CHAPTER ONE

Introduction

Fare buon vino è semplice ma non facile.
(Making good wine is simple but not easy.)
Italian farmer

Requirement Elements / ***Discovery Contexts***	Stakeholders	Goals	Context, Interfaces, Scope	Scenarios	Qualities and Constraints	Rationale and Assumptions	Definitions	Measurements	Priorities
Introduction									
From Individuals									
From Groups									
From Things									
Trade-Offs									
Putting it all Together									

1.1 Summary

Requirements work is simple but not easy: it is a craft, which requires skill. Fortunately, that can be learned.

Requirements are discovered by the use of appropriate inquiry techniques. They are not sitting about, waiting to be 'captured'. Each of the many different situations demands a varied set of techniques.

When requirements are undefined, a project knows neither what it needs to do nor how big a problem it faces. A 'soft systems' process that can handle undefined problems is required, at least at first.

Requirements are more than a list of statements of what a system must do. A better approach is to define a network of related elements, including such things as definitions, goals, rationale and measurements.

This book is organised into two parts:

- Part I, Discovering Requirement Elements;
- Part II, Discovery Contexts.

The book's chapters form the columns (Part I) and rows (Part II) of a matrix that appears at the beginning of each chapter. Any given discovery activity will involve an element from Part I being discovered in a context from Part II. Each chapter in Part I, therefore, cuts across the contexts; each chapter in Part II cuts across the elements.

Requirements work involves a discovery cycle that entails: discovering with stakeholders; documenting; validating with stakeholders. Each chapter in Part 1, Discovering Requirement Elements, is structured according to the discovery cycle.

This book is designed to be entirely practical but, for success, you need to read and reflect as well as work. Books for further reading are suggested.

1.2 Why You Should Read This Book

This book is about what you have to do to discover requirements, whatever kind of business you are in. There are plenty of books and tools designed to help you organise and manage your requirements once you have them, but actual coverage of the critical early stages–of discovery–is far patchier.

The book looks in turn at all the basic elements of requirements, and then at how to discover them. Strangely enough, it seems to be the first book to do this systematically; few popular texts say much about basic requirement

elements like goals, qualities, constraints and rationale and, while there's plentiful coverage on use cases, practical advice on stakeholder analysis and trade-offs is sparse.

Requirements communicate needs from stakeholders to developers on a development project. A development project is a time-limited undertaking with a single purpose: to create the new thing–the product or service–that is named in its mission.

So, if you are involved in a development project, whether as engineer or developer, product manager or business analyst, team leader or project manager, and you need to find out what your project should be doing, this book is for you. We hope the book will be helpful to marketing people, too; after all, marketing discovers requirements, even if it isn't usually expressed in those words.

Words like 'requirement' slip into and out of fashion. Terms like 'systems analyst' have fallen into disuse; new ones like 'business analyst' are becoming popular. In this book, we have tried to focus on the essence of each activity. For instance, we consider different ways to tell the story of what a product should do, including user stories, operational scenarios and use cases. All have their merits but the basic element is the human one of storytelling. We have tried to make this book describe the unchanging core of the work of discovering business needs, so we hope you will find it helpful whatever methodology (and terminology) you prefer.

Because this book's focus is purely practical, we also hope it will be useful to students and lecturers who want to learn about requirements work in industry. The book is arranged in chapters that we have tested as course materials for undergraduate software engineering students. We have included short exercises that could form the basis of tutorial or project work.

Every chapter describes how to apply its techniques to project situations, with practical tips and examples. We have tried hard not to assume prior knowledge on your part, though you will certainly get more from the book if you have project experience. If you are still a student, you will probably find some of the topics obscure until you start trying to discover requirements for yourself. We hope, however, that the examples and illustrations from our own experience will give you an idea of why each technique is useful in industry.

We are aware that busy practitioners have many demands on their time. We have carefully organised this book to make it quick to use.

We believe that for any task, whether you are learning from a book or doing practical work on a project, you need to balance periods of action with periods of reflection. The chapters therefore contain 'grey boxes' that encourage you to reflect on the main text: to read further; to understand the background of an idea or technique; and to improve your own practice.

1.3 Simple but Not Easy

Perhaps your first thought about requirements work is that it is simple, and that you just write down what you need.

Perhaps your experience of trying to write down what you need is that it is not easy, and that you did not get what you wanted.

'Simple but not easy'. Why is that? Things that look simple but are not easy include many traditional skills and crafts.

On a trip to Indonesia some years ago, Ian was lucky enough to be able to watch some traditional boatbuilders at work. They had no workshop other than the beach, and no tools other than hand-adzes, hand-drills and wooden mallets. They used nothing except wood, with moss to fill in the gaps between planks. How did they make the planks fit? How did they know what shape to make each plank? How did they select the wood? They tapped and scraped, a little at a time. They put wooden dowel pegs in the holes in the plank they had just fitted. Then they carefully lined up the holes in the newly shaped plank over the dowels, and gently tapped the plank into place until it was snug and watertight. Each operation was beautifully simple. The final result was a strong, seaworthy boat, like the one illustrated below.

A hand-built outrigger boat, Flores, Indonesia

What are the steps involved in making a set of requirements watertight? How do the parts fit together? How do you create and check each part? What skills do you need for each part? These are the questions this book tries to answer.

