2 The Requirements Process

in which we look at a process for gathering requirements and discuss how you might use it

The requirements process described in this book is the product of our experience. We developed the Volere Requirements Process and its associated specification template from the activities and deliverables we have found effective over years of working on projects and consulting with our clients. The result of this experience is a requirements-gathering and specification process whose principles can be applied to almost all kinds of application types in almost all kinds of development environments.

We want to stress from the very beginning that while we are presenting a process, we are using it as a vehicle for finding requirements. That is, we do not expect you to wave the process around and tell your coworkers that this is "the way to do it." We do expect you will find many useful things to do within the process that will help you to gather requirements more productively and accurately. We are sure of this fact, because we have personally seen hundreds of companies adapt the process to their own cultures and organizations, and we know of thousands more that have done so.

A requirements process is not just for waterfall development. Our clients use XP, RUP, and many other acronyms, as well as traditional waterfall, incremental, and all flavors of agile development processes. Over the years they have agreed with us: If the right product is to be built, then the right requirements have to be discovered. But requirements don't come about by fortuitous accident. To find the correct and complete requirements, you need some kind of systematic process.

The Volere Requirements Process Model in Appendix A contains a detailed model of all of the activities and the connections between them. The model is very detailed. Rather than getting involved in that kind of

Whether you are building custom systems, building systems by assembling components, using commercial off-the-shelf software, accessing open source software, or making changes to existing software, you still need to explore, understand, capture, and communicate the requirements.

See Appendix A for the complete Volere Requirements Process.

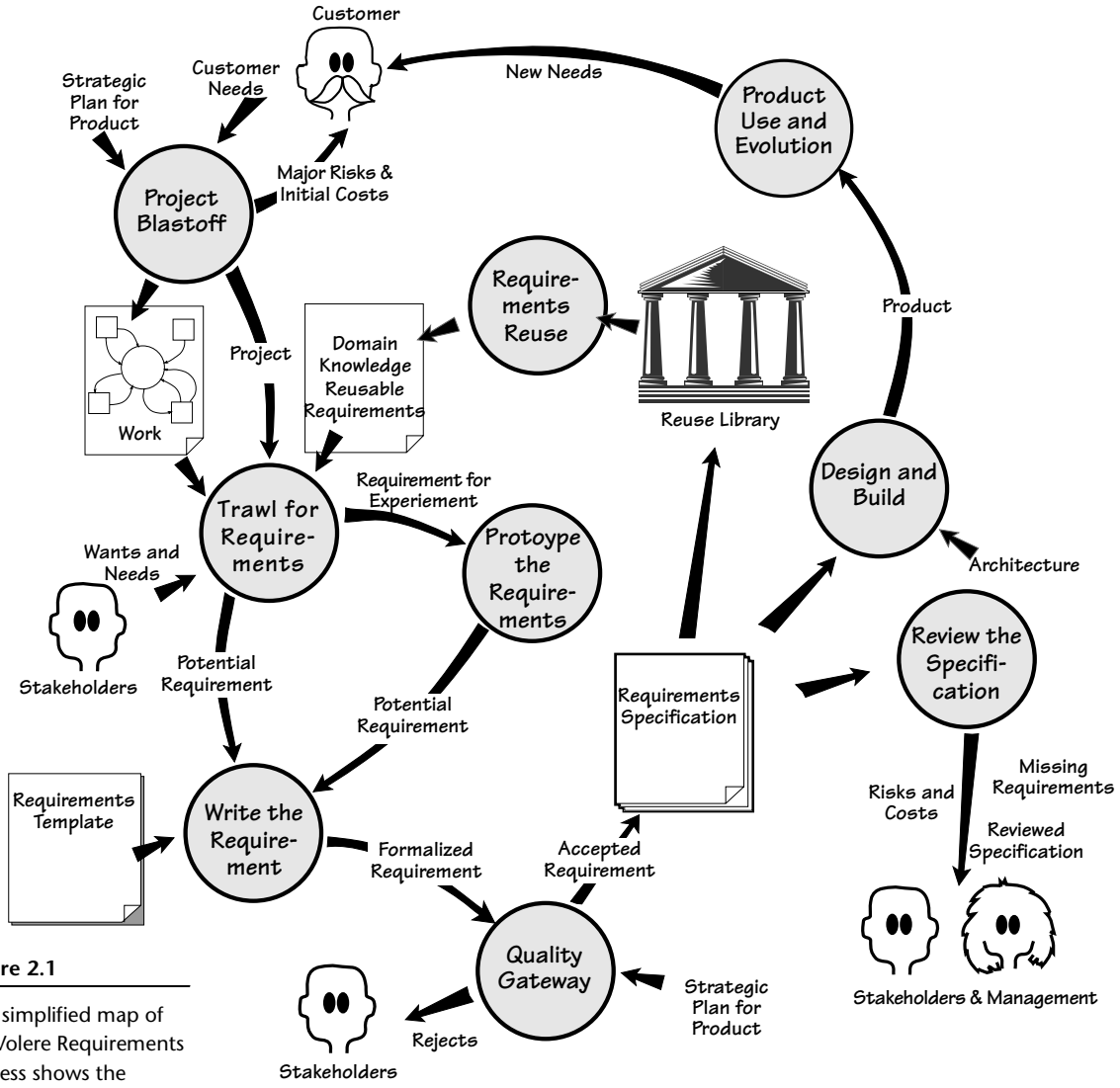


Figure 2.1

This simplified map of the Volere Requirements Process shows the activities and their deliverables. We have used a stylized data flow notation. When you are looking at this diagram, keep in mind that the process is iterative and evolutionary. Each activity (the bubbles) and its deliverables (arrows or documents) are explained in the text.

detail and complexity right away, let's stand back and take a simpler view of the overall process. This simplified view is shown in Figure 2.1.

