C2 The Systems Engineering Approach to Handling Complex Engineering Projects

C2.1 Overview

In the preceding chapter we identified the main sources of complexity in engineering projects, and in Sec. B2.3 we subdivided projects into a number of stages. We can now relate the two, in the sense of determining which sources determine the complexity in the various stages and thereby determine the complexity of the objects created within each stage. These objects are, as we know from Sec. B2.4, processes and the artefacts and descriptions resulting from them, and while our approach to handling them has a common basis, as was already foreshadowed in Ch. A5 and will be developed in more detail in the next section, there are also significant differences, as we shall discuss later in this chapter.

The matrix below is an attempt to indicate in which stages the various sources *introduce* complexity. The emphasis on "introduce" is important, a stage may well *reflect* the complexity of a source, but it was introduced in an earlier stage. And, with the red colour signifying a very significant contribution and the blue colour signifying significant contribution, we recognise the imperative of addressing the complexity of engineering projects in the concept stage. If complexity is not addressed and handled as soon as it is introduced, the result is likely to be, at best, cost and schedule overruns and, at worst, useless or abandoned projects.

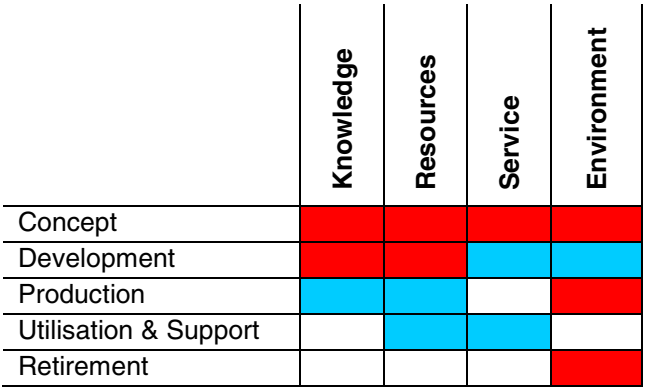

Fig. C2.1 Where and from what sources complexity is being introduced into projects, with the red colour indication very significant, and blue colour indicating significant, levels of complexity.

In order to interpret the above matrix, it is necessary to think of the engineering activity in each stage as a three-step process of problem definition, option identification, and solution selection and documentation. Under this perspective, the extent and, above all, the rapid expansion of the knowledge base (i.e. a large number of options), introduces a high level of complexity into the concept and development stages, but once it is addressed in these stages, only a modest further amount of complexity is introduced in the production stage, and in the following stages the expansion in the knowledge base will, if anything, reduce the reflected complexity (i.e. the work associated with carrying out the planned activities).

The complexity of the service requirements needs to be taken into account in the very earliest part of the project; defined, analysed, and transitioned into requirements on the functionality of the plant. The further development of the design through to data for production or construction can be a complex process, but much of the groundwork for handling this complexity, in particular the structuring (or architecting), should already have been carried out in the concept stage.

The complexity introduced by the environmental requirements also needs to be addressed in the concept stage, but many environmental requirements are focused on the manufacturing and construction processes. And, because the retirement stage comes tens of years after the plant was designed and constructed, the requirements on decommissioning will often have changed (increased) considerably (just think of the requirements on demolishing plant containing asbestos or nuclear material).

Following on from where we left off in the previous Chapter, the matrix in Fig. C1.1 can also be interpreted as showing the relationship between the "external" complexity, represented by the sources, and the "internal" complexity, represented by the stages, and this leads us to the view of the "internal" complexity as belonging to something that is *designed* in *response* to the complexity imposed by the "external" sources. That is, while the concept of complexity remains the same, the characterisation as "internal" or "external" has to do with the degree of control we have. It is important to *understand* the "external" sources of complexity, but when we now turn to the design and management of the engineering process, it is the "internal" complexity we have to be concerned with.

