CHAPTER 2
SOFTWARE REQUIREMENTS

Pete Sawyer and Gerald Kotonya
Computing Department,
Lancaster University
United Kingdom
{sawyer} {gerald}@comp.lancs.ac.uk

Table of Contents

1 INTRODUCTION

This document proposes a breakdown of the SWEBOK Software Requirements Knowledge Area. The knowledge area is concerned with the acquisition, analysis, specification, validation and management of software requirements. It is widely acknowledged within the software industry that software projects are critically vulnerable when these activities are performed poorly. This has led to the widespread use of the term 'requirements engineering' to denote the systematic handling of requirements. This is the term we use in the rest of this document. Software requirements are one of the products of the requirements engineering process.

Software requirements express the needs and constraints that are placed upon a software product that contribute to the satisfaction of some real world application [Kot00]. The application may be, for example, to solve some business problem or exploit a business opportunity offered by a new market. It is important to understand that, except where the problem is motivated by technology, the problem is an artifact of the problem domain and is generally technology-neutral. The software product alone may satisfy this need (for example, if it is a desktop application), or it may be a component (for example, a speech compression module used in a mobile phone) of a software-intensive system for which satisfaction of the need is an emergent property. In fundamental terms, the way in which the requirements are handled for stand-alone products and components of software-intensive systems is the same.

One of the main objectives of requirements engineering is to discover how to partition the system; to identify which requirements should be allocated to which components. In some systems, all the components will be implemented in software. Others will comprise a mixture of technologies. Almost all will have human users and sometimes it makes sense to consider all components of the system to which requirements should be allocated (for example, to save costs or to exploit human adaptability and resourcefulness). Because of this requirements engineering is fundamentally an activity of systems engineering rather than one that is specific to software engineering. In this respect, the term 'software requirements engineering' is misleading because it implies a narrow scope concerned only with the handling of requirements that have already been acquired and allocated to software components. Since it is increasingly common for practicing software engineers to participate in the elicitation and allocation of requirements, it is essential that the scope of the knowledge area extends to the whole of the requirements engineering process.

One of the fundamental tenets of good software engineering is that there is good communication between system users and system developers. It is the requirements engineer who is the conduit for this communication. They must mediate between the domain of the system user (and other stakeholders) and the technical world of the software engineer. This requires that they possess technical skills, an ability to acquire an understanding of the application domain, and the inter-personal skills to help build consensus between heterogeneous groups of stakeholders [Gog93].

We have tried to avoid domain dependency in the document. The knowledge area document identifies requirements engineering practice and identifies when it is and isn't appropriate. We recognise that desktop software products are different from nuclear reactor control systems and the document should be read in this light. Where we refer to particular tools, methods, notations, SPI models,

etc. it does not imply our endorsement of them. They are merely used as examples.

2 DEFINITION OF THE SOFTWARE REQUIREMENTS KNOWLEDGE AREA

This section provides an overview of requirements engineering in which:

- the notion of a 'requirement' is defined;
- motivations for systems are identified and their relationship to requirements is discussed;
- a generic process for analysis of requirements is described, followed by a discussion of why, in practice, organisations often deviate from this process; and
- the deliverables of the requirements engineering process and the need to manage requirements are described.

This overview is intended to provide a perspective or 'viewpoint' on the knowledge area that complements the one in Section 3 – Breakdown of topics for the Software Requirements Knowledge Area.

2.1 What is a requirement?

At its most basic, a requirement is a property that must be exhibited in order to solve some problem of the real world [Pfl98, Kot00, Som01, Tha97]. This document refers to requirements on 'systems' rather than 'solutions' because it is concerned with problems that have software-based solutions. Hence, a requirement is a property that must be exhibited by a system developed or adapted to solve a particular problem. The problem may be to automate part of a task of someone who will use the system, to support the business processes of the organisation that has commissioned the system, to correct shortcomings of an existing system, to control a device and many more. The functioning of users, business processes and devices are typically complex. By extension, therefore, the requirements on a system are typically a complex combination of requirements from different people at different levels of an organisation and from the environment in which the system must operate.

Requirements vary in intent and in the kinds of properties they represent. A distinction can be drawn between *product parameters* and *process parameters*. Product parameters are requirements on the system to be developed and can be further classified as [Kot00, Som97]:

- Functional requirements on the system such as formatting some text or modulating a signal. Functional requirements are sometimes known as capabilities.
- Non-functional requirements that act to constrain the solution. Non-functional requirements are sometimes known as constraints or quality requirements. They can be further classified according to whether they are (for example) performance requirements, maintainability requirements, safety requirements, reliability requirements, electro-magnetic compatibility requirements and many other types of requirements.

A process parameter is essentially a constraint on the development of the system (e.g. 'the software shall be written in Ada'). These are sometimes known as process requirements.

Requirements must be stated clearly and unambiguously and, where appropriate, quantitatively. It is important to avoid vague and unverifiable requirements that depend for their interpretation on subjective judgement ('the system shall be reliable', 'the system shall be user-friendly'). This is particularly important for non-functional requirements. Two examples of quantified requirements are: that a system must increase a call-center's throughput by 20%; and a requirement that a system shall have a probability of generating a fatal error during any hour of operation of less than $1 * 10^{-8}$. The throughput requirement is at a very high level and will need to be used to derive a number of detailed requirements. The reliability requirement will tightly constrain the system architecture [Dav93, Som01].

Some requirements are emergent properties. That is, requirements that can't be addressed by a single component, but which depend for their satisfaction on how all the system components inter-operate. The throughput requirement for a call-centre given above would, for example, depend upon how the telephone system, information system and the operators all interacted under actual operating conditions. Emergent properties are crucially dependent upon the system architecture.

An essential property of all requirements is that they should be verifiable. It may be difficult or costly to verify certain requirements. For example, verification of the throughput requirement on the call-center may necessitate the development of simulation software. The requirements engineering and V&V personnel must ensure that the requirements can be verified within the available resource constraints.

Some requirements generate implicit process requirements. The choice of verification method is one example. Another might be the use of particularly rigorous analysis techniques (such as formal specification methods) to reduce systemic errors that can lead to inadequate reliability. Process requirements may also be imposed directly by the development organization, their customer, or a third party such as a safety regulator.

Requirements have other attributes in addition to the behavioural property that they express. Common examples include a priority rating to enable trade-offs in the face of finite resources and a status value to enable project progress to me monitored. Every requirement must be uniquely

identified so that they can be subjected to configuration control and managed over the entire system life cycle.

2.2 System requirements and process drivers

The literature on requirements engineering sometimes calls system requirements "user requirements". We prefer a restricted definition of the term user requirements in which they denote the requirements of the people who will be the system customers or end-users. System requirements, by contrast, are inclusive of user requirements, requirements of other stakeholders (such as regulatory authorities) and requirements that do not have an identifiable human source. Typical examples of system stakeholders include (but are not restricted to):

- Users – the people who will operate the system. Users are often a heterogeneous group comprising people with different roles and requirements.
- Customers – the people who have commissioned the system or who represent the system's target market.
- Market analysts – a mass-market product will not have a commissioning customer so marketing people are often needed to establish what the market needs and to act as proxy customers.
- Regulators – many application domains such as banking and public transport are regulated. Systems in these domains must comply with the requirements of the regulatory authorities.
- System developers – these have a legitimate interest in profiting from developing the system by, for example, reusing components in different products. If, in this scenario, a customer of a particular product has specific requirements that compromise the potential for component reuse, the developer must carefully weigh their own stake against those of the customer. For mass-market products, the developer is often the primary stakeholder because they wish to maintain the product in as large a market as possible for as long as possible.

In addition to these human sources of requirements, important system requirements often derive from other devices or systems in the environment, which require some services of the system or act to constrain the system, or even from fundamental characteristics of the application domain [Lou95, Tha97]. For example, a business system may be required to inter-operate with a legacy database and many military systems have to be tolerant of high levels of electro-magnetic radiation. We talk of 'eliciting' requirements but in practice the requirements engineer has to systematically extract and inventory the requirements from a combination of human stakeholders, the system's environment, feasibility studies, market analyses, business plans, analyses of competing products and domain knowledge [Som97].

The elicitation and analysis of system requirements needs to be driven by the need to achieve the overall project aims. To provide this focus, a business case should be made which clearly defines the benefits that the investment must deliver. These should act as a 'reality check' that can be applied to the system requirements to ensure that project focus does not drift. Where there is any doubt about the technical, operational or financial viability of the project, a feasibility analysis should be conducted. This is designed to identify project risks and assess the extent to which they threaten the system's viability. Risks should be documented in the project management plan.

Typical risks include the ability to satisfy non-functional requirements such as performance, or the availability of off-the-shelf components. In some specialised domains, it may be necessary to design simulations to generate data to enable an assessment of the project risks to be made. In domains such as public transport where safety is an issue, a hazard analysis should be conducted from which safety requirements can be identified.

2.3 Overview of requirements analysis

Once the aims of the project have been established, the work of eliciting, analysing and validating the system requirements can commence. This is crucial to gaining a clear understanding of the problem for which the system is to provide a solution and its likely cost [Tha97].

The requirements engineer must strive for completeness by ensuring that all the relevant sources of requirements are identified and consulted. It will usually be infeasible to consult everyone. There may be many of users of a large system, for example. However, representative examples of each class of system stakeholder should be identified and consulted. Although individual stakeholders will be authoritative about aspects of the system that represent their interests or expertise, the requirements engineer has the responsibility to create the 'big picture' to permit for the assurance of completeness with all individual stakeholders.

Elicitation of the stakeholders' requirements is rarely easy and the requirements engineer has to learn a range of techniques for helping people articulate how they do their jobs and what would help them do their jobs better. There are many social and political issues that can affect stakeholders' requirements and their ability or willingness to articulate them and it is necessary to be sensitive to them [Gog93]. In many cases, it is necessary to provide a contextual framework that serves to focus the consultation; to help the stakeholder identify what is possible and help the requirements engineer verify their understanding. Exposing the stakeholders to prototypes may help, and these don't necessarily have to be high fidelity. A series of rough sketches on a flip chart can sometimes serve the same purpose as a software prototype, whilst avoiding the pitfalls of distraction caused by cosmetic features of the software. Walking the stakeholder through a small number

of scenarios representing sequences of events in the application domain can also help the stakeholder and requirements engineer to explore the key factors affecting the requirements.

Once identified, the system requirements should be validated by the stakeholders and trade-offs negotiated before further resources are committed to the project. To enable validation, the system requirements are normally kept at a high level and expressed in terms of the application domain rather than in technical terms. Hence the system requirements for an Internet book store will be expressed in terms of books, authors, warehousing and credit card transactions, not in terms of the communication protocols, or key distribution algorithms that may form part of the solution. Too much technical detail at this stage obscures the essential characteristics of the system viewed from the perspective of its customer and users.

Some system requirements may not be satisfiable. Some may be technically infeasible, others may be too costly to implement and some will be mutually incompatible. The requirements engineer must analyse the requirements to understand their implications and how they interact. They must be prioritised and their costs estimated. The goal is to identify the scope of the system and a 'baseline' set of system requirements that is feasible and acceptable. This may necessitate helping stakeholders whose requirements conflict (with each other or with cost or other constraints) to negotiate acceptable trade-offs.

To help the analysis of the system requirements, conceptual models of the system are constructed. These aid understanding of the logical partitioning of the system, its context in the operational environment and the data and control communications between the logical entities. In general, a mix of static (e.g. an object model) and dynamic (e.g. event traces and state diagrams) should be developed to explore different aspects of the system and it's problem domain. However, the choice of which aspects to model is conditioned by the nature of the problem domain.