This simplified version of the process model will reappear at various places throughout this book. For the most part, the simplified version is faithful enough to the complete version in Appendix A. Sometimes, for the sake of making an explanation simpler or clearer, we will make minor changes to the model or show only part of it. Please be aware of this and use the detailed model for any of your important implementation decisions.

Figure 2.1 shows all of the main activities in the Volere Requirements Process, including how they are linked by their deliverables. The deliverables are shown as moving from one activity to the next. For example, the *Trawling for Requirements* activity would probably gather the requirements for one business use case at a time. The requirements would be *written* to demonstrate that they have been correctly understood and agreed, and then passed to the *Quality Gateway* for testing prior to being included in the *Requirements Specification*. Any rejects would be sent back to the originator, who probably would take them back to the trawling activity for clarification and further explanation. Don't take this apparent waterfall approach too literally—the activities usually iterate and overlap before producing their final output. As we go through this process in detail, we will explain where and how iteration and incremental delivery can be used.

Agility Guide

When referring to development processes, *agility* is normally taken to mean the absence of a set process that must be followed regardless of the product being developed. However, it does *not* mean the absence of all process. Agile development means selecting the appropriate process, or parts of a process, that are appropriate for the product and the project.

Not all projects can be as agile as others. Large numbers of stakeholders, the need for documentation, scattered development teams, and other factors will inevitably dictate the degree of agility that can be applied by the project team. As a consequence, you must determine the degree of agility appropriate to your current project. In Chapter 1, we introduced symbols to represent your aspirations for agility. These symbols are intended to guide you as you select the most appropriate way to use the information in this book.

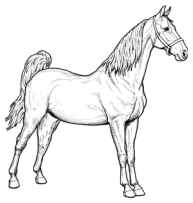
Rabbit projects are those where circumstances allow for the highest degree of agility. They are typically, but not necessarily, smaller projects where close stakeholder participation is possible. Rabbit projects usually include a smaller number of stakeholders.

The fastest way to learn your customer's and his organization's requirements is not at the keyboard but at the whiteboard.

Participants in rabbit projects may think it odd to consider using any kind of process at all for requirements. However, as you look at the process in this chapter, think not of a process that delivers a *requirements specification*, but rather of a process that delivers a requirement, or at least requirements, one use case at a time. If you are using eXtreme Programming (most likely with rabbit projects), the fastest way to learn your customer's and (importantly) his organization's requirements is not at the keyboard but at the whiteboard. Pay particular attention to the part that prototyping and scenarios play in the process and to the idea of *essence* of the system. It is crucial that you understand the difference between a requirement and a solution.

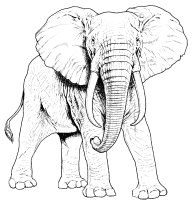
Rabbit projects are iterative. They gather requirements in small units (probably one business use case at a time) and then implement the solution

piecemeal, using the implementation to get feedback from the stakeholders. However, we stress that this feedback should not be used to find out what the stakeholders wanted in the first place. That is the role of requirements, and it is far more effective and efficient if done using requirements methods. Read on.



Horse projects are the "halfway house" of agility. They use as much agility as possible, but have constraints imposed by the project and the organization. Horse projects should use most of the process we are about to describe, keeping in mind that you could easily use an iterative approach to requirements gathering. That is, the requirements for one unit of work—probably one business use case—are gathered and then the designers start work on those requirements. This strategy needs the overall architecture to be in place before it can work. The advantage is that while the requirements analysts are gathering the requirements for one business use case, the developers are busy building a solution for the requirements from the previous business use case.

The sections on trawling, writing, and the Quality Gateway will be of great interest to horse projects. If these activities are done correctly and iteratively, your project can achieve a considerable effectiveness without becoming bogged down in its own process.



Elephant projects are the least agile, but—like their namesake—are large, are dependable, and have long memories. In such a case, your aspirations toward agility may be limited by the organization of the project—for example, you may have a large number of scattered stakeholders—or the need to produce formal requirements documentation such as for pharmaceutical or aeronautical projects, projects that entail some contractual obligation, or projects where you are outsourcing some tasks to another organization.

Most of the elements of the requirements process outlined in this chapter are used by elephant projects. But always be on the lookout for opportunities in your project to increase your agility by gathering requirements in an iterative manner.

Requirements Process in Context

Just as the product evolves on its own, so you may choose to make it evolve by building the early versions with a minimal amount of functionality, and later augmenting it by a planned series of releases

There is no end to the requirements process. When a product, or partial product, is delivered and your users start using it, evolution kicks in. As people use the product, they discover new needs and uses for it, and they then want it to be extended. This raises new requirements that, in turn, go through the same requirements process. Just as the product evolves on its own, so you may choose to make it evolve by building the early versions with a minimal amount of functionality and later augmenting it by a planned series of releases. The Volere Requirements Process is designed with evolution in mind.

