2
REQUIREMENTS ENGINEERING FRAMEWORK

2.1 INTRODUCTION

Before considering a detailed requirements-engineering methodology in the remaining chapters, this chapter provides an overview of a number of the important aspects of the requirements-engineering body of knowledge [1].

2.2 NEEDS AND REQUIREMENTS

When describing a system, we make the distinction between 'needs' and 'requirements and, as illustrated in Figure 2-1, there are needs and requirements at a number of levels. There are four major views: an enterprise view, in which enterprise leadership set the enterprise strategies and concepts of operations; a business management view, in which business management derive business needs and constraints as well as formalize their requirements; a business operations view, in which stakeholders define their needs and requirements; and a systems view, in which system designers define the system in logical and physical views. Because terms such as system and subsystem (system element) are level-specific, we define an *entity* "… a single thing to which a need or requirement refers: an enterprise, business unit, project, system, or system element (which could be a product, process, human, or organisation)" [2].

As shown in Figure 2-1, the enterprise, business management, and business operations views are in the problem domain; the system and system element views are in the solution domain. As discussed in Chapter 1, the problem domain is generally considered to be the responsibility of those who have ownership of the problem to be solved, so the descriptions of the system are predominantly in the language of the customer's business management and business operations, focusing on what the system needs to be able to do, how well it should be done, and why. We saw in Chapter 1 that these descriptions are called *logical* (or often *functional*) descriptions. On the other hand, the solution domain is generally considered to be the responsibility of the developer implementing the solution, most commonly a contractor, so the descriptions of the system in that domain are predominantly in engineering and physical terms, focusing on how the problem is to be solved—that is, how the solution will look once it has been implemented. In Chapter 1 we saw that these latter descriptions are called *physical* descriptions. In many large organisations, there is a soft boundary between the problem domain and the

solution domain in that the customer prepares a draft SyRS which is taken to tender, responded to be tenderers, negotiated with the successful tenderer (the preferred contractor) and then revised to be the SyRS that represents the selected solution of that preferred contractor.

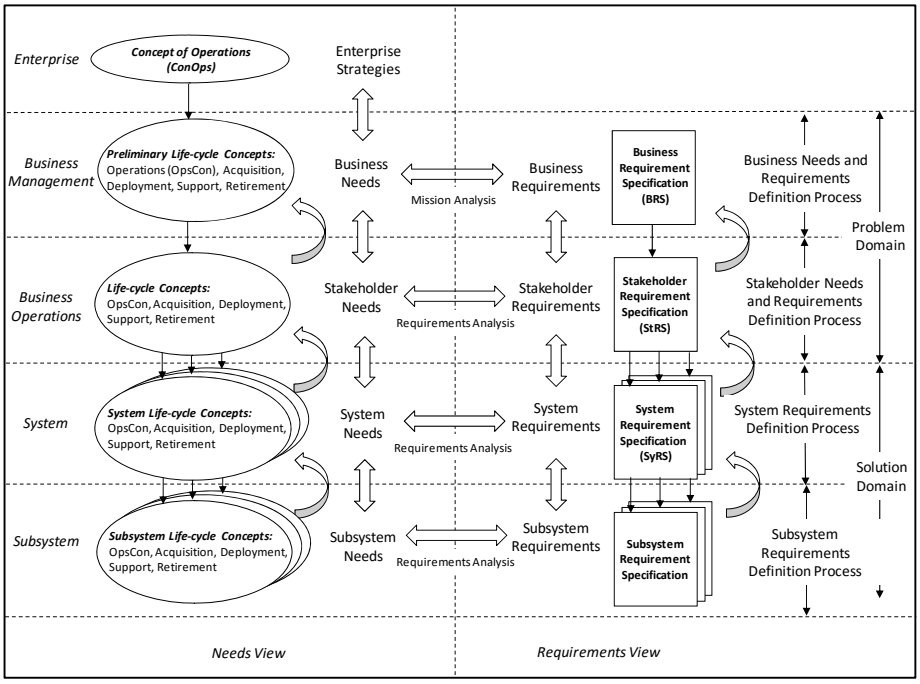

Figure 2-1. Views, domains, processes and artefacts associated with the definition of needs and requirements [3].

In addition to these vertical views of the process, there are two horizontal views shown in Figure 2-1: the *Needs View* and the *Requirements View*. While they tend to coexist at each level of consideration, *needs* and *requirements* are different:

- A *need* is "… the result of a formal transformation of one or more concepts into an agreed-to expectation for an entity to perform some function or possess some quality (within specified constraints)" [4]. Needs are generally capabilities stated as business needs in the language of business management, as stakeholder needs in the language of stakeholders at the business operations levels, or as needs of the system or system element.
- A *requirement statement* is "… the result of a formal transformation of one or more needs into an agreed-to obligation for an entity to perform some function or possess some quality (within specified constraints)" [5]. Requirements are therefore formal statements that are structured and can be validated. There may be more than one requirement that can be defined for any need. Requirements are

elaborated from needs through a process of requirements analysis (also called business analysis [6] or mission analysis [7] at the highest level of application).

At the highest level, there is an enterprise view in which enterprise leadership sets business vision, strategic goals and objectives in the form of a Concept of Operations (ConOps), which is also called a Strategic Business Plan (SBP). At the next two levels levels, there are the dual views: at the business management level there are *Business Needs* and *Business Requirements*; and at the business operations level there are *Stakeholder Needs* and *Stakeholder Requirements*. The transformation of needs into requirements occurs at each level. Here we are interested in the three upper-level major processes:

- *Business Needs and Requirements (BNR) Definition Process.* Business management start with the guidance contained in the ConOps, from which they define Business Needs, largely in the form of Preliminary Life-cycle Concept Documents (PLCD) which capture the Preliminary Acquisition Concept, Preliminary Operational Concept (OpsCon) [8], Preliminary Deployment Concept, Preliminary Support Concept, and Preliminary Retirement Concept. Business Needs are elaborated and formalized into Business Requirements, which are documented in the *Business Requirements Specification (BRS)*, which is also called the *Business Requirement Document* [9].
- *Stakeholder Needs and Requirements (SNR) Definition Process.* Using the Preliminary OpsCon and the other PLCD as guidance, requirements engineers lead stakeholders from the business operations level through a structured process to elicit Stakeholder Needs—in the form of a refined OpsCon document and the completion of the other Life-cycle Concept Documents (LCD)—and transform them into the formal set of Stakeholder Requirements, which are documented in the *Stakeholder Requirement Specification* (*StRS*).
- *System Requirements Definition Process.* The requirements in the StRS are then transformed by requirements engineers into *System Requirements*, which are contained in the *System Requirement Specification (SyRS)* [10]. This document is also referred to as the *Solution Requirement Specification Document* [11], or simply the *System Specification* [12]. Note that Figure 2-1 illustrates that a number of SyRS may be derived from the StRS—that is, perhaps, one SyRS is produced for each element of a capability. Figure 2-1 also illustrates that some organisations may prepare individual LCD for each of a number of systems that are developed to meet the Business Needs. The process continues for the system elements.

ANSI/AIAA G-043A-2012 [13] identifies that the terms ‘concept of operations’ and ‘operational concept’ are often used interchangeably but notes

that an important distinction exists because each has a separate purpose and is used to meet different ends. It is essential that these terms are used so that they are consistent with ANSI/AIAA G-043A-2012 and ISO/IEC/IEEE 29148 [14], as well as the way in which the terms are used in the US DoD and many other defence forces.

ISO/IEC/IEEE 29148 describes the ConOps as:

> *The ConOps, at the organization level, addresses the leadership's intended way of operating the organization. It may refer to the use of one or more systems, as black boxes, to forward the organization's goals and objectives. The ConOps document describes the organization's assumptions or intent in regard to an overall operation or series of operations of the business with using the system to be developed, existing systems, and possible future systems. This document is frequently embodied in long-range strategic plans and annual operational plans. The ConOps document serves as a basis for the organization to direct the overall characteristics of the future business and systems, for the project to understand its background, and for the users of this International Standard to implement the stakeholder requirements elicitation.* [15]

ISO/IEC/IEEE 29148 describes the OpsCon as:

> *A system OpsCon document describes what the system will do (not how it will do it) and why (rationale). An OpsCon is a user-oriented document that describes system characteristics of the to-be-delivered system from the user's viewpoint. The OpsCon document is used to communicate overall quantitative and qualitative system characteristics to the acquirer, user, supplier and other organizational elements.* [16]

The ConOps is developed by the organization leadership at the enterprise level. The ConOps provides the context and guidance for the OpsCon, which is prepared at the business management level. Business management begins with the preparation of the Preliminary OpsCon, which summarizes the needs of the system within its business context—the Business Needs. The Preliminary OpsCon is then elaborated and refined into the OpsCon by engagement with stakeholders at the business operations level during the Stakeholder Needs and Requirements Definition Process—the OpsCon therefore contains Stakeholder Needs.

The OpsCon, however, is just one of the concepts required to address the Stakeholder Needs across the system life cycle. In addition to the operational aspects, other related life-cycle concepts are required to address the following concepts:

- The *Acquisition Concept* describes the way the system will be acquired including aspects such as stakeholder engagement,

requirements definition, solicitation and contracting issues, design, production, and verification.

- The *Deployment Concept* describes the way the system will be validated, delivered, and introduced into operations.
- The S*upport Concept* describes the desired support infrastructure and manpower considerations for supporting the system after it is deployed. A support concept addresses operating support, engineering support, maintenance support, supply support and training support.
- The *Retirement Concept* describes the way the system will be removed from operation and retired, including the disposal of any hazardous materials used in or resulting from the process.

PLCD are prepared by business management and contain the Business Needs defined as part of the Business Needs and Requirements Definition Process. Preliminary life-cycle concepts in the PLCD are elaborated and refined in much more detail in the LCD by stakeholders at the business operations level in the Stakeholder Needs and Requirements Definition Process.

2.2.1 What is a Requirement?

We should begin our consideration of requirements engineering with a number of definitions of a requirement from the major related standards:

- *MIL-STD-499B* [17]: 'Characteristics that identify the accomplishment levels needed to achieve specific objectives for a given set of conditions.'
- *IEEE-STD-1220* [18]: 'A statement that identifies a product or process operational, logical, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).'
- *EIA-632* [19]: 'Something that governs what, how well, and under what conditions a product will achieve a given purpose.'
- *IEEE-610.12:* 'A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.'
- *A Guide to the Business Analysis Body of Knowledge ® (BABOK® Guide)* [20]*:* 'A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents.'
- *ISO/IEC/IEEE 29148* [21]: 'A statement which translates or expresses a need and its associated constraints and conditions.'

Perhaps more usefully, for the purposes of this book, a requirement can be defined as a statement of:

- the services and functions that the system should provide, the things it should do, or some action it should take (often called *functional requirements*);
- the qualities, properties, or attributes that the system must possess (often called *non-functional requirements)*; and
- any restrictions or bounds under which the system should operate, or on the way in which the system is to be developed (also called *constraints* and sometimes also included in the category of non-functional requirements).

Each requirement statement may be supported by:

- *Performance*, *verification*, *and rationale* statements supporting each requirement.
- Definitions of other systems with which the system must integrate and to which it must interface.
- Information about the application domain within which the system must operate.

The requirements in the BRS and the StRS are not as formal as those in the SyRS but they have the same structure and follow the same rules as presented in the remainder of this chapter.

As well as these broad categories, an adjective is often included to describe the nature of the requirement. So, reference is often made to operational requirements, stakeholder requirements, environmental requirements, interface requirements, system requirements, regulatory requirements, safety requirements, design requirements, and many more other categories of requirements.

In this text we simply consider *requirements*, since the selection of categories is relatively artificial. All other groupings are used loosely to describe various categories of requirement which are not part of a structured approach to writing a requirements document but are often convenient perspectives to take of the various subsets of requirements.

2.2.2 Requirements Characteristics and Attributes

However it is gathered or elaborated, each requirement is to be examined as it is collected or elaborated to ensure that it has a number of important *characteristics*. Further, the single statement of each requirement is not sufficient on its own. To support analysis and management, requirements should be recorded with associated *attributes*—the combination of the requirement statement; the associated performance, verification and rationale statements; and the associated attributes results in a *requirements expression*.

This section examines the characteristics of a good requirement statement and the attributes that should be associated with each requirement.

