1 Introduction

There is no fair wind for one who knows not whither he is bound.
Lucius Annaeus Seneca, philosopher, 3–65 AD

1.1 Introduction to Requirements

If ever systems development projects needed a "fair wind", they certainly do so today. Fast-changing technology and increased competition are placing ever-increasing pressure on the development process. Effective requirements engineering lies at the heart of an organization's ability to guide the ship and to keep pace with the rising tide of complexity.

Software is currently the dominant force of change of new products. The trend is driven by three key factors:

1. *Arbitrary complexity.* The most complex systems tend to be those with software, often integrated deep inside the system's components. The complexity of such products is limited only by the imagination of those who *conceive them.*
2. *Instant distribution.* Today a company can think of a new product, implement it in software, and rapidly distribute it around the world. For example, a car manufacturer can improve the software in its diagnostic system and then transmit it electronically around the world to tens of thousands of car showrooms in a day.
3. *"Off-the-shelf" components.* Systems are now constructed from bought-in technology and ready-made components with a corresponding reduction in the product development cycle.

The net impact of these trends is a sudden intensity of competition and the ability to monopolize the rewards from the new technology without needing large factories. The result is pressure to reduce the development cycle and the time to deploy technology. However, "time to market" is not sufficient. The real goal is "time to market with the right product". Establishing the requirements enables us to agree on and visualize the "right product". A vital part of the systems engineering process, requirements engineering first defines the problem scope and then links all subsequent development information to it. Only in this way can we expect to control and direct project activity; managing the development of a solution that is both appropriate and cost-effective.

Requirements are the basis for every project, defining what the stakeholders – users, customers, suppliers, developers, businesses – in a potential new system need from it and also what the system must do in order to satisfy that need. To be well understood by everybody they are generally expressed in natural language and herein lies the challenge: to capture the need or problem completely and unambiguously without resorting to specialist jargon or conventions. Once communicated and agreed, requirements drive the project activity. However, the needs of the stakeholders may be many and varied, and may indeed conflict. These needs may not be clearly defined at the start, may be constrained by factors outside their control or may be influenced by other goals which themselves change in the course of time. Without a relatively stable requirements base, a development project can only flounder. It is like setting off on a sea journey without any idea of the destination and with no navigation chart. Requirements provide both the "navigation chart" and the means of steering towards the selected destination.

Agreed requirements provide the basis for planning the development of a system and accepting it on completion. They are essential when sensible and informed tradeoffs have to be made and they are also vital when, as inevitably happens, changes are called for during the development process. How can the impact of a change be assessed without an adequately detailed model of the prior system? Otherwise, what is there to revert to if the change needs to be unwound?

Even as the problem to be solved and potential solutions are defined we must assess the risks of failing to provide a satisfactory solution. Few sponsors or stakeholders will support product or systems development without a convincing risk management strategy. Requirements enable the management of risks from the earliest possible point in development. Risks raised against requirements can be tracked, their impact assessed and the effects of mitigation and fallback plans understood long before substantial development costs have been incurred.

Requirements therefore form the basis for:

- project planning;
- risk management;
- acceptance testing;
- tradeoff;
- change control.

The most common reasons for project failures are not technical and Table 1.1 identifies the main reasons why projects fail. The data is drawn from surveys conducted by the Standish Group in 1995 and 1996, and shows the percentage of projects that stated various reasons for project failure. Those marked with an bullet are directly related to requirements.

The problems fall into three main categories:

- *Requirements* – either poorly organized, poorly expressed, weakly related to stakeholders, changing too rapidly or unnecessary; unrealistic expectations.
- *Management problems of resources* – failure to have enough money, and lack of support or failure to impose proper discipline and planning; many of these arise from poor requirements control.
- *Politics* – which contributes to the first two problems.

All these factors can be addressed at fairly low cost.

Table 1.1 Reasons for project failure

• Incomplete requirements	13.1%
• Lack of user involvement	12.4%
Lack of resources	10.6%
• Unrealistic expectations	9.9%
Lack of executive support	9.3%
• Changing requirements/specification	8.7%
Lack of planning	8.1%
• Didn't need it any longer	7.5%

Sources: Standish Group, 1995 and 1996; *Scientific American*, September 1994.

Table 1.2 Project success factors

• User involvement	15.9%
Management support	13.9%
• Clear statement of requirements	13.0%
Proper planning	9.6%
• Realistic expectations	8.2%
Smaller milestones	7.7%
Competent staff	7.2%
• Ownership	5.3%

Sources: Standish Group, 1995 and 1996; *Scientific American*, September 1994.

Project success factors are not quite the inverse of the failure factors, but as can be seen in Table 1.2. Management support and proper planning are clearly seen as important here – the larger the project and the longer its schedule, the greater is the chance of failure (*Scientific American*, September 1994).

This book considers an engineering approach to requirements in general and requirements management in particular. It explains the differences between stakeholder requirements and system requirements and indicates how requirements can be used to manage system development. It also shows how traceability from stakeholder requirements through system requirements to design can be used to measure progress, manage change and assess risks. Throughout, the reader will be exposed to the testability aspects of requirements and the components designed to satisfy them, and how to formulate validatability or verifiability requests. It stresses the need to produce designs that can be integrated and tested easily.

Requirements management has important interfaces to project management, which is recognized in the book through the presence of Chapter 8, "Management Aspects of Requirements Engineering".

