when viewed in its entirety, this may be considered a system of systems. This same system of systems may be represented at a higher level of abstraction, by a single context. When considering a system of systems, we look at the system from differing points of view, but this time they may be based on levels of abstraction, rather than from specific stakeholders' points of view.

The diagram in Figure 1.10 shows a higher-level context for the same system of systems, but this time from the country's (home nation's) point of view. In this case, a single context is shown, compared with the many interacting systems that were shown in Figure 1.9. Each of these two approaches is valid and, again because of the concept of the context, the more appropriate one will depend on the point of view being adopted.

The main point here is to demonstrate that there are several ways that the same concepts – systems of systems – can be visualized, depending on why we need to visualize and from which point of view we are visualizing.

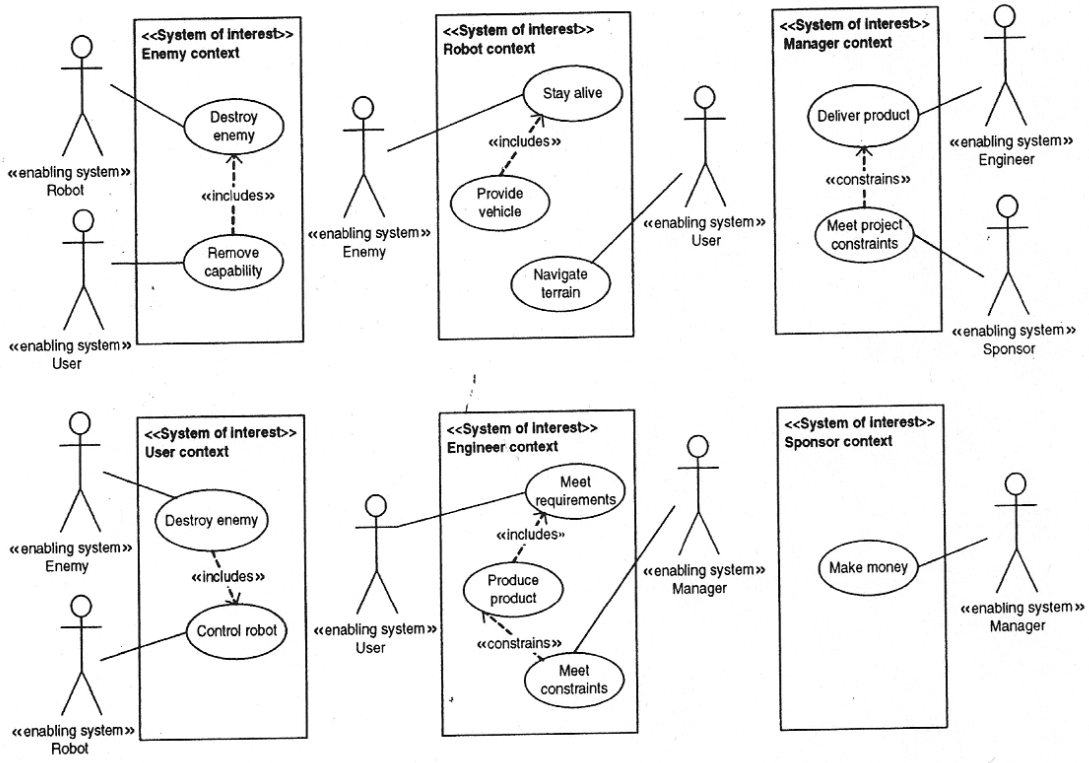

Figure 1.9 Several contexts

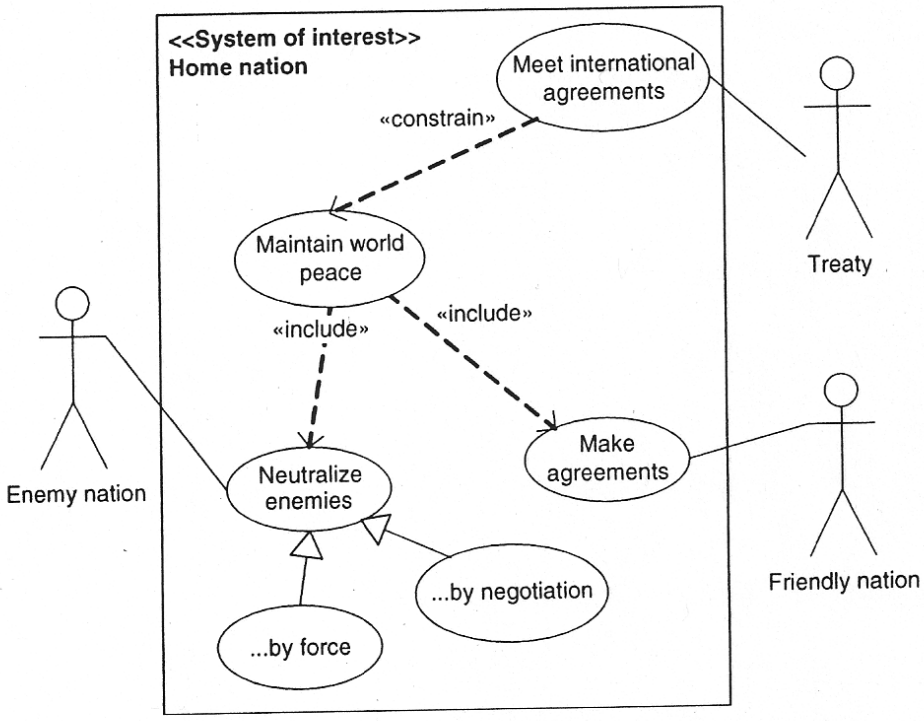

Figure 1.10 Single, high-level context

In summary, therefore, it is essential to look at things from different points of view in order to get a full understanding of the system as a whole.

1.6.4 Life cycles

Perhaps one of the most widely misunderstood, and most important, aspects of systems engineering is that of the life cycle.

Everything in life has a life cycle, whether it is a person, a product or a project. Just to confuse matters, there are very often a number of life cycles within life cycles. For example, consider the life cycle of a passenger train (a product) and then consider the number projects that will be run in order to cover the whole life cycle of the train. Each of the projects has its own life cycle.

For the purposes of this book, the main emphasis will be on project life cycles.