2.2.2.1 Requirements Characteristics

Each requirement should exhibit the following characteristics (based on [22]):

- *Necessary.* The requirement must be necessary in that the system could not function in the desired way without this requirement. It also follows that a requirement cannot be necessary if it cannot be traced back to a higher-level requirement (and the associated need).
- *Singular.* A requirement cannot be a combination of two or more requirements—that is, the requirements set must be normalized so that requirements do not overlap. This also implies that the requirement is concise and clear.
- *Correct.* That the requirement must be correct is axiomatic—an incorrect requirement will result in a system that does not meet the real needs.
- *Unambiguous.* All readers of the requirement should come to the same understanding of what is meant by the requirement—there can be only one interpretation. This is difficult when the requirement is written in natural language, particularly English, where a single statement can have more than one meaning.
- *Feasible.* A requirement must be achievable using extant design and engineering processes, extant technologies, and extant manufacturing procedures. A requirement should also be achievable within organisational constraints such as budget and schedule unless we take care to be unambiguous.
- *Appropriate to level.* Each requirement must be appropriate to the level at which it is stated. For example, a system-level requirement must be independent of the method of implementation (process) or physical solution—that is, it should say what is required, rather than how it should be implemented.
- *Complete.* The requirement must stand alone and not need any further explanation.
- *Conforming.* Each requirement must conform to a standard formal structure as defined by the organisation, or perhaps in accordance with an external standard.
- *Verifiable.* Each requirement must be verifiable, in that it can be proved that the system meets or possesses the requirement. Verification may be by one or more means.

2.2.2.2 Requirements Attributes

So far, we have been discussing requirement statements. However, a requirement is more than just a requirement statement. The full expression of a requirement includes associated *attributes* that aid in the management of the requirement (the attributes may not be visible with the requirement statement but may be appended to it in the requirement document or in an accompanying or hosting database). So, we can define a *requirement expression*: "A requirement expression comprises a requirement statement and a set of associated attributes." [23].

An attribute is "… additional information included with a requirement statement, which is used to aid in the management of that requirement." [24]. Attributes can be organized within four broad categories [25]:

- *Attributes to help define the requirement and its intent.* Examples include: rationale, system of interest primary verification method, system of interest verification approach, parent requirement, source, condition of use, and states and modes.
- *Attributes associated with the system of interest verification.* Examples include: system of interest verification level, system of interest verification phase, system of interest verification results, and system of interest verification status.
- *Attributes to help maintain the requirements.* Examples include: unique identifier, unique name, originator/author, date requirement entered, owner, stakeholders, change board, change status, version number, approval date, date of last change, stability, responsible person, requirements verification status, requirement validation status, status (of requirement), status (of implementation), trace to interface definition, trace to peer requirements, priority, criticality, risk, key driving requirement (KDR), additional comments, type/category (including functional/performance, interactions with external systems - input, output, external interfaces, environmental, facility, ergonomic, compatibility with existing systems, logistics, users, training, installation, transportation, storage; the -ilities (quality), reliability, availability, maintainability, accessibility, safety, security, transportability, quality provisions, growth capacity; physical characteristics; standards and regulations—policy and regulatory, constraint, business rule, business requirement)
- *Attributes to show applicability and allow reuse.* Examples include: applicability, region, country; state/province, application, market segment, business unit, business line.

There is not sufficient space to discuss fully here a taxonomy for attributes (of which there may be a large number—for example, the INCOSE *Guide for Writing Requirements* defines some 44 attributes), nor for how such attributes

could be defined and used. Here, we simply note that the following attributes (based on [26]) are recommended:

- *Unique identifier.* Each requirement must be able to be identified as a unique statement. Consequently, when a requirement is recorded, it must be allocated a unique identifier that remains with the requirement throughout the design process. This is essential for requirements management and for requirements traceability in particular. Although managing the allocation of unique numbers is difficult in a paper-based system, the process is straightforward when conducted within the database underlying an automated requirements management tool. Each requirement also has performance and verification statements associated with it. While these require separate identifiers, it is useful to use a system whereby the association between the three types of requirements is immediately obvious. One recommended system to use the same stem number for all three types of requirements and the letters *P* and *V* are appended to denote the performance and verification statements associated with the requirement statement. See Chapters 8 and 9 for more detail.
- *Short title.* As well as being identified by a unique identifier and a unique sentence, it is useful to give the requirement a unique short title. This short title makes it easy to refer to the content of the requirement as well as making it simpler to display in a graphical format such as in the RBS. The short title is most useful when it is a contraction of the requirement statement.
- *Priority.* Requirements are often stated in terms of their importance to individual stakeholders. The priority of each requirement must therefore be captured with the requirement so that requirements engineers (and designers subsequently) have some design space within which to work during requirements analysis and negotiation. This priority value is allocated from a business management or stakeholder perspective and indicates how important the achievement of each requirement is in the context of the overall system. Stakeholders are tempted to state all requirements as having the highest priority (mostly for the fear that the lower-priority requirements will be subsequently discarded), but they should be encouraged to allow some flexibility within which design can be conducted. The most important (mandatory) requirements are often stated as *shall*; the next most important are stated as *shall, where practicable*; the next most important can be stated as *preferred* or *should*; and the next most important can be stated as *may*. These levels are often simplified to a three-level (mandatory, highly desirable, desirable) or a two-level (mandatory, optional) scheme. When initially collecting requirements, it may be sufficient to ask

stakeholders to rate the requirement's priority as high, medium, or low. A more detailed system could ask them to rate priority on a scale of 1 to 10, which would allow some numerical analysis of priority during subsequent design, if that is desirable. In baseline specifications, priority can therefore be allocated to the requirement in two ways—either in the text of the statement using *shall*, *should*, and *may*; or all statements could be stated using a term such as 'shall' and an indication of priority could be appended as an attribute in a numerical form or in the form of high, medium, or low. The latter method is preferred as it avoids errors that may creep in during re-typing the requirement when the priority is changed.

- *Criticality.* It is also useful to ask the business management and stakeholders to identify those requirements that are critical in terms of their contribution to system performance. While seemingly related to priority, this category identifies show-stoppers—those requirements that are essential to the success of the overall system. In other words, a requirement that is deemed critical must be met or possessed by the system or else the system cannot be introduced into service. In some cases it may be useful to have a rating system, but it is most often sufficient to identify each requirement in a binary fashion as critical or non-critical.
- *Feasibility.* Most system developments and acquisitions have some elements that are to be based on new technology or require the development of new manufacturing and production techniques. It may be useful in these cases to establish the feasibility of meeting the requirement based on an understanding of current technology and procedures. The major use for this category is in the functional analysis tasks associated with the development of the draft SyRS—it is most likely that the utility of the category beyond that stage is marginal and it may well not be included in the approved SyRS.
- *Risk.* Each requirement contributes to overall system risk in its own way and the management of risk is a critical component of systems engineering and project management. It is useful to consider the risk associated with each requirement as it is collected, either by allocating a risk rating (1–10, for example) or by writing an associated risk statement.
- *Source.* There are likely to be a number of sources of requirements. The originator (person, organization, document, or process) must be recorded to ensure that all requirements can be justified. Additionally, sources may need to be consulted when negotiation and compromises are required during requirements analysis.
- *Type.* While not essential, it is very useful to attach to each requirement a tag associated with the type of requirement. Example

are such types as input, output, external interfaces, reliability, availability, maintainability, accessibility, environmental conditions, ergonomic, safety, security, facility, transportability, training, documentation, testing, quality provision, policy and regulatory, compatibility with existing systems, standards and technical policies, conversion, growth capacity, and installation. The type field is most useful because it allows the requirements database to be viewed by a large number of designers and stakeholders for a wide range of uses. For example, maintainers could review the database by asking to be shown all the maintenance requirements, engineers developing test plans could extract all verification requirements, and so on.

- *Rationale.* As each requirement is elicited/elaborated, a rationale must be attached to record the reason for the requirement's existence. Rationale is essential to guide subsequent design.
- *History.* This portion of the database records when the requirement was raised, by what means it was raised, and by whom. It also identifies any changes that are made to the requirement throughout the design process and records when the requirement was modified and by whom.
- *Relationship to other requirements.* The relationship with other requirements must be identified to assist in forward and backward traceability.
- *Other information.* If desired, each requirement should be stored with reference to the date of raising the requirement, the status (proposed, reviewed, accepted, rejected, and so on), and comments.

2.2.3 What Not How

In general, business, stakeholder and system requirements should be statements of *what* a system should do, rather than *how* it should do it. However, for a number of reasons, this is often too simplistic in practice. First, it may be essential that the system should perform a function in a particular way (such as for interoperability, or to meet extant standards). We could also specify a solution to meet broader concepts such as support and commonality with existing solutions. Additionally, a specific statement is often less confusing than an abstract statement of the problem—a specific statement is therefore the best guidance that a conceptual designer can give a lower-level designer. Further, for good business reasons, business management may well dictate not only what is required but also how it is to be implemented. Finally, the people who specify the system are often the domain experts—they are therefore best placed to state how the system should be developed and how it should operate. Still, unless one of those reasons

prevails, we should ensure that the BRS, StRS and SyRS provide logical specifications (because they are in the problem domain), not physical specifications (which belong in the solution domain).

2.2.4 Emergent Properties

Not all properties of a system are exhibited separately by its constituent elements. That is, the system may have properties that only exist when the system elements are integrated—these are called the *emergent properties* of a system. Emergent properties:

- are those that are possessed by the system as a whole;
- only emerge after the individual system elements have been integrated;
- cannot be exhibited by elements of the system in isolation; and
- depend on interactions among system elements, including their environment.

Examples of emergent properties are the 'ilities' such as: maintainability, reliability, availability, and usability. Other examples include safety, security, comfort, and ease of use. Emergent properties must therefore be defined from the top down—they cannot be obtained by specifying requirements for individual elements and then hoping that the desired properties will exist on integration.

Perhaps the bicycle is one of the best examples of a system that possesses emergent properties, since it can only perform its principal function when all system elements, including the rider, have been integrated. If any one of the major elements is removed, the system cannot function.

We also must consider the possibility of the completed system exhibiting emergent properties that were not considered as part of its design and, in some cases, may be undesirable in the end result.

2.3 WHAT IS REQUIREMENTS ENGINEERING?

Requirements engineering can be described most simply as the process by which we identify a problem's context, locate the business and stakeholder needs and requirements within that context, and deliver system requirements that meet those needs [27]. As illustrated in Figure 2-2, that process requires a considerable reduction of the complexity of the real world, and that subset within which the customer organisation operates, into a set of requirements that define the scope of the project. There is therefore considerable judgement required in identifying BNR and SNR and then converting them into a SyRS. Further, as Andriole points out, most requirements-engineering methodologies have been created by people who like precision, diagnosticity, and rigour—yet, requirements management is fundamentally a political process, not a technical one [28].

Dorfman and Thayer describe requirements engineering as the "science and discipline concerned with analysing and documenting requirements" [29]. Kotonya and Sommerville define requirements engineering as "the systematic process of eliciting, understanding, analysing, documenting, and managing requirements" [30]. ISO/IEC/IEEE 29148 defines requirements engineering as the "interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system ... [and] ... is concerned with discovering, eliciting, developing, analysing, determining verification methods, validating communicating, documenting, and managing requirements." [31].

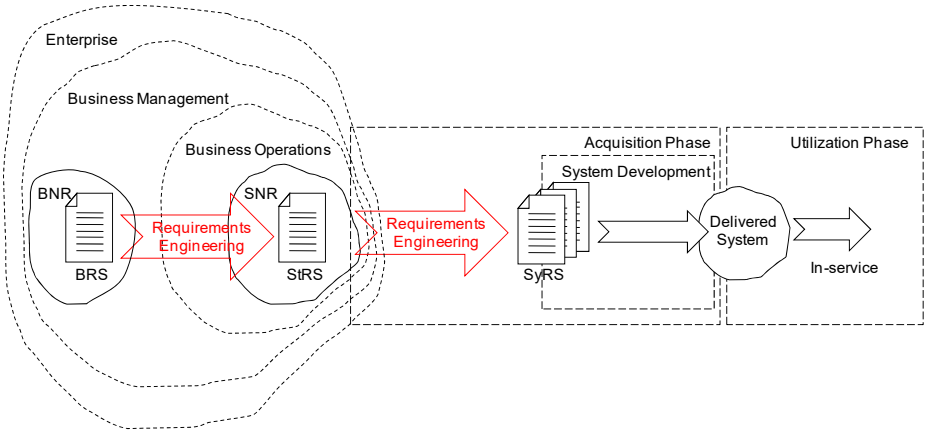

Figure 2-2. The role of requirements engineering in translating BNR into SNR and then into system requirements in one or more SyRS that define the capability system.

2.3.1 Why We Need Requirements

It may not always be obvious why requirements are necessary in the first place. An agreed set of requirements is essential, however, from a number of different perspectives.

From the point of view of the business management, the set of requirements provides a mechanism whereby they can agree that the organisation's resources can be allocated to the project that has been established to bring the system into being. Unless the requirement set is well defined, the project has no firm base and the resources allocated to it will be inappropriate. Worse still, without sufficient oversight, the business may be held to ransom by stakeholders insisting on unrealistic requirements that business management have not approved.

From the project manager's perspective, the set of requirements is an essential part of the definition of the scope of the project—it is from this scope that a suitable system will be developed. Scope management is one of the ten major functions of a project manager [32], and a well defined scope

(set of requirements) is necessary to able to justify any expenditure of funds or effort within the project, to be able to report adequately on the progress of work, and to eventually be able to determine when the project is complete. Clearly then, if there is not a well defined, universally agreed, set of requirements at the outset of the project (or phase), the project (or phase) is doomed to fail.