The system requirements must be analysed in the context of all the applicable constraints. Constraints come from many sources, such as the business environment, the customer's organizational structure and the system's operational environment. They include budget, schedule, technical (non-functional requirements), regulatory and other constraints. Hence, the requirements engineer's job is not restricted to eliciting stakeholders' requirements, but includes making assessments of their feasibility. Requirements that are clearly infeasible should be rejected and the reason for rejection recorded. Requirements that are merely suspected of being infeasible are more difficult. A feasibility study may be justified if, for example, a doubtful requirement is strongly advocated by stakeholders [Kot00, Lou95].

Project resources should be focused on the most important priority requirements. In principle, the requirements should be both *necessary and sufficient* – there should be nothing left out or anything that doesn't need to be included. Achieving this is, of course, difficult. The absence of important requirements information can only be detected by rigorous analysis. Similarly, it may take considerable effort to reach consensus on requirement priorities because one stakeholder's essential requirement may have only cosmetic value to another. In practice, the existence of sufficient resources will allow some non-essential requirements to be satisfied, while insufficient resources may force even strongly advocated requirements to be excluded. Regardless of how the baseline is identified, requirements and V&V personnel must derive acceptance tests that will assure compliance with the requirements before delivery or release of the product.

Eventually, a complete and coherent set of system requirements will emerge as the result of the analysis process. At this point, the principal areas of functionality should be clear. Subsystems or components are defined to handle each principle area of functionality. The system requirements are then allocated or distributed to subsystems/components.

This activity of partitioning and allocation is part of architectural design. Architectural design is a skill that is driven by many factors such as the recognition of reusable architectural 'patterns' or the existence of off-the shelf components. Derivation of the system architecture represents a major milestone in the project and it is crucial to get the architecture right. In particular, the interaction of the system components crucially affects the extent to which the system will exhibit the desired emergent properties. At this point, the system requirements and system architecture are documented, reviewed and 'signed off' as the baseline for subsequent development, project planning and cost estimation.

Except in small-scale systems, it is generally infeasible for software developers to begin detailed design of system components from the system requirements document. The requirements allocated to components that are complex systems in themselves will need to undergo further cycles of analysis in order to add more detail, and to interpret the domain-oriented system requirements for developers who may lack sufficient knowledge of the application domain to interpret them correctly. Hence, a number of detailed technical requirements are typically derived from each high-level system requirement. It is crucial to record and maintain this derivation to enable requirements to be traced. Tracing is crucial to requirements management because it allows, for example, the impact of any subsequent changes to the requirements to be assessed.

Refinement of the requirements and system architecture is where requirements engineering merges with software design. There is no clear-cut boundary but it is rare for requirements analysis to continue beyond 2 or 3 levels of architectural decomposition before responsibility is handed over to the design teams for the individual components.

2.4 Requirements engineering in practice

The overview of requirements analysis given in section 2.3 described the process of eliciting and analysing requirements and deriving the system architecture as if it was a linear sequence of activities. This is an idealised view of the process. This section examines some reasons why a linear process is seldom practicable in the context of real software projects.

There is a general pressure in the software industry for ever-shorter development cycles, and this is particularly pronounced in highly competitive market-driven sectors. Moreover, most projects are constrained in some way by their environment and many are upgrades to or revisions of existing systems where the system architecture is a given. In practice, therefore, it is almost always impractical to implement requirements engineering as a linear, deterministic process where system requirements are elicited from the stakeholders, baselined, allocated and handed over to the software development team. It is certainly a myth that the requirements for large systems are ever perfectly understood or perfectly specified [Som97].

Instead, requirements typically iterate toward a level of quality and detail that is sufficient to permit design and procurement decisions to me made. In some projects, this may result in the requirements being baselined before all their properties are fully understood. This risks expensive rework if problems emerge late in the development process. However, requirements engineers are necessarily constrained by project management plans and must therefore take steps to ensure that the requirements' quality is as high as possible given the available resources. They should, for example, make explicit any assumptions that underpin the requirements, and any known problems.

Even where requirements engineering is well resourced, the level of analysis will seldom be uniformly applied. For example, early in the analysis process experienced engineers are often able to identify where existing or off-the-shelf solutions can be adapted to the implementation of system components. The requirements allocated to these need not be elaborated further, while others, for which a solution is less obvious, may need to be subjected to further analysis. Critical requirements, such as those concerned with public safety, must always be analyzed rigorously.

In almost all cases requirements understanding continues to evolve as design and development proceeds. This often leads to the revision of requirements late in the life cycle. Perhaps the most crucial point of understanding about requirements engineering is that a significant proportion of the requirements *will* change. This is sometimes due to errors in the analysis, but it is frequently an inevitable consequence of change in the 'environment': the customer's operating or business environment; or in the market into which the system must sell, for example. Whatever the cause, it is important to recognise the inevitability of change and adopt measures to mitigate the effects of change. Change has to be managed by ensuring that proposed changes go through a defined review and approval process, and by applying careful requirements tracing, impact analysis and version management. Hence, the requirements engineering process is not merely a front-end task to software development, but spans the whole development life cycle. In a typical project the activities of the requirements engineer evolve over time from elicitation to change management.

2.5 Products and deliverables

Good requirements engineering requires that the products of the process - the deliverables - are defined. The most fundamental of these in requirements engineering is the requirements document. This often comprises two separate documents (an architecture description may also be developed at this stage - see the knowledge area description for software design):

A document that specifies the system requirements. This is sometimes known as the requirements definition document, user requirements document or, as defined by IEEE std 1362-1998, the concept of operations (ConOps) document. This document serves to define the high-level system requirements from the stakeholders' perspective(s). It also serves as a vehicle for validating the system requirements. Its readership includes representatives of the system stakeholders. It must therefore be couched in terms of the customer's domain. In addition to a list of the system requirements, the requirements definition needs to include background information such as statements of the overall objectives for the system, a description of its target environment and a statement of the constraints and non-functional requirements on the system. It may include conceptual models designed to illustrate the system context, usage scenarios, the principal domain entities, and data, information and work flows [Tha97].

A document that specifies the software requirements. This is sometimes known as the software requirements specification (SRS). The purpose and readership of the SRS is somewhat different than the requirements definition document. In crude terms, the SRS documents the detailed requirements derived from the system requirements, and which have been allocated to software. The non-functional requirements in the requirements definition should have been elaborated and quantified. The principal readership of the SRS can be assumed to have some knowledge of software engineering concepts. This can be reflected in the language and notations used to describe the requirements, and in the detail of models used to illustrate the system. For custom software, the SRS may form the basis of a contract between the developer and customer [Kot00, Tha97].

Requirements documents must be structured so as to minimize the effort needed to read and locate information within them. Failure to achieve this reduces the likelihood that the system will conform to the requirements. It also

hinders the ability to make controlled changes to the document as the system and its requirements evolve over time. Standards such as IEEE std 1362-1998 and IEEE std 830-1998 provide templates for requirements documents. Such standards are intended to be generic and need to be tailored to the context in which they are used.

Care must also be taken to describe requirements as precisely as possible. Requirements are usually written in natural language but in the SRS this may be supplemented by formal or semi-formal descriptions. Selection of appropriate notations permits particular requirements and aspects of the system architecture to be described more precisely and concisely than natural language. The general rule is that notations should be used that allow the requirements to be described as precisely as possible. This is particularly crucial for safety-critical and certain other types of dependable systems. However, the choice of notation is often constrained by the training, skills and preferences of the document's authors and readers.

Natural language has many serious shortcomings as a medium for description. Among the most serious are that it is ambiguous and hard to describe complex concepts precisely. Formal notations such as Z or CSP avoid the ambiguity problem because their syntax and semantics are formally defined. However, such notations are not expressive enough to adequately describe every system aspect. Natural language, by contrast, is extraordinarily rich and able to describe, however imperfectly, almost any concept or system property. A natural language is also likely to be the document author and readerships' only *lingua franca*. Because natural language is unavoidable, requirements engineers must be trained to use language simply, concisely and to avoid common causes of mistaken interpretation. These include:

- long sentences with complex sub-clauses;
- the use of terms with more than one plausible interpretation (ambiguity);
- presenting several requirements as a single requirement;
- inconsistency in the use of terms such as the use of synonyms.

To counteract these problems, requirements descriptions often adopt a stylized form and use a restricted subset of a natural language. It is good practice, for example, to standardize on a small set of modal verbs to indicate relative priorities. For example, 'shall' is commonly used to indicate that a requirement is mandatory, and 'should' to indicate a requirement that is merely desirable. Hence, the requirement 'The emergency breaks shall be applied to bring the train to a stop if the nose of the train passes a signal at DANGER' is mandatory.

The requirements documents(s) must be subject to validation and verification procedures. The requirements must be validated to ensure that the requirements engineer has understood the requirements. It is also important to verify that a requirements document conforms to company standards, and is understandable, consistent and complete. Formal notations offer the important advantage that they permit the last two properties to be proven (in a restricted sense, at least). The document(s) should be subjected to review by different stakeholders including representatives of the customer and developer. Crucially, requirements documents must be placed under the same configuration management regime as the other deliverables of the development process [Byr94, Ros98].

The requirements document(s) are only the most visible manifestation of the requirements. They exclude information that is not required by the document readership. However this other information is needed in order to manage them. In particular, it is essential that requirements are traced.

One method for tracing requirements is through the construction of a directed acyclic graph (DAG) that records the derivation of requirements and provides audit trails of requirements. As a minimum, requirements need to be traceable backwards to their source (e.g. from a software requirement back to the system requirement(s) from which it was elaborated), and forwards to the design or implementation artifacts that implement them (e.g. from a software requirement to the design document for a component that implements it). Tracing allows the requirements to be managed. In particular, it allows an impact analysis to be performed for a proposed change to one of the requirements.

Modern requirements management tools help maintain tracing information. They typically comprise a database of requirements and a graphical user interface:

- to store the requirement descriptions and attributes;
- to allow the trace DAGs to be generated automatically;
- to allow the propagation of requirements changes to be depicted graphically;
- to generate reports on the status of requirements (such as whether they have been analysed, approved, implemented, etc.);
- to generate requirements documents that conform to selected standards;
- and to apply configuration management to the requirements.

It should be noted that not every organisation has a culture of documenting and managing requirements. It is common for dynamic start-up companies which are driven by a strong 'product vision' and limited resources to view requirements documentation as an unnecessary overhead. Inevitably, however, as these companies expand, as their customer base grows and as their product starts to evolve, they discover that they need to recover the requirements that motivated product features in order to assess the impact

of proposed changes. Hence, requirements documentation and management are fundamental to the any requirements engineering process.

3 BREAKDOWN OF TOPICS FOR SOFTWARE REQUIREMENTS

The knowledge area breakdown we have chosen is broadly compatible with the sections of ISO/IEC 12207-1995 that refer to requirements engineering activities. This standard views the software process at 3 different levels as primary, supporting and organizational life cycle processes. In order to keep the breakdown simple, we conflate this structure into a single life cycle process for requirements engineering. The separate topics that we identify include primary life cycle process activities such as requirements elicitation and requirements analysis, along with requirements engineering-specific descriptions of management and, to a lesser degree, organizational processes. Hence, we identify requirements validation and requirements management as separate topics.

We are aware that a risk of this breakdown is that a waterfall-like process may be inferred. To guard against this, the first topic, the requirements engineering process, is designed to provide a high-level overview of requirements engineering by setting out the resources and constraints that requirements engineering operates under and which act to configure the requirements engineering process.