1.4 Discovered, Not Found

Requirements cannot be observed or asked for from the users,
but have to be created together with all the stakeholders.
Vesa Torvinen

People have talked for years about requirements capture, gathering, trawling, or elicitation. The words paint attractive pictures of requirements as things you could imprison in a net like fishes or butterflies, harvest like berries, or coax from taciturn experts (Figure 1.1).

But requirements are discovered in a wide range of ways. They are more often created by collaborative work than casually found. As Hugh Beyer and Karen Holtzblatt (1998) write in their book *Contextual Design* [1]: 'Requirements and features don't litter the landscape out at the customer site.'

Often, people do not know what they want either, so it isn't enough just to ask them. You might have to show them prototypes (Chapter 13), devise tests, or work steadily with people towards defining what they want, using the techniques described in Part 1 such as goal and context modelling.

Discovery may also sound somewhat too easy. Like most words, it has different meanings. By discovery, we do not mean stumbling onto things by chance. When teams of scientists discover the quark or the double helix or giant magnetoresistivity or the golden-mantled tree-kangaroo, it is generally not by accident. Discovery, however surprising and delightful the actual moment of realisation, comes as the result of a deliberate search. That search is guided by theory, which itself grows and adapts to whatever is discovered, and is built on evidence.

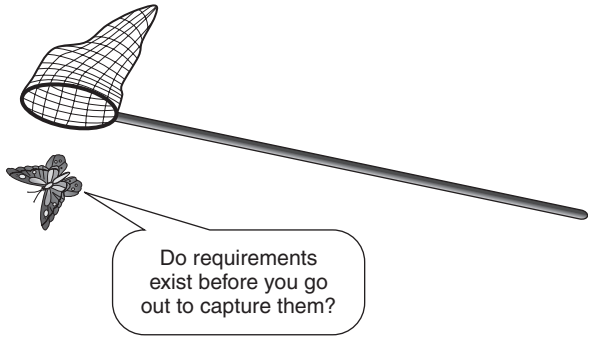

Figure 1.1: Capturing requirements.

A Moment of Discovery

A development project is not a controlled scientific experiment. It is very difficult to prove–and rare even to become aware of–what a project would have been like had requirements not been discovered by the normal systematic use of techniques like those described in this book.

One such moment of conscious discovery came while I was working on a railway project. I was in a workshop, walking through the 'big picture' scenario of how a new train control box was to be brought into service. A conversation ensued, which went something like this:

Me: 'And then you'll be testing the box on the train during a quiet time of day, in between in-service trains?'

Client: 'Um, well, no, we can't run it on the track while other trains are in service in case it fails and causes delays . . . it's against the regulations until it has its safety certificate.'

Me: 'Ah, I see. So you'll be running tests during the night, then, in engineering hours?'

Client: 'Oh no, we won't be able to get enough hours for that. Engineering hours are in demand for all sorts of other purposes. Anyway, most nights the power is turned off to enable the line to be cleaned and the track to be maintained.'

Me: 'Could you maybe just run it around the depot, then?'

Client: 'We could . . . but there are no signals there, so we can't test most of the functions of the box . . . in any case, we're only allowed to run trains in slow manual (5 mph) in the depot.'

Me: 'Well, presumably you have a nice loop of test track with a couple of signals on it so you can try things out properly?'
(Here the workshop participants shifted uncomfortably in their seats and looked at each other nervously.)

Client: 'That would be nice, but we don't have one.'
(A penny dropped in a slot in my brain.)

Me: 'In that case, we'll have to build a simulator. We can tell the contractor to build some software to tell the train control box it is leaving the station, coming to a signal, and arriving at the next station.'
(I was still not prepared for what came next.)

Client: 'Yes, but every station is different. Some have track sloping down, so the train tends to accelerate as it comes in, so the signals have to be further apart on the approach; others slope up, so the signals can be closer together. Some are underground and so are dry; others are above ground and can get wet or icy, so again we allow more distance between the signals there.'

Me: 'In that case, there is no choice: we need a whole-line simulator. Are there plans of the track and signals on the line?'

This discovery was rare, in the sense that it was sudden and large, and everyone realised that a missed requirement had been discovered. Presumably, the omission would finally have been noticed when the test campaign was planned. That would have been most inconveniently late in the project. It would certainly have cost millions of pounds and caused many months of delay.

The need for the simulator had been missed by a specification process that largely followed tradition; for example, there were written requirements and state transition diagrams. But the project's wider context and its bigger picture scenarios had not been thought through.

If there is a lesson to be learned from this, it is that projects need to pay attention to discovering their requirements, using a battery of complementary techniques that include, for instance, not just analysis but also scenarios. That way, the chance of anything large falling through the net is greatly reduced.

Discovery, then, means many different things, such as:

- looking at the evidence;
- being open to new ideas;
- applying creative effort;
- working as a team;
- asking questions that focus the search;
- intending to find particular kinds of thing;
- fitting whatever is found into a reasoned framework;
- relating whatever is found to similar discoveries.

In these senses, requirements discovery is what you need to create a solid foundation for your project.

1.4.1 Many Different Situations

There is no single technique for discovering universal requirements. Projects are of very different kinds, so requirements have to be discovered in diverse ways, most often by working with the appropriate people (Table 1.1).

Projects range from the strictly formal and contractual (as in a custom development or subcontract), to the brief and sketchy (as in small-scale software redevelopment in what is wrongly called software maintenance).

Table 1.1: Where requirements come from.