From the systems engineering perspective, the set of system requirements instantiated in the SyRS represents the transition from the business world into the systems engineering and engineering worlds. Once the set has been established, it is not possible to rectify a poor set of requirements with good design, or good engineering, or good production/process control—particularly when that set of requirements is the basis of a fixed-price contract between the acquirer and the contractor. Consequently, a project can rarely recover from poor definition, regardless of how much good work is performed subsequently.

2.3.2 Why We Need Requirements Engineering

An appropriate requirements-engineering process allows us to guarantee that we begin the project with a set of requirements that is complete, consistent, comprehensible, feasible, and able to be validated. A useful process ensures that all stakeholders have had input and the various points of view have been reconciled. To do that, we must therefore be able to trade off functionality for cost—which implies that we understand the required functionality, priorities, and costs. We also need to be able to use the process to manage changes in requirements for a wide variety of reasons.

Requirements engineering focuses on what is needed by the system, rather than how it is to be designed. As discussed in Chapter 1, the cost of developing requirements is estimated to be 10–15% of the total system cost, yet it is estimated that it costs some 200 times more to rectify an error in requirements during utilization than it does during early Conceptual Design [33]. As discussed in the previous section, poor requirements definition cannot be rectified by good design, so that it invariably follows that rigorous development of requirements is essential for the successful system—requirements engineering provides this rigour through the application of a formal methodology. When requirements-engineering processes are ad hoc, the end result is almost always an unsatisfactory product or a cancelled project [34]. Unfortunately, few organisations have a well defined requirements-engineering process and there are few, if any, standard methodologies that assist.

2.4 REQUIREMENTS ELICITATION AND ELABORATION

Requirements elicitation and elaboration involve working with business managers and stakeholders to investigate the problem to be solved, and to

identify their needs and requirements. At the beginning of this chapter, we considered the three levels of requirement statement:

- *business requirements* in the BRS;
- *stakeholder requirements* in the StRS; and
- *system requirements* in the SyRS.

We then discussed requirements engineering by referring to 'requirements' in general terms. Before discussing in subsequent chapters how these requirements are part of systems engineering, it is useful here to discuss how the design moves through those three levels of requirement statement. In general, requirements in the BRS, StRS and SyRS are described hierarchically through the processes of *elicitation* and *elaboration* [35].

Elicited requirements. Elicited requirements can be attributed to the source (business manager or stakeholder), either directly or through negotiation. These requirements are normally gathered via interview or a structured workshop.

Elaborated requirements. Elaboration is used to move between the levels in the requirements hierarchy—the process is also often referred to as *requirements flowdown*. Elaborated requirements are of two types:

- *Decomposed requirements.* Elicited requirements tend to be at a high level because they were obtained from those directly involved in the business management and business operations supported by the system of interest. Decomposition entails breaking a higher-level requirement into those lower-level requirements that are explicitly required by it. For example, if an aircraft needs to be able to land on a certain class of airfield, then it must be able to land within the appropriate runway length, be less than the maximum allowable weight, be able to interface to the appropriate air traffic control systems, and so on. In decomposition, the need for the children requirements is obvious in the parent. Requirements engineers take an elicited requirement and decompose it into two or more subordinate requirements whose total meaning is equivalent to that of the original requirement. It should be noted that the original requirement normally becomes redundant and is replaced by the lower-level requirements. The essence of the upper-level requirement often becomes a heading or is noted in the rationale for the decomposed requirements, although it is good practice to retain the hierarchical structure of the RBS by abstracting the original requirement to a heading in the specification structure.
- *Derived requirements.* In addition to being high level, elicited requirements also tend to be from the business management and operations perspectives and will not necessarily address all aspects of a systems design. Derivation entails requirements engineers drawing some inference from the higher-level requirements to

> obtain lower-level statements. That is, business management or the stakeholders did not state the requirement directly, but the derived requirement is a necessary part of the system design if one or more of the directly stated requirements are to be met. Unlike decomposed requirements, derived requirements do not replace the original parent requirement.

The use of elaboration (decomposition and derivation) is discussed in more detail in Chapter 5, which includes examples of each activity.

Business, stakeholder and system requirements are not collated randomly—requirements engineers undertake requirements analysis which involves, inter alia, elaboration of a statement of business and stakeholder needs through a systematic exposition of what the system must do in order to meet those needs [36]. Gathering, analysing, and elaborating requirements is an essential step to ensure that the system, once built, actually meets business needs and stakeholder needs [37]. High-level requirements are elaborated further into a number of lower-level requirements which are then grouped, along with other elaborated requirements, through the requirements analysis process [38] into the business-management-level descriptions in the BRS, the business-operations-level descriptions in the StRS, and the system-level description in the SyRS.

The requirements in the SyRS are then further elaborated into lower-level requirements which are then re-grouped from the logical groupings of the SyRS into physical groupings of requirements (the CIs). The CIs are documented in a number of Development (Subsystem) Specifications, which then define 'how' the system will be implemented. The mapping of the logical groupings of requirements in the SyRS into the physical groupings in the CIs in the Development (Subsystem) Specifications informs the definition of project deliverables.

For well-behaved systems, the requirements in each of the BRS, StRS, SyRS and system element specifications are normally grouped in a hierarchical fashion (albeit grouped logically in the first three, and physically in the last). This hierarchy is useful because it allows requirements engineers to decompose the system into tasks of a complexity that can be more easily be understood and communicated, as well as managed within the limitations of humans and our organisations.

A hierarchical description of a system is also useful in that it supports requirements traceability, which is an essential element of effective systems engineering as well as project management. Traceability is assisted by a hierarchical numbering of requirements. As illustrated in Figure 2.3, Requirements 1.1.1, 1.1.2 and 1.1.3 are children requirements of parent Requirement 1.1, which is a child of Requirement 1. For Requirements 1.1.1, 1.1.2 and 1.1.3 to be children of parent Requirement 1.1, they must be able to be derived or decomposed from it—that is, Requirement 1.1 is a superset of

Requirements 1.1.1, 1.1.2, and 1.1.3; and Requirement 1 is a superset of Requirements 1.1 and Requirements 1.2.

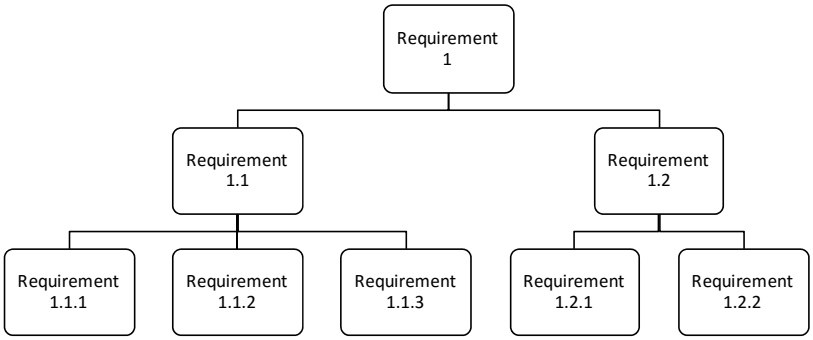

Figure 2.3. The hierarchical elaboration of requirements.

2.4.1 Elicitation Dimensions

Requirements elicitation involves engagement with the stakeholders in order to establish their requirements for the new system. There are a number of techniques for this engagement, such as facilitated structured workshops, brainstorming, interviews, surveys, questionnaires, process models, use cases and user scenarios, simulations, prototypes, observation of work studies (time and motion studies), participation in work activities, observation of the system's organisational and political environment, technical documentation review, benchmarking, competitive system assessment, reverse engineering, and market analysis (see Section 2.5 for more detail).

Requirements elicitation is not at all straightforward. It is not just a matter of "collecting" requirements—that is, simply writing down the requirements as they are gathered from stakeholders. Requirements engineers must be more than simple scribes in this process—they must be able to value-add to what the stakeholders say. There are therefore seven dimensions to requirements elicitation [39]:

- *Understanding the business.* Requirement engineers must understand how the new system contributes to the business or organization. To do so requires them to understand the needs of business management and the nature of the business and those aspects of it that will impact on the new system.
- *Understanding the application domain.* Requirement engineers must also understand the general application domain within the business in which the new system is to be implemented.
- *Understanding the specific problem.* Understanding of the general application domain must be focused further, so requirements engineers must also understand the specific problem addressed by the system.

- *Understanding the needs and constraints of system stakeholders.* The specific needs and constraints of stakeholders must be understood by requirements engineers, as there may be several different sets of criteria by which the final system is judged. Additionally, the specific context within which stakeholders describe their requirements must also be understood, as must the language they use, their predilections, and their biases. Further, the need to trade off various requirements relies on an understanding of the needs of stakeholders.
- *Understanding acquisition and project management.* Requirements engineers are heavily involved in defining the system requirements which are, in effect, the system scope upon which project managers develop the project work breakdown structure (and the consequent schedule and budget). Additionally, the SyRS developed by requirements engineers becomes the centrepiece of the statement of work underpinning the contract signed with the supplier or developer. Requirements engineers must therefore understand how systems are acquired and projects are managed.
- *Understanding requirements engineering and systems engineering.* Requirements engineers stand on the boundary between the business and the project. As well as understanding the business environment in which the system will be deployed, the requirements engineer must also be able to translate that environment into the systems engineering and engineering domains. It is very difficult to contribute to a process that is not understood. Those undertaking requirements elicitation must therefore understand, and preferably have significant experience with, systems engineering and requirements engineering.
- *Understanding the technologies and engineering involved.* Although it is common to assert that a good manager can manage in any field, even at a high level a manager is not performing to full potential if the managed processes and contributing technologies are not understood. This is certainly true at the level of requirements elicitation, where the requirements engineer must understand the technologies involved as well as be familiar with their engineering.

For further detail, Young [40] provides an excellent coverage of the skills required, the characteristics, and the job description, of requirements analysts.

It should be noted that the understanding outlined above is very rare and is not naturally occurring—requirements engineers must be trained. There are two broad approaches: engineers can be trained to become systems engineers and requirements engineers and then be provided with exposure to business and business practices; or business managers can be educated in technology, engineering, systems engineering, and then requirements

engineering. While the latter is not impossible, the former is the most probable.

2.4.2 Difficulties in Eliciting and Elaborating Requirements

Unfortunately, requirements elicitation is not a simple matter of transferring knowledge from stakeholders to requirements engineers, which means that the effective development of requirements can be difficult and challenging. Stakeholders will probably not have given the new system much consideration and will not have had a chance to develop a clear view of their requirements—they may even be hearing about the system for the first time when contacted by requirements engineers requesting their input to the design. Even if they have an informed view, different stakeholders have conflicting requirements, they are not normally aware of the requirements-engineering process and of the acquisition process in general, they are often not willing to compromise their expectations to meet any budget, and they are rarely aware of the limitations that technology has on their expectations. This section briefly discusses these and other difficulties.

Instead of focusing on the logical architecture of the system, stakeholders sometimes identify particular physical solutions that appear to meet their requirements. That is, instead of concentrating on what the system is required to do, there is an ever-present urge to describe how the system should be designed or built. While this is dangerous because it can preclude effective solutions from consideration, requirements engineers can use the predilection to some advantage because the stakeholder can at least visualize (and therefore hopefully describe) something that meets their perceived needs.

Further, stakeholders sometimes become confused between a desire and a need. It is vital that system-level requirements are based on the true needs of the system and not merely a wish list of additional functionality. This reinforces the need for forward and backward traceability between the endorsed business needs and requirements and the SyRS. Additional (unendorsed) functionality may add complexity and expense onto the system without enhancing the system's ability to meet the true system-level requirements. This gold-plating of the requirements must be avoided.

Another difficulty is that stakeholders themselves do not always contribute positively to the requirements-engineering process:

- Stakeholders usually do not really know what they need, do not know what questions need to be asked or answered, and almost never have considered the problem in terms of the detail that must be specified [41].
- Stakeholder expectations can also often be unrealistic.
- Stakeholders often have no desire to compromise their unrealistic demands when faced with cost constraints.

- Stakeholders do not understand the requirements-engineering process and often do not participate in reviews when required, or with the regularity or required.
- Stakeholders will often not commit to a set of written requirements because they fear that they will then be committed to a particular solution—this is particularly evident when they either have a poor understanding of the problem, or have had insufficient time to devote significant effort to its solution.
- Stakeholders often insist on modifying or adding new requirements after the requirements set has been endorsed and the cost and schedule have been fixed.
- Stakeholders are not normally willing to devote the time to understand properly the intricacies of the products and the technologies involved; nor are they normally interested in understanding the engineering, systems-engineering, and requirements-engineering processes involved in developing the requirements set.
- Different stakeholders often have different perspectives resulting in conflicting requirements, or at least similar requirements expressed differently. Even a single stakeholder may have different perspectives over time. Additionally, as key positions change, the new incumbent may have a significantly different view of requirements to that of the previous stakeholder representative.

