

17. Requirements Completeness

in which we decide whether our specification is complete, and set the priorities of the requirements

At some stage during your requirements process, you need to release all or part of your specification—other people, such as developers, testers, marketers, and suppliers, need it. To be released, the specification does not have to contain all the requirements: It could be a partial version with the requirements just for the next iteration, a version of the specification you want to publish for marketing reasons, an extract to use with a request for proposal (RFP), or any portion that you release for any other reason. Nevertheless, before releasing the specification, you need to ensure that it is complete for its intended purpose.

We use the term “specification” here to mean whatever collection of requirements you have. This material does not have to be a formally written specification—it does not even have to be formal. It could be a wiki or a set of story cards; it might hold only the requirements for a partial release of the product. Whatever your intention, this review ensures the specification is sufficient before you hand it over to anyone else.

Figure 17.1 illustrates how the Quality Gateway and the Specification Review work together. The Quality Gateway tests an individual requirement—ensuring that it is correctly stated, unambiguous, within scope, testable, traceable, and not gold plating—thereby confirming that only correct atomic requirements are included in the specification.

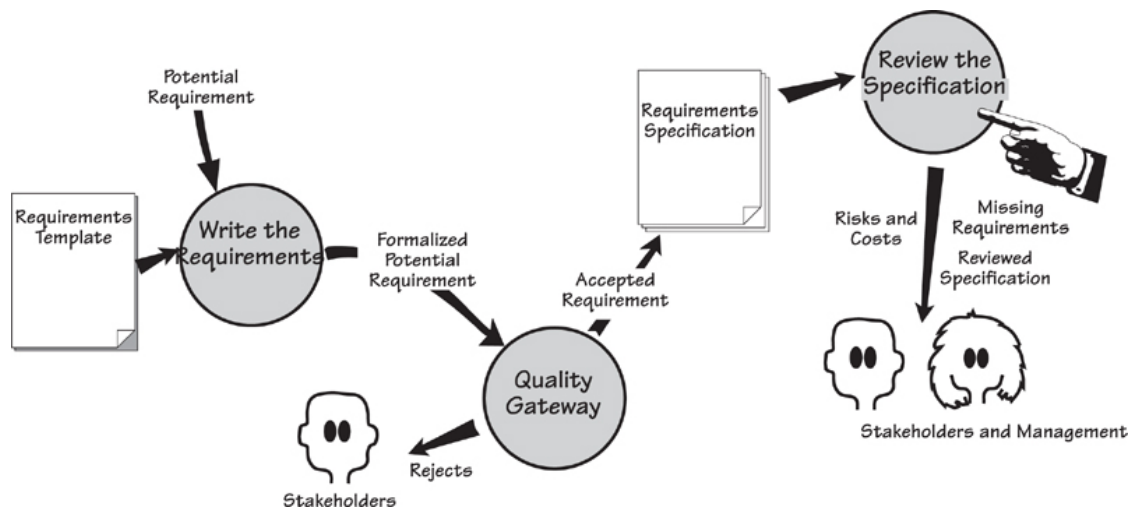


Figure 17.1. You have arrived at the point in the process where you want to consider the specification as a whole. The Quality Gateway has tested and accepted individual requirements, and added them to the specification. Now it is time to assess whether you have a complete specification. This review can be done iteratively—ideally for one product use case worth of requirements at a time.

 **Chapter 11** discusses the Quality Gateway.

Now you have to consider whether the specification is complete, which means reviewing the specification as a whole and ensuring that all the parts that should be there are there. This review makes sense when you consider that it performs these tests:

- Determine whether any requirements are missing.
- Prioritize the requirements so the builders understand their importance and urgency.
- Check for conflicts between requirements that could prevent one or the other from being satisfied.

Additionally, your project management might undertake some other useful tasks at this stage:

- Estimate the cost of construction.

- Evaluate the risks faced by the project.

Reviewing the requirements specification can be done at any time, not just before a release—it can be an ongoing activity. You might, for example, review the specification as a check on the progress of the requirements activity. The quality (or lack thereof) of the specification tells you more about progress than the volume of the specification does.

Formality Guide

Rabbit projects rarely package all of their requirements into a complete specification; instead, they act on each tranche of requirements as it comes along. The review process discussed in this chapter is useful for progressively checking completeness in such projects. Rabbits should look at the sections of this chapter covering non-events and the CRUD check. The section on prioritization, while created with written requirements in mind, is also relevant for rabbit projects.



Horse projects almost always have some kind of written specification. It does not have to be as formal as a specification for an elephant project, but knowing it is complete and relevant is desirable. Horses may not build all of the models we describe here, but that's okay: The review process still works without all of them being present. Horse projects should definitely prioritize their requirements.



Elephant projects always need a complete specification. This is either for statutory reasons or because you are outsourcing the development of the product. If there are statutory demands, then it is incumbent upon you to ensure that you have a complete and correct specification, and in some cases to demonstrate how you confirmed its completeness and correctness. If you are outsourcing and you lack an accurate specification, then

you will probably be disappointed with the end result—the supplier can do no more than build what you ask for.



In this review process we make use of several models, most notably a model of the stored data (UML class model, entity relationship diagram, or your choice of model). Elephant projects almost always make use of models, so here we present another way that you can reap the benefits from such representations.

Reviewing the Specification

This review process is iterative. Finding errors or omissions, and correcting them, could mean introducing new errors; as a consequence, it might be necessary to iterate through this process once or twice to ensure that the specification is watertight. It is useful to maintain a record of the errors you encounter; the types of errors you discover in this review suggest where you need to improve your requirements process.

This review gives you an ideal opportunity to reassess your earlier decision on whether to go ahead with the project. A seriously flawed or incomplete specification, or measurements that say the costs and the risks outweigh the benefits, are almost always indications that you need to consider project euthanasia.

Inspections

One fairly effective way of reviewing the specification is a formalized process called a *Fagan inspection*. Fagan inspections have been around for quite some time, and much has been written about them, so we do not propose to add much to that body of literature here—a brief outline of the process will be sufficient for our purposes.

The inspection process kicks off with a moderator determining the material to be inspected and the inspectors to inspect it. The inspectors are given an overview of the document under consideration, and they have a

day or so to study the material. The inspection meeting proper—limited to two hours—studies the document using checklists of previously found errors. The checklist is applied to the document: “Does this error exist? Does that error exist?” This has proved itself to be a very effective way of trapping errors, yielding a higher detection rate than other review techniques. The checklists are updated when new errors—those not already on the list—are discovered. The author reworks the document, and the moderator ensures all defects have been removed. If necessary, the moderator arranges a follow-up inspection.