There are, of course, many other ways to structure the breakdown. For example, instead of a process-based structure, we could have used a product-based structure (system requirements, software requirements, prototypes, use-cases, etc.). We have chosen the process-based breakdown to reflect the fact that requirements engineering, if it is to be successful, must be considered as a process with complex, tightly coupled activities (both sequential and concurrent) rather than as a discrete, one-off activity at the outset of a software development project. The breakdown is compatible with that used by many of the works in the recommended reading list (Appendices C and D). See section 4. for an itemised rationale for the breakdown.

The breakdown comprises 6 topics as shown in Table 1 [Kot00, Lou95, Tha97].

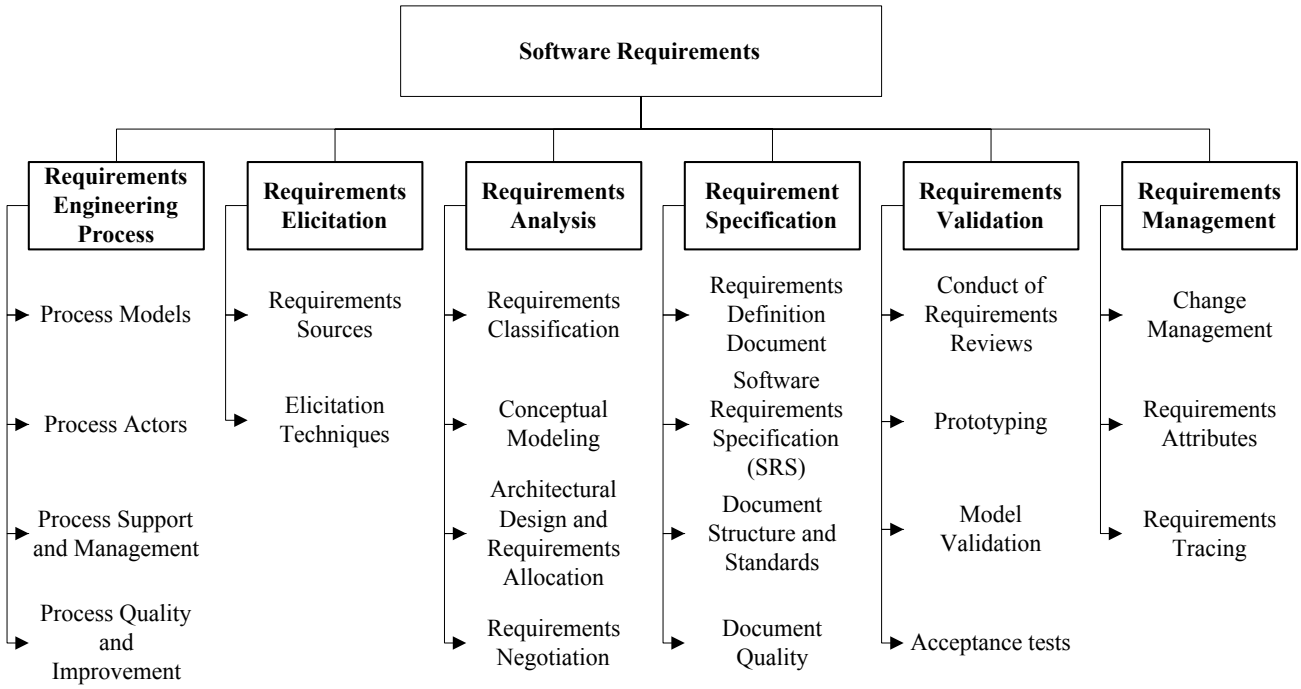

Table 1 Knowledge area breakdown

Figure 1 shows conceptually, how these activities comprise an iterative requirements engineering process. The different activities in requirements engineering are repeated until an acceptable requirements specification document is produced or until external factors such as schedule pressure or lack of resources cause the requirements engineering process to terminate. It is important to note that terminating the requirements engineering process prematurely can have a detrimental effect on the system design. After a final requirements document has been produced, any further changes become part of the requirements management process.

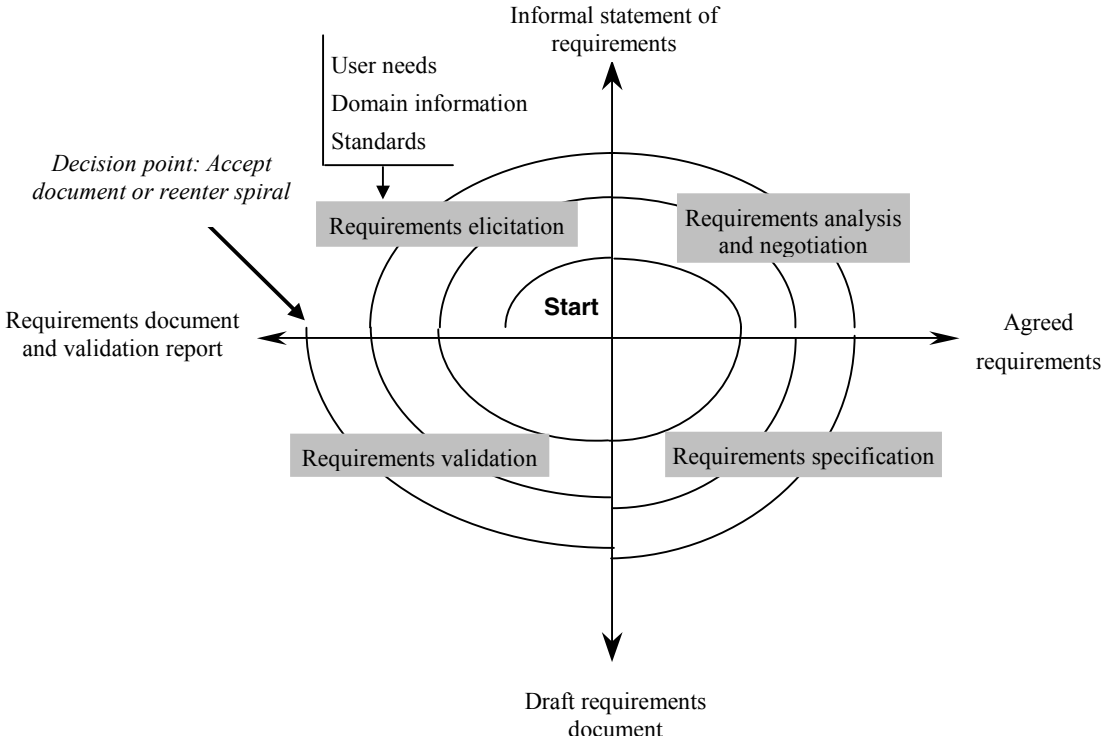

Figure 1 A spiral model of the requirements engineering process

3.1 The requirements engineering process

This section introduces the requirements engineering process, orienting the remaining 5 topics and showing how requirements engineering dovetails with the overall software engineering process.

3.1.1 Process models.

The objective of this subtopic is to provide an understanding that the requirements engineering process:

- is not a discrete front-end activity of the software life cycle, but rather, a process that is initiated at the beginning of a project and continues to be refined throughout the life cycle of the software process;
- must identify requirements as configuration items, and manage them under the same configuration regime as other products of the development process;
- will need to be tailored to the organisation and project context.

In particular, the subtopic is concerned with how the activities of elicitation, analysis, specification, validation and management are configured for different types of project and constraints. The subtopic is also with activities that provide input to the requirements engineering process such as marketing and feasibility studies.

3.1.2 Process actors.

This subtopic introduces the roles of the people who participate in the requirements engineering process. Requirements engineering is fundamentally interdisciplinary and the requirements engineer needs to mediate between the domains of the user and software engineering. There are often many people involved besides the requirements engineer, each of whom have a stake in the system. The stakeholders will vary across different projects but always includes users/operators and customer (who need not be the same) [Gog93]. These need not be homogeneous groups because there may be many users and many customers, each with different concerns. There may also be other stakeholders who are external to the user's/customer's organisation, such as regulatory authorities, whose requirements need to be carefully analysed. The system/software developers are also stakeholders because they have a legitimate interest in

profiting from the system. Again, these may be a heterogeneous group in which (for example) the system architect has different concerns from the system tester.

It will not be possible to perfectly satisfy the requirements of every stakeholder and the requirements engineer's job is to negotiate a compromise that is both acceptable to the principal stakeholders and within budgetary, technical, regulatory and other constraints. A prerequisite for this is that all the stakeholders are indentified, the nature of their 'stake' is analysed and their requirements are elicited.

3.1.3 Process support and management.

This subtopic introduces the project management resources required and consumed by the requirements engineering process. This topic merely sets the context for topic 3 (Initiation and scope definition) of the software management KA. Its principal purpose is to make the link from process activities identified in 3.1.1 to issues of cost, human resources, training and tools.

Table 2 shows the links to common themes in other KAs.

3.1.4 Process quality and improvement.

This subtopic is concerned with requirements engineering process quality assessment. Its purpose is to emphasize the key role requirements engineering plays in terms of the cost, timeliness and customer satisfaction of software products [Som97]. It will help to orient the requirements engineering process with quality standards and process improvement models for software and systems. Process quality and improvement is closely related to the software quality KA and the software process KA. Of particular interest are issues of software quality attributes and measurement, and software process definition. This subtopic covers:

- requirements engineering coverage by process improvement standards and models;
- requirements engineering measures and benchmarking;
- improvement planning and implementation

Links to common themes	
Quality	The process quality and improvement subtopic is concerned with quality. It contains links to SPI standards such as the software and systems engineering capability maturity models, the forthcoming ISO/IEC 15504 and ISO 9001-3 guideline. Requirements engineering process is at best peripheral to these and the only work to address requirements engineering processes specifically, is the requirements engineering good practice guide [Som97].
Standards	SPI models/standards as described in the quality theme above. In addition, the life cycle software engineering standard ISO/IEC 12207-1995 describes requirements engineering activities in the context of the primary, supporting and organizational life cycle processes for software.
Measurement	At the process level, requirements measures tend to be relatively coarse-grained and concerned with (e.g.) counting numbers of requirements and numbers and effects of requirements changes. If these indicate room for improvement (as they inevitably will) it is possible to measure the extent and rigour with which requirements 'good practice' is used in a process. These measures can serve to highlight process weaknesses that should be the target improvement efforts.
Tools	General project management tools. Refer to the software management KA.

Table 2 Process quality links to other KAs

3.2 Requirements elicitation

This topic covers what is sometimes termed 'requirements capture', 'requirements discovery' or 'requirements acquisition'. It is concerned with where requirements come from and how they can be collected by the requirements engineer. Requirements elicitation is the first stage in building an understanding of the problem the software is required to solve. It is fundamentally a human activity and is where the stakeholders are identified and relationships established between the development team (usually in the form of the requirements engineer) and the customer.

3.2.1 Requirements sources

In a typical system, there will be many sources of requirements and it is essential that all potential sources are identified and evaluated for their impact on the system. This subtopic is designed to promote awareness of different requirements sources and frameworks for managing them. The main points covered are:

- Goals. The term 'Goal' (sometimes called 'business concern' or 'critical success factor') refers to the overall, high-level objectives of the system. Goals provide the motivation for a system but are often

vaguely formulated. Requirements engineers need to pay particular attention to assessing the value (relative to priority) and cost of goals. A feasibility study is a relatively low-cost way of doing this [Lou95].