1.2 Introduction to Systems Engineering

This book is not just about requirements for software. The principles and practice of requirements engineering apply to complete systems in which software may play only a small part.

For example, consider a railway system such as the West Coast Mainline from London to Glasgow. A high-level requirement on the system may be to achieve a journey time from Euston Station in London to Glasgow in Scotland in less than 250 minutes. Satisfaction of this single requirement arises from the coordinated interaction of every major component of the system:

- the trains, and their speed;
- the tracks, and their ability to support high-speed trains;
- the stations and station staff, and the waiting time they impose on the trains;
- the drivers, and their ability to control the trains;
- the signalling subsystems;
- the train control and detection subsystems;
- the power delivery subsystems.

Although the software in the signalling and control subsystems plays a vital part in achieving this requirement, it cannot deliver alone. The complete solution involves the whole system. In fact, most requirements are satisfied by the properties that emerge from the way the system as a whole behaves.

What then do we mean by a "system"?

A system is a:

- collection of components – machine, software and human –
- which cooperate in an organized way –
- to achieve some desired result – the requirements.

Thus systems include people. In the West Coast Mainline, the drivers and station staff – the training they receive and the procedures they use – are just as important as the software and machine components.

Since components must cooperate, interfaces between components are a vital consideration in system (and requirements) engineering – interfaces between people and machine components, between machine components and between software components. An example of a machine-to-machine interface in a railway system is the way in which train wheels interface with the track. Apart from the physical arrangements (which are designed to allow the train to be guided along the track without sliding off), electrical currents across the rails may be used to detect the presence of the train as part of the train control subsystem.

At the heart of the concept of a "system" lies the idea of "emergent properties". This refers to the fact that the usefulness of a system does not depend on any particular part of the system, but emerges from the way in which its components interact. Emergent properties may be desirable, in that they have been anticipated and designed into the system so as to make the system useful; or they may be undesirable, in other words unanticipated side effects, such as harm to the environment. The trick in systems engineering is to be able to harness desirable emergent properties and avoid the undesirable ones.

Another important concept is that of "systems of systems". Every system can be construed as being part of a larger, enclosing system. For example, the West Coast Mainline is part of a wider railway system and intersects with other major and minor routes. The entire railway system is part of the wider transport system

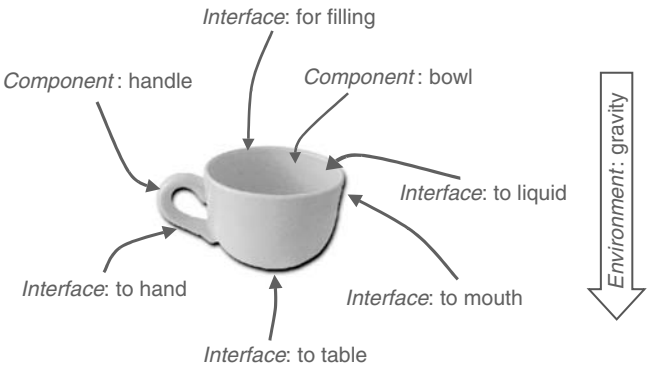

Figure 1.1 A cup as a very simple system.

and interacts in all kinds of ways with the road and air transport networks. The transport system itself provides essential infrastructure for the transport of goods and people as part of the economy of the country. And the country is part of the wider world, and so forth.

To understand the requirements of a system properly is to understand its enclosing system. Often the correct functioning of a system depends on provisions of the enclosing system. For example, the ability of a helicopter to fly depends on the environment provided by the Earth, its gravitation field and its atmosphere.

Take another, very simple, example: a cup (Figure 1.1). It is evident that it has components: a handle and a bowl-shaped container. What purpose do these components serve? The bowl is for containing liquid and the handle is to allow the bowl to be held by someone without getting burnt. We may deduce that the purpose of – or requirement for – the cup is to allow a human being to transfer hot liquid into the mouth without spilling it or getting burnt.

The cup is rich in interfaces. It can be placed on a flat surface for stability; it can be held in a human hand; it can be filled with fluid and emptied; it must interface with the fluid for sustained containment; and it must deliver fluid to the human mouth.

However, there are other observations to be made:

- The cup is no good on its own. It depends on the motor movement of the human arm to achieve its purpose.
- The bowl part of the cup depends crucially on the presence of gravity for its correct functioning. It also has to be used correctly: holding the cup upside down would cause spilling, and may cause scalding.

At the end of the day, the ability of this simple cup to fulfil its purpose depends on:

- the properties that emerge from the interaction of its components;
- appropriate interfaces to external components;
- its correct embedding in the enclosing system – being held in the human hand and lifted by the arm;

- the presence of the proper environment – another solution would be necessary in weightless conditions.

In summary, the engineering of requirements must take the nature of systems into account. Essential considerations are emergent properties, the constraints and provisions of the external environment and the interfaces with surrounding systems.

1.3 Requirements and Quality

The consequences of having no requirements are many and varied. There is ample evidence around us of systems that failed because requirements were not properly organized. However well the system may appear to work at first, if it is not the system that users want or need then it will be useless.

It is interesting to consider the relationship between requirements and quality. The term "quality" may be understood in a variety of ways. When asked about quality cars, one might mention Rolls Royce, Mercedes or Jaguar. This inherent confusion between "quality" and "luxury" is exposed if consideration is given to choosing the best car for the annual RAC rally. Neither Rolls Royce, Mercedes nor Jaguar are chosen, since they do not exhibit the right weight/power ratio, ground clearance and robustness properties. Recent history shows that the best quality car in its class is a Skoda – not a luxury car, but the right quality of car for the job.