The original paper on Fagan inspection (one of the most cited papers in software history) is this: Fagan, Michael. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, vol. 15, no. 3 (1976): 258–287. (We said this process has been around for a while.)

You can easily adopt some of the Fagan rules. Try these:

- Assign a moderator (probably the business analyst) to take responsibility for arranging the inspection and distributing the materials.
- Build a checklist of the most likely errors. You will augment this list with subsequent inspections.
- Give inspectors some time to read the document and prepare for the inspection.
- Limit inspections to two hours and no more than two inspections a day.
- Have between three and eight inspectors.

Fagan inspections can be a very effective weapon to ensure the correctness and completeness of your requirements. Try them.

Find Missing Requirements

The review determines whether all of the requirement types appropriate to your product have been discovered. Use the Volere Requirements Specification Template and its requirement types as a guide when determining whether your specification contains the types of requirements called for by the nature of the product. For example, if you are developing a financial product but you have no security requirements, something is definitely missing. Similarly, a Web product that lacks either usability or look and feel requirements is certainly in trouble.



The Volere Requirements Specification Template appears in [Appendix A](#).

The functional requirements should be sufficient to complete the work of each use case. To check this aspect, play through each of the product use cases as if you were the product. If you do everything the requirements call for, do you arrive at the outcome for the use case? Are your users (you should have them with you when you perform this role-play) satisfied the product will do what they need for their work?

Look for exceptions to the normal things the product must do. Have you generated enough exception and alternative scenarios to cover these eventualities, and do your functional requirements reflect this coverage? Revisit your scenarios and, for each step, determine whether exceptions can occur there or whether an exception might prevent that step from being reached.



Scenarios are discussed in [Chapter 6](#).

Check each product use case against the non-functional requirement types. Does it have the non-functional requirements that it needs and that are appropriate for this kind of use case? Use the requirements template as a checklist. Go through the non-functional requirement types, read

their descriptions, and ensure that the correct non-functional requirements have been included.

Have All Business Use Cases Been Discovered?

For each business event, you determined the business response (the business use case [BUC]) and decided how much of that response will be carried out by the product (the product use case [PUC]). We suggested that you discover the requirements one PUC at a time, and continue until you have covered all of the business events. This approach works well, providing, of course, that you have discovered all of the business events.

You do not have to produce more documents to do the review, but just make more use of what you have.

So how do you know whether you have discovered all of the business events? There is a short procedure that uses the outputs of your requirements process and system modeling. In other words, you do not have to produce more stuff, but just make more use of what you already have.

Figure 17.2 illustrates the procedure for confirming the completeness of your list of business events. Refer to the model of this procedure as we describe its activities.

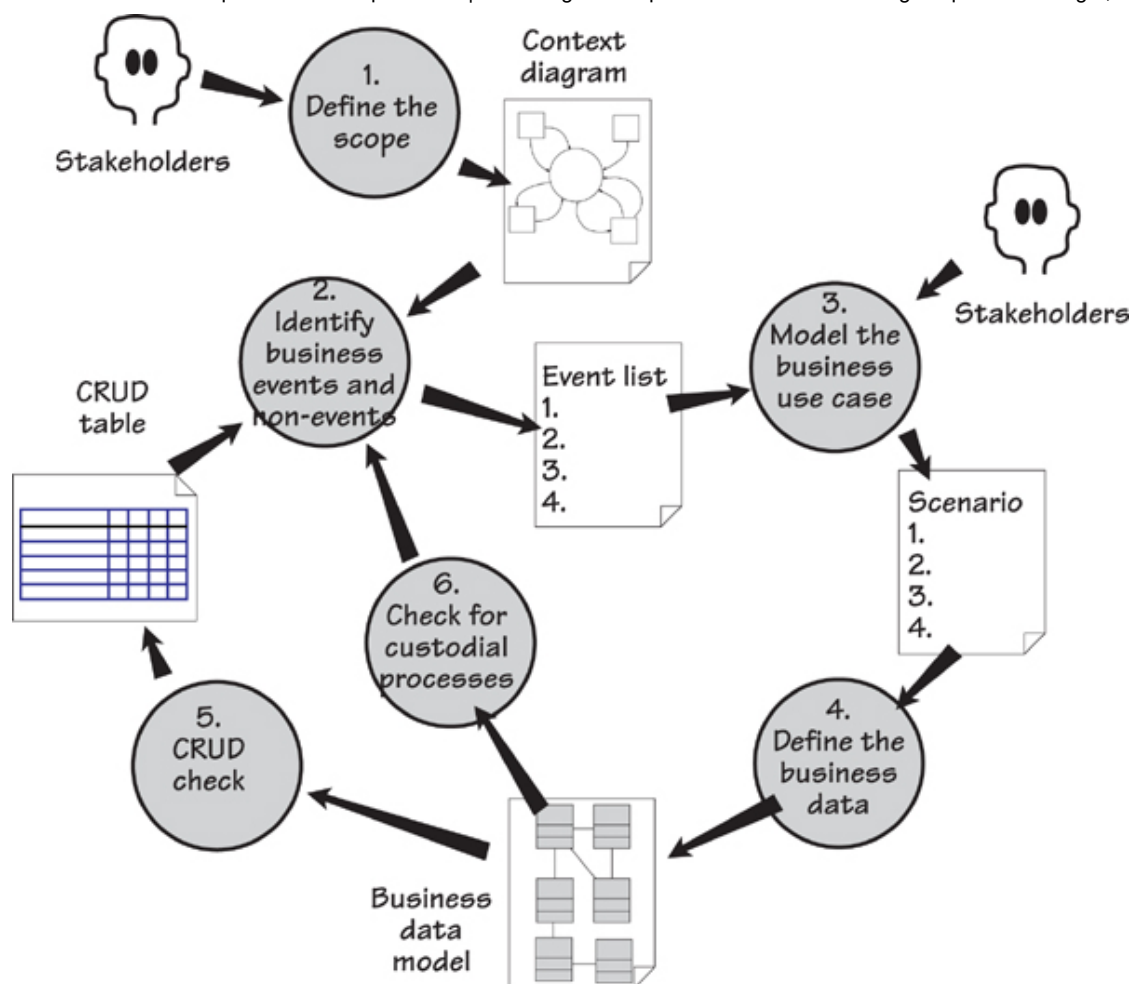


Figure 17.2. The procedure for determining that you have found all of the business events. The process is iterative, going through the activities until the *Identify business events and non-events* activity fails to discover anything new.