- Domain knowledge. The requirements engineer needs to acquire or to have available knowledge about the application domain. This enables them to infer tacit knowledge that the stakeholders do not articulate, assess the trade-offs that will be necessary between conflicting requirements and sometimes to act as a 'user' champion.
- System stakeholders (see 3.1.2). Many systems have proven unsatisfactory because they have stressed the requirements for one group of stakeholders at the expense of others. Hence, systems are delivered that are hard to use or which subvert the cultural or political structures of the customer organisation. The requirements engineer needs to identify represent and manage the 'viewpoints' of many different types of stakeholder [Kot00].
- The operational environment. Requirements will be derived from the environment in which the software will execute. These may be, for example, timing constraints in a real-time system or interoperability constraints in an office environment. These must be actively sought because they can greatly affect system feasibility, cost, and restrict design choices [Tha97].
- The organizational environment. Many systems are required to support a business process and this may be conditioned by the structure, culture and internal politics of the organisation. The requirements engineer needs to be sensitive to these since, in general, new software systems should not force unplanned change to the business process.

3.2.2 Elicitation techniques

When the requirements sources have been identified the requirements engineer can start eliciting requirements from them. This subtopic concentrates on techniques for getting human stakeholders to articulate their requirements. This is a very difficult area and the requirements engineer needs to be sensitized to the fact that (for example) users may have difficulty describing their tasks, may leave important information unstated, or may be unwilling or unable to cooperate. It is particularly important to understand that elicitation is not a passive activity and that even if cooperative and articulate stakeholders are available, the requirements engineer has to work hard to elicit the right information. A number of techniques will be covered, but the principal ones are [Gog93]:

- Interviews. Interviews are a 'traditional' means of eliciting requirements. It is important to understand the advantages and limitations of interviews and how they should be conducted.
- Scenarios. Scenarios are valuable for providing context to the elicitation of users' requirements. They allow the requirements engineer to provide a framework for questions about users' tasks by permitting 'what if?' and 'how is this done?' questions to be asked. There is a link to 3.3.2. (conceptual modeling) because recent modeling notations have attempted to integrate scenario notations with object-oriented analysis techniques.
- Prototypes. Prototypes are a valuable tool for clarifying unclear requirements. They can act in a similar way to scenarios by providing a context within which users better understand what information they need to provide. There is a wide range of prototyping techniques, which range from paper mock-ups of screen designs to beta-test versions of software products. There is a strong overlap with the use of prototypes for requirements validation (3.5.2).
- Facilitated meetings. The purpose of these is to try to achieve a summative effect whereby a group of people can bring more insight to their requirements than by working individually. They can brainstorm and refine ideas that may be difficult to surface using (e.g.) interviews. Another advantage is that conflicting requirements are surfaced early on in a way that lets the stakeholders recognise where there is conflict. At its best, this technique may result in a richer and more consistent set of requirements than might otherwise be achievable. However, meetings need to be handled carefully (hence the need for a facilitator) to prevent a situation where the critical abilities of the team are eroded by group loyalty, or the requirements reflecting the concerns of a few vociferous (and perhaps senior) people to the detriment of others.
- Observation. The importance of systems' context within the organizational environment has led to the adaptation of observational techniques for requirements elicitation. The requirements engineer learns about users' tasks by immersing themselves in the environment and observing how users interact with their systems and each other. These techniques are relatively new and expensive but are instructive because they illustrate that many user tasks and business processes are too subtle and complex for their actors to describe easily.

Table 3 shows the elicitation techniques links to common themes in other KAs.

Links to common themes	
Quality	The quality of requirements elicitation has a direct effect on product quality. The critical issues are to recognise the relevant sources, to strive to avoid missing important requirements and to accurately report the requirements.
Measurement	Very little work on measurement of requirements elicitation has been carried out.

Table 3 Elicitation techniques links to other KAs

3.3 Requirements analysis

This subtopic is concerned with the process of analysing requirements to:

- detect and resolve conflicts between requirements;
- discover the bounds of the system and how it must interact with its environment;
- elaborate system requirements to software requirements.

The traditional view of requirements analysis was to reduce it to conceptual modeling using one of a number of analysis methods such as SADT or OOA. While conceptual modeling is important, we include the classification of requirements to help inform trade-offs between requirements (requirements classification), and the process of establishing these trade-offs (requirements negotiation) [Dav93].

3.3.1 Requirements classification

There is a strong overlap between requirements classification and requirements attributes (3.6.2). Requirements can be classified on a number of dimensions. Examples include:

- Whether the requirement is functional or non-functional (see 2.1).
- Whether the requirement is derived from one or more high-level requirements, an emergent property (see 2.1), or at a high level and imposed directly on the system by a stakeholder or some other source.
- Whether the requirement is on the product or the process. Requirements on the process constrain, for example, the choice of contractor, the development practices to be adopted, and the standards to be adhered to.
- The requirement priority. In general, the higher the priority, the more essential the requirement is for meeting the overall goals of the system. Often classified on a fixed point scale such as *mandatory*, *highly desirable*, *desirable*, *optional*. Priority often has to be balanced against cost of development and implementation.
- The scope of the requirement. Scope refers to the extent to which a requirement affects the system and system components. Some requirements, particularly certain non-functional ones, have a global scope in that their satisfaction cannot be allocated to a discrete component. Hence a requirement with global scope may strongly affect the system architecture and the design of many components, one with a narrow scope may offer a number of design choices with little impact on the satisfaction of other requirements.
- Volatility/stability. Some requirements will change during the life cycle of the software and even during the development process itself. It is useful if some estimate of the likelihood of a requirement changing can be made. For example, in a banking application, requirements for functions to calculate and credit interest to customers' accounts are likely to be more stable than a requirement to support a particular kind of tax-free account. The former reflect a fundamental feature of the banking domain (that accounts can earn interest), while the latter may be rendered obsolete by a change to government legislation. Flagging requirements that may be volatile can help the software engineer establish a design that is more tolerant of change.

Other classifications may be appropriate, depending upon the development organization's normal practice and the application itself.

3.3.2 Conceptual modeling

The development of models of the problem is fundamental to requirements analysis (see 2.3). The purpose is to aid understanding of the problem rather than to initiate design of the solution. Hence, conceptual models comprise models of entities from the problem domain configured to reflect their real-world relationships and dependencies.

There are several kinds of models that can be developed. These include data and control flows, state models, event traces, user interactions, object models and many others. The factors that influence the choice of model include:

- The nature of the problem. Some types of application demand that certain aspects be analysed particularly rigorously. For example, control flow and state models

are likely to be more important for real-time systems than for an information system.

- The expertise of the requirements engineer. It is often more productive to adopt a modeling notation or method that the requirements engineer has experience with. However, it may be appropriate or necessary to adopt a notation that is better supported by tools, imposed as a process requirement (see 3.3.1), or simply 'better'
- The process requirements of the customer. Customers may impose a particular notation or method on the requirements engineer. This can conflict with the previous factor.
- The availability of methods and tools. Notations or methods that are poorly supported by training and tools may not reach widespread acceptance even if they are suited to particular types of problem.

Note that in almost all cases, it is useful to start by building a model of the system context. The system context provides an understanding between the intended system and its external environment. This is crucial to understanding the system's context in its operational environment and to identify its interfaces to the environment.

The issue of modeling is tightly coupled with that of methods. For practical purposes, a method is a notation (or set of notations) supported by a process that guides the application of the notations. Methods and notations come and go in fashion. Object-oriented notations are currently in vogue but the issue of what is the 'best' notation is seldom clear. There is little empirical evidence to support claims for the superiority of one notation over another.

Formal modeling using notations based upon discrete mathematics and which are tractable to logical reasoning have made an impact in some specialized domains. These may be imposed by customers or standards or may offer compelling advantages to the analysis of certain critical functions or components.

This topic does not seek to 'teach' a particular modeling style or notation but rather to provide guidance on the purpose and intent of modeling.

3.3.3 Architectural design and requirements allocation

At some point the architecture of the solution must be derived. Architectural design is the point at which requirements engineering overlaps with software or systems design and illustrates how impossible it is to cleanly decouple both tasks [Som01]. This subtopic is closely related to topic 2, in Chapter 3 (software architecture). In many cases, the requirements engineer acts as system architect because the process of analysing and elaborating the requirements demands that the subsystems and components that will be responsible for satisfying the requirements be identified. This is requirements allocation – the assignment of responsibility for satisfying requirements to subsystems.

Allocation is important to permit detailed analysis of requirements. Hence, for example, once a set of requirements have been allocated to a component, they can be further analysed to discover requirements on how the component needs to interact with other components in order to satisfy the allocated requirements. In large projects, allocation stimulates a new round of analysis for each subsystem. As an example, requirements for a particular braking performance for a car (braking distance, safety in poor driving conditions, smoothness of application, pedal pressure required, etc.) may be allocated to the braking hardware (mechanical and hydraulic assemblies) and an anti-lock braking system (ABS). Only when a requirement for an anti-lock system has been identified, and the requirements are allocated to it can the capabilities of the ABS, the braking hardware and emergent properties (such as the car weight) be used to identify the detailed ABS software requirements.

Architectural design is closely identified with conceptual modeling. The mapping from real-world domain entities to computational components not always obvious, so architectural design is identified as a separate sub-topic. The requirements of notations and methods are broadly the same for conceptual modeling and architectural design.

3.3.4 Requirements negotiation

Another name commonly used for this subtopic is 'conflict resolution'. It is concerned with resolving problems with requirements where conflicts occur; between two stakeholders' requiring mutually incompatible features, or between requirements and resources or between capabilities and constraints, for example [Kot00, Som97]. In most cases, it is unwise for the requirements engineer to make a unilateral decision so it is necessary to consult with the stakeholder(s) to reach a consensus on an appropriate trade-off. It is often important for contractual reasons that such decisions are traceable back to the customer. We have classified this as a requirements analysis topic because problems emerge as the result of analysis. However, a strong case can also be made for counting it as part of requirements validation.

Table 4 shows the requirements negotiation links to common themes in other KAs.

Links to common themes	
Quality	The quality of the analysis directly affects product quality. In principle, the more rigorous the analysis, the more confidence can be attached to the software quality.
Measurement	Part of the purpose of analysis is to quantify required properties. This is particularly important for constraints such as reliability or safety requirements where suitable measures need to be identified to allow the requirements to be quantified and verified.

Table 4 Requirements negotiation links to other KAs

3.4 Software requirements specification

This topic is concerned with the structure, quality and verifiability of the requirements document. This may take the form of two documents, or two parts of the same document with different readership and purposes (see 2.5): the requirements definition document and the software requirements specification. The topic stresses that documenting the requirements is the most fundamental precondition for successful requirements handling.

3.4.1 The system requirements definition document

This document (sometimes known as the user requirements document or concept of operations) records the system requirements. It defines the high-level system requirements from the domain perspective. Its readership includes representatives of the system users/customers (marketing may play these roles for market-driven software) so it must be couched in terms of the domain. It must list the system requirements along with background information about the overall objectives for the system, its target environment and a statement of the constraints, assumptions and non-functional requirements. It may include conceptual models designed to illustrate the system context, usage scenarios, the principal domain entities, and data, information and workflows.

3.4.2 The software requirements specification (SRS)

The benefits of the SRS include:

- It establishes the basis for agreement between the customers and contractors or suppliers (in market-driven projects, these roles may be played by marketing and development divisions) on what the software product is to do and as well as what it is not expected do. For non-technical readership, the SRS is often accompanied by the requirements definition document.
- It forces a rigorous assessment of requirements before design can begin and reduces later redesign.
- It provides a realistic basis for estimating product costs, risks and schedules.
- Organisations can use a SRS to develop their own validation and verification plans more productively.
- Provides an informed basis for transferring a software product to new users or new machines.
- Provides a basis for software enhancement

3.4.3 Document structure and standards

Several recommended guides and standards exist to help define the structure of requirements documentation. These include IEEE P1233/D3 guide, IEEE Std. 1233 guide, IEEE std. 830-1998, ISO/IEC 12119-1994. IEEE std 1362-1998 *concept of operations* (ConOps) is a recent standard for a requirements definition document.