The people around the periphery of the process play an important part in it. These people supply information to the process or receive information from it. They are some of the *stakeholders*—the people who have an interest in the product, but not necessarily a financial one. They participate in the requirements process by providing requirements and receiving deliverables from the process. Additionally, some stakeholders do not show up on Figure 2.1—the consultants and other interested parties who have knowledge needed to gather the requirements for the product. As we discuss the requirements process throughout this book, we also discuss the different stakeholder roles and responsibilities.

The Process

The requirements process is not applicable just to new products.

The requirements process is not applicable just to new products you are developing from the ground up. Most product development that is done today is aimed at maintaining or enhancing an existing product or at making a major overhaul to an existing product or suite of products. A lot of today's development involves commercial off-the-shelf (COTS) products, open source products, or other types of componentware. Whatever your development method, understanding the requirements for the final outcome is still necessary.

Let's look briefly at each of the activities. Subsequent chapters cover them in more detail. The intention of this chapter is to give you a gentle introduction to the process, its components, its deliverables, and the ways that they fit together. If you want more detail on any of the activities, feel free to jump to the relevant chapter before completing this overview.

As we go through the process, we describe it as if you were working with a brand-new product—that is, starting from scratch. We take this tack simply to avoid becoming entangled in the constraints that are part of all maintenance projects. For the latter kind of project, look ahead to Chapter 15, Whither Requirements?, where we discuss projects that are already under way and projects where the requirements are changing.

A Case Study

We shall explain the Volere Requirements Process by talking you through a project that uses it: The IceBreaker project is to develop a product that predicts when and where ice will form on roads, and that schedules trucks to treat the roads with de-icing material. The product will enable road authorities to be more accurate with their predictions, schedule road treatments more precisely, and thus make the roads safer. The road authorities also anticipate they can reduce the amount of de-icing materials used.

> *"The likelihood of frost or ice forming is determined by the energy receipt and loss at the road surface. This energy flow is controlled by a number of environmental and meteorological factors (such as exposure, altitude, road construction, traffic, cloud cover, and wind speed). These factors cause significant variation in road surface temperature from time to time and from one location to another. Winter night-time road surface temperatures can vary by over 10 °C across a road network in a county."*
>
> Source: *Vaisala News*

Project Blastoff

Blastoff is also known as "project initiation," "kickoff," "charter," "project launch," and many other things. We use the term "blastoff" to describe what we are trying to achieve—getting the project launched and flying.

Imagine launching a rocket. 10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – 2 – 1 – blastoff! If all it needed was the ability to count backward from ten, then even Andorra[1] would have its own space program. The truth of the matter is that before we get to the final ten seconds of a rocket launch, a lot of preparation has taken place. The rocket has been fueled, the course plotted—in fact, everything that needs to be done before the rocket can be launched.

The blastoff meeting prepares the project and ensures its feasibility before launching the detailed requirements effort. The principal stakeholders—the client, the main users, the lead requirements analyst, technical and business experts, and other people who are crucial to the success of the project—gather to come to a consensus on the crucial project issues. For the IceBreaker project, Saltworks Systems is the developer of the product, and its employees are aiming for worldwide sales. Northumberland County Highways Department has agreed to be the company's first customer, and it is helping with the requirements. Naturally, key Northumberland people are present at the blastoff meeting.

In the blastoff meeting, the principals work together until they have achieved the blastoff's objectives. That is, they gather enough facts to ensure the project has a clearly defined scope and a worthwhile objective, is possible to achieve, and has commitment from the stakeholders.

It is usually more convenient to define the scope of the business problem first. The lead requirements analyst coordinates the group as they come to a consensus on what the scope of the work is—that is, the business area to be studied—and how this work relates to the world around it. The meeting participants draw a context model on a whiteboard to show the functionality included in the work, the items they consider to be outside the scope of the ice forecasting business, and the connections between the work and the outside world. This model is illustrated in Figure 2.2. Later, as the requirements activity proceeds, it will reveal the optimal product to help with this work.

Once they have reached a reasonable agreement on the scope of the business area to be studied, the group identifies the stakeholders. The stakeholders are the people who have an interest in the product, or who have knowledge pertaining to the product, and thus have requirements. The group identifies the various people who have some interest in IceBreaker: the road engineers, the truck depot supervisor, weather forecasting people, road safety experts, ice treatment consultants, and so on. They do so because if they don't identify all of the stakeholders, the requirements analysts won't find all of the requirements. The context diagram usually identifies many of the stakeholders. We look at how this identification occurs in Chapter 3.

FOOTNOTE 1: Andorra is a tiny principality in the Pyrenees mountains between France and Spain. It became famous in the 1960s for having a defense budget of $4.50, a tale that has become the stuff of legend. Today Andorra's defense budget is zero.

Refer to Chapter 3 for a detailed discussion of project blastoff.

The project blastoff deliverables are the first of eight sections of the Volere Requirements Specification Template in Appendix B.