We also discussed, briefly, the difference between complexity and risk, and noted that, while they are coupled in the sense that increasing complexity leads to an increasing number of hazards (i.e. things that can go wrong), risk involves both the probability and the consequence of such a hazard. Both of those parameters are not intrinsic to the objectives of a project and their complexity, but determined mainly by how we carry out a project. The key to handling the risks associated with complexity is therefore careful and detailed *planning*. It is my experience that many of the causes of unsatisfactory project outcomes can be traced back to inadequate planning. Issues become problems only because no thought have been given to how to resolve them and what resources will be needed, and then, more often than not, a "quick fix" is the result. Some contractors even put their ability to "get stuck right into it" and do projects "on the run" forward as a desirable

quality. The approaches to handling complexity put forward in the following sections all assume that they will be applied within a rigorous planning framework; anything else will reduce their effectiveness greatly and often render them purely cosmetic.

C2.2 Plant and Work–Two Complex Entities

It is beneficial to view an engineering project as containing two complex entities; the plant to be created and operated in order to satisfy the stakeholder requirements, and the body of work required to create and operate it. They are often called "the system" and "the project", or also "the Works" and "the Work"; we shall stay with our previous nomenclature of plant and work. These two entities are obviously tightly coupled, but they are of very different natures. The relationship is somewhat analogous to that of a glove and a hand; the size and shape of the glove is determined completely by the hand, but they are otherwise very different.

The approach to handling any complex entity is always the same: describe it as a system of less complex, interacting elements. However, this is not generally as straight-forward as it sounds, as there is usually a choice of elements and of the way in which they interact, so that we are faced with three problems: Identifying sets of elements that, when interacting, can represent the complex entity; for each set determining which set of interactions (i.e. what structure) provides the best representation; and, above all, formulating an operational definition of what "best" means. The obvious solution to the last problem is to define the best architecture as the one that provides the greatest reduction in complexity, and while this is, indeed, the underlying criterion in most cases, it is too general to be an operational criterion, and it masks practical aspects that depend on the nature of the complex entity and what we want to do with it. In particular, we need to have a very clear understanding of the difference between the plant and the work, and we can start out by recalling the development of an engineering ontology in Sec. B2.4. In terms of the top level categories in Fig. B2.4, the plant is an object and the work is a process; that is, the plant is a continuant and the work is an occurrent. Work is defined completely in terms of what it does; it has no substance, nothing that exists when the work is not being carried out. The plant, on the other hand, has both an enduring physical presence and a functionality; it *is* something and has the ability to *do* something; it has a dual nature. With a bit of simplification, we might say that the work exists in time, whereas the plant exists in space. This difference is reflected in the manner in which the system approach is used to handle complexity in the two cases. Let us first look at what it means for the work.

The description of the work as a set of elements is captured in the Work Breakdown Structure (WBS), where the elements are Work Packages (WPs), defined in terms of their results or outputs, the activities required to produce these outputs, and the resources or inputs required to support the activities and complete them in a certain time period. However, it is often overlooked that the word "structure" has two quite different meanings here. The one directly linked to

"breakdown" arises because the work is broken into WPs in a step-wise fashion; first the whole body of work is broken down into a few main, very large WPs. Then, in the next step, each of these are broken down into smaller WPs, and then, in the next step, each of these are again broken down into smaller WPs, and so on. This is often represented graphically as a tree structure, but this is not the structure involved in representing the work as a system. To describe the work as a system, we choose a complete set of elements (i.e. that cover the whole body of work, but not necessarily all from the same level of the breakdown) and define the interfaces between them; there are a number of well known ways to do this, such as PERT diagrams, N^2 matrices, etc. There is a choice of elements and structure within the constraints imposed by the overall outputs required as a result of the work, the inputs present at the beginning of the work, the overall timeframe, and the resources available. In the next section we shall look at some rules for developing the "best" WBS.

Describing the plant as a system is also generally a step-wise process, but due to its dual nature mentioned above, it is best viewed as two parallel and closely coupled processes. One is the development of the description of the plant in terms of its components. Depending on the industry and the type of project, the resulting artefact has various names, such as Bill of Materials, Schedule of Works, or Works Definition Document (WDD), but in any case it should fulfil two essential requirements:

a. As it is developed in steps, from the description as a single component, the plant, to descriptions in terms of more detailed components, each level must be a *complete* description of the plant. When a component at one level is broken down into a set of smaller components at the next lower level, that set of components is equivalent in all respects to the original component.
b. The document must reflect this step-wise process. It must not be just a simple list of all the components on the lowest level, but define the components at each level and display their hierarchical ordering, i.e. the Plant Breakdown Structure (PBS). This is analogous to the WBS.