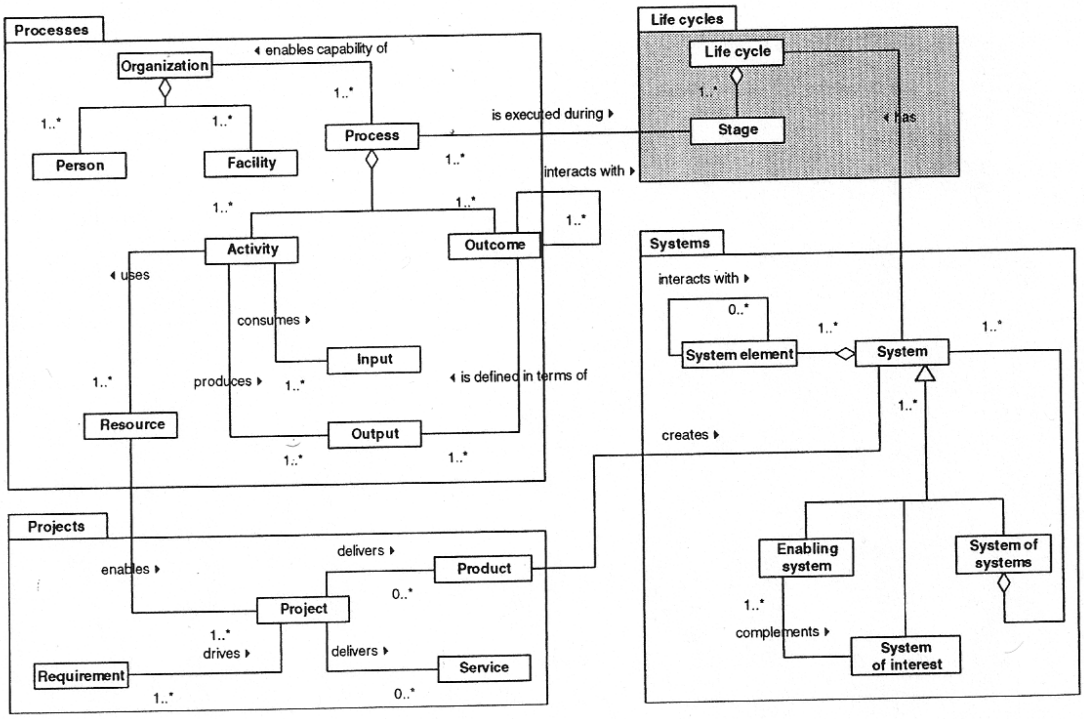

Figure 1.11 Systems-engineering concepts with a focus on 'life cycles'

The diagram in Figure 1.11 shows that a life cycle is made up of a number of stages, the names of which will vary according to whatever standard or best-practice model is being used. A life cycle starts from when the idea is first thought of, and ends when the project or product is decommissioned.

People often use informal phrases for life cycles, such as 'cradle to grave', 'lust to dust' and 'sperm to worm'. Interestingly enough, all these analogies relate to the life of a human, but each can be viewed as being incorrect, as they all assume that the life cycle starts at conception. Although this is very often the case, there are many cases where people may be 'trying' for a baby for many years involving all sorts of complex procedures, in which case there is a long time spent in the life cycle before any conception takes place.

During each stage, a number of processes will be executed. A very common misconception is to confuse the terms *process* and *stage* which is a hang-up from historical life-cycle models, such as the waterfall model, where there was a simple one-to-one ratio between each stage and the processes executed within it. For example, many textbooks will still talk about a 'requirements' stage, where a single set of activities is carried out. In more up-to-date life-cycle models, such as an iterative-style approach, there will be many processes executed within a single stage.

Another common misconception is between the terms *life cycle* and *life-cycle model*. A life cycle is a structural concept and simply identifies a number of stages – in SysML this would be realized using a block diagram. A life-cycle model, on the other hand, is a behavioural concept that describes what actually is supposed to happen or has happened. This would be realized in SysML using an interaction diagram, such as a sequence diagram.

Life cycles and life-cycle models will be discussed in more detail in Chapter 6.

1.6.5 Projects

Systems would not come into existence without projects. Projects are the things that deliver products and services and every project will have a life cycle.

A project will deliver a number of products or services. In the case of a product, this is usually something tangible that many people will think of as a system. On the other hand, although a service may be viewed as being far less tangible, it is nonetheless a system that is being delivered.

Figure 1.12 shows that any project requires a number of resources to enable it, whether they are human or physical.