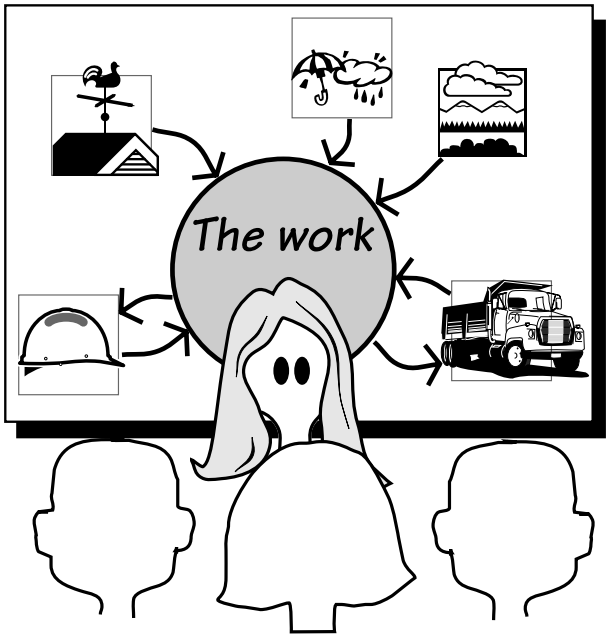


Figure 2.2

The context model is used to build a consensus among the stakeholders as to the scope of the work that needs to be studied. The eventual product is used to do part of this work.

The blastoff also confirms the goals of the project. The blastoff group comes to an agreement on the business reason for doing the project, and it derives a way to measure the advantage the new product will bring. The group also must agree that the product is worthwhile, and that the organization is capable of building and operating it.

It is sensible project management practice at this stage to produce a preliminary estimate of the costs involved for the requirements part of the project. This can be done by using the information contained in the model of the scope of the work. It is also sensible project management to make an early assessment of the risks that the project is likely to face. Although these risks may seem like depressing news, it is always better to get an idea of the downside of the project (its risk and cost) before being swept away by the euphoria of the benefits that the new product is intended to bring.

Finally, the group members arrive at a consensus on whether the project is worthwhile and viable. This is the "go/no go" decision. We know from bitter experience that it is better to cancel a project at an early stage than to have it stagger on for months or years consuming valuable resources with no chance of success. The group must carefully consider whether the product is viable and whether its benefits outweigh its costs.

Alternatively, if many unknowns remain at this point, the blastoff group may decide to start the requirements investigation. It can then review the requirements in a month or so and reassess the value of the project.

The relevant part of the Volere Requirements Process model (Appendix A) is Project Blastoff (Diagram 1).

It is always better to get an idea of the downside of the project (its risk and cost) before being swept away by the euphoria of the benefits that the new product is intended to bring.

READING

DeMarco, Tom, and Tim Lister. *Waltzing with Bears: Managing Risk on Software Projects.* Dorset House, 2003.

Yourdon, Ed. *Death March* (second edition). Prentice Hall, 2003.

Trawling for Requirements

Refer to Chapter 4 for a detailed discussion of business events and use cases, and an exploration of how to use them.

Refer to Chapter 5 for details of the trawling activity.

Once the blastoff is completed, the requirements analysts start trawling for requirements. They learn the work being done by the business area identified by the blastoff. For convenience and consistency, they partition the work context diagram into business use cases. Each business use case is the functionality needed by the work to make the correct response to a business event. A requirements analyst is assigned to each of the business use cases (the analysts can work almost independently of one another) for further detailed study. The analysts use techniques such as apprenticing, scenarios, and use case workshops, among many others, to discover the true nature of the work. These techniques are described in Chapter 5, Trawling for Requirements, and are favored because they involve the stakeholders closely in capturing their requirements. Once they understand the work, the requirements analysts work with the stakeholders to decide the best product to help with this work. That is, they determine how much of the work to automate or change, and what effect those decisions will have on the work. Once they know the extent of the product, the requirements analysts write its requirements. See Figure 2.3.

When they are trawling to discover the requirements, the analytical team members sit with the hands-on users as they describe the work that they do and the work that they hope to do. Some of the team members act as apprentices to the users: They learn how to do the work and, along the way, develop ideas about how improve it. The requirements analysts also consult with other interested stakeholders—usability people, security, operations, and so on—to discover other requirements for the eventual product.