The other process is the development of the set of requirements on these components. It is contained in documents with various names, such as Technical Requirements, Requirements Definition Document (RDD), or Technical Specification. This document (or set of documents) is developed in parallel with the PBS, and as with the PBS, there is a requirement for completeness at each level. However, this "completeness" is partly explicit and partly implicit, with the fraction of requirements that are explicit increasing with each step in the development until, at some level, all the requirements at the top level, i.e. the stakeholder requirements, are explicitly satisfied by the requirements on the components at this level.

In Sec. C2.4 we look at some issues involved in developing the "best" PBS.

In the application of the systems approach to both the plant and the work there is a step-wise "un-hiding" of requirements as the partitioning into more detailed elements progresses. This is one side of the manner in which the systems approach handles the complexity of engineering projects. The other side arises from the fact that at each level of the top-down process, the components are not a collection of unrelated components, but interact in a particular way to form a *system* with a particular structure; this system can have properties, the *emergent properties* we discussed in Sec. A4.1, that are not present in any of the components. As the overall complexity is a combination of the complexity of the elements and the complexity of their interactions, the art of system design is to reduce this combined complexity, not just the one or the other.

However, it must again be emphasized that there are two very significant differences between the plant and the work that have greatly influenced the extent of development and approach to their execution. They will also determine the further course of our development of the application of the system concept to engineering. The first one is that while plants can differ substantially in almost all aspects, from the nature of their product to the technology employed and the size and structure of the physical realisation, the work is basically always quite similar. Not in the details and extent of the work, but it always consists of the same type of activities, has the same structure, uses the same management processes, and so on. As a result, the handling of the complexity of the work has been developed into a well-established set of processes; this is the application of the system concept to the management of engineering projects, as we have alluded to earlier, and any inadequacy or inefficiency in the execution of the work is more often due to not following these processes than to anything else. The process of designing the plant is much less well developed; while there are a number of design methodologies, few, if any, apply the system concept in a consistent manner.

The second difference is that in the case of plants, we can consider what a plant must do, i.e. its functionality, without considering a particular physical realisation; a plant has an "image" in both the functional and the physical domain. That is not the case with work; as we mentioned above, work is defined completely in terms of what it does; it has only this one "image".

Because of these two differences, the subsequent chapters will focus on exploiting the existence of the functional domain and the consistent application of the system concept in that domain as a means of handling the complexity in designing a plant to meet a demanding set of stakeholder requirements.

C2.3 Work Breakdown Structure and Contracting Strategy

As we saw earlier, the Work Breakdown Structure (WBS) for a project is a hierarchical ordering of Work Packages (WPs), with the whole body of work as a single WP at the top level, and as a set of WPs at whatever level is deemed appropriate for each part of the work. The WPs at this lowest level of breakdown are sometimes called *tasks*. However, connected to what might look like a simple (but structured) tabulation of WPs is a number of central features of the project.

This central position of the WBS in the project explains why it acts as the *key* to the data base that contains the data defining the various types of entities constituting the project.

Firstly, with each WP is associated an *effort*, measured e.g. in the number of person-hours (estimated or actually) required to perform the work (this may be subdivided into disciplines). This provides a relationship between resourcing and *duration* of the WP. Secondly, besides manpower, many WPs will require other *resources*, such as computer time, manufacturing facilities, construction equipment, test facilities, and so on. Thirdly, with each WP we can associate a *risk profile* in terms of the maturity of the technology employed, the experience of the allocated personnel, etc, and there are numerous other fields that can be added to this project data base. Finally, as each WP is defined in terms of its outputs and the inputs required to deliver these outputs, this defines the *interfaces* between the WPs (and the interaction of the project with its environment, in the form of project inputs and deliverables). This then provides the description of the work as a system, i.e. as a set of elements with particular interactions and thereby a particular structure. This, and not the hierarchical structure of the WBS, is the structure of the work as a system, and the temporal aspect of this structure is what is usually called the project *program*.