Quality, then, is "fitness for purpose" or conformance to requirements – it is providing something that satisfies the customer and in doing so ensures that the needs of all the stakeholders are taken into account.

As will be seen in Chapter 8, requirements engineering acts as a complement to other management considerations, such as cost and schedule, by providing a vital focus on the delivery of quality. Every management decision is a compromise between cost, schedule and quality, three inter-related axes.

Since requirements engineering is a discipline that applies from the start of the development lifecycle, the leverage on quality that can be exercised by proper requirements management is proportionately greater. Relatively little effort expended in early stages of development can reap dividends in the later stages. The adage "Quality is Free" (the title of a book by Phil Crosby) holds true, in that getting it right at the outset can save huge amounts of effort that would have been necessary to put things right later. Improving requirements means improving the quality of the product.

1.4 Requirements and the Lifecycle

There is a common misconception that requirements engineering is just a single phase that is carried out and completed at the outset of product development. The purpose of this section is to demonstrate that requirements engineering has a vital role to play at every stage of development.

As an initial approach, consider one of the very last activities in the development process: acceptance testing. What is a system accepted against? – the stakeholder

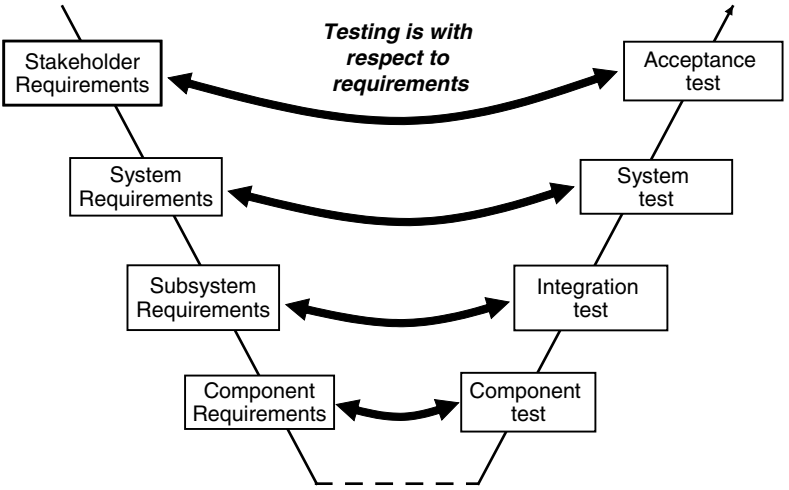

Figure 1.2 Requirements in the V-model.

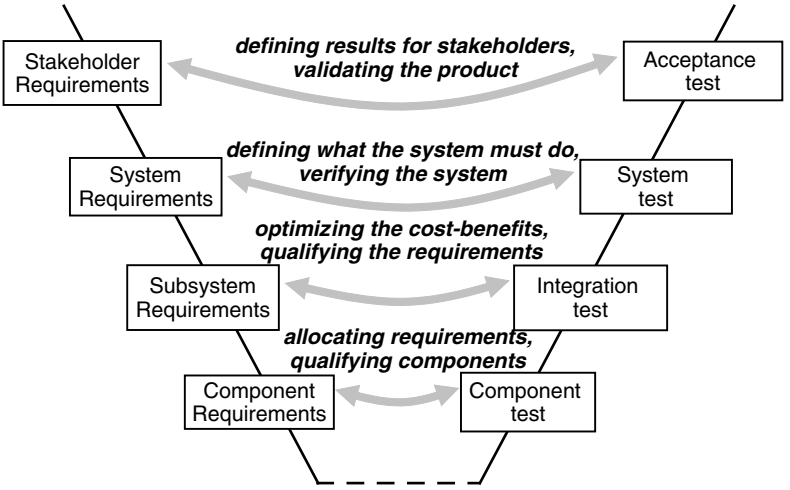

Figure 1.3 Requirements engineering in layers.

requirements. So it can be seen straight away that requirements developed at the outset are still in use in the final stages of development.

The classic V-model, which is used to portray the various stages of development, has its basis in this relationship between testing and requirements. Figure 1.2 shows this relationship at every stage of development.

The V-model also views development in terms of layers, each layer addressing the concerns proper to the corresponding stage of development. Although slightly different processes may be used at each level, the basic pattern of requirements use is the same – a point reinforced through the introduction of a generic process in Chapter 2. Figure 1.3 shows the main concern of requirements engineering at each layer.

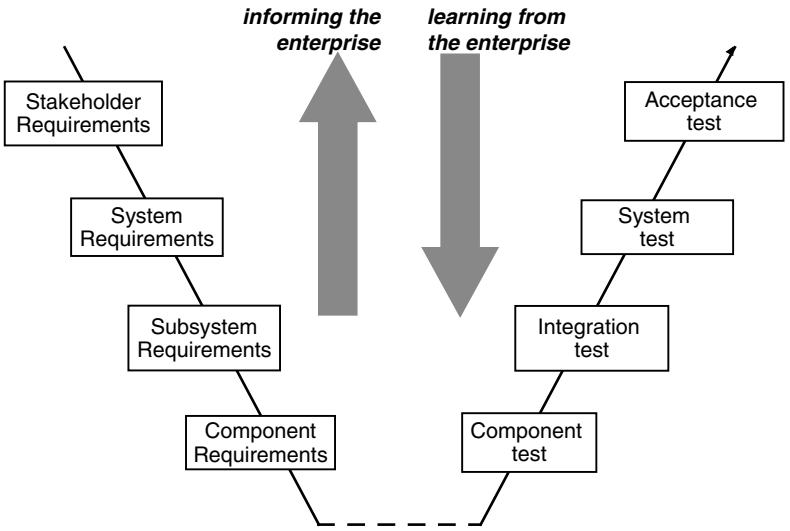

Figure 1.4 Enterprise requirements engineering.