Perhaps the hardest part of requirements gathering is discovering the

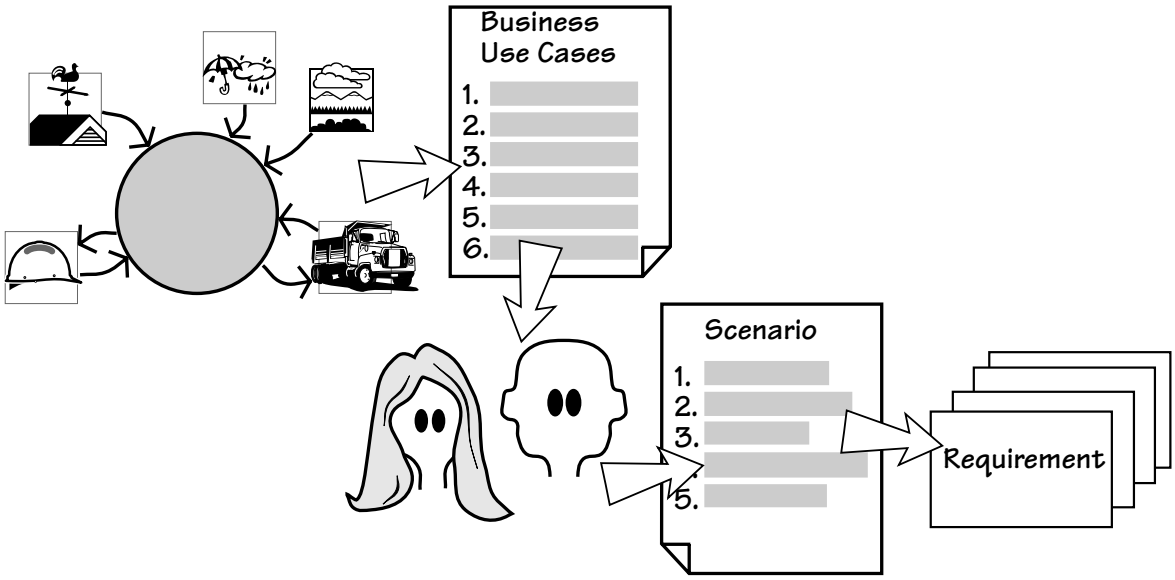


Figure 2.3

The blastoff determines the scope of the work to be studied. The business use cases can be formally derived from the scope. Each of the business use cases is studied by the requirements analysts and the relevant stakeholders to discover the desired way of working. When this is understood, the appropriate product can be determined and requirements written for it.

essence of the system. Many stakeholders inevitably talk about their perceived *solution* to the problem. The essence, by contrast, is the underlying business reason for having the product. Alternatively, you can think of it as the *policy* of the work, or what the work would be if there were no technology (that includes people). We will have more to say about essence in Chapter 5, Trawling for Requirements.

The IceBreaker product must not be simply the automation of the procedures that are done at the moment. To deliver a truly useful product, the analytical team should help to invent a better way to do the work, and build a product that helps with this better way of working. With this goal in mind, they hold creativity workshops where the team members use creative thinking techniques and creative triggers to generate new and better ideas for the eventual product.

READING

Robertson, Suzanne, and James Robertson. *Requirements-Led Project Management.* Addison-Wesley, 2005.

Maiden, Neil, Suzanne Robertson, Sharon Manning, and John Greenwood. *Integrating Creativity Workshops into Structured Requirements Processes.* Proceedings of DIS 2004, Cambridge, Mass., ACM Press.

Prototyping the Requirements

We look at innovative products in Chapter 5, Trawling for Requirements.

Sometimes requirements analysts get stuck. Sometimes there are requirements that are not properly formed, or the user can't explain them, or the requirements analysts can't understand them. Or maybe the product is so groundbreaking that nobody really knows the requirements. Or the analysts and stakeholders just need to work with something more concrete than a written requirement. This is when prototypes are the most effective requirements technique.

A *prototype* is a quick and dirty representation of a potential product—probably only part of the product. It is intended to present the user with some kind of simulation of the requirements. There are two approaches to building requirements prototypes: High-fidelity prototypes use specialized software tools and result in a partially working piece of software, and low-fidelity prototypes use pencil and paper, whiteboards, or some other familiar means, as shown in Figure 2.4. Teams generally like using the low-fidelity prototypes because they can generate them quickly and the users enjoy the spontaneous nature and inventiveness of these prototypes.

Prototyping as a requirements-gathering technique is fully explained in Chapter 12, Prototyping the Requirements.

This part of the process is shown in detailed model form in Prototype the Requirements (Diagram 5) in the Volere Requirements Process model in Appendix A.

Scenarios

Scenarios are stories. The analysts use them when working with stakeholders to arrive at an understanding of the functionality of a use case. The scenario shows, step by step, how a business use case or a product use case is performed. The analysts find that scenarios are a useful neutral language for talking to the relevant stakeholders about the use cases. The scenarios, once they are agreed upon, form the foundation for the requirements.

Refer to Chapter 6, Scenarios and Requirements, for a detailed discussion of scenario modeling.

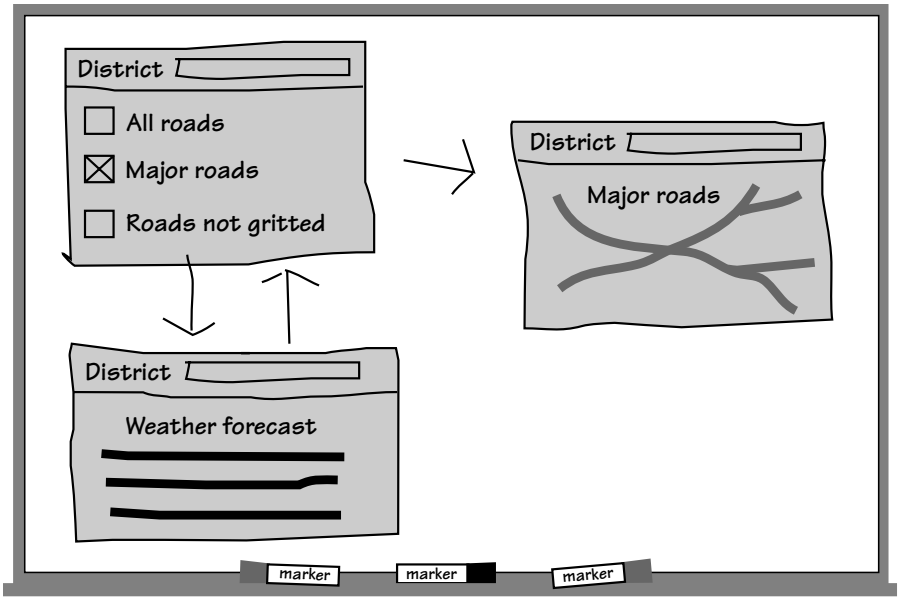


Figure 2.4

A low-fidelity prototype built on a whiteboard to provide a quick visual explanation of some of the requirements, and to elicit misunderstood or missing requirements.