Now, to address the question of what is the "best" representation of the work as a system, it is useful to think of the work as consisting of two types or categories of work: the work in the narrowest sense; i.e. the work directly involved in creating the plant, such as the actual design and construction activities, and the work involved in the management of these activities. This separation is also often reflected in the program, in that the WPs containing activities of the first type have definitive durations, whereas many of the WPs containing the management activities are tied to the duration of the phases or stages of the work. The point of this separation is that the totality of the work in the first category is, at least to a first order, independent of how it is subdivided into WPs, whereas the management work is highly dependent on the particular representation of the work as a system, and so an obvious initial choice for the criterion for "best" system is the one that minimises the management effort, which leads directly to a number of simple rules:

1. The breakdown of a WP should be according to a single criterion only. The only exception to this rule is at the first level, where management is normally separated out from the rest of the work (the work being managed).
2. Minimise the number of interfaces and the number of parameters involved in defining each interface. This can also be phrased as requiring the WPs to be as *self-contained* as possible.
3. Ensure that the management responsibilities (i.e. completion on time and within budget) are mapped onto the structure of the technical responsibilities (i.e. meeting the technical requirements) to the greatest extent possible. This means that, at the lowest level of the WBS, i.e. at the task level, there should be no distinction between management and technical responsibility; one person is responsible for both.

4. A corollary to this last point is that tasks should be discipline based.
5. Group the low risk activities into the same area of the WBS and the program, thereby allowing the management effort to be more focused and the change management effort to be minimised.
6. Examine the sequential vs. parallel character of the structure to ensure that it reflects how it is intended to carry out the work.
7. The size of the tasks (WPs on the lowest level), i.e. the extent of the breakdown, should reflect the desired progress monitoring accuracy. If a task is properly defined, it is easy to determine its completion; estimating progress within a task is always somewhat subjective.

Minimising the management effort is not the only criterion that can apply; depending on the project, such criteria as minimising the completion time or maximising local content may play a dominant role in determining the work structure. In general all such criteria will have to be considered and given an appropriate weighting.

However, overlaid on all of these considerations are a number of commercial considerations that impose a structure on the work in the form of a *contracting strategy*. Engineering projects are carried out within a contractual framework between the various bodies involved in the project, such as the Owner, the Engineer, the Constructor (or Contractor), various Suppliers, the Operator, the Maintainer, etc. The contracts and the interfaces between them form the top level representation of the work as a system, and for many large projects, such as major infrastructure projects, this system can already be complex enough to warrant a formal system design approach [1]. The rules given above will still apply to a certain extent, but there are also some additional considerations:

1. The financing structure. If the Owners are financing the project off their balance sheet, they have a free hand in choosing the contracting strategy, but if there are debt providers involved, they will often impose certain requirements. Alliance Contracts may have very individual contract structures, Public Private Partnerships (PPPs) have other requirements, and so on.
2. The availability of an adequate number of contractors with all the required competencies in order to ensure competitive bidding. There is no sense in asking firms to bid, of whom it is known that they are already fully (or over-) committed.
3. The ability of the Owner or Proponent to manage contracts.
4. The need for highly specialised abilities. This may make it unavoidable to engage a number of smaller firms.

The “best” contracting strategy is the one that results in the lowest cost when all factors are considered, including the Owners’ total risk exposure. And when one considers the performance record of complex projects, such as large software projects [2] or large infrastructure projects [3], it becomes obvious that minimising the risk exposure may be at least as important as minimising the cost. Through

their interactions or interdependencies, the contracts form a complex system, and the Owners' risk exposure is an emergent property of this system. As with any system property, it is determined by both the properties of the individual elements, in this case the likelihood of non-performance of the individual contractors, and the interfaces between them, and for a particular system implementation, a performance model can be developed in a manner quite similar to that used in the well-known FMECA [4]. Various possible implementations can then be considered in order to determine the "best" one.