3.4.4 Document quality

This is one area where measures can be usefully employed in requirements engineering. There are tangible attributes that can be measured. Moreover, the quality of the requirements document can dramatically affect the quality of the product.

A number of quality indicators have been developed that can be used to relate the quality of an SRS to other project variables such as cost, acceptance, performance, schedule, reproducibility etc. Quality indicators for individual SRS statements include imperatives, directives, weak phrases, options and continuances. Indicators for the entire SRS document include size, readability, specification depth and text structure [Dav93, Ros98, Tha97].

There is a strong overlap with 3.5.1 (the conduct of requirements reviews). Table 5 shows the document quality links to common themes in other KAs.

Links to common themes	
Quality	The quality of the requirements documents dramatically affects the quality of the product.
Measurement	Quality attributes of requirements documents can be identified and measured. See 3.4.4.

Table 5 Document quality links to other KAs

3.5 Requirements validation

It is normal for there to be one or more formally scheduled points in the requirements engineering process where the requirements are validated. The aim is to pick up any problems before resources are committed to addressing the requirements. Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right system (i.e. the system that the user expects) [Kot00]. There are four important subtopics.

3.5.1 The conduct of requirements reviews.

Perhaps the most common means of validation is by inspection or formal reviews of the requirements document(s). A group of reviewers is constituted with a brief to look for errors, mistaken assumptions, lack of clarity and deviation from standard practice. The composition of the group that conducts the review is important (at least one representative of the customer should be included for a customer-driven project, for example) and it may help to provide guidance on what to look for in the form of checklists.

Reviews may be constituted on completion of the system requirements definition document, the software requirements specification document, the baseline specification for a new release, etc.

3.5.2 Prototyping.

Prototyping is commonly employed for validating the requirements engineer's interpretation of the system requirements, as well as for eliciting new requirements. As with elicitation, there is a range of prototyping techniques and a number of points in the process when prototype validation may be appropriate. The advantage of prototypes is that they can make it easier to interpret the requirements engineer's assumptions and give useful feedback on why they are wrong. For example, the dynamic behaviour of a user interface can be better understood through an animated prototype than through textual description or graphical models. There are also disadvantages, however. These include the danger of users' attention being distracted from the core underlying functionality by cosmetic issues or quality problems with the prototype. For this reason, several people recommend prototypes that avoid software – such as flip-chart-based mockups. Prototypes may be costly to develop. However, if they avoid the wastage of resources caused by trying to satisfy erroneous requirements, their cost can be more easily justified.

3.5.3 Model validation.

The quality of the models developed during analysis should be validated. For example, in object models, it is useful to perform a static analysis to verify that communication paths exist between objects that, in the stakeholders domain, exchange data. If formal specification notations are used, it is possible to use formal reasoning to prove properties of the specification (e.g. completeness).

3.5.4 Acceptance tests.

An essential property of a system requirement is that it should be possible to validate that the finished product satisfies the requirement. Requirements that can't be validated are really just 'wishes'. An important task is therefore planning how to verify each requirement. In most cases, this is done by designing acceptance tests.

Identifying and designing acceptance test may be difficult for non-functional requirements (see 2.1). To be validated, they must first be analysed to the point where they can be expressed quantitatively.

Table 6 shows the acceptance tests links to common themes in other KAs.

Links to common themes	
Quality	Validation is all about quality - the quality of the requirements.
Measurement	Measurement is important for acceptance tests and definitions of how requirements are to be verified.

Table 6 Acceptance tests links to other KAs

3.6 Requirements management

Requirements management is an activity that spans the whole software life cycle. It is fundamentally about change management and the maintenance of the requirements in a state that accurately mirrors the software to be, or that has been, built [Kot00, Lou95].

There are 3 subtopics concerned with requirements management.

3.6.1 Change management

Change management is central to the management of requirements. This subtopic describes the role of change management, the procedures that need to be in place and the analysis that should be applied to proposed changes. It has strong links to the configuration management knowledge area.

3.6.2 Requirements attributes

Requirements should consist not only of a specification of what is required, but also of ancillary information that helps manage and interpret the requirements. This should include the various classification dimensions of the requirement (see 3.3.1) and the verification method or acceptance test plan. It may also include additional information such as a summary rationale for each requirement, the source of each requirement and a change history. The most fundamental requirements attribute, however, is an identifier that allows the requirements to be uniquely and unambiguously identified. A naming scheme for generating these IDs is an essential feature of a quality system for a requirements engineering process.

3.6.3 Requirements tracing

Requirements tracing is concerned with recovering the source of requirements and predicting the effects of requirements. Tracing is fundamental to performing impact analysis when requirements change. A requirement should be traceable backwards to the requirements and stakeholders that motivated it (from a software requirement back to the system requirement(s) that it helps satisfy, for example). Conversely, a requirement should be traceable forwards into requirements and design entities that satisfy it (for example, from a system requirement into the software requirements that have been elaborated from it and on into the code modules that implement it).

The requirements trace for a typical project will form a complex directed acyclic graph (DAG) of requirements. In the past, development organizations either had to write bespoke tools or manage it manually. This made tracing a short-term overhead on a project and vulnerable to expediency when resources were short. In most cases, this resulted in it either not being done at all or being performed poorly. The availability of modern requirements management tools has improved this situation and the importance of tracing (and requirements management in general) is starting to make an impact in software quality.

Table 7 shows the requirements tracing links to common themes in other KAs.

Links to common themes	
Quality	Requirements management is a level 2 key practice area in the software CMM and this has boosted recognition of its importance for quality.
Measurement	Mature organizations may measure the number of requirements changes and use quantitative measures of impact assessment.

Table 7 Requirements tracing links to other KAs

4 BREAKDOWN RATIONALE

The criterion mentioned below are the criterion described in Appendix A of the Guide: Knowledge Area Description Specifications for the Trial Version of the Guide to the SWEBOK.

Criterion (a): Number of topic breakdowns

One breakdown provided

Criterion (b): Reasonableness

The breakdown is reasonable in that it covers the areas discussed in most requirements engineering texts and standards.

Criterion (c): Generally accepted

The topic breakdowns (shown in Table 1) are generally accepted in that they cover areas typically in texts and standards.

At level A.1 the breakdown is identical to that given in most requirements engineering texts, apart from process improvement. Requirements engineering process improvement is an important emerging area in requirements engineering. We believe this topic adds great value to any the discussion of the requirements engineering as its directly concerned with process quality assessment.

At level A.2 the breakdown is identical to that given in most requirements engineering texts. At level A.3 the

breakdown is similar to that discussed in most texts. We have incorporated a reasonably detailed section on requirement characterization to take into account the most commonly discussed ways of characterizing requirements. A.4 the breakdown is similar to that discussed in most texts, apart from document quality assessment. We believe this an important aspect of the requirements specification document and deserves to be treated as a separate sub-section. In A.5 and A.6 the breakdown is similar to that discussed in most texts.

Criterion (d): No specific domains have been assumed

No specific domains have been assumed

Criterion (e): Compatible with various schools of thought

Requirements engineering concept at the process level are general mature and stable.

Criterion (f): Compatible with industry, literature and standards

The breakdown used here has been derived from literature and relevant standards to reflect a consensus of opinion.

Criterion (g): As inclusive as possible

The inclusion of the requirements engineering process A.1 sets the context for all requirements engineering topics. This level is intended to capture the mature and stable concepts in requirements engineering. The subsequent levels all relate to level 1 but are general enough to allow more specific discussion or further breakdown.

Criterion (h): Themes of quality, tools, measurement and standards

The relationship of requirements engineering product quality assurance, tools and standards is provided in the breakdown.

Criterion (i): 2 to 3 levels, 5 to 9 topics at the first level

The proposed breakdown satisfies this criterion.

Criterion (j): Topic names meaningful outside the guide

The topic names satisfy this criterion

Criterion (k): Version 0.1 of the description

Criterion (l): Text on the rationale underlying the proposed breakdowns

This document provides the rationale

5 MATRIX OF TOPICS VS. REFERENCE MATERIAL FOR SOFTWARE REQUIREMENTS

In Table B.1 shows the topic/reference matrix. The table is organized according to requirements engineering topics in section 3. A 'X' indicates that the topic is covered to a reasonable degree in the reference. A 'X' in appearing in main topic but not the sub-topic indicates that the main topic is reasonably covered (in general) but the sub-topic is not covered to any appreciable depth. This situation is quite common in most software engineering texts, where the subject of requirements engineering is viewed in the large context of software engineering.

TOPIC / REFERENCE	[Bry94]	[Dav93]	[Gog93]	[Kot98]	[Lou95]	[Pfl98]	[Ros98]	[Som96]	[Som97]	[Tha97]
Requirements engineering process		X		X				X	X	
Process models				X				X	X	
Process actors		X		X					X	
Process support									X	
Process improvement				X					X	
Requirements elicitation		X	X	X	X	X				
Requirements sources		X	X	X	X	X				
Elicitation techniques		X	X	X	X	X				
Requirements analysis		X		X				X		
Requirements classification		X		X				X		
Conceptual modeling		X		X				X		
Architectural design and requirements allocation		X						X		
Requirements negotiation				X						

TOPIC / REFERENCE	[Bry94]	[Dav93]	[Gog93]	[Kot98]	[Lou95]	[Pfl98]	[Ros98]	[Som96]	[Som97]	[Tha97]
Requirement specification	X	X		X		X	X	X		X
The requirements definition document	X	X		X			X	X		X
The software requirements specification (SRS)	X	X		X			X	X		X
Document structure	X	X		X			X			X
Document quality	X	X		X			X			
Requirements validation		X						X		X
The conduct of requirements reviews				X						X
Prototyping		X		X						X
Model validation		X		X						X
Acceptance tests		X								
Requirements management		X		X				X		
Change management				X						
Requirement attributes				X						
Requirements tracing				X						

Table B.1 Topics and their references

Key	*Reference*
[Byr94]	[Byrne 1994]
[Dav93]	[Davis 1993]
[Gog93]	[Goguen and Linde 1993]
[Kot00]	[Kotonya and Sommerville 2000]
[Lou95]	[Loucopoulos and Karakostas 1995]
[Pfl98]	[Pfleeger 1998]
[Ros98]	[Rosenberg 1998]
[Som01]	[Sommerville 2001]
[Som97]	[Sommervelle and Sawyer 1997]
[Tha97]	[Thayer and Dorfman 1997]

6 RECOMMENDED REFERENCES FOR SOFTWARE REQUIREMENTS

[Byrne 1994]. Byrne, E., "IEEE Standard 830: Recommended Practice for Software Requirements Specification," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 58.

Describes the IEEE Standard 830-1993 for requirements specification.

[Davis 1993]. Davis, A.M., Software Requirements: Objects, Functions and States. Prentice-Hall, 1993.

Provides a way of categorizing software requirements techniques--objects, functions, and states. The author takes an analytical approach by helping the reader analyze which technique is best, rather than imposing one specific technique. Discussion of a wide variety of techniques and their uses is augmented with application illustration using three case studies.

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

This paper is an attempt to address the failings of traditional requirements practice, particularly in eliciting requirements. The paper explores a different paradigm for understanding requirements engineering: the process is seen essentially as a social process, in which requirements emerge and evolve from the discourse between users and developers. The paper describes a number of techniques for requirements elicitation and examines their strengths and weaknesses.

[Kotonya and Sommerville 2000]. Kotonya, G., and I. Sommerville, Requirements Engineering: Processes and Techniques. John Wiley and Sons, 2000.

Introduces requirements engineering to undergraduate and graduate students in computer science, software engineering, and systems engineering. Part I is process-oriented and describes different activities in the

requirements engineering process. Part II focuses on requirements engineering techniques, covering the use of structured methods, viewpoint-oriented approaches, and specification of non- functional requirements and of interactive systems. A final chapter presents a case study illustrating a viewpoint-oriented approach. Includes chapter key points and exercises.