1. Define the Scope

The scope we are concerned with here is the scope of the work to be studied. In [Chapter 3](#), *Scoping the Business Problem*, we built a context model to show the scope of the IceBreaker work, and we reproduce it in [Figure 17.3](#). The context model is mainly completed during the blastoff activity, then refined further as requirements discovery progresses. The review process we describe here checks the completeness of your context diagram and, where necessary, updates it.

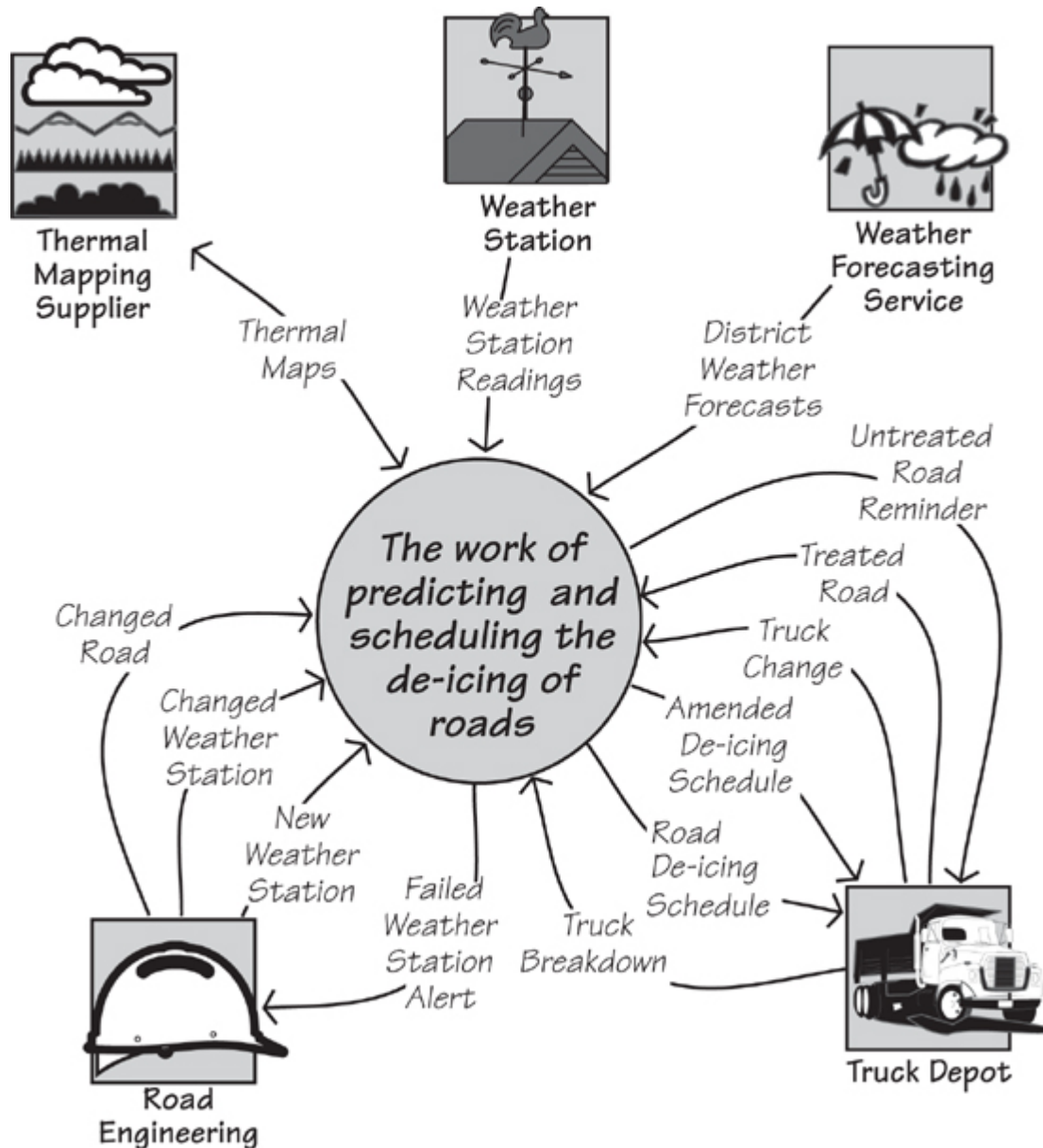



Figure 17.3. The context diagram of the work shows the data entering and leaving the scope of the work. These data transfers are referred to as boundary data flows. We use these flows to determine the business events.

2. Identify Business Events and Non-events

During the blastoff, or at the beginning of trawling, you determined the business events by looking at the boundary data flows on the context diagram. If a business event happens outside the work, the adjacent system sends a data flow (which we are calling a boundary data flow) to the work, and the work responds by processing the data contained in the flow. Thus a business event is associated with each incoming boundary data flow. The outgoing boundary flows are either part of the response to an externally triggered business event (the work responds by processing the incoming flow and produces the outgoing flow) or the result of a time-

triggered event (such as reporting and sending out reminders or alerts). In short, each flow on the context model is connected to a business event. When you have found all of the boundary flows from the context model, you have determined all of the possible business events . . . for the moment.

 **Chapter 4**, Business Use Cases, tells you how to determine business events from the context model.

The output of this activity is a list of business events. The list of the IceBreaker business events appears in **Table 17.1**.

Table 17.1. The Business Event List for the IceBreaker Work

Event Name	Input Data Flow	Output Data Flow
1. Weather Station transmits reading	Weather Station Readings	
2. Weather Bureau forecasts weather	District Weather Forecasts	
3. Road Engineering advises changed roads	Changed Road	
4. Road Engineering installs new Weather Station	New Weather Station	
5. Road Engineering changes Weather Station	Changed Weather Station	
6. Time to test Weather Stations		Failed Weather Station Alert
7. Truck Depot changes a truck	Truck Change	
8. Time to detect icy roads		Road De-icing Schedule
9. Truck treats a road	Treated Road	
10. Truck Depot reports problem with truck	Truck Breakdown	Amended De-icing Schedule
11. Time to monitor road de-icing		Untreated Road Reminder

Non-events

Now look for non-events. The term “non-events” is a play on words, and here we use it to mean events that happen if another event does not hap-

pen. **Table 17.1** lists Event 9, *Truck treats a road*. What happens if the truck does *not* treat the road? The work has to do something, and what it does is a non-event (it happened because another event did not happen), so now we have identified Event 11, *Time to monitor road de-icing*. The work responds to this (non) event by checking whether all roads have been treated as directed and, if they have not, issues an *Untreated Road Reminder*.