Another role that requirements can play in an organization is to act as a means of communicating between projects. This is a good idea, because many organizations wish to:

- maximise reuse of artefacts across projects;
- manage families of similar products;
- use programme management to coordinate activities;
- optimize process by learning from the experiences of other projects.

A good set of stakeholder requirements can provide a concise, non-technical description of what is being developed at a level that is accessible to senior management. Similarly, the system requirements can form an excellent technical summary of a development project. These descriptions can serve as a basis for comparison with other activities. This is illustrated in Figure 1.4.

If requirements are to play such a central role in systems development, they need to be maintained. To change the design of a product without having also updated the requirements to reflect that change is to store up huge problems for later stages of development. Hence requirements engineering connects strongly with change management.

Whether change originates from within a project – for example, technical issues arising from details of the design – or from without – such as evolving stakeholder needs – the impact of that change on quality, cost and schedule needs to be assessed. This assessment forms the basis on which to:

- accept or reject the change (where that is a choice);
- negotiate the cost of the change with the customer/suppliers;
- organize the redevelopment work.

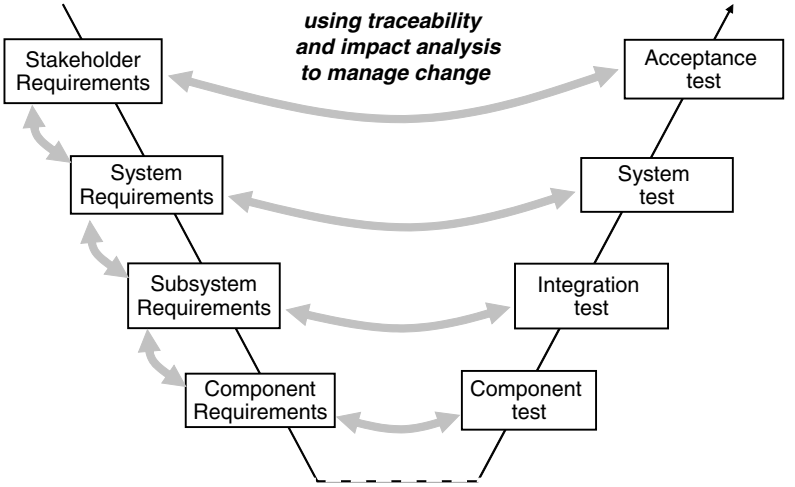

Figure 1.5 Risk of traceability in change management.

The key concept that enables this kind of impact analysis is requirements traceability, a topic treated in greater detail in Section 1.5 and in Chapters 2 and 7. Suffice to say that change management is an integral part of the requirements engineering process. This role is illustrated in Figure 1.5.

Quite apart from change management, a manager's ability to control a project is considerably enhanced by good requirements engineering. Without requirements, project managers have no means of gauging how well the project is going, or even if it is going in the right direction. When it comes to changes there is nothing against which change can be judged. What is more, when they do come to intervene, their only approach is at a technical level, which is inappropriate to their role, and which interferes with the technical role properly played by the engineers. Requirements well expressed at the appropriate level give managers just the right view of the project to be able to perform their role.

In summary, requirements are essential to the health of every system development. They influence the whole development from beginning to end and from top to bottom. Without effective requirements engineering, development projects are like ships drifting rudderless in a storm! Above all else, with good requirements management, hearing the voice of the users and customers ceases to be a game of Chinese whispers, and becomes a matter of clear lines of communication throughout the development process.

1.5 Requirements Traceability

In the requirements engineering context, traceability is about understanding how high-level requirements – objectives, goals, aims, aspirations, expectations, needs – are transformed into low-level requirements. It is therefore primarily concerned with the relationships between layers of information.

In a business context, one may be interested in how

- business vision
  is interpreted as
- business objectives
  are implemented as
- business organization and processes.

In an engineering context, the interest may focus on how

- stakeholder requirements
  are met by
- system requirements
  are partitioned into
- subsystems
  are implemented as
- components.

Using traceability can contribute to the following benefits:

- *Greater confidence in meeting objectives.* Establishing and formalizing traceability engenders greater reflection on how objectives are satisfied.
- *Ability to assess the impact of change.* Various forms of impact analysis become possible in the presence of traceability information.
- *Improved accountability of subordinate organizations.* Greater clarity of how suppliers contribute to the whole.
- *Ability to track progress.* It is notoriously difficult to measure progress when all that you are doing is creating and revising documents. Processes surrounding traceability allow precise measures of progress in the early stages.
- *Ability to balance cost against benefit.* Relating product components to the requirements allows benefit to be assessed against cost.