Writing the Requirements

A major problem in system development is misunderstood requirements. To avoid this dilemma, the analysts must write their requirements in a testable manner and ensure that the originating stakeholder understands and agrees with the written requirement before it is passed downstream. In other words, the analysts are writing the requirements to ensure that all parties have achieved the identical understanding of what is needed. Although the task of writing down the requirements may seem an onerous burden, we have found it is the only way to ensure that the essence of the requirement has been captured and communicated, and that the delivered product can be tested. (See Figure 2.5.)

The requirements analysts are writing for the stakeholders. That is, the requirements are the business requirements, and they must be written using business language so that the nontechnical stakeholders can understand them and verify their correctness. Of course, the requirements also need to be written so that the product designers and other technicians can build precisely what the client wants. To ensure this correctness, and to make the requirement testable, the analysts add a *fit criterion* to each requirement. A fit criterion is a quantification or measurement of the requirement so the testers can determine precisely whether an implementation meets—in other words, fits—the requirement.

Chapter 9 describes fit criteria in detail.

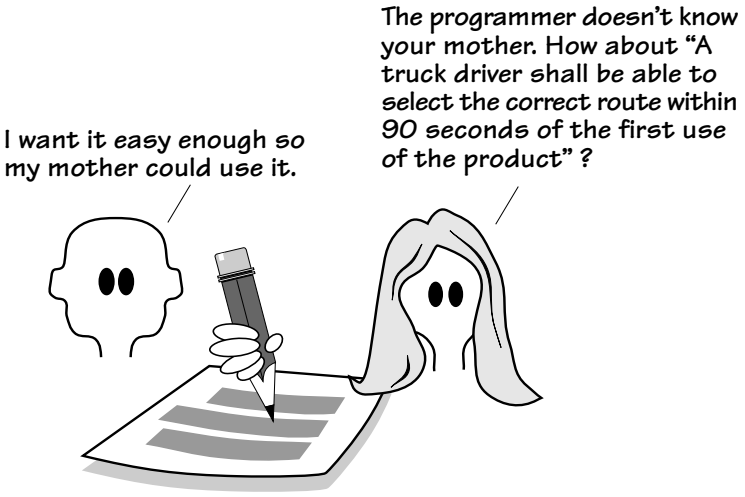


Figure 2.5

The requirements are captured in written form so as to communicate effectively between the stakeholders and the analysts. Only by writing them down can the team ensure that the required product is built.

The analysts use two devices to make it easier to write a specification. The first device, the requirements specification template, is an outline of a requirements specification. The analysts use it as a checklist of which requirements they should be asking for and as a guide to writing their specification documents. The second device is a shell, also known as a snow card. Each requirement is made up of a number of components, and the shell is a convenient layout for ensuring that each requirement has the correct constituents.

See Appendix B, the Volere Requirements Specification Template.

Of course, the writing activity is not really a separate activity. In reality, it is integrated with the activities that surround it—trawling, prototyping, and the Quality Gateway. However, for the purposes of understanding what is involved in getting the correct requirements in a communicable form, we have chosen to look at it separately.

Refer to Chapter 10 for a detailed discussion of writing the requirements.

An alternative to writing functional requirements is building models. Numerous kinds of models are available, and we do not intend this book to describe how to build all of them. While we encourage the use of models for requirements work, we must issue a caution about the tendency of some modelers, and some models, to leap straight into a solution without firstly demonstrating an understanding of the problem. Also bear in mind that models do not specify the nonfunctional requirements. As a result, any models you build must be augmented by written requirements for the nonfunctional requirements.

Lastly, we must consider the primary reason for wanting written requirements. The point is not to *have* written requirements (although that is often necessary), but rather to *write* them. The act of writing the requirement, together with its associated fit criterion, means the analyst has to correctly

understand the requirement. If the requirement is not correctly understood, and agreed to by the relevant stakeholders, then any attempt to write it will result in a nonsense—one that is quickly detected when the requirement reaches the Quality Gateway.

The Quality Gateway

Requirements are the foundation for all that is to follow in the product development cycle. It therefore stands to reason that the requirements must be correct before they are given to the designers/developers. The Quality Gateway (Figure 2.6) tests the requirements. It is a single point that every requirement must pass through before it can become a part of the specification. Quality Gateways are normally set up so that one or two people, probably the lead requirements analyst and a tester, are the only people authorized to pass requirements through the gateway. Working together, they check each requirement for completeness, relevance, testability, coherency, traceability, and several other qualities before they allow it to become part of the specification.

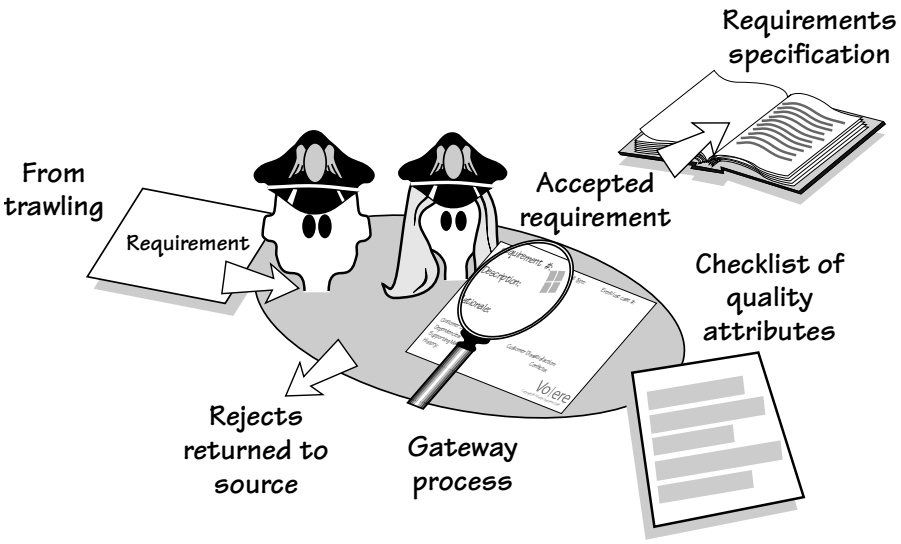
The Quality Gateway is detailed in Diagram 4 of the Volere Requirements Process model in Appendix A.