A more common example found in many businesses is the business event called “Customer pays invoice.” You are no doubt familiar with this event, probably by being the payer rather than the payee. So what happens if the customer does not pay the invoice? There is a corresponding non-event called “Time to send reminder notice to nonpayers.”

During your specification review, go through your list of business events and ask, “What happens if this event does not happen?” Not all business events have a non-event—most don’t—but checking the list for their existence will reveal missing events.

Add any new business events discovered through this exercise to the list of business events, and update the context model with the appropriate flows. Continue searching the list of existing events for more non-events, but don’t be overly concerned if you don’t find one for every event. Often when you ask the question, “What happens if this event does not happen?”, the answer is “Nothing.”

3. Model the Business Use Case

Activity 3 (**Figure 17.2**) is not part of the requirements review, but something you have already done. As part of your requirements investigation, you will have built models or scenarios to help you and your stakeholders understand the desired response to the business event. Given that scenarios are the most commonly used models, we show a scenario in the schematic of the review process. However, many business analysts prefer to use UML activity diagrams or similar forms. The type of model you use is not important. What is important is that your model shows the functionality of the business use case; from this information, you can determine the stored data used by that functionality.



Reading

Robertson, James, and Suzanne Robertson. *Complete Systems Analysis: The Workbook, the Textbook, the Answers*. Dorset House, 1998. This book is a thorough treatment of business use case modeling.

4. Define the Business Data

The next part of the review process is a step you might have already performed: building a model of the stored data needed by the work. You can use a class diagram, an entity relationship model, a relational model, or any other data-modeling notation you prefer. As long as it shows classes, entities, or tables, and the associations or relationships between them, it will suffice.

Figure 17.4 shows a sample model of the data used by the IceBreaker work.

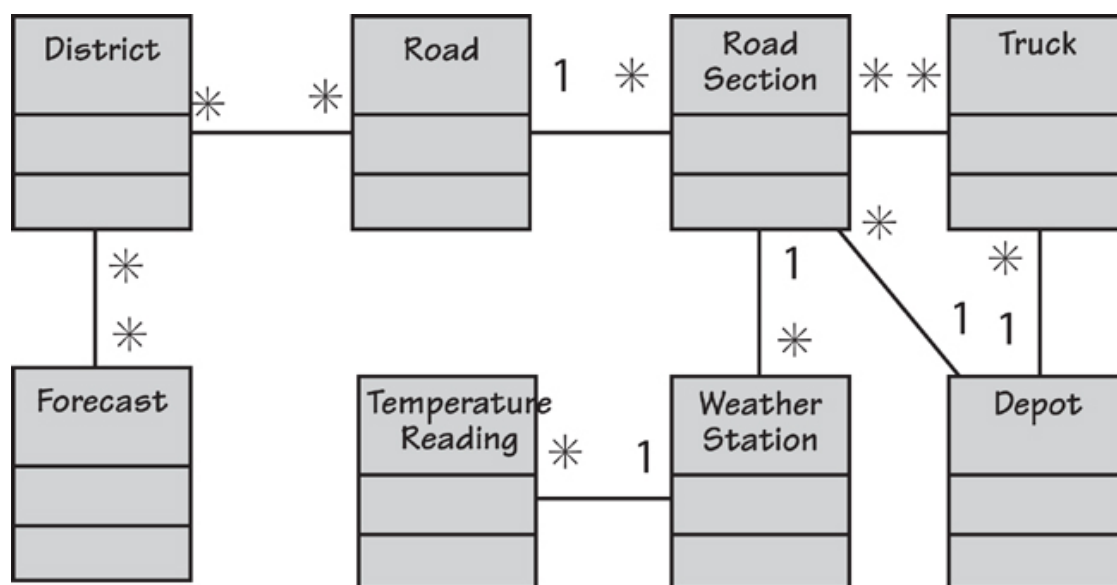


Figure 17.4. This model shows the stored data used to predict and schedule the de-icing of roads. It uses UML class model notation.

If building this kind of model is a task you do not normally tackle, ask one of the database people to do it for you. They have to build such a model at some stage of the development, and they may as well do it now when it can serve more purposes than just aiding the logical database design.

However, you must insist that the database person builds a model of the *business data*, and does not start designing a database; these are two different things.

If you don't want to build a model of the stored data, a simple alternative is to make a list of the classes of data used by the business. Data classes (also called "entities") are the subject matter of stored data—they are the things that we store data about. You can think of a class as a collection of elementary data items (their correct name is "attributes") for something that is important to the business. The "something" can be real, such as a customer or goods you sell, or it can be abstract, such as an account, a contract, or an invoice. The important thing to note is that the class does not have an alphanumeric value. For example, an account has no alphanumeric value, but its attributes—the account number, the account balance, and so on—are items that do have alphanumeric values. This is a useful rule for those times when you are wondering whether something is a class or an attribute.

Classes are uniquely identified.

Another, perhaps more useful rule is that classes are uniquely identified. Thus anything to which your organization attaches an identifier is a class—accounts, credit cards, cars, shipments, flights, and so on—and all have unique identifiers.

Do not agonize too long over identifying the data classes—just do as well as you can without spending the rest of the month on it. Some heuristics are generally helpful in identifying classes by defining their properties:

- Things, concrete or abstract, used by the business
- Things that are identified—accounts, sales opportunities, customers
- Subjects of data, not the data itself
- Nouns with a defined business purpose
- Products or services—mortgages, service agreements

- Branches of organizations, locations, or constructions
- Roles—case officer, employee, manager
- Events that are remembered—agreements, contracts, purchases
- Adjacent systems from the context diagram

You can also find the classes from your context model. The stored data used by the work comes in via data flows and leaves via other data flows. Think about it this way: If data resides inside the work, there must be some flow of data to bring it there. In turn, you can dissect the data flows found on the context model and look at their attributes: “What is this attribute describing?” or “What is the subject of this data?” These subjects are your classes. Analyze all boundary flows, both inward and outward, looking for “things” that conform to the properties of classes listed earlier. When you have analyzed all of the flows, you have most likely identified all of the business data classes.