Information on the application domain is not always collected in one place and may involve specialist terminology. It is highly unlikely in the development of large systems that any one stakeholder (or any single group of stakeholders, for that matter) is able to articulate all system-level requirements. Additionally, the language employed by stakeholders in most complex systems tends to be arcane to outsiders and even to other specialist stakeholders of the same system. Some considerable time may have to be invested by requirements engineers to be able to comprehend the domain knowledge of all stakeholders.

Requirements are often poorly stated because the real requirements are not known or understood. It cannot be assumed that the potential users of the system know what they need from the system—which is only to be expected since they may have never seen it nor operated it. Requirements engineers must recognize this and try a number of different communication and elicitation techniques. As a corollary, stakeholders sometimes firmly believe that they know exactly what they need until they are presented with a system that has their specified functionality whereupon they realise that they do not require some of the features included and bemoan the lack of others. To overcome this, stakeholders must be presented with prototypes, models, and simulations as soon as possible to ensure that there is a strong correlation between the stakeholders' perceived requirements and the appropriate system

requirement statements encapsulated in the SyRS. As mentioned earlier, the stakeholders' predilection for stating solutions in physical terms can be used to advantage by requirements engineers to confirm requirements through analogy by comparing stakeholders' stated needs with existing systems or prototypes that perform similar functions.

Conflicting requirements often result when two or more requirements are mutually exclusive such that satisfaction of one requirement will result in failing to meet another requirement. Conflicts must be resolved early in the requirements-engineering process, as conflicts can be a source of ambiguity. Conflicts can also occur where the same requirement appears twice, worded in slightly different ways, or requiring different levels of performance for the same function. Every requirement must be unique and stated only once to avoid confusion and conflict.

There are also a number of organisational and political factors that influence requirements elicitation and elaboration:

- Changes in economic and business environment are a constant factor in the work lives of most stakeholders and they may initially show some understandable animosity to the suggestion that they should devote valuable time to contributing to yet another change to the way in which they conduct business.
- Even without considering the constant need to change, the stakeholders who best understand the problem to be solved are invariably not available to help specify requirements for the new system. Stakeholders in modern "right-sized" organizations are probably busier than they have ever been at any other point in time. The pressures of their daily workplace may preclude them from being sufficiently available to contribute to the requirements-engineering process.
- Organizations and systems involve people, and the human element is a strong influence on requirements engineering. Political, organizational, or personal bias is therefore often evident in early statements of requirement. Requirements engineering is not concerned with removing these factors, which will arguably always exist whenever humans are involved in the process. Bias does, however, need to be identified as early as possible to ensure that subsequent decisions are made on an informed basis and the bias is formally recorded as a constraint.
- Additionally, it cannot be assumed that every stakeholder is a supporter of the new system. Some may prefer that available funds are expended on another project that is more important to them. Others may not agree that the system is necessary at all, and consequently fail to contribute adequately to the requirements-engineering process.

The final challenge is to know when enough is enough, as it is difficult to judge when the requirements analysis effort is complete. Although experienced personnel are best placed to make this judgment, no two system developments are likely to be the same, and the stopping criteria will require careful consideration for each system design.

2.4.3 Techniques for Eliciting Requirements

Requirements engineers select an appropriate requirements elicitation methodology to suit the system being developed. Most requirements-engineering literature (see for example, Alexander and Stevens [42], Goguen and Linde [43], Hull [44], Katonya and Sommerville [45], Sommerville and Sawyer [46], Young [47], Young [48], and IEEE-STD-1223 [49]) suggests such techniques as facilitated structured workshops, brainstorming, interviews, surveys, questionnaires, process models, use cases and user scenarios, simulations, prototypes, observation of work studies (time and motion studies), participation in work activities, observation of the system's organisational and political environment, technical documentation review, benchmarking, competitive system assessment, reverse engineering, and market analysis. The selection of technique is critical since inappropriate use can doom a project to failure (Verner et al [50]). Frameworks (see for example, Coulin et al [51], and Tsumaki and Tamai [52]) have been proposed to assist in selecting an appropriate requirements-engineering technique for a particular project.

The following sections describe each of the major requirements elicitation/elaboration techniques in more detail.

2.4.3.1 Facilitated Structured Workshops

Perhaps the fastest (and most efficient) way to elicit requirements is through structured requirement workshops. It must be remembered that workshops are part of the iterative cycle of requirements development—the top-down approach greatly assists here in allowing participants to focus on appropriate levels of abstraction (level of detail). The iterative cycle does not start with a blank sheet of paper, however. A workshop should always start with a draft set of requirements and other early Conceptual Design artefacts, particularly the visual artefacts we describe at the end of this chapter. These various artefacts are therefore very useful as discussion and communication tools as well as requirements analysis tools. If possible, the set of draft upper-level artefacts should be sent out to participants before the workshop to allow them to "read-in". It is therefore essential that a first iteration of requirements engineering is conducted before the workshop is held. Conducting a workshop that is not informed by some well-focused preliminary documents normally leaves stakeholders with a poor opinion of the new system even before Conceptual Design is well under way. Another way to ensure that

stakeholders make valuable contributions is to conduct a period of briefing/training in requirements elicitation before beginning the workshop series.

Significant thought must be given to the administration and the mechanics of the conduct of the workshop. As mentioned earlier, it is essential that the workshop is not only structured but also facilitated to avoid serious digression. The location, resources, and other logistical considerations must be taken into account to ensure that the workshop runs smoothly—the inability to run a simple workshop without problem is not likely to give stakeholders confidence that their new system is in good hands. A popular way of avoiding workplace distractions is to hold the events at a convenient off-site location where stakeholders can devote their attention to articulating their needs. There must also be a scribe nominated to record the design as it is iterated.

The main disadvantage of workshops is the additional travel costs that may be incurred in gathering stakeholders in the one location and the difficulty associated with getting many busy people in the same place at the same time. Technologies that facilitate remote access (such as videoconferencing) may assist in reducing costs.

2.4.3.2 Brainstorming Sessions

Brainstorming sessions are normally also run as workshops but are considerably less formal than structured workshops. It may well be worth conducting a brainstorming session with selected key stakeholders and designers as a means of obtaining the first iteration of the requirements-engineering products, before running a structured workshop. Brainstorming is particularly sensitive to the people involved—the value of the exercise also tends to diminish as the number of participants increases. As with structured workshops, brainstorming sessions should be facilitated, and they should probably be held off-site to allow participants to focus on the outcomes without the distractions of daily work.

2.4.3.3 Interviews, Surveys, and Questionnaires

Interviews, surveys, and questionnaires offer a formal process through which requirements may be garnered from individual stakeholders. Their use should be considered carefully, however, in that useful information is only provided in response to focused, well-balanced questions, which implies that a measure of requirements engineering has been conducted before the interviews are held or before the surveys and questionnaires are sent out. All three processes can also be very time consuming, which again means that careful thought should be given to the questions to ensure that responses add to an understanding of the systems requirements, rather than add additional noise to the requirements definition process.

In some respects, interviews should be considered to be one-on-one structured workshops. Similar preparation is required—the interview should never be approached cold by either the requirements engineer or the stakeholder being interviewed. Interviews can be very effective but are very time-consuming and a structured workshop may be preferred if there is a large number of stakeholders. As an exercise in good stakeholder management, however, it is generally advisable to interview at least the key stakeholders.

While there is no satisfactory replacement for the personal contact possible in an interview, there are occasions when surveys and questionnaires are useful requirements elicitation tools. Their utility is limited, however, to activities such as the collection of statistical data to reinforce or validate requirements collected by interview or workshop.

2.4.3.4 Use Cases and Operational Scenarios

One of the difficulties with asking stakeholders about their requirements for the new system is that stakeholders will probably have been trained on an existing system and may have difficulty in setting aside current operating procedures when considering requirements for the new system. Use cases and operational (or user) scenarios (sometimes also generically called *storyboarding*) are a simple but powerful way of assisting in providing stakeholders with a new frame of reference, particularly at the beginning of the requirements-elicitation process where they may have difficulty in breaking free from existing practices.

Additionally, stakeholders will find it very difficult to understand requirements when they have been written as a series of simple ‘shall’ statements. Use cases and scenarios are given in a narrative (and often visual) style to illustrate how the system will operate and how users will interact with it. Use cases and scenarios are therefore very useful because users often find it easiest to describe their view of the system in terms of examples of how it might be used.

Use cases therefore provide a good framework within which to elicit requirements, as well as serving throughout the project as upper-level descriptions of the required functionality. Use cases and scenarios also perform a very important role of defining the purpose of the system, which is useful during acceptance testing to ensure that the delivered system is ‘fit-for-purpose’—the use cases developed at the start of development to assist in eliciting requirements then effectively become test cases at the end of the development to assist in verification/validation of the system.

To ensure a complete description of a system, it is important that the use cases and scenarios capture the exception conditions as well as the normal operational conditions. That is, users are often keen to describe the way in which the system works in terms of the positive aspects of its implementation, rather than what to do when the normal process is prevented from occurring

due to such events as system failure, actions by a competitor or adversary, or simply operator error.

Use cases and use-case modelling are very popular in software requirements engineering—they are described in a number of excellent references [53].

2.4.3.5 Process Models

Process models provide descriptions of the major activities involved in a particular process as well as their sequencing. Process models are normally provided at a number of different levels and from a number of different perspectives to allow for an understanding at different levels of abstraction from coarse to fine resolution and to ensure that all views of the process are considered.

Process models are particularly well suited to systems such as transaction-based systems where more-formal methodologies such as entity-relation models can be used to define formal database structures. For requirements engineering during Conceptual Design of most other types of systems, however, process models are too low-level to be useful and use cases are preferred.

2.4.3.6 Prototypes

Requirements elicitation can be enhanced by the ability of requirements engineers to mock up a system, or part of a system, that meets the collected requirements and then present that to stakeholders for consideration. Prototypes and simulations can be very complex or they can be as simple as a series of role-playing exercises in which users are asked to describe how they would imagine the system operating under particular circumstances. At the highest level, use cases and operational scenarios could also be considered to be a form of paper-based prototyping (a form of virtual walkthrough) in which stakeholders describe their requirements in a range of typical operations for the system.

Simulations and prototypes can also be used for requirements validation because they are a very useful (visual) way of showing stakeholders how the designers have captured and interpreted their requirements.

Prototypes have two main forms:

- throw-away prototypes that are intended to elicit or validate requirements but not expected to form part of the developed system; and
- evolutionary prototypes, that are intended to deliver a working system to the customer in incremental steps.

It is important to identify the purpose of the prototype. Although both forms are used to assist in eliciting and elaborating requirements, the nature of the

prototype must be explained to the stakeholders when they are viewing it. If it is an evolutionary prototype, they can expect to see it, or elements of it, again. Care must be taken to explain that a throw-away prototype is just that, and they cannot expect to see it again. Some dangers with throw-away prototypes include:

- Stakeholders who are detail-oriented often pay more attention to the structure of the prototype than to its operation. This is particularly true of software prototypes where stakeholders are sometimes blinded by the (often rudimentary) graphical user interface and cannot see past it to the functionality being demonstrated.
- Having seen the prototype in software, stakeholders sometimes find it very hard to understand why it will take so long to develop a system that has the functionality they have just witnessed. They struggle to see the difference between a mocked-up version and a properly tested, robust version. Again, this is particularly true of software prototypes—while a pilot would never expect to take away and fly in a cardboard cut-out prototype used to illustrate cockpit layout, stakeholders often see the functionality in a software prototype and somehow extrapolate it into a working system.
- Developers of code for the prototype are reluctant to throw it away and feel compelled to use it in the final system, despite the fact that, again, the code is not well designed nor properly tested.

2.4.3.7 Modelling and Simulation

Mathematical modelling and simulation are widely used and are becoming increasingly popular as computers and software become more capable. As the name suggests, mathematical models provide a mathematical representation of a physical system or design. Simulation then uses the models to determine both static and dynamic characteristics of the system under varying conditions. An example of a static model would be one that represents the structural characteristics of a bridge. Loads and stresses can be changed in magnitude, direction, and position to ensure that the structure will withstand the loads associated with its intended purpose. An example of a dynamic system is an aircraft's autopilot system, which is a classic example of a control system that can be used to determine characteristics such as stability and response times.

Models and simulations are excellent tools for performing what-if analyses and determining the limits of the design. Variables can be changed and conditions easily altered to allow designers to investigate the likely behaviour of the design in varying conditions. Simulation is widely used for a variety of different synthesis tasks. Environmental design and human-machine interface design are two examples of synthesis tasks that can benefit from simulation. The fidelity of the simulation (and the models) must be defined and confirmed prior to their use as synthesis tools.

To build and use a mathematical model, designers need to define the problem and formulate the model to replicate the problem. The model can be investigated with pen and paper if desired, but the more common approach is to convert the model into a computer program for ease of use and speed. Once this is done, perhaps the most crucial step must be performed. The performance of the model must be validated to ensure that it is an accurate representation of the problem. It is pointless to collect data from a model if the model is incorrect (that is, it is not representative of the problem). Once the model has been built and validated, designers can experiment with it via simulation and collect important design data.