Type of project	Source of requirements
In-house development by an organisation's IT department	People within the organisation: software users, managers and the IT department itself. No contract ('We can't sue our IT department').
Product or service development for the mass market	Marketing, product management (on behalf of the public and the company); technical experts (specialists) in different disciplines.
Custom development for a single client	Business requirements from the client's technical people; terms and conditions from its commercial side.
Commercial off-the-shelf (COTS) package purchase/tailoring	Package selection, often done **in-house** by matching needs to package capabilities; tailoring, as for either **in-house** or **custom development**.
Subcontract e.g. within a product development	Prime contractor, by derivation from the system requirements and design. Most–possibly all - of this work is analysis rather than discovery as such, though gaps may reveal missed business requirements.
Maintenance support for earlier custom development	System/software users within the client organisation (via change requests received once the product is in service–these are often very informal); problem reports.

Perhaps the projects in greatest danger from poor requirements work are those that seem fairly small and simple, but turn out to contain hidden complexities.

Guest Box: Robin Goldsmith on REAL Business Requirements

Robin F. Goldsmith, JD, author of the book *Discovering REAL Business Requirements for Software Project Success* (2004) [6], describes a key concept from his experience of requirements discovery.

Requirements are NOT all the Same

It's not only that requirements must be discovered rather than gathered, but REAL requirements also are not specified. One of the common major

sources of difficulty starts with the failure to distinguish between two fundamentally different types of requirements.

- **Business requirements** are the REAL requirements which provide value when they are met, satisfied, or delivered. Business requirements are from the perspective of, and in the language of, the business or user. I am using the term 'business' broadly - for work or personal, for profit or not. REAL business requirements are conceptual and exist within the business environment, which is why they must be discovered. They are business deliverable whats that provide value by serving business objectives through solving problems, taking opportunities, and meeting challenges. There usually are *many possible ways to accomplish them*.
- **Product requirements** are from the perspective of, and in the language of, a human-defined product which is *one of those possible ways presumably to accomplish* the business requirements. Since these often are phrased in terms of the product's external functions, they are also called 'functional specifications', which embraces the illusory distinction from 'non-functional requirements.' Humans specify designs. Product requirements are design, which is not limited to engineering technical detail. They provide value if and only if they meet the REAL business requirements.

People, including most fellow requirements authors, usually are referring to product requirements when they use the term 'requirements.' Many do use the term 'business requirements' to mean vague high-level requirements, often just purposes and objectives, which they mistakenly believe decompose into detailed product requirements. Widely-accepted conventional wisdom also holds that creep–changes to requirements after they've supposedly been defined–is due to unclear and untestable product requirements.

In fact, much of creep occurs because product requirements, regardless of how clear and testable they are, do not meet the REAL business requirements. The primary reason is that the REAL business requirements have not been defined adequately and in detail, primarily because people concentrate mainly on product requirements.

The Problem Pyramid™ is a powerful tool that helps draw these distinctions and guide discovery of high-level REAL business requirements. These then must be driven down to sufficient detail to make them adequately clear and testable. At every hierarchical level of detail, REAL business requirements are business deliverable *whats* that provide value when delivered. They never are *hows*. The difference is not level of detail.

The *Problem Pyramid*™ consists of six steps which must be performed in sequence:

1. Identify the REAL **Problem, Opportunity, or Challenge** which provides REAL value when addressed appropriately. This is exceedingly difficult. Failure to identify the REAL problem leads many requirements definitions astray from the start.
2. Identify the **Current Measure(s)** which tell us that the Problem is a Problem. Defining measures is part of defining the REAL Problem. Failure and inability to define measures often indicates that the Problem has been not defined correctly.
3. Identify the **Goal Measure(s)** which tell us the Problem has been addressed. Meeting the Goal Measures(s) provides value.
4. Identify the **Cause(s)** of the Problem. Causes are the *As Is* process producing the Current Measures. One doesn't solve a problem directly but rather one solves it by identifying and addressing its causes.
5. Define the business deliverable *whats* that when delivered will provide value by achieving the Goal Measure(s). These are the *Should Be* process and are the REAL **Business Requirements**.
6. Specify a product **Design** *how* to satisfy the REAL Business Requirements.

1.5 A Softer Process, at First

Fortunately, people (in universities and in industry) have developed a broad understanding of how to go from a situation where nobody even knows what problems ought to be solved, to a position where everybody agrees on the problem that is to be solved. After that point, a definite product or service can be designed to solve the problem effectively, and a more procedural process of 'requirements management', supported by database tools, can take over from discovery or 'requirements definition'.

To reach that point, a 'softer' process than product design, focused more on people than on products, is needed (see Chapter 4). That process iterates the nature of the business, its issues and ambitions, until a clear decision–say, to develop a product–emerges.

Once preliminary development funding, at least, is made available, work can begin on discovering the requirements and defining the scope. Later, when

the shape of the product or service is sufficiently well-defined, harder and more definite modelling approaches can take over. Academics sometimes call these the 'early' and 'late' requirements phases. A book on discovering requirements must naturally focus mainly on the earlier, creative aspects of requirements work. As Kotonya and Sommerville (1998) put it in their textbook *Requirements Engineering* [5]:

> 'Structured methods of requirements analysis ... are not particularly useful for the early stages of analysis where the application domain, the problem, and the organisational requirements must be understood. They are based on ''hard'' models ... [which] are inflexible and focus on the automated systems.'