Now we get to the fun part.

5. CRUD Check

Each class of data (check these on your class model) must be Created and Referenced. Some are also Updated or Deleted. Build a CRUD table, such as the one shown in **Table 17.2**, to show whether every class has all the appropriate actions performed on it. These actions are performed by business use cases, so this step is where you reveal any missing events.

Table 17.2. The CRUD Table

Each cell shows the identifier of the business event that creates, references, updates, or deletes the entity. Gaps in the table indicate missing events.

Class	Create	Reference	Update	Delete
Depot		7		
District		2, 8, 10		
Forecast	2	8, 10		
Road	3	4, 8, 9, 10	3	
Road Section	3	4, 8, 10, 11	3, 9	
Temperature Reading	1	6, 8, 10		
Truck	7	8, 9, 11	7, 10	
Weather Station	4	1, 6	5	

***Each class of data must be Created and Referenced.
Some are also Updated or Deleted.***

If a class is referenced without first being created, it means the creation event is missing. If a class is created without being referenced, then either an event is missing or some data has been created that does not have to be created. Some classes (but not all) are updated, and some are deleted. Naturally, for either of these last two events to occur, the data must have been created.

Empty cells in the CRUD table reveal missing business events. For example, the classes Depot and District do not have any creating business event, yet they are referenced. Thus the context is incomplete: It does not have the incoming flows needed to create these classes of stored data.

When you find missing business events, you have to revisit your stakeholders to find out more about them. When you determine what the events are, update the context diagram, add them to the event list, update the CRUD table, and continue the process.

The Delete column of the CRUD table shows classes that are deleted for business policy reasons only. This is not the same as archiving or cleaning

up the database. For example, if a Depot were to be taken out of service, then it is deleted from a business point of view. However, a Forecast is never deleted—there is no essential business policy reason for doing so.

6. Check for Custodial Processes

The work's processes can be fundamental or custodial. *Fundamental processes* are connected to the reason for the product's existence—for example, analyzing the roads, recording the weather forecasts, and scheduling the trucks to treat the roads. In contrast, *custodial processes* exist to maintain—that is, keep custody of—the stored data. These processes make changes to the data solely to keep it up-to-date and are not part of the fundamental processing.

For example, when you hand over your credit card to make a purchase, the credit card company records the amount and other details of your purchase. That is a fundamental process.

If a class has changeable attributes, there is probably a custodial business event to change them.

Now imagine that you move to a new home. After doing so, you advise the credit card company of your change of address, and the company updates its records accordingly. This is a custodial process; it exists just to keep the data up-to-date.

To check for custodial processes, go through the class model and the CRUD table, and ensure that you have sufficient business events and their processes to maintain all of the work's stored data. If a class has changeable attributes, then there is probably a custodial business event to change them.

Repeat Until Done

Activities 1 through 6 of the business event discovery process are iterative. That is, you continue to go through the process—identifying business events; modeling the business use cases; adding to the class model; checking that the classes are created, referenced, updated, and deleted—until

activity 2, *Identify business events and non-events*, fails to reveal any new events. At that point, you can be confident that there are no more business events relevant to your work.


You might also investigate the automated tools at your disposal, as some of this procedure can be automated. It is not that hard to do manually, but if you can get some automated help, why not?

Prioritizing the Requirements

One problem with requirements is that there are always too many of them. Prioritizing gives you a way to choose which ones to implement in which versions of the product. Decisions about prioritization are complex because they involve different factors and these factors are often in conflict with each other. Also, because the various stakeholders probably have different goals, it may prove difficult to reach agreement about priorities.

Despite its difficulty, this task must be done sooner or later, and sooner is best: The earlier you prioritize, the easier it is. But let's go back to being difficult—there are quite a few factors that are considered when prioritizing. We spoke earlier about the customer satisfaction and customer dissatisfaction.

Each requirement should carry a customer satisfaction and customer dissatisfaction rating. These ratings help the customer to consider the relative value of individual requirements, and to prioritize them.

 See **Chapter 16**, Communicating Requirements, for more on customer satisfaction and dissatisfaction.

You can group requirements together and prioritize them as a unit.

To prioritize requirements, you can group them together into logical (to you) groups. These groups are then prioritized as a unit, on the assumption that all of the composing requirements have the same priority as the group as a whole. A group might be a use case, a component, a feature, or any other collection of requirements that it makes sense to prioritize as a unit instead of treating them individually.

To make it easier to read, for the next few pages, we use the term “requirements” to mean “groups of requirements,” “features,” “product use cases,” or any other grouping you care to use.

Prioritization Factors

The following factors commonly affect prioritization decisions:

- The cost of implementation
- Value to the customer or client
- Time needed to implement the product
- Ease of technological implementation
- Ease of business or organization implementation
- Benefit to the business
- Obligation to obey the law

Not all of these factors are relevant to every project, and the relative importance of each factor differs for each project. Within a project, the relative importance of the factors is not the same for all of the stakeholders. Given this combinatorial complexity, you need some kind of agreed-upon prioritization procedure to provide a way of making choices. Part of that procedure is to determine when you will make prioritization decisions.

When to Prioritize

How soon should you make choices? As soon as you have two items to choose between. And keep in mind that the more visible you make your requirements knowledge, the more chances you have to make, and help others make, informed choices.

If your requirements have a well-organized structure, you can prioritize them early in your project. The process described in this book includes a project blastoff (**Chapter 3**) that advocates building a work context model, and then partitioning it using business events. We strongly suggest that you assign a priority rating to each business use case during the blastoff. You can, if you like, attach customer satisfaction and dissatisfaction ratings to the business events at this time. This early prioritization indicates which parts of the business should be investigated first, and which can be safely ignored until later, or in some cases, abandoned. In addition, you use this first prioritization to guide your iterations and version planning.

As you write atomic requirements, you should progressively consider whether to prioritize them. If any requirements obviously have low value, then tag them as such. Use the customer satisfaction and customer dissatisfaction ratings, discussed in the previous section, to help other people make choices.

Your stakeholders often assume the term “requirements” means these capabilities will definitely be implemented. “Requirements” are really desires or wishes that we need to understand well enough to decide whether and how to implement them.