Traceability relationships are usually many-to-many – that is, one lower level requirement may be linked to several higher level requirements and vice versa. The simplest way to implement a form of traceability is to link requirements statements in one layer with statements in another. Requirements management tools typically allow such linking by drag-and-drop between paragraphs of documents. The links are rather like hyperlinks in web pages, but should ideally be traversable in either direction. Figure 1.6 shows traceability downwards through the layers of requirements and across to the test information. The direction of the arrows follows a particular convention: information traces back to the information it responds to. There are a number of reasons for this convention:

- It usually corresponds to the chronological order in which information is created: always link back to the older information.
- It usually corresponds to access rights due to ownership: one owns the outgoing links from a document, someone else owns the incoming links.

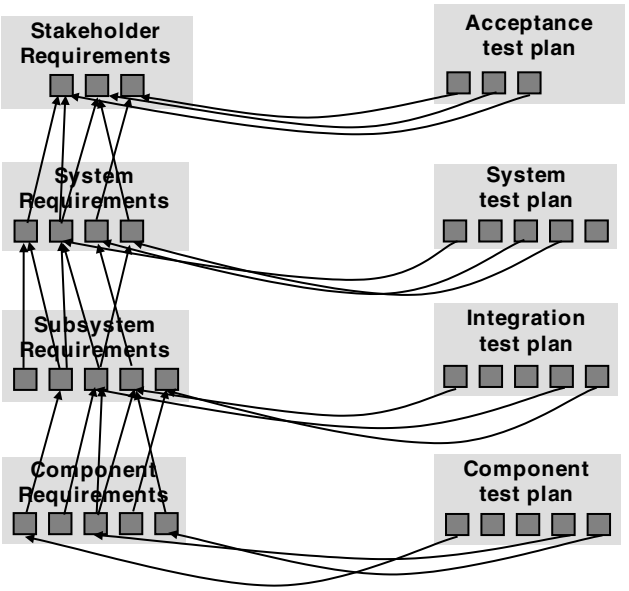

Figure 1.6 Requirements traceability.

Table 1.3 Types of traceability analysis

Type of analysis	Description	Processes supported
Impact analysis	Following incoming links, in answer to the question: "What if this was to change?"	Change management
Derivation analysis	Following outgoing links, in answer to the question: "Why is this here?"	Cost–benefit analysis
Coverage analysis	Counting statements that have links, in answer to the question: "Have I covered everything?" Most often used as a measure of progress	General engineering Management reporting

Various forms of traceability analysis can be used to support requirements engineering processes, presented in Table 1.3.

Impact analysis is used to determine what other artefacts in the development might be affected if a selected artefact changes. This is illustrated in Figure 1.7. The impact is potential; creative analysis has to be carried out by an engineer to determine the exact nature of the impact, if any.

Derivation analysis works in the opposite direction to impact analysis. A low-level artefact – such as a requirement, design element or test – is selected and the traceability links are used to determine what higher level requirements have given rise to it. Elements in the design that do not so trace back are potentially adding cost without benefit.

Finally, coverage analysis can be used to determine that all requirements do trace downwards to lower layers and across to tests. The absence of such a trace is a fairly certain indication that the requirement will not be met or tested. The presence of a link does not, of course, ensure that the requirement *will* be met – that again requires creative engineering judgement.

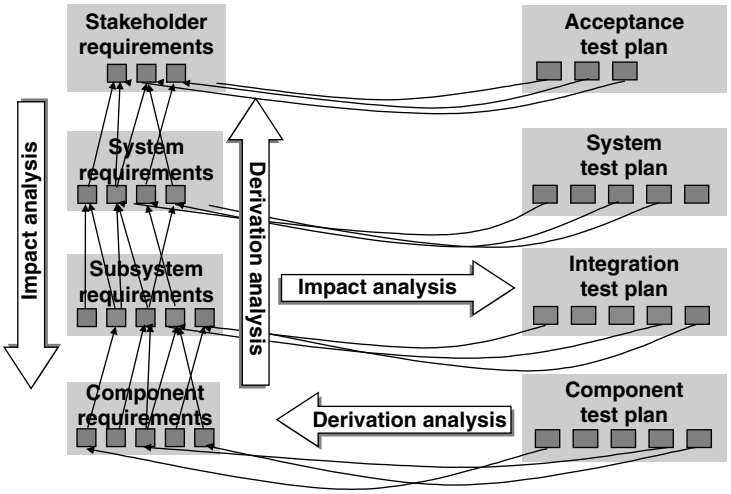

Figure 1.7 Impact and derivation analysis.

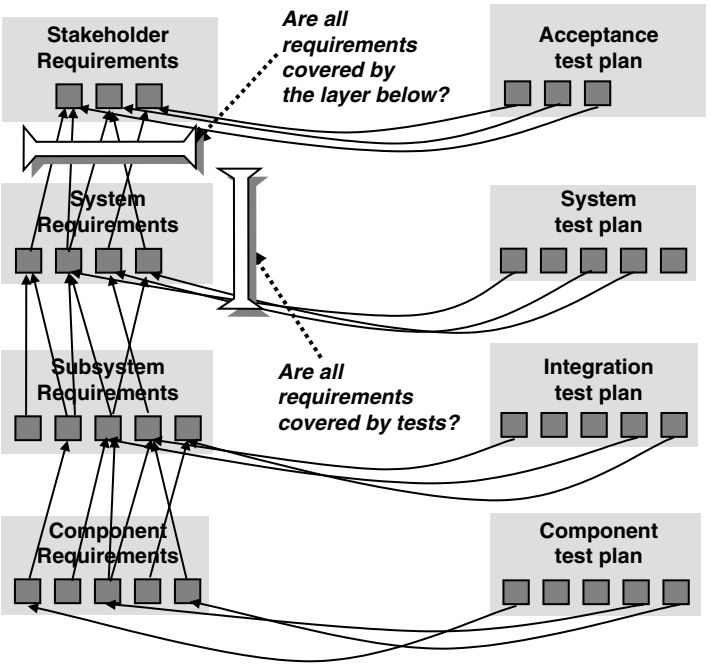

Figure 1.8 Coverage analysis.