The softer processes invariably consist of collaborative techniques that involve a group of people with different backgrounds and experiences. Developers tend to call people in all other roles 'users'. Consultants know the importance of 'clients' or 'customers'. Academics and politicians talk about 'stakeholders', a safely neutral word. (Stakeholder analysis is described in Chapter 2.) Consultants and developers, far from being experts, are often the outsiders in whichever domain or area of expertise (such as finance, telecommunications or aerospace) is being addressed.

People working on requirements have to learn *both* about the domain and about what the stakeholders feel is the problem; requirements touch on both 'soft' and 'hard' systems work (Figure 1.2; and see Chapter 4). For consultants, analysts, developers and (in an effective requirements process) other stakeholders alike, this really is discovery.

There are any number of job titles that cover all or part of this work. 'Systems engineer' applies mainly to more hardware-centred industries, whereas 'business analyst' seems to apply mainly to IT systems, so perhaps 'requirements analyst' is a better term for our purposes. The rather loosely defined relationships of some of these professions to the areas of work we are discussing are indicated, very approximately, on Figure 1.2.

There are numerous soft[1] approaches. Some span all of development (usually of software); others are meant mainly for understanding and reorganising whole businesses. For instance:

- **Contextual design** [1], devised by Hugh Beyer and Karen Holtzblatt (1998), starts by exploring the work context, models influences in the 'culture' of a workplace, and continues right through to user interface design.
- **Soft systems methodology** [2], pioneered by Peter Checkland (1990), is generally applied at a strategic level, spanning many domains and any

[1] 'Soft' here means that the work is ill-defined, involving many partially known concerns, not susceptible to deterministic approaches. It does not mean 'about software'.

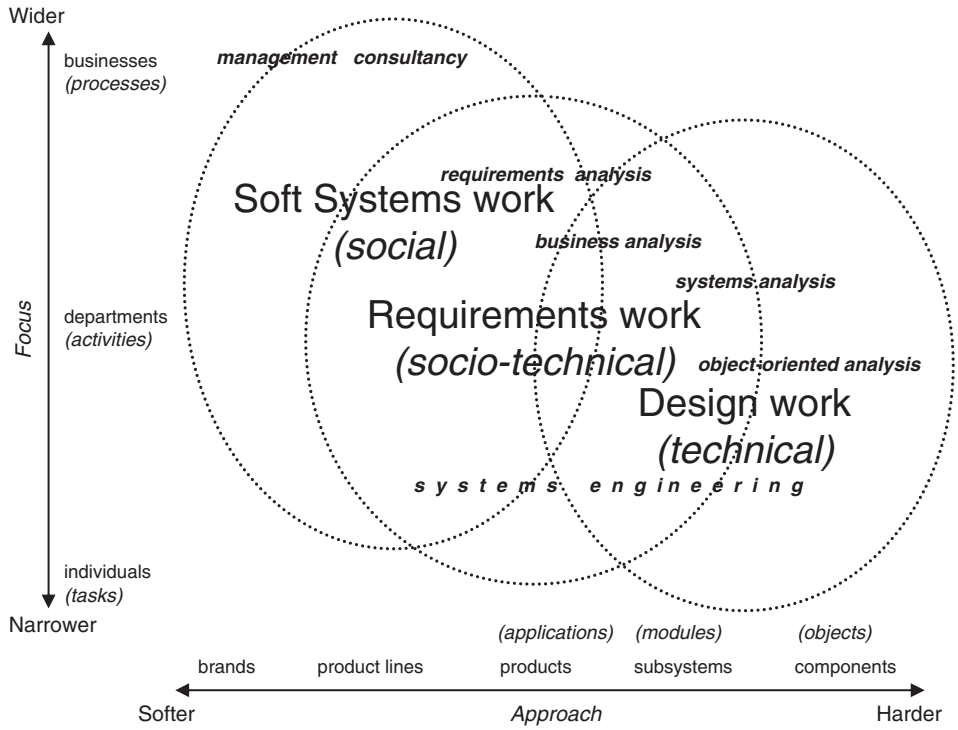

Figure 1.2: Requirements discovery has to span the divide between 'soft' and 'hard', between stakeholder problems and technical solutions.

number of possible future products, but in the main stopping short of individual product specification.

- Similar ideas in strategic planning, business analysis and other fields attempt to understand what a business is trying to achieve, and redesign how it works. For instance, James Dewar's (2002) **Assumption-based planning** [3] offers a fresh insight into discovering assumptions.

The techniques advocated in such processes are not difficult and do not need elaborate tools, but together they offer the requirements analyst the chance of creating better requirements.

None of this predetermines the development life cycle that you should adopt. You can think of the requirements cycle as a smaller wheel inside whatever life cycle you use.

Many organisations write a set of 'stakeholder requirements' at the point in their life cycle where they know they want to create a system but have not chosen its design. They then evaluate alternative designs and select the one they

Requirements 'Engineering'?

Many years ago, the inventor of the DOORS requirements tool, Richard Stevens, phoned me up and said he was working on a tool to do requirements engineering.

'How can you engineer a requirement?', I asked in astonishment. I'd only seen requirements as a list of very dull statements like, 'The system shall enable the user to edit the ''Name'' field.'

What did that possibly have to do with engineering? Richard asked me to suspend my disbelief. He showed me what the tool did, and enthusiastically painted a picture of his vision.