Part of the reason for progressive prioritization is to manage expectations. Your stakeholders often assume the term “requirements” means these capabilities will definitely be implemented. “Requirements” are really desires or wishes that we need to understand well enough to decide whether and how to implement them. For example, we might have a requirement that is really high priority but, due to a mixture of constraints,

we cannot meet its fit criterion 100 percent. However, we do have a solution that will meet the fit criterion at the 85 percent level.

If you have been progressively prioritizing requirements throughout the project, people are able to accept such compromises without feeling cheated. Prioritization prepares stakeholders for the fact you cannot implement all the requirements.

Requirement Priority Grading

You can grade your requirement priority however it suits your way of working. A common way of grading requirements is “high,” “medium,” and “low,” but this approach usually means that all requirements are somehow high priority. There is also the MoSCoW approach, which is popular; this acronym stands for Must have, Should have, Could have, Won’t have.

Some organizations assign their requirements to releases: R1, R2, R3, and so on. The idea is that the R1 requirements are the highest priority or have the highest customer appeal, and are intended to appear in the first release. But having assigned your requirements to releases, suppose you discover you have too many requirements in the R1 category. At that point, you need to prioritize further.



Reading

Davis, Al. *Just Enough Requirements*. Dorset House, 2005.

The idea of sorting the requirements into prioritization categories is often referred to as *triage*. This term (from the French verb *trier*, meaning “to sort”) comes from the field of medicine. It was first adopted during the Napoleonic wars when field hospitals were not capable of treating all soldiers who had been wounded. The doctors used triage to place the patients into one of three categories:

- Those who would live without treatment

- Those who would not survive
- Those who would survive if they were treated

Due to scarce medical resources, the doctors treated the third group only. The idea of triage can be used in project work using the categories:

- Those requirements needed for the next release
- Those requirements definitely not needed or wanted for the next release
- Those requirements you would like if possible

If the first and last categories leave you with more requirements than will fit into your budget, you need to prioritize further.

Prioritization Spreadsheet

A prioritization spreadsheet ([Figure 17.5](#)) enables you to prioritize the overflow requirements. Ideally—and especially if you have done a good job on progressive prioritization—these requirements will fit into the “would like if possible” category.

Volere Prioritisation Spreadsheet										
Copyright © The Atlantic Systems Guild 2006										
Requirement/Product Use Case/Feature	Number	Factor - score out of 10 Value to Customer	%Weight applied 40	Factor - score out of 10 Value to Business	%Weight applied 20	Factor - score out of 10 Minimise Implementation Cost	%Weight applied 10	Factor - score out of 10 Ease of Implementation	%Weight applied 30	Priority Rating
Requirement 1	1	2	0.8	7	1.4	3	0.3	8	2.4	4.9
Requirement 2	2	8	3.2	8	1.6	5	0.5	7	2.1	7.4
Requirement 3	3	7	2.8	3	0.6	7	0.7	4	1.2	5.3
Requirement 4	4	6	2.4	8	1.6	3	0.3	5	1.5	5.8
Requirement 5	5	5	2	5	1	1	0.1	3	0.9	4
Requirement 6	6	9	4	6	1.2	6	0.6	5	1.5	6.9
Requirement 7	7	4	2	3	0.6	6	0.6	7	2.1	4.9

Figure 17.5. This prioritization spreadsheet can be downloaded from www.volere.co.uk.



The downloadable Volere Prioritization Spreadsheet (www.volere.co.uk) offers a way to prioritize requirements. This spreadsheet contains some examples that you can replace with your own data.

Earlier in this chapter, we identified seven prioritization factors (or you may use any other prioritization factors relevant to your project). On our spreadsheet (see [Figure 17.5](#)), we have limited the number of factors to four, as more than that makes it difficult, if not impossible, to agree on a weighting system.

The *% Weight Applied* column shows the relative importance assigned to each factor. You arrive at this percentage weight by stakeholder discussion and voting.

In column 1, list the requirements you want to prioritize. These might be atomic requirements or recognized groups of requirements. Give each requirement–factor combination a score out of 10. This score reflects the positive contribution to the factor made by this requirement, where 1 means no contribution and 10 means the maximum possible contribution. In the example, for requirement 1, we assigned a score of 2 for the first factor because we believe that it does not make a very positive contribution to *Value to Customer*. The same requirement scores a 7 for *Value to Business*, as it makes a significant contribution to the business. The score for *Minimizing the Cost of Implementation* is 3; we think this requirement is relatively expensive to implement. It scored an 8 in terms of its *Ease of Implementation*, reflecting the relative simplicity of this requirement.

For each score, the spreadsheet calculates a weighted score by applying the percent weight for that factor. The priority rating for the requirement is calculated as the total of the weighted scores for the requirement.

You may use a variety of voting systems to arrive at the weights for the factors and the scores for each requirement. To make sure everyone is

heard, issue voting tokens (gold stars, Monopoly money, or some such device) and ask each stakeholder to place his voting tokens on his highest-priority requirements. Of course, the spreadsheet is merely a vehicle for enabling a group of stakeholders to arrive at a consensus when prioritizing the requirements. By making complex situations more visible, you make it possible for people to communicate their interests, to appreciate other individuals' opinions, and to negotiate.

Conflicting Requirements

Two requirements are conflicting if you cannot implement them both—the solution to one requirement prohibits implementing the other. For example, if one requirement asks for the product to “be available to all” and another says it shall be “fully secure,” then both requirements cannot be implemented as specified.

Prioritization, as we discussed earlier, might prevent some conflicts from happening. However, nothing is perfect, so you might need to take the following steps.

As a first pass at finding conflicting requirements, sort the requirements into their types. Then examine all entries that you have for each type, looking for pairs of requirements whose fit criteria are in conflict with each other. See **Figure 17.6**.

		Requirement #						
		1	2	3	4	5	6	7
Requirement #	7			X			X	
	6							
	5							
	4	X						
	3							
	2							
	1							

Figure 17.6. This matrix identifies conflicting requirements. For example, requirements 3 and 7 are in conflict with each other. If we implement a solution to requirement 3, it will have a negative effect on our ability to implement a solution to requirement 7, and vice versa.