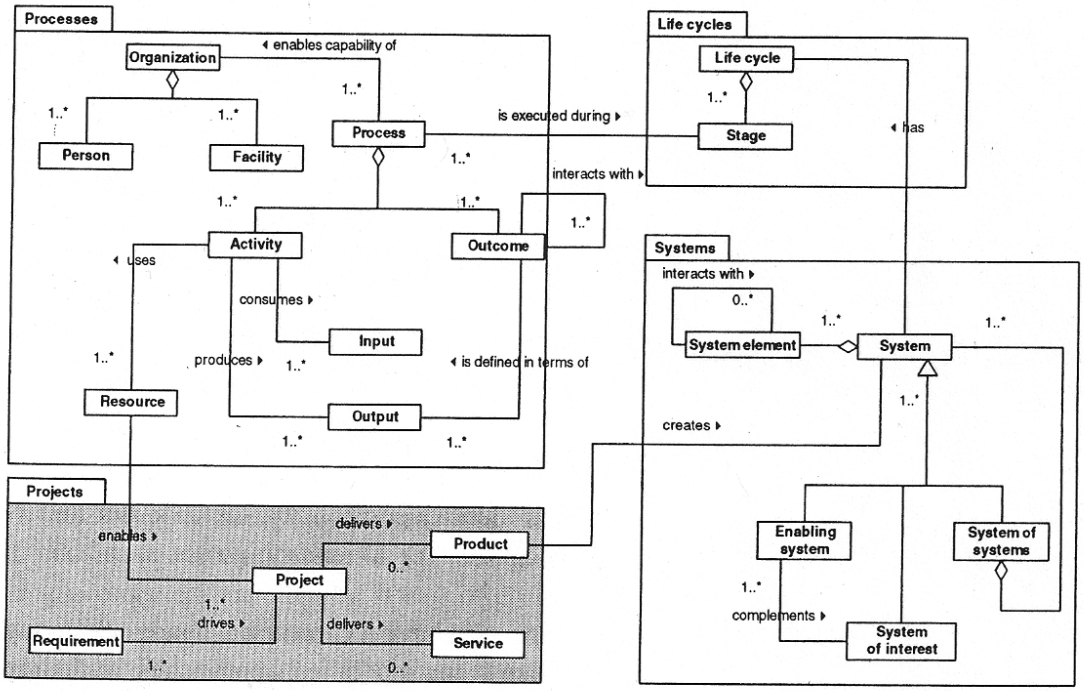

Figure 1.12 Systems engineering concepts with a focus on 'projects'

All projects are driven by requirements. This is a keystone to the whole of systems engineering and, indeed, is a massive field of work and research in itself. Requirements will not only drive the project, but are also a benchmark for quality, as all acceptance tests are based directly on requirements. There are many definitions for the term *quality* but two of the more widely used ones are: 'fitness for purpose' and 'conformance to requirements' [12]. Therefore, if the requirements are not fully understood, then the system cannot be validated properly and, hence, quality cannot be demonstrated.

1.7 Modelling

The three evils of systems engineering were introduced earlier in this chapter and one of the main thrusts of this book is the promotion of modelling. Indeed, modelling is one way that can help us address these evils, since modelling allows us to identify complexity, aid understanding and improve communication.

1.7.1 Defining modelling

In order to understand modelling, it is important to define it and (for a change) this is relatively straightforward. We define a model as a *simplification of reality* [13]. It is important to simplify reality in order to understand the system. This is because, as humans beings, we cannot comprehend complexity.

If a model is a simplification of reality, there are many things then that may be thought of as a model.

- *Mathematical* models, such as equations that represent different aspects of a system, that may be used as part of a formal analysis or proof.
- *Physical* models, such as mock-ups, that may be used to provide a picture of what the final system will look like or may be used as part of a physical simulation or analysis.
- *Visual* models, such as drawings and plans, that may be used as a template for creation or the basis of analysis.
- *Text* models, such as written specifications, that are perhaps the most widely used of the tools at our disposal. Regarding text as a model can be initially quite surprising but, the second we start to describe something in words, we are simplifying it in order to understand it.

This is by no means an exhaustive list, but it conveys the general message.

It is important to model so that we can identify complexity, increase our understanding and communicate in an unambiguous (or as unambiguous as possible) manner.

In order to model effectively, it is essential to have a common language that may be used to carry out the modelling. There are many modelling approaches, including graphical, mathematical and textual, but, regardless of the approach taken, there are a number of requirements for any modelling language.

1.7.2 The choice of model

The choice of model refers to the fact that there are many ways to solve the same problem. Some of these ways will be totally incorrect, but there are always more correct ways than one to solve the problem at hand. Although all these approaches may be correct, some will be more *appropriate* and, hence, more correct for the application. For example, if one wanted to know the answer to a mathematical equation, there are several approaches open: you may simply ask someone else what the answer is; you may guess the answer; you may apply formal mathematical analysis and formulae; or you may enter the equation into a mathematical software application. All may yield a correct answer, but the most appropriate approach will be application-dependent. If you were merely curious as to the answer to the equation, then guessing or asking someone else may be entirely appropriate. If, on the other hand, the equation was an integral part of the control algorithm for an aeroplane, then something more formal would be more appropriate.

It is important that we have a number of different tools available in order to choose the most appropriate solution to a problem, rather than just rely on the same approach every time. Therefore, one requirement for any modelling language is that it must be flexible enough to allow different representations of the same information to allow the optimum solution to be chosen.

1.7.3 The level of abstraction

Any system may be considered at many different levels of abstraction. For example, an office block may viewed as a single entity from an outside point of view. This is what will be referred to as a *high level of abstraction*. It is also possible to view a tiny part of the office block, for example the circuit diagram associated with a dimmer switch in one of the offices. This is what is known as a *low level of abstraction*. As well as the high and low levels of abstraction, it is also necessary to look at many intermediate levels, such as each floor layout on each level, each room layout, the lifts (or elevators), the staircases and so on. Only by looking at something at high, low and in-between levels of abstraction is it possible to gain a full understanding of a system.