C2.4 Developing the Plant Breakdown Structure

As already discussed, the Plant Breakdown Structure (PBS) is a structured description of the physical (or spatial) characteristics of the elements making up the plant, without specifying their performance characteristics; it may be viewed as a structured, high level Bill of Materials. As such, it provides a check list against which the Owners can check that they are receiving everything that was agreed; that the *extent* of the plant is correct at handover. The nature of the PBS is different to that of many other project artefacts, in that it may be viewed as a "book-keeping" artefact, completely void of any engineering content. Of course, that is a highly simplified view, in that the Owners would often be heavily influenced by advice from the Engineer in what they ask for, but in the end, the resulting PBS is the Owners' "shopping list".

However, in addition to its importance as a structured check list, both in contractual terms and, as we shall discuss in a moment, as a dimension of the project data base, it is the progressive development of the PBS that is such a valuable process. Firstly, depending on the starting point of the project, the Owners' perception of what they want to end up with at the completion of the project will vary greatly in both justification and level of detail. Consequently, there is a process of questioning and probing to be completed before a clear picture emerges of what is required to meet the Owners' initial requirements.

Secondly, once an initial version of the PBS has been agreed and the project gets underway, both the Owners and the Engineer recognise omissions and opportunities for improvement and, depending on the stage of the project, the PBS is developed in greater detail as the design progresses. In any case, the PBS is a living document and provides an up-to-date record of the agreed extent of the plant, and it becomes a check list for the updating of any activities and artefacts that are linked to it. In particular, one or more levels of the WBS would normally be structured on items in the PBS, so that, for example, if it is agreed to add another production unit or another access road, corresponding tasks, such as the design, construction, and assurance of these items must be added to the WBS, and from there the changes propagate into budget, resource allocation, program, etc.

Just as was the case for the WBS, there are some useful rules for developing the PBS, and they are (for obvious reasons) quite similar to those for the WBS:

1. At some level, the breakdown should reflect the contracting strategy. For example, if the plant consists of a number of production modules, each involving building services, and a contract is to be let for building services plant wide, then building services should be a separate physical element, subdivided into production modules. However, if a contract is to be let for the production modules, then the modules will be a separate entity, subdivided into such parts as building services.
2. Within a contract, the partitioning into elements should reflect well-understood areas of competence and responsibility within the organisation.
3. The partitioning should be chosen so that the interfaces are minimised and, where possible, in accordance with industry interface standards. This applies to information exchange interfaces as well as physical interfaces, such as voltage levels, flange sizes, standard packaging (container) sizes, etc.
4. Examine the use of Commercial-Off-The-Shelf (COTS) elements at the first opportunity in the development of the PBS, as there would have to be a robust cost-effectiveness argument (taking development risk into account) for choosing anything else.

C2.5 The Project Database

The project database is the most direct manifestation of the description of the project as a system; it is the structured collection of all project-specific data, such as drawings, reports, specifications, plans, calculations, and software programs, and each item is an electronic *file*. The files fall into two categories: those arising out of, or primarily associated with, a management activity, which we shall call *management files*, and the rest, which we shall call *production files*; this separation is due to the very different structures of the management activities and of the other tasks, as mentioned earlier. And furthermore, while files in both categories are identified by the associated WBS number, the production files have a second identifier – their associated PBS numbers. These two identifiers can be thought of as two coordinate axes spanning a surface, and each file represented by a point of this surface, but the two coordinates are not completely orthogonal because, as mentioned, it is common for the partitioning criterion on one or more levels of the WBS to be identical to the partitioning on one or more levels of the PBS. For example, if the project involves the design of a plant consisting of a number of modules, then a file arising out of the design of a module (say, a specification) will be identified by the WBS number of the WP for the design of the module and by the PBS number of the module; any other PBS number would be inadmissible in combination with this WBS number.