Of course, a requirement might potentially conflict with any other requirement in the specification. To help you discover these problems, here are some clues to the situations where we most often find requirements in conflict:

- Requirements that use the same data (search by matching terms used)
- Requirements of the same type (search by matching requirement types)
- Requirements that use the same scales of measurement (search by matching requirements whose fit criteria use the same scales of measurement)

For functional requirements, look for conflicts in outcomes. As an example, suppose one requirement for the IceBreaker project calls for a roads section to be treated by the nearest truck, and another specifies that truck scheduling must rotate the trucks to allow for maintenance and driver rest periods. These two requirements would probably result in different outcomes.

Conflicts between requirements are normal for most requirements-gathering efforts, and indicate the need for

some sort of conflict resolution mechanism.

Conflicts may arise because different stakeholders have asked for different requirements, or because stakeholders have asked for requirements that are in conflict with the client's idea of the requirements. This type of overlap, which is normal for most requirements-gathering efforts, indicates you need to establish some sort of conflict resolution mechanism.

You, as the requirements analyst, have the most to gain by settling conflicts as rapidly as possible, and as early as possible, so we suggest that you take the lead role in resolving them. When you have isolated the conflicting requirements, approach each of the stakeholders separately (this is one reason why you record the originator of each requirement). Go over the requirement with the user and ensure that both of you have the same understanding of it. Reassess the satisfaction and dissatisfaction ratings: If one stakeholder gives low marks to the requirement, then he may not care if you drop it in favor of the other requirement. Do this with both stakeholders and do not, for the moment, bring them together.

When you talk to each stakeholder, explore his reasoning. What does the stakeholder really want as an outcome, and will it be compromised if the other requirement takes precedence? Also, ensure that the requirement is not a solution, as often stakeholders ask for solutions that are within their own realm of experience, and naturally, experiences differ.

Most of the time we have been able to resolve conflicts by talking to the stakeholders. Note that we use the term "conflict" here, not "dispute." There is no dispute. There are no positions taken, no noses put out of joint by the other guy winning. The stakeholder need not even know who the conflicting party is.

If you, as a mediator, are unable to reach a satisfactory resolution, then we suggest that you determine the cost of implementing the opposing requirements, assess their relative risks, and, armed with numbers, call the participants together and see if you can reach some compromise. Except in cases of extreme office politics, stakeholders are usually willing to compromise if they are in a position to do so gracefully without loss of face.

Ambiguous Specifications

The specification should, as far as is practical, be free of ambiguity. You should not have used any pronouns, and should be wary of unqualified adjectives and adverbs—all of these parts of speech introduce ambiguity. Do not use the word “should” when writing your requirements; it infers that the requirement is optional. Nevertheless, even if you follow these guidelines, some problems may remain.

The fit criterion quantifies a requirement, thereby making it unambiguous. We described fit criteria in [Chapter 12](#), explaining how they make each requirement both measurable and testable. If you have correctly applied fit criteria, then the requirements in your specification will be unambiguous.

This leaves the descriptions of the requirements. Obviously, the less ambiguity they contain, the better, but a poor description cannot do too much damage if you have a properly quantified fit criterion for the requirement. However, if you are concerned about this issue, then we suggest that you select 50 requirements randomly. Take one of them and ask a panel of stakeholders to give their interpretation of the requirement. If everyone agrees on the meaning of the requirement, then set it aside. If the meaning of the requirement is disputed, then select five more requirements. Repeat this review until either it becomes clear that the specification is acceptable or the collection of selected requirements to test has grown so large (each ambiguity brings in five new ones) that the problem is obvious to all.

If the problem is truly bad, then consider rewriting the specification using a better-qualified requirements writer. Or, if it is really, really bad, consider aborting the project. A problem with requirements is the most common problem with crippled projects—there appears no point to proceeding when you know you have poor, ambiguous requirements.

The terms used in the requirements must be those defined in the Data Dictionary section of the specification. If every word has an agreed-upon definition and you have used the terms consistently, then the meanings throughout the specification must be consistent and unambiguous.

Risk Assessment

Risk assessment is not really a requirements problem, but rather a project issue. At this stage of the requirements process, you have a complete specification of a product that you intend to build. You have invested a certain amount of time deriving this specification, and you are about to invest even more time in building the product. Now seems like a good time to pause for a moment and consider the risks involved in proceeding.

As a requirements analyst, you do not have to handle the risk assessment by yourself. This task is more likely to be performed by the project manager, and if your organization is of a reasonable size, there should be someone on staff who is trained in risk assessment.



DeMarco, Tom, and Tim Lister. *Waltzing with Bears: Managing Risk on Software Projects*. Dorset House, 2003. This book contains strategies for recognizing and monitoring risks.

The role of the business analyst in risk assessment is to consider requirements from the point of view of whether they contain some risk that could affect the success of the project. Some requirements pose greater risk than others. For example, some requirements might call for a technology or an implementation that the development team has never attempted before. That is not to say that they cannot pull it off, but there is a risk that they can't. Consider the risks that could be present within the following parts of the Volere Requirements Specification Template:

Project Drivers

1. The Purpose of the Project

Is the purpose of the product reasonable? Is it something your organization can achieve? Or are you setting out to do something you have never

done before, with only hysterical optimism telling you that you can deliver the objective successfully?

2. The Client, the Customer, and Other Stakeholders

Is the client a willing collaborator? Or is he uninterested in the project? Is the customer represented accurately? Are all stakeholders involved and enthusiastic about the project and the product? Hostile or unidentified stakeholders can have a very negative effect on your project. What are the chances that everyone will make the necessary contributions? Which risks do you run by not gaining the cooperation you need?

2. Users of the Product

Are the users properly represented? While user representative panels are useful, experience has shown that they are frequently wrong in their assessment of what the real users want and need. Are the users capable of telling you the correct requirements? Many project leaders often cite the quality of user contributions to requirements as their most serious and frequently encountered risk.

Many system development efforts result in substantial changes to the users' work and the way that users work. Have you considered the risk that the users will not be able to adapt to the new arrangements? Remember that humans do not like being changed, and your new product is bringing changes to your users' work. Are the users capable of operating the new product? Consider these risks carefully, as the risk of the users not being prepared to change may turn out to be a substantial obstacle.

Project Constraints

3. Mandated Constraints

Are the constraints reasonable, or do they indicate design solutions with which your organization has no experience? Is the budget reasonable given the effort needed to build the product? Unrealistic schedules and budgets are among the most common risks cited by projects.