Coverage can also be used as a measure of progress: how far have the systems engineers got in responding to the stakeholder requirements? Suppose the task of writing systems requirements in response to stakeholder requirements is given to engineers. As they write system requirements, they link them back to the stakeholder requirements to which they are responding. (By doing it as they go along, the creation of traceability is very little extra overhead – it is much more difficult to establish traceability after both documents have been written!)

At any stage of the task, the engineers' progress can be measured in terms of the percentage of stakeholder requirements that have been covered so far. This is a very useful management tool during the early stages of development.

The same principle can be used to measure progress in planning tests. What percentage of the requirements have tests defined so far? These two dimensions of coverage are summarized in Figure 1.8.

Because of the kinds of analysis that can be carried out, traceability is a simple concept that lies at the heart of the requirements engineering process. More advanced forms of traceability are discussed in detail in Chapter 7.

1.6 Requirements and Modelling

It is important to understand the relationship between requirements management and system modelling. They are mutually supportive activities that should not be equated. Figure 1.9 compares the relationship to a sandwich. In this analogy, requirements management is the "bread and butter" of the development cycle. The "filling" provided by system modelling explains and exposes the analysis and design that has led to subsequent layers of requirements.

Some people talk about requirements modelling. This is a misnomer. You model the system design, not the requirements. Modelling supports the design activity and is where most of the creative work takes place. It assists the engineer in understanding enough of the system to decompose the requirements at a particular level into the next level down. The requirements themselves are a complete snapshot of what is required at each level in increasing levels of detail.

A particular model never says everything about a system – if it did, it would not be a model. For this reason, several different, possibly inter-related, models of systems are often used to cover a variety of different aspects. It is left to the

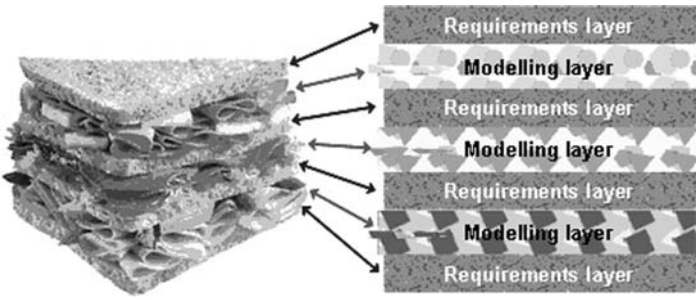

Figure 1.9 The systems engineering sandwich.

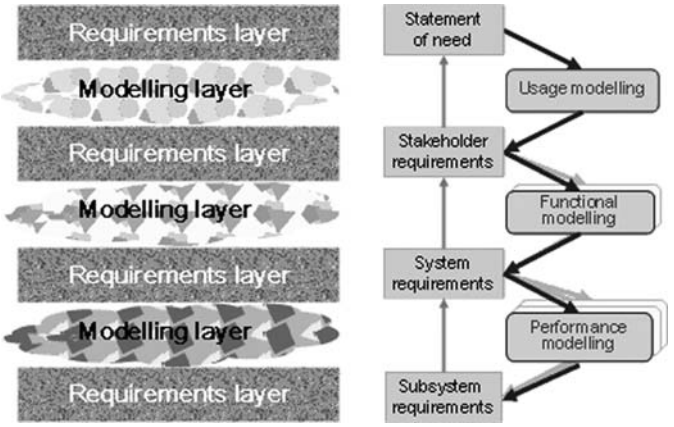

Figure 1.10 Requirements and modelling.

expression of requirements – usually in textual form – to cover those aspects not modelled.

A model is an abstraction of a system that deliberately focuses on some aspects of a system to the exclusion of others. Abstraction is, in this sense, avoidance of distraction – ignoring those details that, although important, are not relevant to a particular model. The advantage of this is that smaller amounts of related information can be collected, processed, organized and analyzed, applying various specific techniques pertinent to the aspects under study.

Where a large amount of complex information has to be managed, modelling provides a means of zooming in, collecting together subsets of the data for a particular purpose and zooming out once more to appreciate the whole. It aids in maintaining a system-wide grasp through focussing on small amounts of information at a time.

Figure 1.10 portrays the inter-related roles that requirements and system modelling play. Models assist the requirements engineer in analysing the requirements at a particular level so as to:

- communicate with the customer and improve mutual understanding of the system to be developed;
- analyze the system to ascertain the presence of desired emergent properties (and the absence of undesirable ones);
- determine how to satisfy the requirements by deriving new requirements at the layer below.

The nature of the models used will vary from layer to layer. At the top layer, usage models such as "stakeholder scenarios" are used to derive the first statement of stakeholder requirements. Following this, various kinds of functional model may be used to derive system requirements from the stakeholder requirements. For software, such models could include UML class diagrams, message sequence charts and state charts. (See Chapter 3 for more details on these modelling techniques.)

Moving from system requirements to architecture, the concerns become focused on various aspects of performance. Multiple models may be used to give confidence that the selected architecture can deliver against both non-functional and functional requirements. Here, models may include queuing theory used to assess performance, wind tunnels for assessing aerodynamics and timetable modelling to assess viability of journey times.