C2.6 Standards and Current Practice

C2.6.1 A Note on The Influence of Software [5]

Software science and software engineering have been the originators of much of what we call systems engineering. This occurred, on the one hand, because software programs soon became so complex that their development became natural applications for the systems approach and, on the other hand, because the abstract nature of software made it relatively easy to apply the abstraction that is central to the system approach. However, if we take systems engineering to literally mean the "engineering" of systems, then we need to recognise that software development is very different from engineering and, as a consequence, carefully consider to what extent some of the software-oriented developments in systems engineering are applicable to, or appropriate for, the broader scope of engineering.

Engineering can be viewed as consisting of two distinct, but closely coupled sets of activities; the development of technology based on natural science, and the application of this technology to meet the needs of society. That is, the end objective of engineering is to create objects that provide services, and a century ago that definition would not have raised any questions. The objects were machines, boats, bridges, substances, etc., and the services they produced were clearly identifiable, even if they were embedded in a greater collection of objects providing the ultimate service. As an example, consider a newspaper. Its service has many aspects, such as bringing information to its readers, allowing businesses to reach their customer base through advertising, and so on. Producing this service requires the interplay of many elements, such as the paper, the printing machine, the journalists, and so on, but there would be no doubt about characterising the printing machine as an engineered object and a story in the paper as not. Nobody would call journalism engineering or characterise the writing of a story as text engineering.

With the advent of computers, a new dimension was introduced, in that the engineered object now consisted of two distinct parts, the hardware and the software, and they exist in a sort of symbiotic relationship, in that each part is useless by itself; only together do they provide a service. However, the nature of this relationship depends on the application; in a simple application, such as the interlocking or automation of a piece of machinery, the instructions that make up the software are identical to the hard-wired connections in a corresponding relay logic, and we would not have any hesitation in calling the development of these instructions an engineering task. But the fact that the development of the instructions or the wiring connections is carried out using a formalism, Boolean algebra, that is part of mathematics, does not make mathematics a part of engineering. We do not speak of "mathematics engineering"; engineering uses mathematics, just as we use the laws of physics, and developing the instructions using Boolean algebra is no different to using calculus to determine the strength of a shell.

If we now move to more complex applications, as the development of an accounting program or a program for calculating the strength of a shell, the relationship between hardware and software has almost disappeared. It is certainly still true that without a computer to run on the software is useless, but the developer does not have to give much, if any, consideration to the characteristics of the hardware. As long as he obeys the rules of the programming language, his program can be compiled to run on any computer. And he does not have to have any domain-specific knowledge; in the case of the accounting program, accountants will specify what the program must do and the rules to be obeyed, in the case of the shell program, engineers will specify what it must do and provide all the equations etc. The situation is quite analogous to that of the journalist writing a story; he does not have to consider what paper it will be printed on or the characteristics of the printing machine, his skill lies in describing the observed facts in a manner that will meet the readers' needs, while adhering to the rules of the language. So, why do we call the activity of the software author software *engineering*, when we do not associate any engineering with the journalist's activity?

This question is not semantic nit-picking nor an attempt at demarcation; it arises out of a concern over the current direction and focus of systems engineering, as exemplified by the recent revision of the ISO standard, ISO 15288 (2002 vs. 2008), and the adaptation of a software modelling language, UML, to systems, as SysML. To me, this focus seems to ignore two fundamental differences between systems engineering and software development; the level of abstraction in the subject matter itself, and the level of creativity. Engineering is about successful outcomes, and where success is measured not by one's peers, but by the stakeholders. Both the problems in executing the projects and the measures of success are often largely of a non-technical nature, and the complexity that systems engineering is called upon to handle is only partly due to the advanced technology required or the multidisciplinary nature of the project. Factors such as market forces, financial backing, politics, union pressures, environmental interest groups, personal preferences, etc. are always significant and often dominating, and these are not factors that are effectively handled by introducing any formalism or abstraction. They require extensive communication with the diverse bodies involved, and natural language is the only realistic option for this. Furthermore, the engineering of hardware is about the engineering of real, physical objects, and that activity has its own elements of abstraction and associated formalisms and languages, in the form of engineering drawings and diagrams, architectures, and ontologies (to use a very *in* word, even if a bastardisation of the philosophical meaning). Software, on the other hand, is already an abstraction; software development can be thought of as a process for expressing given dependencies between actions (or variables) in a language that can be executed by computers, and as such software development lies somewhere within the triangle spanned by mathematics, logic, and linguistics.