[Loucopoulos and Karakostas 1995]. Loucopoulos, P., and V. Karakostas, System Requirements Engineering. McGraw-Hill, 1995.

It provides software professionals with a practical framework for a formal requirements engineering (RE) process. Readers will exchange their RE problem-solving skills in chapters that help them accurately assess the nature of the problems and implement effective solutions.

[Pfleeger 1998]. Pfleeger, S.L., Software Engineering-Theory and Practice. Prentice-Hall, Chap. 4, 1998.

Applies concepts to two common examples: one that represents a typical information system, and one that represents a real-time system. This work features an associated web page containing examples from literature and links to web pages for relevant tool and method vendors.

[Rosenberg 1998]. Rosenberg, L., T.F. Hammer and L.L. Huffman, "Requirements, testing and metrics", 16th Annual Pacific Northwest Software Quality Conference, Oregon, October 1998.

This paper addresses the issue of evaluating the quality of a requirements document. The authors describe a tool developed to parse requirements documents. The Automated Requirements Measurement (ARM) software scans a file containing the text of the requirements specification. The tool searches each line of text for specific words and phrases based on seven quality indicators. ARM has been applied to 56 NASA requirements documents.

[Sommerville 2001]. Sommerville, I. Software Engineering (6th edition), Addison-Wesley, pp. 63-97,

97-147, 2001.

A textbook that presents a general introduction to software engineering, for students in undergraduate and graduate courses and software engineers in commerce and industry. It doesn't describe commercial design methods or CASE systems, but paints a broad picture of software engineering methods and tools.

[Sommerville 1997]. Sommerville, I., and P. Sawyer, Requirements engineering: A Good Practice Guide. John Wiley and Sons, Chap. 1-2, 1997.

Presents guidelines which reflect good practice in requirements engineering, based on the authors' experience in research and in software and systems development. The guidelines range from common sense tips to complex new methods, and can be used in any order, which suits the reader's problems, goals and budget. Guidelines are consistent with ISO 9000 and CMM, are ranked with cost and benefit analysis, include implementation advice, and can be combined and applied to suit an organization's needs.

[Thayer and Dorfman 1997]. Thayer, R.H., and M. Dorfman, Software Requirements Engineering (2nd Ed). IEEE Computer Society Press, pp. 176-205, 389-404, 1997.

A new edition of the comprehensive collection of original and reprinted articles describing the current best practices in requirement engineering focused primarily on software systems but also including hardware and people systems. The 35 papers introduce current issues and basic terminology, and cover the phases of software requirements engineering including elicitation, analysis, specification, verification, and management. Specific discussions feature descriptions of the process developers and users use to review and articulate needs and constraints on development, examine software requirements and documentation, and supply details on management planning and control. Lacks an index.

APPENDIX A – LIST OF FURTHER READINGS

[Ardis 1997]. Ardis, M., "Formal Methods for Telecommunication System Requirements: A survey of Standardized Languages," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Berzins, et al. 1997]. Berzins, V., et al., "A Requirements Evolution Model for Computer Aided Prototyping," Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Skokie, Illinois: Knowledge Systems Institute, June 1997, pp. 38-47.

[Beyer and Holtzblatt 1995]. Beyer, H., and Holtzblatt, K., "Apprenticing with the Customer," Communications of the ACM, 38, 5 (May 1995), pp.45-52.

[Bruno and Agarwal 1995]. Bruno, G., and R. Agarwal, "Validating Software Requirements Using Operational Models," Second Sympoium on Software Quality Techniques and Acquisition Criteria, Florence, Italy, May 1995.

[Bucci, et al. 1994]. Bucci, G., et al., "An Object-Oriented Dual Language for Specifying Reactive Systems," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 6-15.

[Bustard and Lundy 1995]. Bustard, D., and P. Lundy, "Enhancing Soft Systems Analysis with Formal Modeling," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Chechik and Gannon 1994]. Chechik, M., and J. Gannon, "Automated Verification of Requirements Implementation," ACM Software Engineering Notes, Proceedings of the International Symposium on Software Testing and Analysis, Special Issue (October 1994), pp. 1-15.

[Chung and Nixon 1995]. Chung, L., and B. Nixon, "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach," Seventeenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1995.

[Ciancarini, et al. 1997]. Ciancarini, P., et al., "Engineering Formal Requirements: An Analysis and Testing Method for Z Documents," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Crespo 1994]. Crespo, R., "We Need to Identify the Requirements of the Statements of Non-Functional Requirements," International Workshop on Requirements Engineering: Foundations of Software Quality, June 1994.

[Curran, et al. 1994]. Curran, P., et al., "BORIS-R Specification of the Requirements of a Large-Scale Software Intensive System," Conference on Requirements Elicitation for Software-Based Systems, July 1994.

[Darimont and Souquieres 1997]. Darimont, R., and J. Souquieres, "Reusing Operational Requirements: A Process-Oriented Approach," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Davis and Hsia 1994]. Davis, A., and P. Hsia, "Giving Voice to Requirements Engineering: Guest Editors' Introduction," IEEE Software, 11, 2 (March 1994), pp. 12-16.

[DeFoe 1994]. DeFoe, J., "Requirements Engineering Technology in Industrial Education," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 145.

[Demirors 1997]. Demirors, E., "A Blackboard Framework for Supporting Teams in Software Development," Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Skokie, Illinois: Knowledge Systems Institute, June 1997, pp. 232-239.

[Diepstraten 1995]. Diepstraten, M., "Command and Control System Requirements Analysis and System Requirements Specification for a Tactical System," First IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society Press, November 1995.

[Dobson and Strens 1994] Dobson, J., and R. Strens, "Organizational Requirements Definition for Information Technology," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 158-165.

[Duffy, et al. 1995]. Duffy, D., et al., "A Framework for Requirements Analysis Using Automated Reasoning," Seventh International Conference on Advanced Information Systems Engineering (CAiSE '95), Springer-Verlag, 1995.

[Easterbrook and Nuseibeh 1995]. Easterbrook, S., and B. Nuseibeh, "Managing Inconsistencies in an Evolving Specification," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1995.

[Edwards, et al 1995]. Edwards, M., et al., "RECAP: A Requirements Elicitation, Capture, and Analysis Process Prototype Tool for Large Complex Systems," First IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society Press, November 1995.

[El Emam and Madhavji 1995a]. El Emam, K., and N. Madhavji, "Requirements Engineering Practices in Information Systems Development: A Multiple Case Study," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Fairley and Thayer 1997]. Fairley, R., and R. Thayer, "The Concept of Operations: The Bridge From Operational Requirements to Technical Specifications," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Fickas and Feather 1995]. Fickas, S., and M. Feather, "Requirements Monitoring in Dynamic Environments," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Fields, et al. 1995]. Fields, R., et al., "A Task-Centered Approach to Analyzing Human Error Tolerance Requirements," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Ghajar-Dowlatshahi and Varnekar 1994]. Ghajar-Dowlatshahi, J., and A. Varnekar, "Rapid Prototyping in Requirements Specification Phase of Software Systems," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 135-140.

[Gibson 1995]. Gibson, M., "Domain Knowledge Reuse During Requirements Engineering," Seventh International Conference on Advanced Information Systems Engineering (CAiSE '95), Springer-Verlag, 1995.

[Goldin and Berry 1994]. Goldin, L., and D. Berry, "AbstFinder: A Prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology and Evaluation," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 84-93.

[Gotel and Finkelstein 1997]. Gotel, O., and A. Finkelstein, "Extending Requirements Traceability: Lessons Learned from an Industrial Case Study," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Heimdahl 1996]. Heimdahl, M., "Errors Introduced during the TACS II Requirements Specification Effort: A Retrospective Case Study," Eighteenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1996.

[Heitmeyer, et al. 1996]. Heitmeyer, C., et al., "Automated Consistency Checking Requirements Specifications," ACM Transactions on Software Engineering and Methodology, 5, 3 (July 1996), pp. 231-261.

[Holtzblatt and Beyer 1995]. Holtzblatt, K., and H. Beyer, "Requirements Gathering: The Human Factor," Communications of the ACM, 38, 5 (May 1995), pp. 31-32.

[Hudlicka 1996]. Hudlicka, E., "Requirements Elicitation with Indirect Knowledge Elicitation Techniques: Comparison of Three Methods," Second IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1996.

[Hughes, et al. 1994]. Hughes, K., et al., "A Taxonomy for Requirements Analysis Techniques," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 176-179.

[Hughes, et al. 1995]. Hughes, J., et al., "Presenting Ethnography in the Requirements Process," Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, April 1995.

[Hutt 1994]. Hutt, A., Object-Oriented Analysis and Design, New York, New York: Wiley, 1994.

[Jackson 1995]. Jackson, M., Software Requirements and Specifications, Reading, Massachusetts: Addison Wesley, 1995.

[Jackson 1997]. Jackson, M., "The Meaning of Requirements," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Jones and Britton 1996]. Jones, S., and C. Britton, "Early Elicitation and Definition of Requirements for an Interactive Multimedia Information System," Second IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1996.

[Kirner and Davis 1995]. Kirner, T., and A. Davis, "Nonfunctional Requirements for Real-Time Systems," Advances in Computers, 1996.

[Klein 1997]. Klein, M., "Handling Exceptions in Collaborative Requirements Acquisition," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Kosman 1997]. Kosman, R., "A Two-Step Methodology to Reduce Requirements Defects," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Krogstie, et al. 1995]. Krogstie, J., et al., "Towards a Deeper Understanding of Quality in Requirements Engineering," Seventh International Conference on Advanced Information Systems Engineering (CAiSE '95), Springer-Verlag, 1995.

[Lalioti and Theodoulidis 1995]. Lalioti, V., and B. Theodoulidis, "Visual Scenarios for Validation of Requirements Specification," Seventh International Conference on Software Engineering and Knowledge Engineering, Skokie, Illinois: Knowledge Systems Institute, June 1995, pp. 114-116.

[Leite, et al. 1997]. Leite, J., et al., "Enhancing a Requirements Baseline with Scenarios," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Lerch, et al. 1997]. Lerch, F., et al., "Using Simulation-Based Experiments for Software Requirements Engineering," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Leveson, et al. 1994]. Leveson, N., et al., "Requirements Specification for Process-Control Systems," IEEE Transactions on Software Engineering, 20,, 9 (September 1994), pp. 684-707.

[Lutz and Woodhouse 1996]. Lutz, R., and R. Woodhouse, "Contributions of SFMEA to Requirements Analysis," Second IEEE International Conference on Requirements Engineering, Computer Society Press, April 1996.

[Lutz and Woodhouse 1997]. Lutz,R., and R. Woodhouse, "Requirements Analysis Using Forward and Backward Search," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Macaulay 1996]. Macaulay, L., Requirements Engineering, London, UK: Springer, 1996.

[Macfarlane and Reilly 1995]. Macfarlane, I., and I. Reilly, "Requirements Traceability in an Integrated Development Environment," Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, March 1995.

[Maiden and Rugg 1995]. Maiden, N., et al., "Computational Mechanisms for Distributed Requirements Engineering," Seventh International Conference on Software Engineering and Knowledge Engineering, Skokie, Illinois: Knowledge Systems Institute, June 1995, pp. 8-15.

[Mar 1994]. Mar, B., "Requirements for Development of Software Requirements," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 39-44.