The tool sold very well and did an excellent job of managing requirements on some incredibly big systems in some amazingly complex business environments:

- One firm used it via dedicated satellite links to share confidential requirements on sites on opposite sides of the Atlantic Ocean.
- Another used it to monitor numerous contractors, who were each implementing a different part of a huge specification, via an intricate process of locking slices of the database and passing the slices backwards and forwards as files.

It was fascinating to work with. It involved tools, modelling, database design, customisation with scripts (which sometimes meant weeks of programming), training and data handling. This was certainly software engineering. But were we actually engineering the requirements?

I think now that we were engineering the *management* of the requirements. Other people worked on the requirements as such, in an altogether 'softer' way: much less like engineering; much more like discovery.

will use. In the light of that choice, they write a set of 'system requirements',[2] usually much more detailed than the stakeholder requirements, describing what the (known design of) system will have to do. If the system is large, this is followed by a further stage of design, breaking the system down into several subsystems, each specified in its own set of 'subsystem requirements'. The design of life cycles is outside this book's scope, but Appendix B discusses how to identify the level to which a requirement properly belongs. Since it is easier

[2]These are often requirements relating only to the product under design so the name 'system' is not ideal, but it is widely used. Where the requirements cover issues such as staff training as well as hardware and software, system is certainly the right word. See Chapter 4 for discussion.

to think about details than about the big picture, people often state 'system' or 'subsystem' requirements (prematurely) in their stakeholder requirements.

1.6 More than a List of 'The System Shalls'

The old definition of a requirement as a separately verifiable contractual statement is still valid, but is not very useful during requirements discovery.

It is not possible to start writing formal requirements until you know who the stakeholders are, what their goals are, what the context is and so on. These things are not 'requirements' in the old, narrow sense, but defining them does take you up to the point where you know what problem you want to solve, and can communicate that to a supplier. In this broader sense, goals and scenarios and so on certainly form the requirements; individually, we can call them 'requirement elements'.

'The requirements' in the broad sense means a network of interrelated requirement elements: a requirement that satisfies a goal, is justified in a rationale model, using terms defined in the project dictionary, etc. This is a richer structure than an old-fashioned list of statements, and it fulfils its purposes better. Part I of this book describes these requirement elements, and ways of discovering each of them.

1.6.1 A Network of Requirement Elements

Just as no single technique is sufficient for discovering requirements, so there is no single way of documenting a requirement that is suitable for every situation. Instead, there is a set of commonly occurring elements that together can define what is wanted. Figure 1.3 shows these, together with some relationships between them.

Many other interrelationships are possible. For example, a scenario may reveal the need for an interface; an assumption may justify a measurement; a stakeholder may be responsible for some constraints.

These elements should largely be familiar, as they are simply tidied up versions of concepts that everyone uses in daily conversation: roles, stories, events, goals, reasons and so on. Indeed, it seems that many of them are fundamental to the structure of every human language, if the ideas advanced in Steven Pinker's (2007) *The Stuff of Thought* [7] are correct. That Pinker really is close to the truth can be seen in the way that different methods select subsets of these elements (see Chapter 15 for further evidence).

Figure 1.3 is a coarse-grained[3] model of a possible organisation of the requirements information within a project. Each box in the diagram (except

[3]In a finer grained model, more elements would appear. For instance, 'scenario' would resolve into 'scenario step', 'exception', and so on.

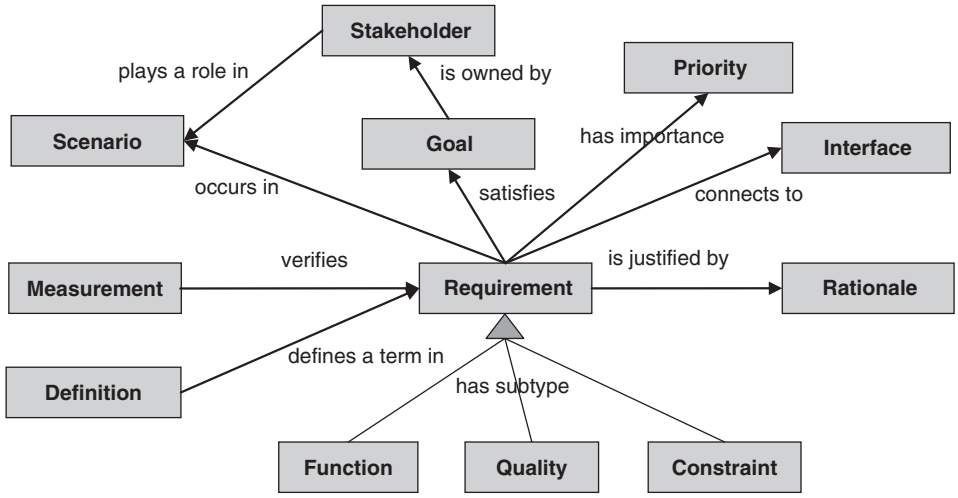

Figure 1.3: Requirements specification as a network of elements.

'requirement' itself) names a requirements element. All but one of the element boxes correspond directly to chapters in Part I of this book. The exception is 'function', which is covered by Chapters 4 and 5. The writing of functions is also described in Ian Alexander and Richard Stevens' (2002) *Writing Better Requirements* [12]. Interfaces are both design (of the containing system) and requirements (on the contained product). Since interfaces may be known in advance, they show that the idea that requirements come before design is not necessarily true.