As is evident from these examples, the nature of the models also varies from application to application. The modelling of timetables may be appropriate for the design of railway systems, but not for aircraft design, where the modelling of aerodynamics is rather more appropriate. (Aerodynamics may also be important to high-speed trains, of course.) Message sequence charts may be used in communications systems, but data-rich applications will find data-focused modelling such as entity–relationship diagramming more appropriate.

Whereas the models may vary, the principles of requirements management remain generic across applications. Since this book is about requirements engineering, it also covers the closely associated subject of modelling and methods.

1.7 Requirements and Testing

As has been discussed above, testing is closely related to requirements at every level. In its broadest sense, testing is any activity that allows defects in the system to be detected or prevented, where a defect is a departure from requirements. So testing activities include reviews, inspections, analysis through modelling in addition to the classical tests of components, subsystem and systems that are carried out.

Because of the diversity of testing activities, the term *qualification* is used in this book to refer to all such activities.

Qualification should begin as early as possible, since waiting until the system is almost complete before carrying out any kind of testing can lead to very expensive design changes and rebuilds. The earliest kinds of qualification action take place during the design of the system, and include requirements reviews, design inspections and various forms of analysis carried out on system models.

Figure 1.11 portrays the qualification strategy along a time-line below the V-model. Early qualification actions relate to the left-hand side of the V-model and later ones to the test stages on the right-hand side.

A single stakeholder requirement will typically give rise to a multitude of qualification activities at various stages of development. Where a requirement is satisfied through useful emergent properties, qualification of components alone is insufficient; tests have to be carried out at the level where emergent properties are manifest.

1.8 Requirements in the Problem and Solution Domains

Systems engineering is concerned with developing and managing effective solutions to problems. As has been discussed, it is a staged process vital for businesses in enabling them to produce the right product within acceptable time-scales and costs.

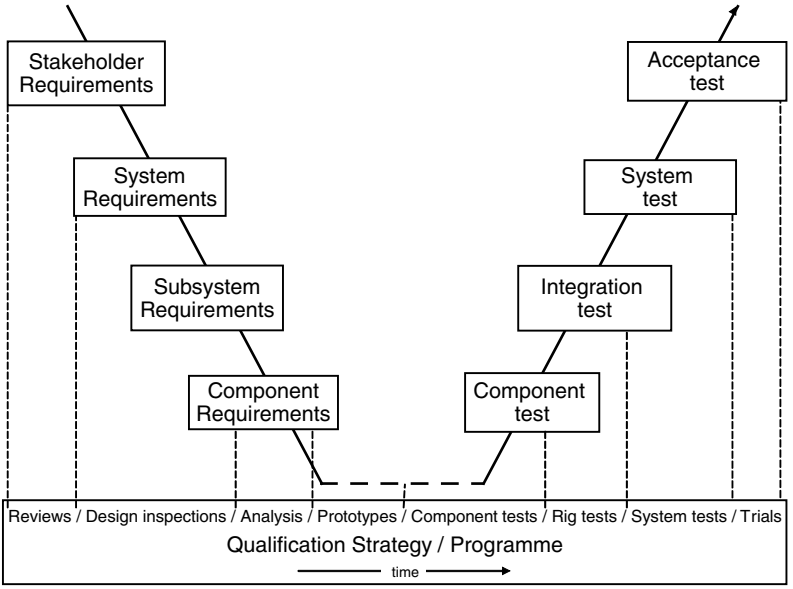

Figure 1.11 Qualification strategy and the V-model.

Table 1.4 Problem and solution spaces

Requirements layer	Domain	View	Role
Stakeholder requirements	Problem domain	Stakeholder's view	State *what* the stakeholders want to achieve through use of the system. Avoid reference to any particular solution
System requirements	Solution domain	Analyst's view	State abstractly *how* the system will meet the stakeholder requirements. Avoid reference to any particular design
Architectural design	Solution domain	Designer's view	State *how* the specific design will meet the system requirements

Early in the process, the definition of the requirements for the product to be built is of prime importance. From a management and engineering point of view, a clear distinction should be made between the "problem domain" and the "solution domain". Those stages of development associated with the highest levels of system description – statement of need, usage modelling and stakeholder requirements – should be firmly rooted in the problem domain, whereas subsequent layers, starting with system requirements, operate in the solution domain.

Table 1.4 portrays the ideal boundary between the problem and solution domains and the roles that the top requirements layers play.

There is an important principle of abstraction at play here. The initial statement of capability should state no more than is necessary to define the problem and avoid any reference to particular solutions. This allows freedom to the systems engineers to carry out their role, which is to devise the best solution without preconceived ideas.

Modelling assists in the derivation of the next layer of requirements and tends to consider possible solutions, even at a high level. To avoid inappropriate solution bias, rather than focus on the system in question, early modelling should focus on the immediately *enclosing* system. For instance, if a radio system is being developed for a sailing boat, then early modelling should focus on the vessel and not so much on the radio. This leads to a statement of the problem to be solved in the context of the enclosing solution.

The same principle applies to the systems engineers: they should allow the designers the freedom to perform their role, that of designing against an abstract solution. The elements of solution introduced through functional modelling remain at a high level, leaving the detail to be defined in subsequent stages.