5. Relevant Facts and Assumptions

Are the assumptions reasonable? Should you make contingency plans for the eventuality that one or more of the assumptions turns out not to be true? It pays to keep in mind that assumptions are really risks.

Functional Requirements

6. The Scope of the Work

Is the scope of the work correct? Do you run the risk of not including enough work to produce a satisfactory product? If the scope is not large enough, then the resulting product will not do enough for the user to make it truly valuable to him. A failure to define the work scope correctly always results in early requests for modifications and enhancements to the product.

7. Business Data Model and Data Dictionary

Is the terminology defined so that everyone has the same interpretation of the terms contained in the requirements?

8. The Scope of the Product


Does the scope of the product include all of the needed functionality, or just the easy stuff? Is it feasible given the budget and time available? Having the wrong product scope risks having many change requests after delivery.

Other commonly cited risks include creeping user requirements and incomplete requirements specifications. Risk analysis does not make all of these risks disappear, but it does ensure that you and management become aware of problems that might arise and can make appropriate plans for monitoring and addressing them. It is far more preferable to raise the alert early than to watch a disaster unfold while knowing that you might have been able to prevent it.

Measure the Required Cost

Measuring the cost or effort needed is not usually the responsibility of the requirements analyst. We mention this topic here because now that the requirements are known, you have an ideal opportunity to measure the size of the product. Common sense suggests that you do not proceed past this point without knowing its size, and thus the effort needed to build the product.

To this end, we have included a short introduction to function point counting in [Appendix C](#). The appendix shows how this technique works and suggests it as an effective way to estimate size.

 [Appendix C](#), Function Point Counting: A Simplified Introduction, gives a brief but sufficient primer on this commonly used technique for measuring the size of your work or your product.

The work you have done in gathering the requirements provides input to the measuring process; your context model is the definitive guide to the size of the work; a data model (if you have one) provides guidance to the effort needed to store the data. You can simply count the number of requirements you have written. All of these are measurements, and any are vastly preferable to guesswork, or blind acceptance of imposed deadlines.

One of the most commonly encountered risks is the risk of poor estimates of time needed to complete the project. This risk almost always manifests itself by turning into a real-world problem—when time starts to run out, the project team usually responds by taking shortcuts, skimping on quality, and ends up delivering a poor product even later than originally planned. It becomes avoidable when you take the short time needed to measure the size of the product, thereby determining—accurately—the required effort to build it.

We suggest that you include some kind of measurement activity in your completeness review.

Summary

The purpose of the review we have been talking about is to assess the correctness, completeness, and quality of the requirements specification. This review also gives you an opportunity to measure the benefit, cost, and risk attached to building the product, and to assess whether it is worthwhile to continue development of the product.

Consider the model shown in **Figure 17.7**. It provides a composite measure of the overall value of the product by measuring the risk, the cost to build and operate the product, and the benefit it brings along each of the corresponding scales. What does your profile look like? If you have high scores for cost and risk but a low score for benefit, you should consider abandoning the product. Conversely, you would love to have high benefit with low costs and low risks, but you probably won't get it. The point is to map these factors and note whether the overall profile of your product indicates that it is one to build or one to avoid.

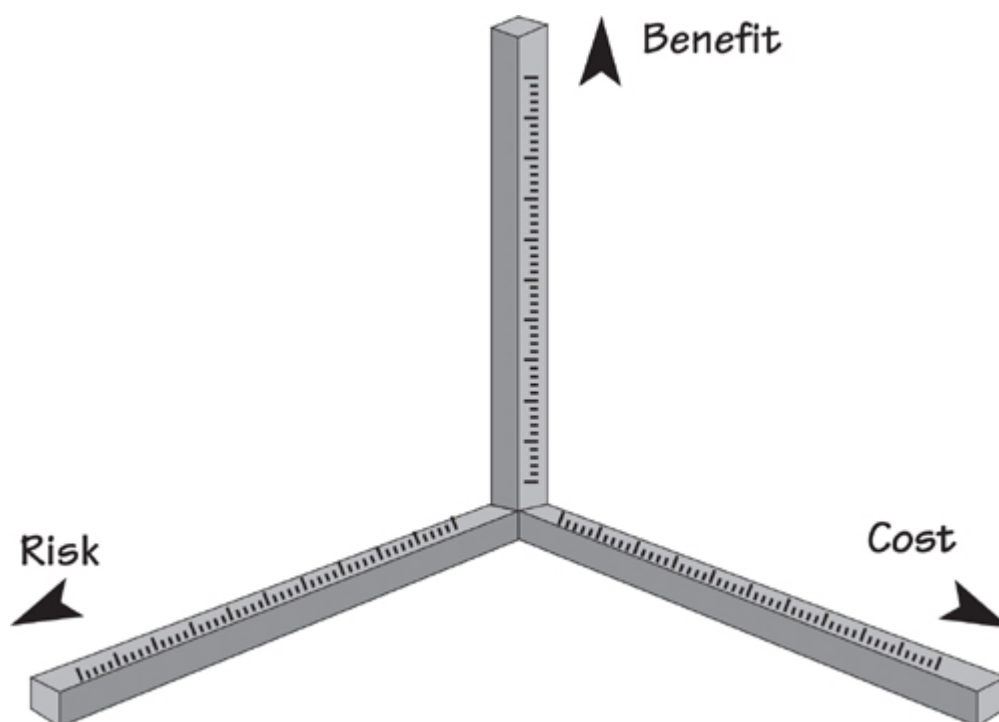


Figure 17.7. Each axis represents one of the factors that determines whether the product is worthwhile. The Cost axis, measuring the cost of construction and operation, can be assessed using function points or some other size measurement. The Benefit axis measures the value to the business and reflects the customer ratings placed by the stakeholders on the product. The Risk axis measures the severity of risks determined by the risk analysis activity.

And that brings us to the end of our book. In the course of getting from page one to here, we have tried to inject our experience of requirements projects that spans many years and many continents. If we have been able to pass along some advice, show you a process, give you some tips, suggest a direction, solve a problem, provide a shortcut, and explain the previously unfathomable, then we have succeeded in meeting our goal for this text.

If we have provided you with a reliable companion for your requirements work, then that is what we set out to do. If we have made some noticeable difference to the way that you go about discovering requirements, and if you feel that those requirements are better than they would have been without this book, then the past year will not have been in vain.

Enjoy our book; we hope it makes some difference, and the difference is beneficial to you.