Not that this process does not require great skill, but it is a skill analogous to that of the journalist; of choosing the right words and constructing sentences and paragraphs that produces the correct understanding in the reader's mind of the

situation being described. It also involves creativity, but it is creativity on a completely different level to that involved in engineering. Whereas creativity in engineering is concerned with creating possible solutions and then selecting the best one of these, the creativity in software development is concerned with giving the computer the most correct and efficient "explanation" of what it has to do as its part of the solution. It involves no interaction with stakeholders at all.

Both engineering and software development can be highly complex processes, and the system approach is equally applicable in both cases. In the case of engineering, the application is called systems engineering; in software development it is called something like structured programming, and while some of the basic features are the same, as is to be expected, given their common origin, there are also many features that are specific to the different natures of the two domains, as discussed above. In particular, within a given engineering project, they are located in quite different parts of the process, as illustrated in Fig. C2.2.

It is also worth while noting that the relative effort expended on engineering and on software development can vary greatly from project to project. On a 2 billion dollar freeway project the software development component may account for 5 million dollars and the engineering 100 million, whereas on a project to develop a new ERP module, there may be no engineering at all. In the latter case, the requirements are developed by business analysts, and the hardware platform is existing. This variability is a further indication of the relative independence of the two activities, and dispels the notion that they are two sides of the same coin and therefore need to have a common approach.

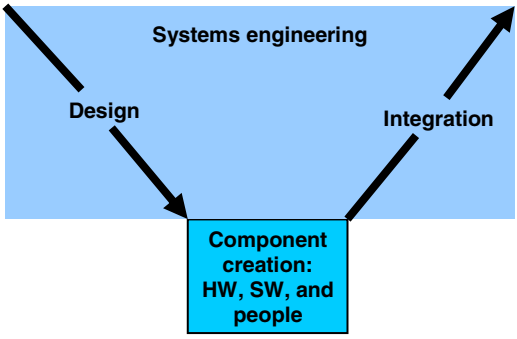

Fig. C2.2 Systems engineering and software development are located in different areas of an engineering project.

In summary, then, the significant differences between engineering and software development in the nature of both their products and their activities give rise to doubts about the current trend in systems engineering of adopting methodologies, representations, and tools from software development without a critical examination of their applicability. You should keep this in mind when assessing the applicability of the standards and current practice to your projects.

C2.6.2 Standards

Over the last fifty year or so, a number of standards and guides have addressed systems engineering, or aspects of systems engineering. Some of these have evolved and are still in use today; others have been superseded and are mainly of historical interest. The early development of systems engineering, at Bell Labs, initially for telecommunications and subsequently for some major defence projects, was supported by various company-internal and project-specific documentation, but the first publicly available standard was MIL-STD-499, *Systems Engineering Management*, released in 1969. The title is interesting, as it reflects the initial priority of the systems engineering effort: that of handling the complexity involved in *managing* these large projects and responding to the imperatives of the Cold War arms race. The title was maintained in the first revision, MIL-STD-499A, released in 1974, but in 1992 a draft version of the second revision, MIL-STD-499B, was released for comment with the title *Systems Engineering*, reflecting the importance placed on doing systems *engineering*, rather than just managing it. However, this version was never released, and the US DoD changed its policy to rely on commercial standards where applicable standards were available. Two such standards appeared in draft form at that time, EIA 632, *Processes for Engineering a System*, and IEEE Std 1220, *Standard for the Application and Management of the Systems Engineering Process*. (For completeness, it should be noted that the US Air Force Space Command issued version C of the standard in 1995 for its own use in supporting materiel acquisitions.)

It is also appropriate to mention the significant contribution of the US Defense Systems Management College, in particular through the publication of the *Systems Engineering Management Guide* (1990), and the impact of the book *Systems Engineering and Analysis*, by B.S. Blanchard and W.J. Fabrycky, first published by Prentice-Hall in 1981and now in its fifth edition.