Different projects vary considerably in terms of the importance of the different requirement elements. In practice, therefore, projects sometimes entirely omit some of the elements shown in Figure 1.3. For example, a project with a simple stakeholder structure might omit 'stakeholder analysis'. The resulting lack could be compensated for by additional explanation of stakeholder issues in 'scenarios', 'definitions' or 'rationale'. For more on tailoring a requirements approach for your own project, see Chapter 15.

The information model drawn in Figure 1.3 only covers requirements. Other related elements in a project's information model include plans, risks, issues/decisions and tests. These are important to projects, will have connections to the requirement elements, and may be appropriately recorded in a requirements database (see Appendix C) but are outside the scope of this book.

Figure 1.3, and indeed the chapter structure of this book, can be seen as a customisable template for organising the requirements on your project. For convenience, a generalised template is provided as a starting point in Appendix D.

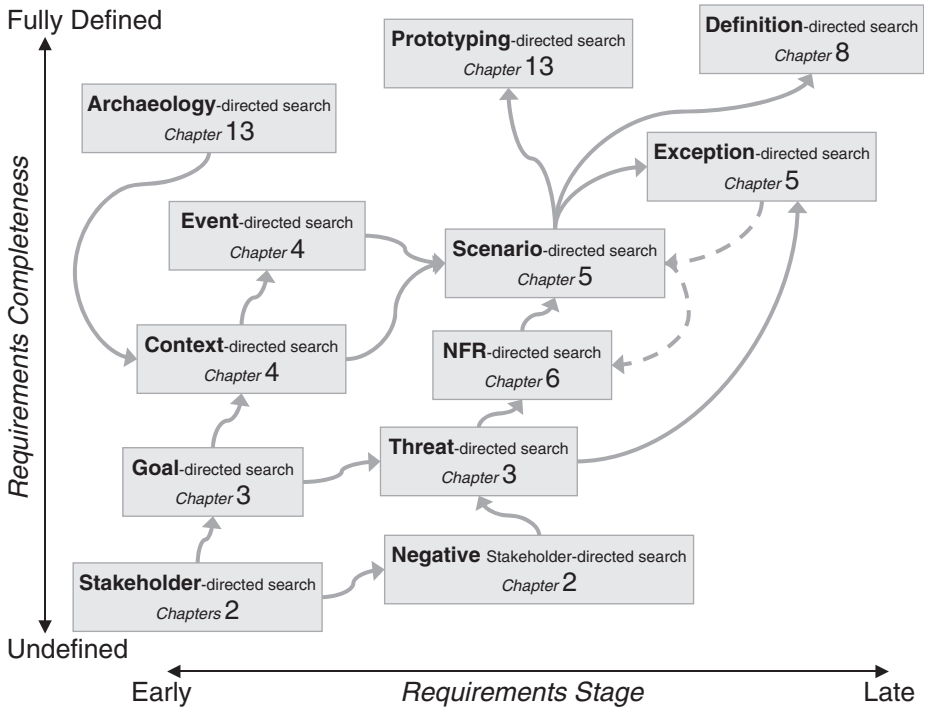

Figure 1.4: Requirements discovery as search, directed by the requirements elements so far discovered.

1.6.2 Discovery as Search

A good way of thinking about discovery is as a search (Figure 1.4). You can use different techniques, related to the requirement elements defined in Part I of this book, to direct your search at different stages. In other words, the structure of what you know drives what you discover next. The better you organise your knowledge of the requirements, the better you can discover what is really needed.

For example, you discover a goal. Then you can use a scenario to explore how to achieve that goal. Then you can search for exceptions that can occur at each step of that scenario.

1.7 A Minimum of Process: The Discovery Cycle

A generic discovery process can be drawn as a simple inquiry cycle (Figure 1.5).

An inquiry cycle is more or less what it says: a cycle of activities, carried out by a team, to enable them to inquire effectively into some subject (such as what precisely should be developed as a product).

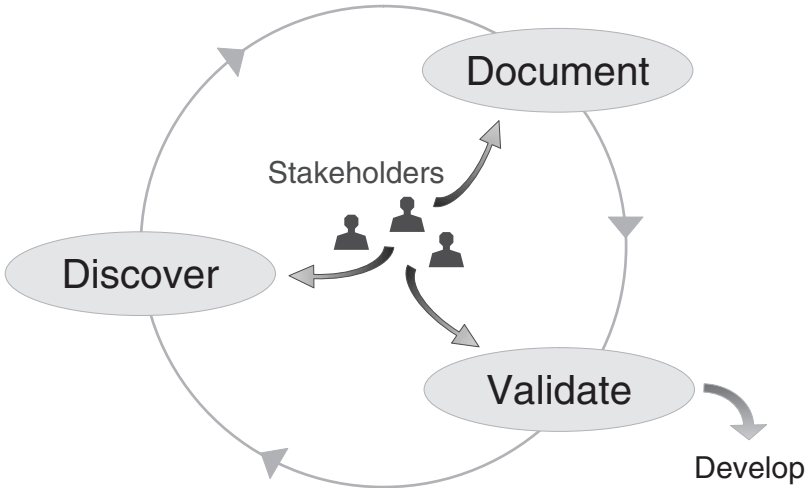

Figure 1.5: Inquiry cycle for creating requirements.