[Massonet and van Lamsweerde 1997]. Massonet, P., and A. van Lamsweerde, "Analogical Reuse of Requirements Frameworks," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[McFarland and Reilly 1995]. McFarland, I., and I. Reilly, "Requirements Traceability in an Integrated Development Environment," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Mead 1994]. Mead, N., "The Role of Software Architecture in Requirements Engineering," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 242.

[Mostert and von Solms 1995]. Mostert, D., and S. von Solms, "A Technique to Include Computer Security, Safety, and Resilience Requirements as Part of the Requirements Specification," Journal of Systems and Software, 31, 1 (October 1995), pp. 45-53.

[Mylopoulos, et al. 1995]. Mylopoulos, J., et al., "Multiple Viewpoints Analysis of Software Specification Process," submitted to IEEE Transactions on Software Engineering.

[Nishimura and Honiden 1992]. Nishimura, K., and S. Honiden, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," submitted to IEEE Transactions on Software Engineering, December 1992.

[Nissen, et al. 1997]. Nissen, H., et al., "View-Directed Requirements Engineering: A Framework and Metamodel," Ninth IEEE International Conference on Software Engineering and Knowledge Engineering, Skokie, Illinois: Knowledge Systems Institute, June 1997, pp. 366-373.

[O'Brien 1996]. O'Brien, L., "From Use Case to Database: Implementing a Requirements Tracking System," Software Development, 4, 2 (February 1996), pp. 43-47.

[Opdahl 1994]. Opdahl, A., "Requirements Engineering for Software Performance," International Workshop on Requirements Engineering: Foundations of Software Quality, June 1994.

[Pinheiro and Goguen 1996]. Pinheiro,F., and J. Goguen, "An Object-Oriented Tool for Tracing Requirements," IEEE Software, 13, 2 (March 1996), pp. 52-64.

[Playle and Schroeder 1996]. Playle, G., and C. Schroeder, "Software Requirements Elicitation: Problems, Tools, and Techniques," Crosstalk: The Journal of Defense Software Engineering, 9, 12 (December 1996), pp. 19-24.

[Pohl, et al. 1994]. Pohl, K., et al., "Applying AI Techniques to Requirements Engineering: The NATURE Prototype," IEEE Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence, IEEE Computer Society Press, May 1994.

[Porter, et al. 1995]. Porter, A., et al., "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," IEEE Transactions on Software Engineering, 21, 6 (June 1995), pp. 563-575.

[Potts and Hsi 1997]. Potts, C., and I. Hsi, "Abstraction and Context in Requirements Engineering: Toward a Synthesis," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Potts and Newstetter 1997]. Potts, C., and W. Newstetter., "Naturalistic Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Potts, et al. 1995] Potts, C., et al., "An Evaluation of Inquiry-Based Requirements Analysis for an Internet Server," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Ramesh, et al. 1995]. Ramesh, B., et al., "Implementing Requirements Traceability: A Case Study," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Regnell, et al. 1995]. Regnell, B., et al., "Improving the Use Case Driven Approach to Requirements Engineering," Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, April 1995.

[Reubenstein 1994]. Reubenstein, H., "The Role of Software Architecture in Software Requirements Engineering," IEEE International Conference on Requirements Engineering, Computer Society Press, April 1994, p. 244.

[Robertson and Robertson 1994]. Robertson, J., and S. Robertson, Complete Systems Analysis, Vols. 1 and 2, Englewood Cliffs, New Jersey: Prentice Hall, 1994.

[Robinson and Fickas 1994]. Robinson, W., and S. Fickas, "Supporting Multi-Perspective Requirements Engineering," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 206-215.

[Rolland 1994]. Rolland, C., "Modeling and Evolution of Artifacts," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 216-219.

[Schoening 1994]. Schoening, W., "The Next Big Step in Systems Engineering Tools: Integrating Automated Requirements Tools with Computer Simulated Synthesis and Test," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 409-415.

[Shekaran 1994]. Shekaran, M., "The Role of Software Architecture in Requirements Engineering," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 245.

[Siddiqi, et al. 1997]. Siddiqi, J., et al., "Towards Quality Requirements Via Animated Formal Specifications," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Spanoudakis and Finkelstein 1997]. Spanoudakis, G., and A. Finkelstein, "Reconciling Requirements: A Method for Managing Interference, Inconsistency, and Conflict," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Stevens 1994]. Stevens, R., "Structured Requirements," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 99-104.

[van Lamsweerde, et al. 1995] van Lamsweerde, A., et al., "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[White and Edwards 1995]. White, S., and M. Edwards, "A Requirements Taxonomy for Specifying Complex Systems," First IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society Press, November 1995.

[Wiley 1999]. Wiley, B., Essential System Requirements: A Practical Guide to Event-Driven Methods, Addison-Wesley, 1999.

[Wyder 1996]. Wyder, T., "Capturing Requirements With Use Cases," Software Development, 4, 2 (February 1996), pp. 36-40.

[Yen and Tiao 1997]. Yen, J., and W. Tiao, "A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements," IEEE International Symposium on Requirements Engineering, Computer Society Press, March 1997.

[Yu 1997]. Yu, E., "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, March 1997.

[Zave and Jackson 1996]. Zave, P., and M. Jackson, "Where Do Operations Come From? A Multiparadigm Specification Technique," IEEE Transactions on Software Engineering, 22, 7 (July 1996), pp. 508-528.

APPENDIX B – REFERENCES USED TO WRITE AND JUSTIFY THE DESCRIPTION

[Acosta 1994]. Acosta, R., et al., "A Case Study of Applying Rapid Prototyping Techniques in the Requirements Engineering Environment," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 66-73.

[Alford 1994]. Alford, M., "Attacking Requirements Complexity Using a Separation of Concerns," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 2-5.

[Alford 1994]. Alford, M., "Panel Session Issues in Requirements Engineering Technology Transfer: From Researcher to Entrepreneur," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 144.

[Anderson 1985]. Anderson, T., Software Requirements: Specification and Testing, Oxford, UK: Blackwell Publishing, 1985.

[Anderson and Durney 1993]. Anderson, J., and B. Durney, "Using Scenarios in Deficiency-Driven Requirements Engineering," International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 134-141.

[Andriole 1992]. Andriole, S., "Storyboard Prototyping For Requirements Verification," Large Scale Systems, 12 (1987), pp. 231-247. 14.[Andriole 1992]

[Andriole 1995]. Andriole, S., "Interactive Collaborative Requirements Management," Software Development, (September 1995).

[Andriole 1996]. Andriole, S.J., Managing Systems Requirements: Methods, Tools and Cases. McGraw-Hill, 1996.

[Anton and Potts 1998]. Anton, A., and C. Potts, "The Use of Goals to Surface Requirements for Evolving Systems," Twentieth International Conference on Software Engineering, IEEE Computer Society, 1998.

[Ardis, et al. 1995]. Ardis, M., et al., "A Framework for Evaluating Specification Methods for Reactive Systems," Seventeenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1995.

[Bickerton and Siddiqi 1993]. Bickerton, M., and J. Siddiqi, "The Classification of Requirements Engineering Methods," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 182-186.

[Blanchard and Fabrycky 1998]. Blanchard, B. and Fabrycky, W.J., Systems Engineering Analysis, Prentice Hall, 1998.

[Blum 1983]. Blum, B., "Still More About Prototyping," ACM Software Engineering Notes, 8, 3 (May 1983), pp. 9-11.

[Blum 1993]. Blum, B., "Representing Open Requirements with a Fragment-Based Specification," IEEE Transaction on Systems, Man and Cybernetics, 23, 3 (May-June 1993), pp. 724-736.

[Blyth, et al. 1993a]. Blyth, A., et al., "A Framework for Modelling Evolving Requirements," IEEE International Conference on Computer Software and Applications, IEEE Computer Society Press, 1993.

[Boehm 1994]. Boehm, B., P. Bose, et al., "Software Requirements as Negotiated Win Conditions," Proc. 1st International Conference on Requirements Engineering (ICRE), Colorado Springs, Co, USA, (1994), pp.74-83.

[Boehm, et al. 1995]. Boehm, B., et al., "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach," Seventeenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1995.

[Brown and Cady 1993]. Brown, P., and K. Cady, "Functional Analysis vs. Object-Oriented Analysis: A View From the Trenches," Third International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, July 1993.

[Byrne 1994]. Byrne, E., "IEEE Standard 830: Recommended Practice for Software Requirements Specification," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 58.

[Burns and McDermid 1994]. Burns, A., and J. McDermid, "Real-Time Safety-Critical Systems: Analysis and Synthesis," IEE Software Engineering Journal, 9, 6 (November 1994), pp. 267-281.

[Checkland and Scholes 1990]. Checkland, P., and J. Scholes, Soft Sysems Methodology in Action. John Wiley and Sons, 1990.

[Chung 1991a]. Chung, L., "Representation and Utilization of Nonfunctional Requirements for Information System Design," Third International Conference on Advanced Information Systems Engineering (CAiSE '90), Springer-Verlag, 1991, pp. 5-30.

[Chung 1999]. Chung, L., Nixon, B.A., Yu. E., Mylopoulos, J., Non-functional Requirements in Software Engineering, Kluwer Academic Publishers, 1999.

[Chung, et al. 1995]. Chung, L., et al., "Using Non-Functional Requirements to Systematically Support Change," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Connell and Shafer 1989]. Connell, J., and L. Shafer, Structured Rapid Prototyping, Englewood Cliffs, New Jersey, 1989.

[Coombes and McDermid 1994]. Coombes, A., and J. McDermid, "Using Quantitative Physics in Requirements Specification of Safety Critical Systems" Workshop on

Research Issues in the Intersection Between Software Engineering and Artificial Intelligence, Sorrento, Italy, May 1994.

[Costello and Liu 1995]. Costello, R., and D. Liu, "Metrics for Requirements Engineering," Journal of Systems and Software, 29, 1 (April 1995), pp. 39-63.

[Curtis 1994]. Curtis, A., "How to Do and Use Requirements Traceability Effectively," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 57-64.

[Davis 1993]. Davis, A.M., Software Requirements: Objects, Functions and States. Prentice-Hall, 1993.

[Davis 1995a]. Davis, A., 201 Principles of Software Development, New York, New York: McGraw Hill, 1995.

[Davis 1995b]. Davis, A., "Software Prototyping," in Advances in Computing, 40, M. Zelkowitz, ed., New York, New York: Academic Press, 1995.

[Davis, et al. 1997]. Davis, A., et al., "Elements Underlying Requirements Specification," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[De Lemos, et al. 1992]. De Lemos, R., et al., "Analysis of Timeliness Requirements in Safety-Critical Systems," Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems, Nijmegen, The Netherlands: Springer Verlag, January 1992, pp. 171-192.

[Dobson 1991]. Dobson, J., "A methodology for analysing human computer-related issues in secure systems," International Conference on Computer Security and Integrity in our Changing World, Espoo, Finland, (1991), pp. 151-170.

[Dobson, et al. 1992]. Dobson, J., et al., "The ORDIT Approach to Requirements Identification," IEEE International Conference on Computer Software and Applications, IEEE Computer Society Press, 1992, pp. 356-361.

[Dorfman and Thayer 1997]. Dorfman, M., and R.H. Thayer, Software Engineering. IEEE Computer Society Press, 1997.

[Easterbrook and Nuseibeh 1996]. Easterbrook, S., and B. Nuseibeh, "Using viewpoints for inconsistency management," Software Engineering Journal, 11, 1, 1996, pp.31-43.

[Ebert 1997]. Ebert, C., "Dealing with Non-Functional Requirements in Large Software Systems," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[El Emam 1997]. EL Amam K., J. Drouin, et al., SPICE: The theory and Practice of Software Process Improvement and Capability Determination. IEEE Computer Society Press, 1997.