2.4.3.8 Analysis of Similar Systems

In a similar vein to the use of prototypes, it may be useful to assist stakeholders to investigate their requirements by asking them to describe what they like and don't like about an existing system that is similar in functionality (even if only in part) to the system under consideration. At the very least, requirements engineers will be able to develop an awareness of the stakeholders' level of understanding of the new system. Even if that is poor, requirements engineers should still gain an understanding of user interface issues and perhaps even of some required functionality in the new system. Additionally, requirements engineers may gain some insight into the new system by scanning problem reports from similar existing systems.

2.4.3.9 Observation of the Workplace

Although time-and-motion studies may not always be a suitable way of eliciting requirements (particularly at the systems level) an understanding of the work environment and the workplace context is essential to the development of stakeholder requirements. By observing stakeholders in their natural environment requirements engineers may often identify requirements that the stakeholder had not stated directly though more formal requirements elicitation techniques such as interviews or workshops. Stakeholders often assume skills, processes and tools and do not describe them explicitly. They are also not always consciously aware of all of their work practices that are relevant to the new or modified system.

If the workplace cannot be observed directly, key stakeholders can be selected to be permanent members of the requirement engineering team if it is considered that stakeholder input is continuously required. In some cases, stakeholder involvement may continue past the requirements stage and a stakeholder representative may be attached to the teams performing the lower-level design, to ensure that the workplace context is always taken into account.

2.4.3.10 Documentation Review

In addition to observing the workplace, requirements engineers may review relevant business and technical documents. Although few organizations have very many information-rich documents, many organizations have standard operating procedures (SOPs) that may describe in some detail how particular operations are conducted. Those documents may therefore be useful as alternate sources of information about the workplace. Care must be taken, however, because the documents may describe the old system and not the new one.

2.4.3.11 Market Analysis

Many systems do not have a readily available, homogeneous set of stakeholders. Consumer products such as motor vehicles, for example, have a wide range of stakeholders whose requirements are many and varied, often with some conflict. Market analysis techniques may be useful in identifying a common subset of achievable requirements that maximize the potential market for the system.

2.4.3.12 Competitive System Assessment and Trials

As with prototypes and simulations, competitive system assessment and trials are aimed at providing stakeholders with some tangible manifestation of early requirements so as to assist in requirements elicitation and validation. One simple, quick way of providing depth to early requirements is to identify a number of existing systems that seem suitable and ask stakeholders to evaluate them in the context of their needs.

2.5 REQUIREMENTS VALIDATION

Requirements must be validated in order to ensure that individual requirements are appropriate and that the requirements set is an adequate description of the desired system. This process is intended to detect any problems and gaps in the requirements before they are used as a basis for system development. In particular, as illustrated in Figure 2-4, the SyRS must be validated to show that it meets the stakeholders' original requirements in the StRS, which must be validated to show that it meets the business requirements in the BRS.

Requirements validation is very difficult—somehow, we have to ensure that we have a complete set of requirements that, in their aggregation, represents the system required by the stakeholders and meets the business needs. This is difficult because there is no significant baseline we can refer back to for the validation. Later on in the life cycle, we will look back to the previous baseline—for example in Preliminary Design, we look back to the

SyRS; in Detailed Design and Development, we look back to the Development Specifications; and in Construction and Production we look back to the Product Specifications. In Conceptual Design, however, we only have the high-level business descriptions as the start point. It is therefore not a straightforward matter to validate that the StRS and then the SyRS represent the complete set of requirements associated with meeting those needs. Validation is normally conducted by a formal requirements review, or a series of reviews.

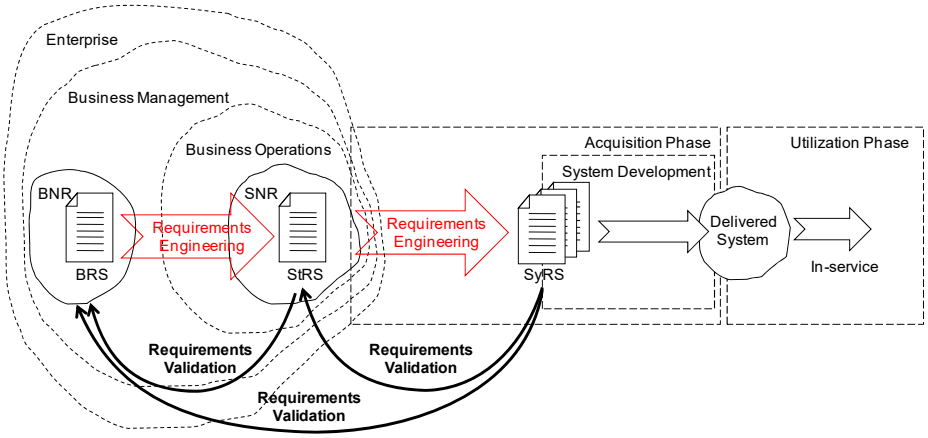

Figure 2-4. Validation of the requirements in the SyRS.

We must also accept that we can only guarantee zero errors in our requirements-engineering process by spending an infinite amount of time—at some point we must draw a line and say that we are sufficiently prepared to be able to continue the systems engineering process.

Broadly, there are two aspects to requirements validation, as discussed in the following sections:

- each requirement must be valid, and
- the complete set of requirements must be valid.

2.5.1 Validating Individual Requirements

We need to be able to convince ourselves that each requirement meets our conditions of being necessary, singular, correct, unambiguous, feasible, appropriate to level, complete, conforming, and verifiable. Since our requirements engineers undertook this task when writing the requirements, validation is probably best achieved by an independent reviewer.

We are also interested in making sure that we are complete in describing all aspects associated with each requirement. That is, we need to ensure that, in addition to the requirement statement, we have articulated performance and verification requirements (as well as added an appropriate rationale). We also need to ensure that we have addressed associated constraints, and exception conditions.

In some senses, however, validating individual requirements is relatively straightforward. Validating the set of requirements is much more difficult, however—as soon as we go past a couple of dimensions our human intuition quickly deserts us.

2.5.2 Validating the Requirement Set

Once we have validated each requirement and the associated statements, we must convince ourselves that the set of requirements meets the overall need. In short, the set must be:

- complete (all requirements included, irrelevant ones excluded);
- consistent (internally in that requirements do not conflict, as well as with external interfaces);
- comprehensible (a reader can 'visualise' the system from it);
- able to be validated (can be shown to meet the needs); and
- feasible (is it achievable, affordable, etc).

These sorts of characteristics were relatively easy to check in individual requirements but are much harder tasks when considering the set as a whole.

2.5.2.1 Characteristics of a Good Requirement Set

This section examines each characteristic in a little more detail and then examines some ways of validating the requirements set.

2.5.2.1.1 Complete

A good requirement set is complete—that is, it contains all of the requirements necessary to describe the system. Being able to assess completeness therefore requires that we can determine when we have finished the elicitation and elaboration, negotiation, and allocation processes. Despite ensuring that we make the best attempt at validating requirements, we must accept that our requirements-engineering process can only guarantee that we have gathered all requirements with zero errors by spending an infinite amount of time—somewhere we must draw a line and say that we are sufficiently prepared to continue the systems engineering process.

When is that point? Well, that is largely a matter of judgement—one of the reasons why the most capable of our systems engineers are allocated to requirements engineering. We can, however, come to some understanding of what our requirements would look like at the point we are happy to continue. We are aided by the fact that requirements engineering is an iterative process that will continue throughout the life cycle—so we only have to be confident at this stage that we have iterated sufficiently to have captured (and adequately organised) the bulk of the requirements so that any further iteration will only produce minor variations.

2.5.2.1.2 Consistent

A consistent set of requirements ensures that individual requirements are unique and do not conflict internally or with external interfaces. It is possible in some formal specification techniques to define a formal property of consistency and prove it using formal methods. These formal techniques are really only useful (cost-effective) in limited applications such as when the problem domain is small and well-bounded, or (as is the case in safety-critical systems) we are very concerned to guarantee consistency.

There are, however, some things that can be done to ensure consistency. While we can't feed the requirements through some automated tool and have a consistency factor returned and all inconsistent requirements flagged, we can use some tools to assist. For example, a requirements-management tool can be used to search for any requirement that does not have a parent requirement, or that does not spawn at least one child requirement.

Tools can also assist in presenting portions of the requirements set (ordered by type, for example) to assist in looking for potential conflicts between requirements. While the tool assistance is useful, consistency checks require considerable intuition and are therefore best conducted by experienced requirements engineers.

2.5.2.1.3 Comprehensible

Like consistency, comprehension is also difficult to describe formally. If natural specification language has been employed, we can look for structure and syntax within each requirement, but we don't have the same sort of assistance in ensuring that the set of requirements will be able to be comprehended by a broad range of stakeholders, requirements engineers, and lower-level designers. Visualisation techniques (such as the use of prototypes) therefore greatly aid the process of ensuring that the stakeholders understand what the collection of 'shall' statements is describing. Inclusion in the SyRS of figures and drawings as well as the visual artefacts of requirements engineering also greatly assists comprehension of the requirements set.

2.5.2.1.4 Able to be Validated

The transformation must be formal and able to be validated, not just for individual requirements, but also for the set. It must be able to be shown that achievement of the set of requirements will result in meeting the set of needs from which it was transformed.

2.5.2.1.5 Feasible

As with consistency, it is relatively easy to check that an individual requirement is realistic. For example, even a non-expert would easily be able

to identify an unrealistic requirement such as a mobile phone with a battery life of 50 days—even at first glance, it does not seem to be feasible and, even if it is, it doesn't seem like it will be practicable or affordable.

The combination of feasible individual requirements does not necessarily sum to a feasible set of requirements, however. For example, is our set of requirements feasible for these seemingly realistic individual requirements for a portable, external hard drive for a computer: weighs less than 1 kg, has a storage capacity of 1 TByte, is directly accessible on an Ethernet network, can be dropped from 1m without damage, can survive in temperatures of ±50°C, and retails for less than $100. While each of those requirements seems perfectly feasible, even at first glance, we cannot be so readily sure that the set is feasible (that is, all requirements can be met simultaneously). As soon as we go past a couple of dimensions our human intuition quickly deserts us.

We need experienced requirements engineers who have seen such systems defined before. Even then, they can probably only determine that the set is feasible by continuing the requirements/design process to the point where the design space is sufficiently well bounded and understood.

Again, then, we have a case for not worrying too much about getting to too low a level as we conduct the design—we should continue to the point where we are happy with the content and then 'write it up' at the (higher) level appropriate to the baseline document.

2.5.2.2 Validation Methods

Although there are few formal techniques that are available, there are a number of ways to assist in validation of the requirement set.

First, of course, a good top-down requirements-engineering process already has a significant measure of validation incorporated into it. As discussed in Chapter 5, when we develop the mission statement, we ask ourselves whether this single sentence was necessary, singular, correct, unambiguous, feasible, appropriate-to-level, complete, conforming, and verifiable. We then develop half a dozen goals and ask ourselves whether the achievement of those goals would result in the achievement of the mission and then whether each goal was necessary, singular, correct, unambiguous, feasible, appropriate-to-level, complete, conforming, and verifiable. We then analyse objectives to ensure that they would result in the achievement of the relevant goals, and to ensure that the objectives themselves are also necessary, singular, correct, unambiguous, feasible, appropriate-to-level, complete, conforming, and verifiable.

Further, the requirements-engineering artefacts have been developed top-down allowing us to validate requirements at each level.

When we were developing use cases and operational scenarios, we should also have developed a series of test cases (these are system validation test cases). Although our design only exists in paper form, we can 'do' some

of these tests. Walkthroughs (inspections) involve experienced systems engineers and designers 'walking through' the requirements using scenarios and test cases to ensure that the system has accommodated such a condition and is likely to produce the required response. Walkthroughs are therefore a form of 'mental' prototyping.

Again, the design artefacts discussed at the end of the chapter assist here as the stakeholder can be walked through any scenario at any chosen level of detail, using a visual format rather than reading centimetres of document containing 'shall' statements. Other forms of soft and hard simulation, prototyping, and modelling can be used to put the virtual system through its paces to determine whether the requirements stand up to a virtual 'trial'.

It is not always possible to produce simulations and models, but any forms of visualisation greatly assist during validation. Visualisation is particularly important for stakeholders who are not familiar with imagining what systems would look like from paper-based descriptions (even two-dimensional drawings).

2.5.2.3 Validation Plan

Since there are a number of ways (no single one of which is guaranteed of success) to validate individual requirements and the requirements set, a suitable validation plan must be developed very early on in the process. The plan should articulate the validation methods, who is going to be involved, who adjudicates conflicts, timings, major events such as review workshops, locations, and so on.

Since requirements engineering is a complex process and the requirements documents are necessarily complex, reviewing the requirements set will also be complex and can quickly become unmanageable unless considerable thought has been given to the process and all participants have had sufficient time to prepare.

2.5.2.4 Review Meetings

As with all formal meetings the major work of validation is conducted before the meeting—reviews are costly in time and care must be taken to minimise the impact on the schedules of the reviewers. Additionally, the reviewers should be chosen with care to ensure a balance across the stakeholders.