What all inquiry cycles have in common is a period of action followed by a period of reflection (Heron 1996) [4]. Both are necessary: action to get on with the work; reflection to consider whether the work is complete, or going in the right direction.

Agility

It is fashionable nowadays to use the word 'agile' a lot. On some kinds of project, you can discover a handful of requirements, implement them in a product at once and then go back to discover a few more. This approach can be very helpful in some situations, as it gives customers something to look at, and this may prompt them to tell you some new requirements that they hadn't known about before.

Agility is not really new; people used to call it 'rapid prototyping', 'iterative development', 'rapid application development' and other such complicated names. Good project managers and project teams have always practised it.

Rapid iteration does not in any way absolve you from discovering the requirements. Agile or not, the decision depends on your situation. But, discovering requirements is crucial.

For example, if you hold a workshop to discover scenarios (Chapter 5), you are immersed in action. Afterwards, quietly analysing models of those scenarios to document the workshop's findings, you have time to reflect on

what you have found. Then, armed with a fresh list of questions, you go back to the relevant stakeholders and validate your findings. This may lead you to further discovery activities.

However, the techniques for discovering, documenting, and validating a set of scenarios (for instance) are not necessarily the same as those for discovering goals, or risks, or stakeholders. Also, it's no good waiting until the end of the requirements phase before you try to validate each finely polished requirement. You need to do that right away, checking each little discovery as you go along. So there isn't one enormous requirements process, but many little inquiry cycles. You could call this 'agile requirements discovery'. For example, Mike Cohn's (2004) *User Stories Applied* [8] offers a fully worked out agile approach (for software). Accordingly, each chapter in Part I covers the discovery, documentation and validation of the element it discusses.

Apart from that very basic process, this book's focus is essentially on what you actually need to do on your project. It does not propose some new overarching process, framework or methodology, because:

- Every project is different; standardised processes can only describe the average or typical case (and a few common variations). The final chapter of this book describes how to tailor your own requirements process from the building blocks described in the earlier chapters.
- The actual techniques don't have to be used in some official, fixed process or method at all. You can scribble a goal model during an interview, or while reflecting afterwards on the train home. You can do a little stakeholder analysis and note down a few scenarios, and then get on with something else. If this solves a problem on a project, that's fine.
- What is missing isn't theory, but practice: projects creating their requirements simply and carefully. This book describes straightforward ways of doing that.

1.8 The Structure of this Book

This book is divided into two main parts:

- Part I describes what you need to discover – different requirement elements–with practical techniques to create, document and validate them. On most projects, you will probably need to use most of these at some stage. The requirement elements are described in roughly the order you are most likely to use them. They do not form a single mandatory process or anything like that. You have to use your common sense and develop the experience to apply those that will work best on your project.
- Part II describes the contexts where you can effectively discover requirements. These include the traditional environments in which consultants

meet stakeholders – interviews and workshops – as well as consulting the public, observation, 'reverse engineering' and reuse. It also describes how to make the transition from discovery to using the requirements to drive your project. This includes trading off what is wanted against what can be achieved.

Each chapter in Parts I and II is structured to help the reader, with:

- a list of questions and answers;
- a goal;
- a summary;
- the main text, illustrated with diagrams, tables and photographs;
- a list of bare minimum activities;
- next steps;
- exercises, with hints on answers at the back of the book;
- further reading.

1.8.1 Part I: Discovering Requirement Elements

Part I describes the things you need to find out, a step at a time. They may be created directly by working with stakeholders in the contexts described in Part II, or they may be analysed quietly in between meetings with stakeholders.

Each chapter describes one element (e.g. scenarios), with worked examples, containing sections saying how to:

- discover it;
- document it;
- validate it for completeness, correctness, and consistency.

The elements of Part I include:

- a list of the stakeholders, and their influences on each other;
- goals, including the negative side;
- context, interfaces and scope of the work, and of possible future products, including a list of the events to be handled;
- stories, scenarios and use cases that describe how the product could be used to deliver the wanted results;
- qualities and constraints that any acceptable product must meet;
- rationale and assumptions, arguing the case for (and against) decisions, such as choosing particular requirements and design options;
- technical terms, data definitions and roles used in a project;
- acceptance criteria and verification methods, or qualities of service, defining how you will know that the requirements have been met;

- priorities, both input (desired by stakeholders) and output (agreed by the project for a given phase of development).

The descriptions in each case are enlivened with brief accounts of practical experience.

1.8.2 Part II: Discovery Contexts

Part II describes practical ways for developers to work together with stakeholders to discover the requirements. The contexts it describes include:

- interviews;
- workshops;
- observation of or participation in the work;
- 'reverse engineering'–discovery from existing systems;
- reuse, where requirements are well enough understood or standardised to be taken over without being re-created;
- trade-offs.

These are the contexts in which you can discover all of the elements described in Part I. These two parts of the book therefore form a matrix of elements against contexts. Your project will follow its own unique path through this matrix of discovery opportunities.

The book ends with a chapter on how to put all the elements and contexts together. Among other things, it describes the use of the element/context matrix itself to guide the planning of the requirements process.