One of the tasks of the Quality Gateway is to ensure that each requirement has a fit criterion attached to it. The fit criterion is a measurement of the requirement that makes it both understandable and testable. The understandability is for the benefit of the client, who has on several occasions said, "I am not going to have any requirements that I do not understand, nor will I have any that are not useful or that don't contribute to my work. I want to understand the contributions that they make. That's why I want to measure each one."

We discuss measurements for requirements in Chapter 9, Fit Criteria.

Figure 2.6

The Quality Gateway ensures a rigorous specification by testing each requirement for completeness, correctness, measurability, absence of ambiguity, and several other attributes before allowing the requirement to be added to the specification.



The requirements analyst has a different, but complementary reason for measuring and testing requirements: "I need to ensure that each requirement is unambiguous; that is, it must have the same meaning to both the client and the developer. I also need to measure the requirement against the client's expectations. If I can't put a measurement to it, then I can never tell if we are building the product the client really needs."

Chapter 11 describes how the Quality Gateway tests the requirements for these qualities.

Another reason the project has a Quality Gateway is to prevent *requirements leakage*. Just as water seeps into a leaky rowing boat and you cannot tell where it is coming from, requirements sometimes seem to leak into the specification without anyone really knowing where they came from or what value they add to the product. By ensuring that the only way for requirements to get into the specification is through the Quality Gateway, the project team is in control of the requirements, and not the other way around.

Reusing Requirements

Requirements for any product you build are never completely unique. We suggest that before starting on any new requirements project, you go through the specifications written for previous projects and look for potentially reusable material. Sometimes you may find dozens of requirements you can reuse without alteration. More often you will find requirements that, although they are not exactly what you want, are suitable as the basis for some of the requirements you will write in the new project.

For example, in the IceBreaker project, the rules for road engineering do not change between products, so the requirements analysts do not have to rediscover them. They also know that the business of vehicle scheduling does not radically change every year, so their trawling process can take advantage of some requirements from previous projects. Similarly, on many projects within an organization, the nonfunctional requirements are fairly standard, so analysts can start with a specification from one of the previous projects and use it as a checklist.

See Chapter 13 for more on reusing requirements.

The point about reusing requirements is that once a requirement has been successfully specified for a product, and the product itself is successful, the requirement does not have to be reinvented. In Chapter 13, we discuss how you can take advantage of the knowledge that already exists within your organization and how you can save yourself time by recycling requirements from previous projects.

This topic is the subject of Diagram 7 of the Volere Requirements Process model in Appendix A.

Reviewing the Specification

The Quality Gateway exists to keep bad requirements out of the specification. But it does this one requirement at a time. When you think your requirements specification is complete, you should review it. This final review

See Chapter 14 for more on reviewing the specification.

checks that there are no missing requirements, that all the requirements are consistent with one another, and that any conflicts between the requirements have been resolved. In short, the review confirms that the specification is really complete and suitable so that you can move on to the next stage of development.

This review also offers you an opportunity to reassess the costs and risks of the project. Now that you have a complete specification, you know a lot more about the product than you did at blastoff time. Once the requirements specification is complete, you have a precise knowledge of the scope and functionality of the product, so this is the time to remeasure its size. From that size, and from your knowledge of the project's constraints and solution architecture, you can estimate the cost to construct the product.

You also know at this stage which types of requirements are associated with the greatest risks. For example, the users may have asked for an interface that their organization has not built before. Or perhaps they want to use untried technology to build the product. Does the developer have the people capable of building the product as specified? By reassessing the risks at this point, you give yourself a better chance to build the desired product successfully.

Iterative and Incremental Processes

One common misconception in the requirements world is that you have to gather *all* the requirements before moving on to the next step of design and construction. In some circumstances this is necessary, but not always. On the one hand, if you are outsourcing or if the requirements document forms the basis of a contract, then clearly you need to have a complete requirements specification. On the other hand, providing the overall architecture is known, construction can often begin before all the requirements are gathered. We suggest that you consider this point when working on your own requirements projects.

Let's go back to the IceBreaker project. The developers are ready to start building the product, so right after the blastoff meeting the key stakeholders select a few (let's say three or four) of the highest-priority business use cases. The requirements analysts gather the requirements for only those business use cases, ignoring the remainder for the meantime. It is feasible to ignore them because there is always a minimal functional connection between the business use cases, so the analysts do not interfere with one another's work. Then, when the first tranche of requirements have successfully passed the Quality Gateway, the developers can start their work. The intention is to implement a small number of use cases as early as possible to get the reaction of the stakeholders. If there are to be nasty surprises, then the IceBreaker team members want to get them as early as possible. While the first use cases

are being developed and delivered, the analysts are working on the requirements for the next-highest-priority ones. Soon they have established a rhythm for delivery, with new use cases being delivered every few weeks.