[El Emam and Madhavji 1995]. El Emam, K., and N. Madhavji, "Measuring the Success of Requirements Engineering," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Fagan 1986]. Fagan, M.E., "Advances in Software Inspection," IEEE Transactions on Software Engineering 12, 7, 1986, pp. 744-51.

[Feather 1991]. Feather, M., "Requirements Engineering: Getting Right from Wrong," Third European Software Engineering Conference, Springer Verlag, 1991.

[Fenton 1991]. Fenton, N. E., Software metrics: A rigorous approach. Chapman and Hall, 1991.

[Fiksel 1991]. Fiksel, J., "The Requirements Manager: A Tool for Coordination of Multiple Engineering Disciplines," CALS and CE '91, Washington, D.C., June 1991.

[Finkelstein 1992]. Finkelstein, A., Kramer, J., B. Nuseibeh and M. Goedicke, "Viewpoints: A framework for integrating multiple perspectives in systems development," International Journal of Software Engineering and Knowledge Engineering, 2, 10, (1992), pp.31-58.

[Garlan 1994]. Garlan, D., "The Role of Software Architecture in Requirements Engineering," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, p. 240.

[Gause and Weinberg 1989]. Gause, D.C., and G. M. Weinberg, Exploring Requirements : Quality Before Design, Dorset House, 1989.

[Gilb and Graham 1993]. Gilb, T., and D. Graham, Software Inspection. Wokingham: Addison-Wesley, 1993.

[Goguen and Linde 1993]. Goguen, J., and C. Linde, "Techniques for Requirements Elicitation," International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 152-164.

[Gomaa 1995]. Gomaa, H., "Reusable Software Requirements and Architectures for Families of Systems," Journal of Systems and Software, 28, 3 (March 1995), pp. 189-202.

[Grady 1993a]. Grady, J., Systems Requirements Analysis, New York, New York: McGraw Hill, 1993.

[Graham 1998]. Graham, I., Requirements Engineering and Rapid Development: An Object-Oriented Approach, Addison Wesley, 1998.

[Hadden 1997]. Hadden, R., "Does Managing Requirements Pay Off?," American Programmer, 10, 4 (April 1997), pp. 10-12.

[Hall 1996]. Hall, A., "Using Formal Methods to Develop an ATC Information System," IEEE Software 13, 2, 1996, pp.66-76.

[Hansen, et al. 1991]. Hansen, K., et al., "Specifying and Verifying Requirements of Real-Time Systems," ACM SIGSOFT Conference on Software for Critical Systems, December 1991, pp. 44-54.

[Harel 1988]. Harel, D., "On Visual Formalisms," Communications of the ACM, 31, 5 (May 1988), pp. 8-20.

[Harel 1992]. Harel, D., "Biting the Silver Bullet: Towards a Brighter Future for System Development," IEEE Computer, 25, 1 (January 1992), pp. 8-20.

[Harel and Kahana 1992]. Harel, D., and C. Kahana, "On Statecharts with Overlapping," ACM Transactions on Software Engineering and Methodology, 1, 4 (October 1992), pp. 399-421.

[Harwell 1993]. Harwell, R., et al, "What is a Requirement," Proc 3rd Ann. Int'l Symp. Nat'l Council Systems Eng., (1993), pp.17-24.

[Heimdahl and Leveson 1995]. Heimdahl, M., and N. Leveson, "Completeness and Consistency Analysis of State-Based Requirements," Seventeenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1995.

[Hofmann 1993]. Hofmann, H., Requirements Engineering: A Survey of Methods and Tools, Technical Report #TR-93.05, Institute for Informatics, Zurich, Switzerland: University of Zurich, 1993.

[Honour 1994]. Honour, E., "Requirements Management Cost/Benefit Selection Criteria," Fourth International Symposium on Systems Engineering, Sunnyvale, California: National Council on Systems Engineering, August 1994, pp. 149-156.

[Hooks and Stone 1992] Hooks, I., and D. Stone, "Requirements Management: A Case Study — NASA's Assured Crew Return Vehicle," Second Annual International Symposium on Requirements Engineering, Seattle, Washington: National Council on Systems Engineering, July 1992.

[Hsia, et al. 1997]. Hsia, P. et al., "Software Requirements and Acceptance Testing," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Humphery 1988]. Humphery, W.S., "Characterizing the Software Process," IEEE Software 5, 2 (1988), pp. 73-79.

[Humphery 1989]. Humphery, W., Managing the Software Process, Reading, Massachusetts: Addison Wesley, 1989.

[Hutchings 1995]. Hutchings, A., and S. Knox, "Creating products customers demand," Communications of the ACM, 38, 5, (May 1995), pp. 72-80.

[IEEE 1998a]. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications.

[IEEE 1998b]. IEEE Std 1362-1998. IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document.

[Ince 1994]. Ince, D., ISO 9001 and Software Quality Assurance. London: McGraw-Hill, 1994.

[Jackson and Zave 1995]. Jackson, M., and P. Zave, "Deriving Specifications from Requirements: An Example," Seventeenth IEEE International Conference on Software Engineering, IEEE Computer Society Press, 1995.

[Jarke and Pohl 1994]. Jarke, M., and K. Pohl, "Requirements Engineering in 2001: Virtually Managing a Changing Reality," IEE Software Engineering Journal, 9, 6 (November 1994), pp. 257-266.

[Jarke, et al. 1993]. Jarke, M., et al., "Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 19-31.

[Jenkins 1994]. Jenkins, M., "Requirements Capture," Conference on Requirements Elicitation for Software-Based Systems, July 1994.

[Jirotka 1991]. Jirotka, M., Ethnomethodology and Requirements Engineering, Centre for Requirements and Foundations Technical Report, Oxford, UK: Oxford University Computing Laboratory, 1991.

[Kotonya 1999]. Kotonya, G., "Practical Experience with Viewpoint-oriented Requirements Specification," Requirements Engineering, 4, 3, 1999, pp.115-133.

[Kotonya and Sommerville 1996]. Kotonya, G., and I. Sommerville, "Requirements Engineering with viewpoints," Software Engineering, 1, 11, 1996, pp.5-18.

[Kotonya and Sommerville 1998]. Kotonya, G., and I. Sommerville, Requirements Engineering: Processes and Techniques. John Wiley and Sons, 1998.

[Lam, et al. 1997a]. Lam, W., et al., "Ten Steps Towards Systematic Requirements Reuse," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Leveson 1986]. Leveson, N.G., "Software safety - why, what, and how," Computing surveys, 18, 2, (1986), pp. 125-163.

[Leveson 1995]. Leveson, N.G., Safeware: System Safety and Computers. Reading, Massachusetts: Addison-Wesley, 1995.

[Loucopulos and Karakostas 1995]. Loucopulos, P., and V. Karakostas, Systems Requirements Engineering. McGraw-Hill, 1995.

[Lutz 1993]. Lutz, R., "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 126-133.

[Lutz 1996]. Lutz, R., "Targeting Safety-Related Errors During Software Requirements Analysis," The Journal of Systems and Software, 34, 3 (September 1996), pp. 223-230.

[Maiden and Sutcliffe 1993]. Maiden, N., and A. Sutcliffe, "Requirements Engineering By Example: An Empirical Study," International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 104-111.

[Maiden, et al., 1995] Maiden, N., et al., "How People Categorise Requirements for Reuse: A Natural Approach," Second International Symposium on Requirements Engineering, IEEE Computer Society Press, 1995.

[Mazza 1996]. Mazza, C., J. Fairclough, B. Melton, D. DePablo, A. Scheffer, and R. Stevens, Software Engineering Standards, Prentice-Hall, 1996.

[Mazza 1996]. Mazza, C., J. Fairclough, B. Melton, D. DePablo, A. Scheffer, R. Stevens, M. Jones, G. Alvisi, Software Engineering Guides, Prentice-Hall, 1996.

[Modugno, et al. 1997]. Modugno, F., et al., "Integrating Safety Analysis of Requirements Specification," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1997.

[Morris, et al. 1994]. Morris, P., et al., "Requirements and Traceability," International Workshop on Requirements Engineering: Foundations of Software Quality, June 1994.

[Paulk 1996]. Paulk, M.C., C.V. Weber, et al., Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.

[Pfleeger 1998]. Pfleeger, S.L., Software Engineering-Theory and Practice. Prentice-Hall, 1998.

[Pohl 1994]. Pohl, K., "The Three Dimensions of Requirements Engineering: A Framework and Its Applications," Information Systems 19, 3 (1994), pp. 243-258.

[Pohl 1999]. Pohl, K., Process-centered Requirements Engineering, Research Studies Press, 1999.

[Potts 1993]. Potts, C., "Choices and Assumptions in Requirements Definition," International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, p. 285.

[Potts 1994]. Potts, C., K. Takahashi, et. al., "Inquiry-based Requirements Analysis," IEEE Software, 11, 2, 1994, pp. 21-32.

[Pressman 1997]. Pressman, R.S. Software Engineering: A Practitioner's Approach (4 edition). McGraw-Hill, 1997.

[Ramesh et al. 1997]. Ramesh, B., et al., "Requirements Traceability: Theory and Practice," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Roberston and Robertson 1999]. Robertson, S., and J. Robertson, Mastering the Requirements Process, Addison Wesley, 1999.

[Rosenberg 1998]. Rosenberg, L., T.F. Hammer and L.L. Huffman, "Requirements, testing and metrics, " 15th Annual Pacific Northwest Software Quality Conference, Utah, October 1998.

[Rudd and Isense 1994]. Rudd, J., and S. Isense, "Twenty-two Tips for a Happier, Healthier Prototype," ACM Interactions, 1, 1, 1994.

[Rzepka 1992]. Rzepka, W., "A Requirements Engineering Testbed: Concept and Status," 2nd IEEE International Conference on Systems Integration, IEEE Computer Society Press, June 1992, pp. 118-126.

[SEI 1995]. A Systems Engineering Capability Model, Version 1.1, CMU/SEI95-MM-003, Software Engineering Institute, 1995.

[Siddiqi and Shekaran 1996]. Siddiqi, J., and M.C. Shekaran, "Requirements Engineering: The Emerging Wisdom," IEEE Software, pp.15-19, 1996.

[Sommerville 1996].Sommerville, I. Software Engineering (5th edition), Addison-Wesley, pp. 63-97, 117-136, 1996.

[Sommerville and Sawyer 1997]. Sommerville, I., and P. Sawyer, "Viewpoints: Principles, Problems, and a Practical Approach to Requirements Engineering," Annals of Software Engineering, 3, N. Mead, ed., 1997.

[Sommerville, et al. 1993]. Sommerville, I., et al., "Integrating Ethnography into the Requirements Engineering Process," International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 165-173.

[Sommerville 1997].Sommerville, I., and P. Sawyer, Requirements engineering: A Good Practice Guide. John Wiley and Sons, 1997

[Stevens 1998]. Stevens, R., P. Brook, K. Jackson and S. Arnold, Systems Engineering, Prentice Hall, 1998.

[Thayer and Dorfman 1990]. Thayer, R., and M. Dorfman, Standards, Guidelines and Examples on System and Software Requirements Engineering. IEEE Computer Society, 1990.

[Thayer and Dorfman 1997]. Thayer, R.H., and M. Dorfman, Software Requirements Engineering (2nd Ed). IEEE Computer Society Press, 1997.

[White 1993]. White, S., "Requirements Engineering in Systems Engineering Practice," IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, January 1993, pp. 192-193.

[White 1994]. White, S., "Comparative Analysis of Embedded Computer System Requirements Methods," IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994, pp. 126-134.