Before the meeting, documents are distributed early to give attendees an opportunity to identify and document any conflicts, omissions, or ambiguities. These responses are returned and an agenda created. The meeting is then held to discuss any problems with the individual requirements or the set of requirements. Action items are identified and allocated to appropriate individuals. The meeting chair follows up after the meeting to ensure that all action items have been completed.

2.6 REQUIREMENTS MANAGEMENT

Requirements management provides the processes by which the requirement set and any changes to that set are managed throughout the system life cycle. Requirements management involves establishing and ensuring compliance with a formal procedure for the elicitation/elaboration, verification, and traceability of requirements. The many detailed requirements in lower-level specifications have been derived from a relatively simple statement made by a stakeholder; these lower-level requirements must be managed as they are developed to ensure that each of the lower-level requirements can be justified. Additionally, a formal procedure is required for the control of changes to requirements.

2.6.1 Requirements Change Management

Throughout the entire life cycle of the system, we are very interested in managing the configuration of the system elements. This configuration management activity begins as soon as configuration items are defined at the end of Preliminary Design (in the ABL). Before that, however, we must manage the configuration of the requirements set—this process involves a formal change-management process to ensure that changes to the requirements set occur in a controlled way.

2.6.1.1 Change-management Policies

Change-management policies are defined to define the procedures, processes, and standards that are to be used to manage changes to requirements during the requirements-engineering process. Change-management policies should cover:

- The process by which a change is proposed.
- Definition of the information required to support each change request.
- The process used to analyse the reason for the change and to assess the impact of the change.
- The change-control authority (person or board) that considers and approves the change.
- The process by which the change-control authority receives, considers, and approves the change.
- Tool support for the change-management process.

2.6.1.2 Change-management Process

The requirements change-management process has four main activities:

- *Problem identification and analysis*. A problem with one or more requirements could arise during requirements analysis or allocation,

through a revised stakeholder need, from operational problems, or as a result of a change in technology. Once identified, the problem is analysed and a change is proposed. Change proposals (change request forms) would normally contain:

- a statement outlining the change,
- reasons for the change,
- alternatives available,
- preferred alternative detail,
- impact of the change especially interface impacts, and
- draft design documentation reflecting the change.

- *Change analysis*. The proposed change is analysed to determine which requirements are affected by the change and to analyse the impact (cost, time, and other effects) of the change.
- *Change approval.* The change is submitted to the change authority for approval. The change is described in a change proposal outlining why the change is required, what options are available, and which option is preferred. The change proposal may be rejected if the change is not considered to be necessary (or has already been proposed), the change is unacceptable to the stakeholders, or there are insufficient funds to support the change.
- *Change implementation.* Once the change has been approved the documentation set is amended accordingly. Any amendments are validated (in accordance with the current validation status).

2.6.2 Change Management—Traceability

Requirements cannot be managed effectively without *requirements traceability*, which allows us to be able to identify where each requirement came from, what requirements are related to it, and which requirement or requirements stemmed from it (that is, which is its child, or are its children). If a requirement must change for some reason, good traceability allows us to identify all requirements affected by the change. It has been estimated that some 35% of requirements for the average project will have been added after the requirements-engineering process is deemed to have ended [54]. Consequently, traceability is essential to support change management and supports the project management function of project scope management.

Broadly, there are two main sorts of traceability:

- *Forward traceability.* Forwards traceability gives us confidence that each requirement has been addressed in the design. Through forward traceability, design decisions can be traced from any given high-level requirement (a parent requirement) down to a detailed design decision (a child requirement). For each requirement, we must be able to find at least one child in the subordinate design documents.

If there is more than one child, we must be able to identify all of them.

- *Backward traceability*. We must be able to trace from each requirement back to at least one requirement in the parent requirement specification. This backward traceability ensures that additional requirements (not formally endorsed by the customer) have not crept (through *requirements creep*) into the design. Any aspect of the design that cannot be traced back to a higher-level requirement is likely to represent unnecessary work for which the customer is most probably paying a premium. That is, an orphan requirement must, by definition in such a process, be out-of-scope.

Backwards and forwards traceability can mean that an enormous amount of information must be recorded. Consequently, one of the key decisions to be made early in the requirements-engineering process is the degree to which traceability is implemented and the point at which traceability and change management start. If change management starts too early (with the mission, goals and objectives, for example) the large number of changes in the early iterations are not important and tracking them leads to a huge amount of information that is relatively useless. It is generally better to allow the design to mature to a reasonable degree, than it is to insert traceability information as soon as the requirements are entered into a database.

2.6.3 Requirements Management Tools

Due to the large number of requirements elicited/elaborated during the development of the SyRS, a requirements management tool is generally necessary to assist in the management of requirements. In particular, it is almost impossible to have requirements traceability without implementing the requirements in some automated context.

A simple Internet search reveals that there are approximately two dozen requirements management tools commercially available (such as Artisan Studio, CORE, and DOORS, to name only three) [55]. Rather than attempt to review a subset of the tools, this section describes some of the most important features that a requirements management tool should have to assist during the system-development process. In particular, a requirements management tool has the following desirable attributes:

- *Supports requirements elicitation.* The selected requirements management tools must support the initial capture of requirements, which may involve importing requirements statements from existing requirements documentation or extracting individual requirements from within a larger, more general source document. Once the requirements have been captured, the management tool must support the editing and addition of requirements attributes. Because the requirements management tool will trace all changes and there is a

very large rate of change of requirements during elicitation, there is some advantage in not populating a requirements management tool until the rate of requirements churn has reached an acceptably low level.

- *Allows readers to browse.* It is important for stakeholders to be able to relate to the requirements database in a natural way, so requirements management tools must allow the stakeholders to browse through the requirements in any order.
- *Allows specific requirements to be retrieved.* The tool must allow a specific requirement to be retrieved directly by its unique identifier, short title, or any part of its text.
- *Supports forward and backward traceability.* Traceability is critical to the systems engineering and requirements-engineering processes. Requirements management tools must support and maintain traceability between the different levels of requirements (from business to stakeholder to system to subsystem to assembly to component, as well as in the reverse direction), and between logical design and physical architecture.
- *Supports the generation of good requirements.* Some requirements management tools assist with the aim of producing good requirements by providing automated functions to check for inconsistencies and ambiguities. These functions are valuable, but stakeholders should establish a degree of confidence in the process initially and always cross check that the requirements are well written.
- *Generates reports.* A good requirements management tool can generate reports of requirements based on selected criteria (a selected attribute, selected combinations of attributes, or some textual string) to assist in validation of the requirements set.
- *Allows import and export of requirements in various formats*. It is also essential that the tool allows import and export of requirements in various database, word-processing and spread-sheet formats, and to interface with other requirements management tools. This ensures that the overall requirements management process can be maintained even when business managers, stakeholders, contractors, and sub-contractors use different tools.
- *Supports change control and change impact assessment.* Requirements change over time. Performance levels may change, new requirements may be added, and old requirements may be deleted. The requirements management tool must support this configuration management function. Other configuration management functions such as impact analysis, change proposals and approvals, and change history should also be available. For example, support for some form of impact analysis is necessary to

enable stakeholders to investigate the potential impact on the system of adding, deleting, or modifying requirements. As well as enabling changes, the tool should support the ability to restrict rights of users (to read-only, for example) in order to prevent unauthorised changes, and to keep track of which changes were made by whom.

- *Supports allocation and logical-to-physical translation.* During Preliminary Design, a physical architecture is established and logical requirements allocated to it. Some requirements management tools allow the physical architecture to be documented within the tool and support the process of requirements allocation. This feature, combined with an ability to perform impact analysis, becomes extremely valuable as the system development passes through the more-detailed design stages.
- *Does not enforce a particular requirements-engineering process.* It is essential that the requirements engineers do not have to modify their preferred methodologies and processes because they have adopted a tool that instantiates a different set of processes. Although most tools are delivered with built-in specification templates based on popular specification standards, requirements engineers should also be able to select their own specification format, which will require an ability to perform some formatting of the information.

2.7 REQUIREMENTS-ENGINEERING TOOLS

There are a large number of tools that may assist in requirements engineering, including the context diagram, functional flow block diagrams (FFBD), requirements breakdown structure (RBS), and N2 diagrams. Other tools include structured analysis, data flow diagrams (DFD), control flow diagrams (CFD), IDEF diagrams, behaviour diagrams, action diagrams, state/mode diagrams, process flow diagrams, function hierarchy diagrams, state transition diagrams (STD), entity relationship diagrams (ERD), structured analysis and design, object-oriented analysis (OOA), unified modelling language (UML), structured systems analysis and design methodology (SSADM), and quality function deployment (QFD). Each of these techniques focuses on gathering requirements in a formal systematic way.

In this text we focus on the philosophy of requirements engineering using the upper-level tools of the context diagram, FFBD, and RBS, leaving the reader to consult the many excellent references for the more detailed tools [56].

This section provides a brief introduction to the RBS and FFBD, which are used in the remaining chapters to illustrate the processes, activities and steps necessary to implement the methodology presented. The use of a context diagram is discussed in more detail in Chapter 6.

2.7.1 Requirements Breakdown Structure (RBS)

The requirements-engineering process can be assisted by the use of a requirements framework around which the logical description of the system can be based. Here we call the requirements framework the *requirements breakdown structure* (RBS) because, as illustrated in Figure 2.5, the requirements are shown in a tree-structured elaboration of the mission. The words are deliberately chosen to differentiate this structure from the well-known project management document called the *work breakdown structure* (WBS) [57]—the RBS is grouped logically (functionally), the WBS is structured by physical work packages (including physical system elements) and contains other project-related work—the logical groupings of the RBS are then allocated to the physical groupings of the physical system elements in the WBS. The RBS is also called a *functional hierarchy*.

In addition to being a useful requirements analysis tool, the RBS can be used to capture the business requirements in the BRS, which can be further refined to reflect stakeholder requirements in the StRS, and then the system requirements in the SyRS. When completely populated, the RBS for a system provides an outline of the SyRS. In the first instance, the system is described by the words or short phrases of the RBS; in the SyRS, these short titles are elaborated into well formed requirement statements articulating functional, performance, and verification requirements.

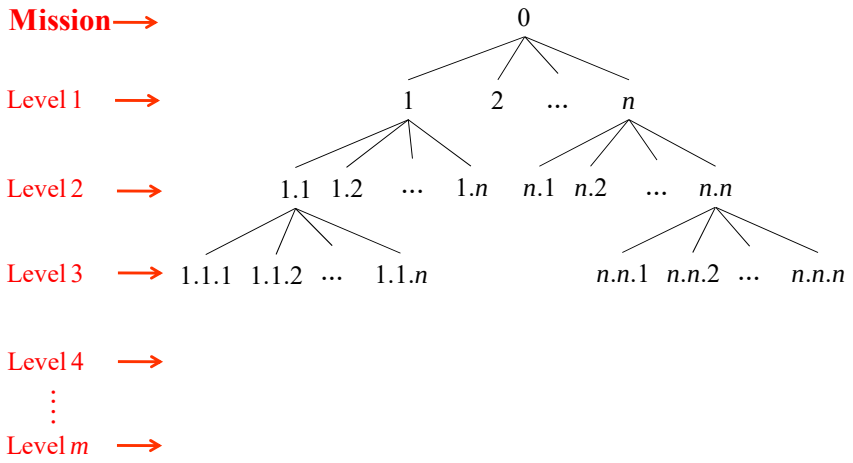

Figure 2.5. A simple RBS showing the hierarchical elaboration of requirements for the mission statement down to *m* levels.

There are a number of advantages associated with the establishment of a suitable framework such as the RBS early in the requirements analysis activity. First, the framework will act as a reference source during requirements analysis to ensure that all aspects of the system requirements are addressed and that important areas are not omitted. The framework allows

multiple people to work on the analysis simultaneously as it facilitates the effective allocation to individuals of responsibility for sections of requirements. The framework assists in avoiding duplication of requirements in different sections, which is undesirable as it raises the probability of conflict between requirements and often leads to ambiguity and confusion.

It should be noted, of course, that not all complex systems can be represented adequately in a hierarchical decomposition and we must remain cognisant that we might need to extend the representation to describe fully the system to reflect its possible emergent properties. Still, almost all workable human-made systems are hierarchical, so the RBS is an adequate representation [58].

When developing the RBS, some general rules should be followed:

- The RBS is to be displayed hierarchically. The level of the RBS illustrated should be such that the portion displayed should be visible on a single A4 sheet of paper, at no less than 10-point font, with no lines leaving or entering the page.
- Naming of RBS elements should be consistent so that the key-word phrases are either verb-based or noun-based. The discipline of using three words is useful, however, as it forces the author to stay focused on the function, particularly the verb (which is the function that has to be performed).
- The length of the key-word phrases should be consistent (three or four words are normally sufficient).
- The RBS elements are to be logical (functional) not physical.
- Each key-word phrase must be unique (since it must be fleshed out into a unique requirement).
- As illustrated in Figure 2.3 and Figure 2.5, the RBS elements should be numbered with a hierarchical numbering system so that parent requirements can be traced to their children and vice versa.
- There must be more than one child at each level—having only one child indicates that the child is redundant (it is simply a copy of the parent) or additional children must be elaborated.