For example, in a traffic control system:

- The stakeholders may express the problem in terms of maximizing traffic flow while minimizing the risk of accidents at a traffic junction and minimizing cost of maintenance.
- The systems engineers may consider a variety of solutions, such as traffic-lights or roundabouts and a bridge as the approach that best solves the problem within constraints of development and maintenance costs.
- The designers then set to work designing the bridge within the physical constraints presented by the physical environment.

It is frequently the case that the stakeholders will express the problem in terms of a preconceived solution. It then becomes the requirements engineers' job to determine whether there is a good reason for mandating a particular solution or whether it is an unnecessary constraint. For example, the customer starts by trying to procure traffic lights; the supplier asks questions that lead to an understanding of the underlying objectives – maximize traffic flow and minimize risk for drivers and pedestrians – leading to a solution-independent expression of the problem; the reasons for the choice of solution are now better understood and perhaps confirmed through appropriate modelling, leading to a precise and well-informed specification of the abstract solution.

When it comes to procuring systems, a judgement needs to be made as to whether to procure against the problem domain (stakeholder requirements) or against the abstract solution domain (system requirements). Often the nature of the solution is known in advance and it makes sense to procure against system requirements framed in terms of that solution. However, even if procuring against a particular solution, the discipline of capturing a statement of the pure problem prior to a solution still offers important advantages.

Without a clear distinction between problem and solution, the following may result:

- lack of understanding of the real problem;
- inability to scope the system and understand which functions to include;

- domination of debate about the system by the developers and suppliers, because the only descriptions of the system are expressed in terms of solutions;
- inability to find optimal solutions due to lack of design freedom.

For these reasons, the book makes the distinction between stakeholder and system requirements, in terms of how requirements are captured, modelled and expressed.

1.9 How to Read This Book

This book is concerned with engineering requirements and how this process may help those systems engineers and software engineers to create better requirements. Chapter 1 has discussed the importance of requirements and has investigated the role of requirements engineering in all parts of the development lifecycle.

Because of multiple dependencies between chapters, the ordering of material has been carefully chosen to reduce the number of forward references. Although it is best to read the chapters in the sequence presented, some guidelines are given here to assist readers with particular objectives to make efficient use of the book.

Chapter 2, "A Generic Requirements Engineering Process", presents requirements engineering in a generic form that is applicable to all layers of development. Although this approach assists the reader in gaining a good understanding of the essence of requirements engineering, it remains, of necessity, fairly abstract. The generic process is made more concrete, however, in Chapters 5 and 6, where it is adapted to the stakeholder and system layers of development using numerous examples.

Chapter 3, "System Modelling for Requirements Engineering", talks about system modelling, covering various techniques and methods in wide use. This is again in preparation for Chapters 5 and 6, where particular modelling techniques are placed in the context of stakeholder and system requirements.

Chapter 4, "Writing and Reviewing Requirements", addresses the structuring of requirements documents and the expression of requirements statements. Here the language of different kinds of requirement is discussed.

Chapter 5, "Requirements Engineering in the Problem Domain", instantiates the generic process to address the problem domain, in which stakeholder requirements are the primary focus.

Chapter 6, "Requirements Engineering in the Solution Domain", then does the same for requirements in the solution domain, from system requirements downwards through subsystems and components.

Chapter 7, "Advanced Traceability", presents further approaches to traceability, aimed at improving the way in which rationale for traceability is captured, and discusses metrics that can be derived from traceability.

Chapter 8, "Management Aspects of Requirements Engineering", addresses project management in a requirements management context, covering a variety of organization types.

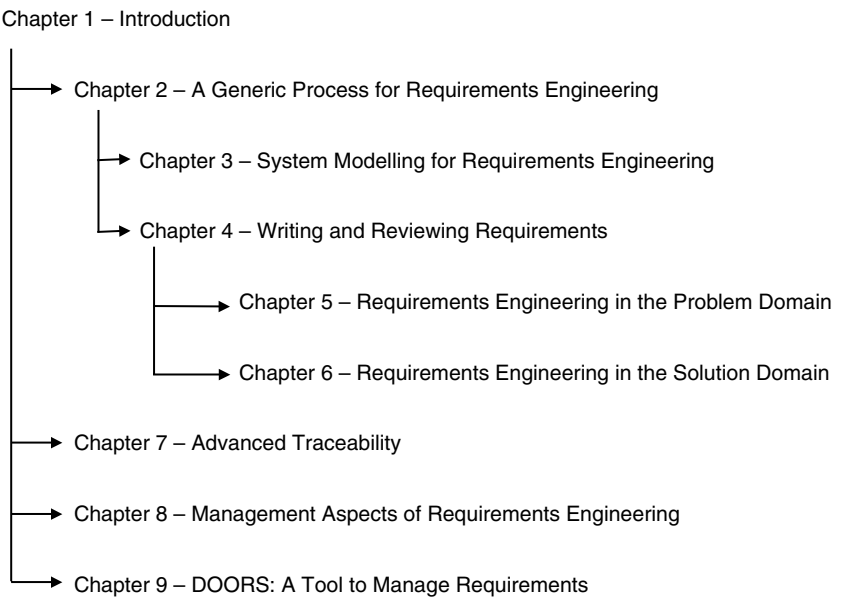

Figure 1.12 Chapter dependencies.

Finally, Chapter 9, "DOORS: A Tool for Requirements Management", provides an overview of DOORS as an example of a software tool which serves as an enabler of a requirements management process. A case study is used to illustrate the processes presented in the book and features of the tool.

Figure 1.12 depicts the chapter dependencies.