Therefore, the second requirement for any modelling language is that any system must be able to be represented at different levels of abstraction.

1.7.4 Connection to reality

It has already been stated that, by the very nature of modelling, we are simplifying reality and there is a very real danger that we may oversimplify to such a degree that the model loses all connection to reality and, hence, all relevant meaning.

One type of modelling where it is very easy to lose the connection to reality is that of mathematical modelling. Mathematical modelling is an essential part of any engineering endeavour, but it can often be seen as some sort of dark art, because very few people possess a sufficient knowledge to make it usable and, indeed, many people are petrified of maths! Consider the example of the mathematical operation of differentiation, which is used to solve differential equations. As every schoolchild knows, differentiation can be applied in a quite straightforward manner to achieve a

result. What this means in real life, however, is another matter for discussion entirely. Differentiation allows us to find the slope of a line that, when taken at face value, and particularly when at school, can be viewed as being utterly meaningless. To take this example a little further, we are then told that integration is the *opposite* of differentiation (what is the opposite of finding the slope of a line?), which turns out to be measuring the area underneath a line. Again, when first encountered, this can be viewed as being meaningless. In fact, it is not until later in the educational process, when one is studying subjects such as physics or electronics, that one realizes that finding the slope of a line can be useful for calculating rate of change, velocity, acceleration, etc. It is this application, in this example, that provides the connection to reality and hence helps communicate 'why'.

The third requirement for any modelling language, therefore, is that it must be able to have a strong connection to reality and, hence, be meaningful to observers who, in many cases, should require no specialist knowledge, other than an explanation, to understand the meaning of any model.

1.7.5 Independent views of the same system

Different people require different pieces of information, depending on who they are and what their role is in the system. It is essential that the right people get the right information at the right time. Also, for the purpose of analysing a system, it is important to be able to observe a system from many different points of view. For example, consider the office block again, where there would be different types of people requiring different information: the electricians require wiring diagrams, not colour charts or plumbing data; the decorators require colour charts, not wiring diagrams; and so on.

There is a potentially very large problem when considering things from different points of view, and this is consistency. Consistency is the key to creating a correct and consistent model and, without any consistency, it is not possible to have or demonstrate any confidence in the system.

The fourth requirement, therefore, for any modelling language is that it must allow any system to be looked at from different points of view and that these views *must* be consistent.

1.8 SysML – the system modelling language

It has been established, therefore, that there is a clear need for a common language that can be used as an enabler for our systems-engineering activities. The most widely used language, to date, to achieve this has been the Unified Modelling Language, or UML, as it is widely known. However, there were many arguments why UML is suitable or not for systems engineering, but the irrefutable facts of the matter are these.

- The UML has been used very successfully for systems-engineering activities in many application domains for many years.
- There are some areas of the UML that can be improved upon.

If these two statements can be accepted as being true, then the SysML may very well be the language for you!

The SysML provides an excellent set of extension mechanisms to the UML that can be used to augment and enhance the basic capabilities of UML. There are some aspects of the SysML that are contentious and there are many strong arguments against, for example, leaving out major parts of the original UML modelling language.

It must also be pointed out that everything that can be modelled in SysML is an extension of what already exists in UML. In fact, it is possible to represent all SysML constructs in existing UML constructs, as is shown in Appendix B.

This book intends to look at all aspects of the SysML, both the good and bad, and allow the reader to decide which parts of the SysML will provide them with the most benefits.

1.9 Using this book

When using this book, it is important to remember that it is not, in itself, intended to be any sort of ultimate reference for systems engineering. It cannot possibly hope to cover the modelling of all the areas of systems engineering that are necessary to an organization. Nor can it cover every modelling concept within the Systems Modelling Language.

However, when used in conjunction with some of the information sources mentioned earlier, this book should form part of the greater systems knowledge.

In using this book as part of the greater systems knowledge, there are also a number of points to consider concerning the other information sources used.