Numerous other organisations in the defence and aerospace industry, such as NASA and the European Cooperation for Space Standardization, have published their own systems engineering standards, and the Australian DoD issued its *Capability Systems Life Cycle Management Manual* in 2002.

In 2002 the International Standards Organisation issued ISO 15288, *Systems engineering – System life cycle processes*. It was updated in 2008, with its title change to *Systems and software engineering – system life cycle processes*, and it has now become the central standard for systems engineering. As the title indicates, it presents systems engineering as a collection of processes, which results in a heavy emphasis on management aspects. Through the intent of the 2008 revision to harmonise it with ISO 12207, the standard's direct applicability to non-software-intensive industries has perhaps been somewhat reduced. Nevertheless, with judicious tailoring, the standard provides the framework for the application of systems engineering in most industries.

However, when applying ISO 15288 to any organisation, be it a permanent organisation, such as a company or a government body, or a temporary project

organisation, there are a couple of issues that need to be considered. The first is that, as with many other ISO standards, in particular ISO 9001 *Quality Systems*, ISO 15 288 defines general features of the processes that must be met in order for an organisation to be able to claim conformance with any one of those processes; it does not say anything about the extent of the processes. Secondly, it says nothing about the procedures and tools that would be required to support an efficient execution of the processes.

C2.6.3 Current Practice

The heading of this section, "Current Practice", is a convenient short-hand for the headings of a number of more detailed, but related topics, and perhaps the most immediate one could be phrased as "What do most of the people who consider themselves to be systems engineers actually do?". The answer to this is found by considering the evolution of systems engineering. We touched briefly on this in Chapter B1, where we noted that a formative influence on systems engineering was its role as an enabler in the arms and space race of the Cold War. As such, considerable resources in the aerospace and defence industries and their government clients were invested in developing a framework of processes and procedures that would significantly increase the probability of completing complex projects to specification and within tight timeframes, and it resulted in a Body of Knowledge (BoK) embodied in numerous specifications, Data Item Descriptions (DIDs) [6], text books, and guides, as well as numerous courses provided by universities and other training institutions. It is the practice of this BoK, which we might call *classical* systems engineering, that still today constitutes what the majority of systems engineers do, and the reason is that the aerospace and defence industries are the only industries where a systems engineering framework is mandated, and therefore where the majority of professional systems engineers are found.

There is no single, authoritative guide to this classical BoK, although such products as the previously referenced *Systems Engineering Handbook*, published by the International Council on Systems Engineering (INCOSE), and the book by Blanchard and Fabrycky contain references to a significant part of the literature.

Notes and References

1. The complex contracting arrangements in a number of large infrastructure projects are discussed in section 6.7 of Managing Large Infrastructure Projects, ref. 3 below
2. A well-known assessment of the success of software project is the CHAOS Manifest of the Standish Group, `http://standishgroup.com`, but it has also been criticised by a number of sources, many of which can be found by simply Google on "standish report". However, there seems to be no doubt about the fact that large software projects have had a relatively poor rate of completion on time, within budget, and to agreed performance criteria, whatever may be valid reasons for this

3. The book Managing Large Infrastructure Projects, published by A.T. Osborne BV, 2008, is a most interesting and valuable documentation of lessons learned on 15 major infrastructure products in Europe, with the findings clearly organised into eight groups. The study specifically addresses Project Management, but because these are large and very complex projects, many of the problems encountered are those that arise in complex systems in general
4. Failure Modes, Effects, and Criticality Analysis (FMECA) is a well-established process, documented in numerous textbooks and articles, and supported by different many tools. The best introduction is to look it up in Wikipedia, `http://en.wikipedia.org/wiki/Failure_mode,_effects,_and_criticality_analysis`
5. This text appeared in the Newsletter of the Systems Engineering Society of Australia (SESA), No. 48 (July 2009); under the title Why Software is Different
6. Data Item Descriptions (DIDs) were originally defined as part of MIL-STD-498, which consisted of two parts, Overview and Tailoring Guidebook and Application and Reference Guidebook, and aimed at software development. However, the concept has proved to be of enduring value and applicability, and current defence contracts often require compliance with a large number of DIDs