Requirements Retrospective

You are reading this book about a requirements process, presumably with the intention of improving your own process. Retrospectives are one of the most effective tools for discovering the good and bad of a process, and suggesting remedial action. Retrospectives for requirements projects consist of a series of interviews with stakeholders and group sessions with the developers. The intention is to canvas all the people involved in the process and ask tough questions:

- What did we do right?
- What did we do wrong?
- If we had to do it again, what would you do differently?

"If we did the project again tomorrow, what would we do differently?"

By looking for honest answers to these questions, you give yourself the best chance of improving your process. The idea is very simple: Do more of what works and less of what doesn't.

Keep a record of the lessons learned from your retrospective. The next project can then use that record as a starting point, so the lessons learned from previous projects are passed along.

Your retrospective can be very informal: a coffee-time meeting with the project group, or the project leader collecting e-mail messages from the participants. Alternatively, if the stakes are higher, it can be formalized to the point where it is run by an outside facilitator who canvases the participants, both individually and as a group, and publishes a retrospective report.

The most notable feature of retrospectives is this: Companies that regularly conduct retrospectives consistently report significant improvements in their processes. Retrospectives are probably the cheapest investment you can make in your own process.

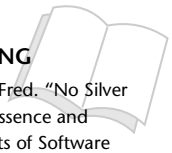
Your Own Requirements Process

The itinerant peddler of quack potions, Doctor Dulcamara, sings the praises of his elixir, which is guaranteed to cure toothache, make you potent, eliminate wrinkles and give you smooth beautiful skin, destroy mice and bugs, and make the object of your affections fall in love with you. The rather fanciful libretto from Donizetti's opera *L'Elisir d'Amore* points out something that, although very obvious, is often disregarded: There is no such thing as the universal cure.

We have distilled our experiences from a wide variety of projects to provide you with a set of foundation activities and deliverables that will apply to any project.

We really would like to be able to present you with a requirements process that has all the attributes of Doctor Dulcamara's elixir—a process that suits all projects for all applications in all organizations. But we know from experience that every project has different needs. At the same time, we have learned that some fundamental principles hold good for any project. Thus, instead of attempting to provide you with a one-size-fits-all magic potion, we have distilled our experiences from a wide variety of projects to provide you with a set of foundation activities and deliverables that will apply to any project.

We are using a process here to describe the things that have to be done to successfully gather requirements, and the deliverables that are the foundation for any kind of requirements activity. As you read this book, think of adapting them to your own culture, your own environment, your own organizational structure, and your own chosen way of product development.



READING

Brooks, Fred. "No Silver Bullet: Essence and Accidents of Software Engineering" and "'No Silver Bullet' Refired." *The Mythical Man-Month: Essays on Software Engineering* (twentieth anniversary edition). Addison-Wesley, 1995.

For instance, projects using eXtreme Programming are not supposed to produce a requirements specification, but there is still a clear need to understand the requirements. This understanding cannot be achieved effectively by writing code. To invest in writing an individual requirement, complete with its fit criterion, remains the fastest way of understanding that requirement. (Writing code is building a *solution* to satisfy the requirement, and it does not guarantee that the real requirement is ever discovered.) In the Volere Requirements Process, we provide scenarios as a way of modeling the functionality of the use case. This is almost always a quicker way to discover requirements, particularly when you start to consider the exceptions and alternatives for a use case. For a nonfunctional requirement, writing it down, complete with its fit criterion, remains the fastest way of understanding it.

Defining the scope of the business area affected by the product is still the most effective way of keeping the requirements and the development work focused. Learning about the work, and not just the product, is the best way of building a relevant product. Of course, we do not intend that you use the Volere process straight out of the box. Instead, we urge you to adopt the most beneficial practices, adapting the process as necessary to make it relevant to your project and your organization.

To adapt this process you need to understand each of the deliverables it produces—the rest of this book will discuss these in detail. Once you understand the content and purpose of each deliverable, ask how each one (provided it is relevant) would best be produced within your project environment using your resources:

- What is the deliverable called within your environment? Use the definitions of the terms used in the generic process model and identify the equivalent deliverable in your organization.

- Is this deliverable relevant for this project?
- How much do you already know about this deliverable? Do you know enough to be able to avoid devoting additional time to it?
- Who produces the deliverable? Understand which parts of the deliverable are produced by whom. Also, when several people are involved, you need to define the interfaces between them.
- When is the deliverable produced? Map your project phases to the generic process.
- Where is the deliverable produced? A generic deliverable is often the result of fragments that are produced in a number of geographical locations. Define the interfaces between the different locations and specify how they will work.
- Who needs to review the deliverable? Look for existing cultural checkpoints within your organization. Do you have recognized stages or phases in your projects when peers, users, or managers must review your specification?

The generic model describes deliverables and procedures for producing them. You decide how to use them.

In Conclusion

We have described—rather briefly—a process for gathering and verifying requirements. The remainder of this book describes the activities in detail, along with their deliverables. Feel free to jump to any chapter that is of immediate concern—we wrote the chapters in more or less the order in which you would arrive at the activities, but you don't have to read them that way.

Keep in kind that the model in Appendix A is a complete record of the Volere Requirements Process.