- Both the handbook and ISO 15288 are evolving entities and, therefore, are subject to change. However, that said, it is unlikely there will be any major conceptual changes in either (one would certainly hope not), but it may be that much of the terminology and actual techniques used may change as time goes on.
- Both *The INCOSE Systems Engineering Handbook* and ISO 15288 are intended to be used as guides only and should not be taken as being carved in stone. Therefore, do not simply accept everything in them, but look for the parts that are relevant and suitable for the systems that you work on, rather than taking everything verbatim from them.
- Both represent best practice and, therefore, must be tailored to suit your requirements. Too many people want complete off-the-shelf solutions to all life's problems but, unfortunately, life is more complex than that. Indeed, if it were possible simply to pick up a best-practice manual that applied to everyone and solved all life's problems, there would be no need for systems engineering.

1.9.1 Competency

The world of competencies and competency frameworks is a varied and complex one. It is yet another term where there are many different definitions, but one thing that

is agreed by all sources is that, in order to carry out systems engineering effectively, competent people are needed to carry out the roles in a project.

At the highest level, competency may be seen as being able to demonstrate three things:

- *knowledge*, concerning application, and domain knowledge that is essential to understand our systems;
- *skills*, such as techniques and tools that are essential to be able to realize our project successfully; and
- *attitude*, such as behavioural and professional attitudes that allow engineers to work in an efficient manner and to understand the value in what we do.

It is the intention of this book to address all three at some level, so that people may become, in time, competent in applying the SysML to systems-engineering projects. There is only so much that a book can achieve, and an essential part of evolving one's competence from beginner to expert is to practise these techniques.

For those interested in competency frameworks, the one that is used as a basis for this book is the INCOSE competency framework.

1.10 Conclusions

This chapter has introduced, at a high level, the world of systems engineering. There were a number of key concepts that were introduced that were classified into four main groupings: projects, processes, systems and life cycles. It has shown how there are three evils that affect us all, which are complexity, a lack of understanding and communication issues. One way to address these three issues is to apply modelling. In order to model effectively, it is essential to choose a common modelling language that can be used by all relevant people. In order to choose an appropriate modelling language there are certain requirements that must be satisfied in that the modelling language must present a number of modelling options, be able to represent information at different levels of abstraction, have strong connection to reality and be able to represent the system from different points of view.

The modelling language that will be described in this book is the systems-engineering modelling language, or SysML, which is a visual modelling language, based on the UML that satisfies all the above-mentioned requirements.

The remainder of this book looks in detail at the language itself and then explores some common uses and applications for modelling.

1.11 References

1 International Standards Organization. 'ISO 15288, Lifecycle management – system life cycle processes'. ISO Publishing; 2003
2 International Council on Systems Engineering (INCOSE). 'INCOSE systems engineering handbook – a guide for system life cycle processes and activities'. Version 3.0; June 2006

3 International Standards Organization, op. cit.
4 Federal Aviation Agency (USA FAA). *Systems Engineering Manual* (definition contributed by Simon Ramo); 2006
5 Eisner, Howard. *Essentials of Project and Systems Engineering Management*. New York: Wiley; 2002
6 International Council on Systems Engineering, op. cit.
7 International Council on Systems Engineering (INCOSE). 'INCOSE competencies framework'. Issue 2.0; November 2006
8 Holt J. *UML for Systems Engineering: Watching the Wheels*. 2nd edn. London: IEE Publishing; 2004 (reprinted, IET Publishing; 2007)
9 Elk A. 'The brontosaurus sketch', *Monty Python's Flying Circus*. BBC TV; 1974
10 Pressman R. *Software Engineering: A Practitioner's Approach: European Adaptation*. Maidenhead: McGraw-Hill Publications; 2000
11 Brookes F.P. *The Mythical Man-Month*. Boston, MA: Addison-Wesley; 1995
12 International Standards Organization. 'ISO 9001, Model for quality assurance in design, development, production, installation and servicing'. ISO Publishing; 1994
13 Booch G., Rumbaugh J. and Jacobson I. *The Unified Modelling Language User Guide*. Boston, MA: Addison-Wesley; 1999