2.7.2 Functional Flow Block Diagrams (FFBDs)

The hierarchical representation of requirements in the RBS can be supported by FFBDs [59], a simple example of which is shown in Figure 2-6. While the RBS is very useful to show a hierarchical decomposition of the requirements, FFBDs are useful in showing the interrelationship of the functions to be performed by the system in accomplishing its mission. Some functions must be performed sequentially; some may be performed in parallel; sometimes alternative paths may be taken based on certain conditions—the FFBDs can show these relationships much more easily than the RBS.

As with the RBS, FFBDs are developed top-down through hierarchical decomposition. Each top-level block can be expanded (decomposed) as

necessary to ensure that the functionality of the system is adequately defined, as illustrated in Figure 2-6.

When developing FFBDs, some general rules should be followed:

- Each function is to be presented in a single box enclosed in a single solid line.
- As with the RBS, naming of the FFDB elements should be consistent so that the key-word phrases are unique, either verb-based or noun-based, and are no more than three or four words long. The same names should be used for the function as it is displayed in both the FFBD and the RBS.
- As shown in Figure 2-6, the FFBDs should be displayed as a hierarchical elaboration.
- The level of the FFBD illustrated should be such that the portion displayed should be visible on a single A4 sheet of paper, at no less than 10-point font.
- As with the RBS, the FFBDs also make use of a hierarchical numbering system. Where possible, the same numbering system is to be used in the RBS and the FFBDs.
- As with the RBS, naming of the FFDB elements should be consistent so that the key-word phrases are unique, either verb-based or noun-based, and are no more than three or four words long. The same names should be used for the function as it is displayed in both the FFBD and the RBS.
- The lines that connect the blocks have arrows to show the flow of functions, not the elapsed time between them. Consistency in drawing is required—conventionally, the flow is from left to right of the FFBD.

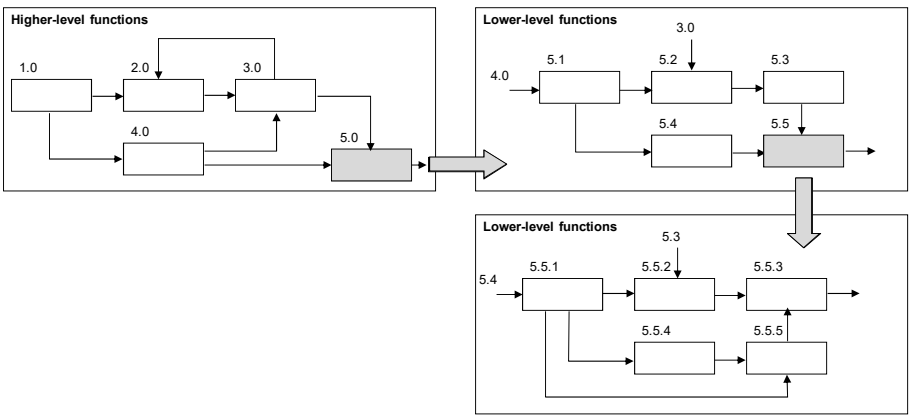

Figure 2-6. Example hierarchical decomposition of FFBDs.

If useful, additional blocks may be used to show such additional control constructs as decision points, repetition, iteration, concurrent flow (logical AND), and alternative flow (logical OR)—an example is illustrated in Figure 2-7.

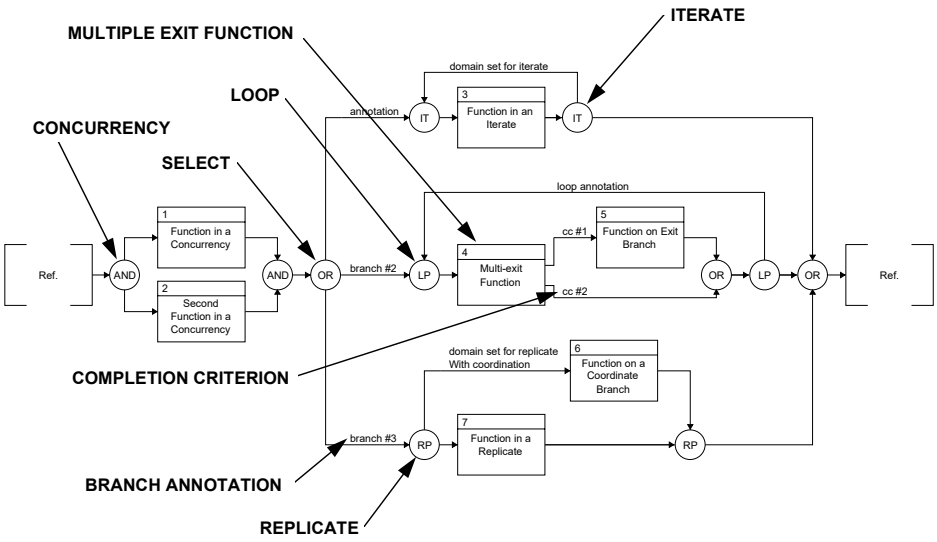

Figure 2-7. Example FFBD showing additional control constructs [60]. (reproduced with permission of Vitech Corporation)

2.8 REVIEW QUESTIONS

1. Define what is meant by a *need* and a *requirement*.
2. Briefly describe the *Business Needs and Requirements Definition Process*, including the roles of the *Preliminary Life-cycle Concept Documents (PLCD)* and the *Business Requirement Specification (BRS)*.
3. Briefly describe the *Stakeholder Needs and Requirements Definition Process*, including the roles of the *Life-cycle Concept Documents (LCD)* and the *Stakeholder Requirement Specification (StRS)*.
4. Briefly describe the *System Requirements Definition Process*, including the roles of the *System Requirement Specification (SyRS)*.
5. Briefly describe the role and content of the *concept of operations (ConOps)* and the *operational concept (OpsCon)*.
6. Briefly describe what is meant by a *functional requirement*, a *non-functional requirement*, and a *constraint*.
7. Briefly describe the nine *characteristics* of a good requirement.
8. Briefly explain the eleven main *attributes* that may be associated with each requirement.

9. Briefly explain why system-level requirements should describe *what* the system should do rather than *how* it should be done. Outline why this rule can sometimes be broken.
10. Briefly describe what is meant by the term *emergent properties*.
11. Briefly outline why we need requirements.
12. Briefly outline why we need requirements engineering.
13. Briefly describe what is meant by the terms *elicitation*, *elaboration*, *decomposition*, and *derivation*, in the context of requirements engineering.
14. Briefly describe the seven dimensions to *requirements elicitation*.
15. Briefly describe the major difficulties in eliciting and elaborating requirements.
16. Briefly explain the major techniques for eliciting requirements.
17. Briefly describe the requirements-engineering activity of *requirements validation*. Describe the difference between validating each requirement and validating the complete set of requirements.
18. Briefly describe the characteristics of a good set of requirements: *complete*, *consistent*, *comprehensible*, *able to be validated*, and *feasible*.
19. Briefly describe what is involved in *requirements management*.
20. Briefly describe *requirements change-management*, including the principal activities.
21. Briefly describe *requirements traceability* (including *forward traceability* and *backward traceability*).
22. List ten desirable features of a requirements management tool.
23. Briefly describe the *requirement breakdown structure (RBS)* and its role as a requirements framework.
24. Briefly describe *functional flow block diagrams (FFBDs)* and their role in defining functional requirements.

ENDNOTES

[1] In addition to information contained in the general systems engineering references listed at the end of Chapter 1, specific detail on requirements engineering can be found in:
Adzic, G., *Specification by Example*, Shelter Island NY: Manning Publications Co., 2011.
Alexander I.F., and L. Beus-Dukic, *Discovering Requirements: How to Specify Products and Services*, Chichester, England: John Wiley & Sons, 2007.
Alexander I.F., and R. Stevens, *Writing Better Requirements*, London: Addison Wesley, 2002.
Andriole, S., *Managing Systems Requirements: Methods, Tools and Cases*, New York: McGraw-Hill, 1996.
Berenbach, B., D.J. Paulish, J. Kazmeier, A. Rudolf, *Software and Systems*

Requirements Engineering in Practice, New York: McGraw-Hill, 2009.
Bray I.K., *An Introduction to Requirements Engineering*, Boston, MA: Addison Wesley, 2002.
Davis, A.M., *Software Requirements: Analysis and Specification*, Engelwood Cliffs, NJ: Prentice Hall International, 1990.
Davis, A.M., *Software Requirements: Objects, Functions and States*, Engelwood Cliffs, N.J.: Prentice Hall International, 1993.
Davis, A.M., *Just Enough Requirements Management: Where Software Development Meets Marketing*, New York, NY: Dorset House Publishing, 2005.
Dorfman, M. and R.H. Thayer, *System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.
Ferdinandi, P.L., *A Requirements Pattern: Succeeding in the Internet Economy*, Boston, MA: Addison-Wesley, 2001.
Fitchett, P. and J. Haslam, *Writing Engineering Specifications*, London, UK: Spon Press, 1988.
Gause, D. and G. Weinberg, *Exploring Requirements: Quality Before Design*, New York: Dorset House Publishing Company Incorporated, 1989.
Gilb, T., *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Butterworth-Heinemann, 2005.
Goldsmith, R.F., *Discovering Real Business Requirements for Software Project Success*, Norwood, M.A: Artech House, 2004.
Gottesdiener, E., *Requirements by Collaboration: Workshops for Defining Needs*, Reading, MA: Addison Wesley, 2002.
Grady, J., *Systems Requirements Analysis*, New York: McGraw-Hill Book Co., 2006.
Hass, K.B., D.J. Wessels, and K. Brennan, *Getting it Right: Business Requirement Analysis Tools and Techniques*, Vienna, VA: Management Concepts, 2008.
Hatley, D.J., P. Hruschka, and I.A. Pirbhai, *Process for System Architecture and Requirements Engineering*, New York: Dorset House Publishing Company, 2000.
Hay, D.C., *Requirements Analysis: From Business Views to Architecture*, Upper Saddle River, NJ: Prentice Hall, 2003.
Hood, C., S. Wiedemann, S. Fichtinger, and U. Pautz, *Requirements Management: The Interface Between Requirements Development and All other Systems Engineering Processes*, Oberhaching: Springer, 2008.
Hooks, I.F. and K.A. Farry, *Customer-centred Products: Creating Successful Products Through Smart Requirements Management*, New York: AMACOM, 2001.
Hossenlopp, R. and K.B. Hass, *Unearthing Business Requirements: Elicitation Tools and Techniques*, Vienna, VA: Management Concepts, 2008.
Hull, M.E.C., K. Jackson, and A.J.J. Dick, *Requirements Engineering*, London: Springer, 2011.
Jackson, M., *Software Requirements and Specifications: A Lexicon of Software Practice, Principles and Prejudices*, Reading, M.A.: Addison Wesley, 1995.
Jonasson, H., *Determining Project Requirements, Second Edition: Mastering*

the BABOK® and the CBAP® Exam, Boca Raton, FL: Auerbach Publications, 2012
Kossmann, M., *Requirements Management: How to Ensure You Achieve What You Need From Your Projects*, Gower, 2013.
Kotonya, G. and I. Sommerville, *Requirements Engineering: Processes and Techniques*, West Sussex, England: John Wiley & Sons, 2000.
Kulak, D. and E. Guiney, *Use Cases: Requirements in Context*, Boston, MA: Addison-Wesley, 2004.
Laplante, P.A., *Requirements Engineering for Software and Systems*, Boca Raton, FL: CRC Press, 2009.
Larson, E. and R. Larson, *Practitioner's Guide to Requirements Engineering—Part 1: Requirements Planning*, Minneapolis, MN: Watermark Learning Publications, 2009.
Leffingwell, D., and D. Widrig, *Managing Software Requirements: A Unified Approach*, Reading, M.A.: Addison-Wesley, 2000.
Loucopoulos, P., and V. Karakostas, New York: McGraw-Hill, 1995.
Lutowski, R., *Software Requirements: Encapsulation, Quality, and Reuse*, Boca Raton, FL: Taylor and Francis, 2005.
Macaulay, L., *Requirements Engineering*, London, England: Springer, 1996.
Mate J.L., and A. Silva (editors), *Requirements Engineering for Sociotechnical Systems*, Hershey, PA: Information Science Publishing, 2005.
Pohl, K., *Requirements Engineering Fundamentals*, Santa Barbara CA: Rocky Nook, 2011.
Pohl, K., and C. Rupp, *Requirements Engineering: Fundamentals, Principles, and Techniques*, Heidelberg: Springer-Verlag, 2010.
Pohl, K., *Requirements Engineering: Fundamentals, Principles, and Techniques*, Heidelberg: Springer-Verlag, 2010.
Purdy, D., *A Guide to Writing Successful Engineering Specifications*, New York: McGraw-Hill, 1991.
Rinzler, B., *Telling Stories: A Short Path to Writing Better Software Requirements*, Wiley, 2009.
Robertson S., and J. Robertson, *Mastering the Requirements Process*, Harlow, England: Addison-Wesley, 2013.
Sommerville, I., and P. Sawyer, *Requirements Engineering*, Chichester: England, John Wiley & Sons, 1997.
Thayer, R.H. and M. Dorfman (editors), *Software Requirements Engineering*, Los Alamitos, CA: IEEE Computer Society Press, 1997.
van Lamsweerde, A., *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Chichester, England: John Wiley & Sons, 2009.
Wiegers, K.E. and J. Beatty, Software Requirements, Redmond, WA: Microsoft Press, 2013.
Wieringa, R., *Requirements Engineering: Frameworks for Understanding*, Wiley, 1996.
Withall, S., *Software Requirements Patterns (Best Practices)*, Redmond, MA: Microsoft Press, 2007.
Young, R., *Effective Requirements Practices*, Boston, M.A.: Addison-Wesley, 2001.