1.9 Further Reading

1.9.1 Books on 'Softer' Approaches

1. Beyer, H. and Holtzblatt, K. (1998) *Contextual Design: Defining Customer-Centered Systems,* London: Morgan Kaufmann.
   A good practical book from industry experts with a user interface design background. They describe a process that moves from the 'customer' to understanding the work context of a future product, to designing and prototyping a software product and its user interface.
2. Checkland, P. and Scholes, J. (1990) *Soft Systems Methodology in Action,* Chichester: John Wiley & Sons, Ltd.
   Checkland essentially founded the Soft Systems Methodology (SSM) approach, and his books present the thinking and experience behind it. Checkland's 'rich pictures' and way of thinking are useful for understanding a problem and the many pressures on systems and their stakeholders.

SSM has now taken on a life of its own, but Checkland explains the basic concepts well. SSM is a valuable precursor to requirements work, for example when people don't agree on what problem is to be solved.

3. Dewar, J. (2002) *Assumption-Based Planning,* Cambridge: Cambridge University Press.
   Dewar's book describes a set of effective techniques to explore and improve strategic plans. It shows how you can discover the unspoken assumptions that plans depend on, and work out what to do if those assumptions should break. This leads to the reasons for decisions, and to robust requirements.
4. Heron, J. (1996) *Co-operative Inquiry: Research into the Human Condition,* London: Sage.
   Heron has written a rather intellectual and academic book, but it describes a practical way for people (groups of stakeholders) to work together in an inquiry cycle process for any purpose. The approach can be seen as a human-centred version of the widely used 'plan, do, check' management cycle proposed by W. Edwards Deming. It is also a kind of 'action research'. A similar approach was used by Peter Checkland and his colleagues to develop Soft Systems Methodology.
5. Kotonya, G. and Sommerville, I. (1998) *Requirements Engineering, Processes and Techniques,* Chichester: John Wiley & Sons, Ltd.
   This useful textbook pioneered the coverage of 'early' requirements work. Its authors who, like Checkland, were researchers at Lancaster University, were aware of that university's tradition of collaborative work including ethnographic observation and soft systems.
6. Goldsmith, R.F. (2004) *Discovering REAL Business Requirements for Software Project Success,* Boston: Artech House.
   This is one of the few books explicitly about requirements discovery (and is very different from this book). Goldsmith offers many practical suggestions for checking and evaluating proposed requirements so as to weed out any that are not 'REAL Business Requirements'. The book is simply and engagingly written. It is very clear on some of the common traps and pitfalls in writing requirements.

1.9.2 Books on the Philosophical Background

7. Pinker, S. (2007) *The Stuff of Thought,* London: Allen Lane.
   Pinker's wonderful book explains the structures and concepts that seem to lie below the surface of all human languages. What is fascinating is that many of the elements that go to make up requirements (as in Part I of *Discovering Requirements*) seem to be inherent in every language.

For example, sentences have place-holders for (stakeholder) roles in the form of expected subjects and objects; roles are expected to have goals, and to push for those, against opposing forces (threats, hostile stakeholders); special verbs describe (interface) events and have implied (scenario) timelines associated with them, and so on. If Pinker is right, then the described requirement elements are fundamental to human thought, and will always be the natural way to express needs.

8. Cohn, M. (2004) *User Stories Applied: For Agile Software Development,* Boston: Addison-Wesley.
   Cohn has written a radical, funny and powerfully argued book. It effectively demolishes the traditional approach of, say, IEEE 830 (Institute of Electrical and Electronics Engineers Software Requirements Specifications), that requirements can be written as a straightforward set of 'shall statements'. Cohn shows that more is needed. He favours short, informal scenarios (user stories) but is wise enough to see that other elements, such as goals, have their uses (e.g. page 135 of his book). The approach as written is strongly tailored to software, e.g. with very rapid build cycles, but many of the ideas have wider application.
9. Gause, D.C. and Weinberg, G.M. (1989) *Exploring Requirements: Quality Before Design,* New York: Dorset House.
   This was one of the very first books that was explicitly about 'requirements'. It remains stimulating and engaging all these years after it was published because it looks, very simply, at the basic issues that underlie requirements discovery, even if it raises many more questions than it answers. Here is a taste: 'In one requirements review of a single eight-page piece of an on-line banking system, we turned up 121 ambiguities that were interpreted in at least two ways by different reviewers.'

1.9.3 Books on 'Harder' Approaches

10. Simon, H.A. (1996) *The Sciences of the Artificial,* 3rd edition, Cambridge, Mass: MIT Press.
    This small paperback contains a set of essays; Simon describes it as a 'fugue, whose subject and countersubject were first uttered in lectures . . . but are now woven together as . . . alternating chapters'. In other words, it is an intellectual account of the underpinnings of the hard, rational, systems view of the world of both engineering and society. Chapters 5 and 8, on design and complexity respectively, are the most obviously relevant. Simon writes fluently and persuasively, but from an extremely 'hard' standpoint.

11. Stevens, R., Brook, P., Jackson, K. and Arnold, S. (1998) *Systems Engineering, Coping with Complexity,* London: Prentice Hall.
'The green book', as it is known by systems engineers, is a simple account of how a large systems project for a product such as an aircraft should be organised. (*The Sciences of the Artificial* is listed in its references.) The book starts with a set of requirements to be managed; it is consistent with the philosophy of requirements database tools such as Stevens' creation, DOORS, but focuses on processes rather than tool support.
12. Alexander, I. and Stevens, R. (2002) *Writing Better Requirements,* London: Addison-Wesley.
Alexander and Stevens provide simple advice on framing requirements in words, with a minimum of process.