Young, R., *The Requirements Engineering Handbook*, Norwood, M.A, Artech House, 2004.
Young, R., *Project Requirements: A Guide to Best Practices*, Vienna VA: Management Concepts, 2006.

[2] Ryan, M.J. and L.S. Wheatcraft, "On a Cohesive Set of Requirements Engineering Terms", *Systems Engineering*, 2017.

[3] Ryan, M.J., "An Improved Taxonomy for Major Needs and Requirements Artefacts", *INCOSE International Symposium IS2013*, June 2013.

[4] Ryan, M.J. and L.S. Wheatcraft, "On a Cohesive Set of Requirements Engineering Terms", *Systems Engineering*, 2017.

[5] Ryan, M.J. and L.S. Wheatcraft, "On a Cohesive Set of Requirements Engineering Terms", *Systems Engineering*, 2017.

[6] International Institute for Business Analysis, *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide)*, Version 2, 2009.

[7] *International* Institute for Business Analysis, *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide)*, Version 2, 2009.

[8] ANSI/AIAA G-043A-2012, *Guide for the Preparation of Operational Concept Descriptions*, American National Standards Institute, American Institute of Aeronautics and Astronautics (sponsor), 2012.

[9] *International* Institute for Business Analysis, *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide)*, Version 2, 2009.

[10] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[11] International Institute for Business Analysis, *A Guide to the Business Analysis Body of Knowledge® (BABOK®* Guide*)*, Version 2, 2009.

[12] MIL-STD-499B, *Military Standard—Systems Engineering—Draft*, Washington D.C.: United States of America Department of Defense, 1994.

[13] ANSI/AIAA G-043A-2012, *Guide for the Preparation of Operational Concept Descriptions*, American National Standards Institute, American Institute of Aeronautics and Astronautics (sponsor), 2012.

[14] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[15] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[16] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[17] MIL-STD-499B, *Military Standard—Systems Engineering—Draft*, Washington D.C.: United States of America Department of Defense, 1994.

[18] IEEE-STD-1220-2005, *IEEE Standard for Application and Management of the Systems Engineering Process*, New York: IEEE Computer Society, 2005.

[19] ANSI/EIA-632-1998, *Processes for Engineering a System*, Washington, D.C.: Electronic Industries Association (EIA), 1999.

[20] International Institute for Business Analysis, *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide)*, Version 2, 2009.

[21] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[22] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148, Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2017.

[23] Ryan, M.J. and L.S. Wheatcraft, "On a Cohesive Set of Requirements Engineering Terms", *Systems Engineering*, 2017.

[24] Ryan, M.J. and L.S. Wheatcraft, "On a Cohesive Set of Requirements Engineering Terms", *Systems Engineering*, 2017.

[25] Wheatcraft L.S., M.J. Ryan and J. Dick, "On the Use of Attributes to Manage Requirements", Volume 19, Issue 5, September 2016, pp. 448–458.

[26] IEEE-STD-1233-1998, *IEEE Guide for Developing System Requirements Specifications*, New York: The Institute of Electrical and Electronic Engineers (IEEE), 1998. Note that a similar list is in IEEE 29148.

[27] Verner, J., K. Cox, S. Bleistein, and N. Cerpa, Requirements Engineering and Software Project Success: An Industrial Survey in Australian and the U.S.", *Australasian Journal of Information Systems*, Vol. 13, No. 1, September 2005.

[28] Andriole, S., "The Politics of Requirements Management", *IEEE Software*, November/December 1998, pp. 82−84.

[29] Dorfman, M. and R.H. Thayer, *System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.

[30] Kotonya, G. and I. Sommerville, *Requirements Engineering*, John Wiley & Sons, Chichester, 2000.

[31] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148 FDIS Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, 2011.

[32] Project Management Institute Standards Committee, *A Guide to the Project Management Body of Knowledge*, Upper Darby P.P: Project Management Institute, 2013.

[33] Davis, A.M., *Software Requirements: Objects, Functions and States*, Englewood Cliffs, NJ: Prentice Hall, 1993.

[34] Verner, J., K. Cox, S. Bleistein, and N. Cerpa, "Requirements Engineering and Software Project Success: an Industrial Survey in Australia and the U.S.", *Australasian Journal of Information Systems*, Adelaide, Vol. 13, No. 1, September 2005.

[35] Hitchins, D., *Systems Engineering: A 21st Century Systems Methodology*, Chichester, *England*: John Wiley & Sons, 2007.

[36] Grady, J.O., *System Requirements Analysis*, Burlington M.A.: Academic Press, 2006.

[37] Daniels, J. and T. Bahill., "The Hybrid Process That Combines Traditional Requirements and Use Cases", *Systems Engineering*, Vol. 7, No. 4, 2004.

[38] IEEE Computer Society, *IEEE-STD-1220-2005—IEEE Standard for Application and Management of the Systems Engineering Process*, New York: IEEE Computer Society, 2005.

[39] See also a similar taxonomy in: Kotonya, G. and I. Sommerville, *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, Chichester, 2000.

[40] Young, R., *The Requirements Engineering Handbook*, Norwood, M.A, Artech House, 2004.

[41] Brooks, F., *The Mythical Man Month: Essays on Software Engineering*, Chapel Hill: Addison-Wesley, 1995.

[42] Alexander, I.F. and R. Stevens, *Writing Better Requirements*, Addison-Wesley, London, 2002.

[43] Goguen, J. and C. Linde, "Techniques for Requirements Elicitation", *International Symposium on Requirements Engineering*, San Diego, California: IEEE Computer Society Press, pp. 152–164, 1993.

[44] Hull, M.E.C., Jackson, K., and Dick, A.J.J., et al., *Requirements Engineering,* Springer, London, 2011.

[45] Katonya, G. and I. Sommerville, *Requirements Engineering: Processes and Techniques*, John Wiley and Sons, Chichester, 2000.

[46] Sommerville, I. and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley and Sons, Chichester, 1997.

[47] Young, R.R., *Effective Requirements Practices*, Addison-Wesley, London, 2001.

[48] Young, R.R., *The Requirements Engineering Handbook*, Artech House, Norwood, MA, 2004.

[49] IEEE-STD-1233-1998, *IEEE Guide for Developing System Requirements Specifications*, New York: The Institute of Electrical and Electronic Engineers (IEEE), 1998.

[50] Verner, J., K. Cox, S. Bleistein, and N. Cerpa, "Requirements Engineering and Software Project Success: an Industrial Survey in Australia and the U.S.", *Australasian Journal of Information Systems*, Adelaide, Vol. 13, No. 1, September 2005.

[51] Coulin, C., D. Zowghi, and A.E.K. Sahraoui, "A Situational Method Engineering Approach to Requirements Elicitation Workshops in the Software Development Process", *Software Process Improvement and Practice 11*, 451–464, 2006.

[52] Tsumaki, T. and T. Tamai, "Framework for Matching Requirements Elicitation Techniques to Project Characteristics", *Software Process Improvement and Practice 11*, pp. 505–519, 2006.

[53] Specific detail on use-case modelling can be found in:
Bittner, K. and I. Spence, *Use Case Modelling*, Boston, MA: Addison-Wesley, 2003.
Cockburn, A., *Writing Effective Use Cases*, Boston, MA: Addison-Wesley, 2001.
Kulak, D. and E. Guiney, *Use Cases: Requirements in Context*, Boston, MA: Addison-Wesley, 2004.
Rinzler, B., *Telling Stories: A Short Path to Writing Better Software Requirements*, Wiley, 2009.

[54] Jones, C., *Software Quality—Analysis and Guidelines for Success*, International Thomson, 1997.

[55] See also: Andriole, S., *Managing Systems Requirements: Methods, Tools and Cases*, New York: McGraw-Hill, 1996.

[56] Specific detail on analysis tools can be found in:
Blanchard, B. and W. Fabrycky, *Systems Engineering and Analysis*, Upper Saddle River, N.J.: Prentice Hall, 2011.
Coad P. and E. Yourdon, *Object Oriented Analysis*, Upper Saddle River, N.J.: Prentice Hall, 1990
Defense Systems Management College (DSMC), *Systems Engineering Management Guide*, Washington, D.C.: U.S. Government Printing Office, 1990.
Eisner, H., *Essentials of Project and Systems Engineering Management*, New York: John Wiley & Sons, 2008.
Gause, D. and G. Weinberg, *Exploring Requirements: Quality Before Design*, New York: Dorset House Publishing Company Incorporated, 1989.
Goodland, M. and C. Slater, *SSADM: A Practical Approach*, London: McGraw-Hill Publishing Company, 1995.
Grady, J.O., *Systems Requirements Analysis*, Burlington, M.A., Academic Press, 2006.
Heap, G., J. Stanway and A. Windsor, *A Structured Approach to Systems Development*, London: McGraw-Hill Book Company, 1992.
Hitchins, D., *Putting Systems to Work*, Chichester, England: John Wiley & Sons, 1992.
Kotonya, G. and I. Sommerville, *Requirements Engineering: Processes and Techniques*, West Sussex, England: John Wiley & Sons, 2000.
Lacy, J., *Systems Engineering Management: Achieving Total Quality*, New York: McGraw-Hill Book Co., 1992.
Leffingwell, D. and D. Widrig, *Managing Software Requirements: A Unified Approach*, Reading, M.A.: Addison-Wesley, 2000.
Macaulay, L., *Requirements Engineering*, London, England: Springer, 1996.
Maier, M. and E. Rechtin, *The Art of Systems Architecting*, Boca Raton, F.L.: CRC Press, 2009.
Martin, J., *Principles of Object-oriented Analysis and Design*, Upper Saddle River, N.J.: Prentice Hall, 1993.
Martin, J., *Systems Engineering Guidebook: A Process for Developing Systems and Products*, Boca Raton, F.L.: CRC Press, 1997.
Martin, J. and J. Odell, *Object-oriented Analysis and Design*, Upper Saddle River, N.J.: Prentice Hall, 1992.
Rumbaugh, J., et al, *Object-oriented Modeling and Design*, Upper Saddle River, N.J.: Prentice Hall, 1991.
Sage, A. and J. Armstrong, *Introduction to Systems Engineering*, New York: John Wiley & Sons, 2000.
Shlaer, S. and S. Mellor, *Object-oriented Systems Analysis*, Upper Saddle River, N.J.: Prentice Hall, 1988.
Stevens, R., P. Brook, K. Jackson, and S. Arnold, *Systems Engineering: Coping with Complexity*, Hertfordshire, England: Prentice Hall Europe, 1998.
Young, R., *Effective Requirements Practices*, Boston, M.A.: Addison-Wesley,

2001.
Westerman, H., *Systems Engineering Principles and Practice*, Boston, M.A.: Artech House, 2001.

[57] Project Management Institute (PMI), *Guide to the Project Management Body of Knowledge*, Newtown Square, PA: Project Management Institute (PMI), 2013.

[58] Simon, H., *The New Science of Management Decision*, Englewood Cliffs, NJ: Prentice-Hall, 1977.

[59] Blanchard and Fabrycky, (Blanchard, B. and W. Fabrycky, *Systems Engineering and Analysis*, Upper Saddle River, N.J.: Prentice-Hall, 2011) note that these generic FFBDs can be prepared by any one of the graphical methods including Integrated DEFinition (IDEF) modeling method, the Behavioral Diagram Method, and the N^2 Charting Method, which are compared and discussed in Blanchard, B., D. Verna and E. Peterson, *Maintainability: A Key to Effective Serviceability and Maintenance Management*, New York: John Wiley & Sons, 1995. The DSMC management guide (Defense Systems Management College (DSMC), *Systems Engineering Management Guide*, Washington, D.C.: U.S. Government Printing Office, 1990, p. 6.3–6.5) explains how FFBDs can be used during function identification.

[60] J. Long, *Relationship Between Common Graphical Representations Used in Systems Engineering*, National Council on Systems Engineering (NCOSE) Symposium, 1995.