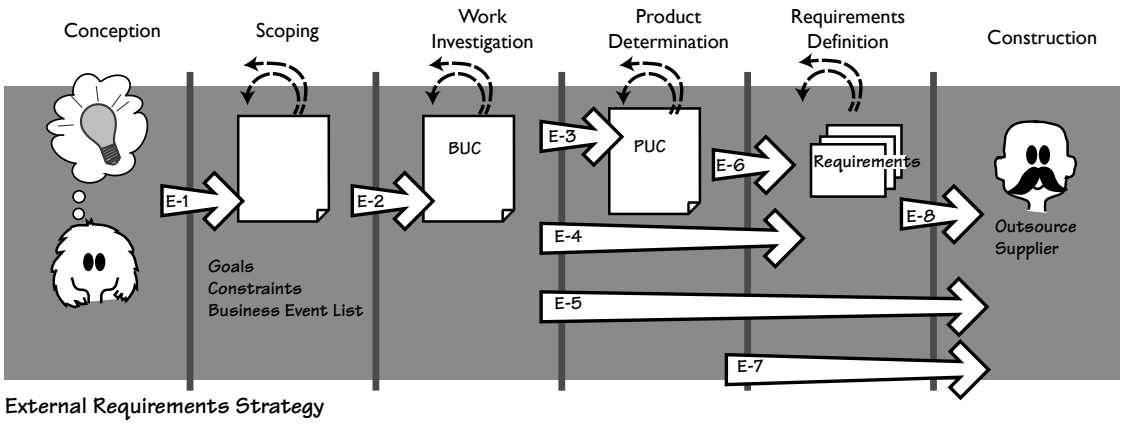
THIRD EDITION

MASTERING THE REQUIREMENTS PROCESS

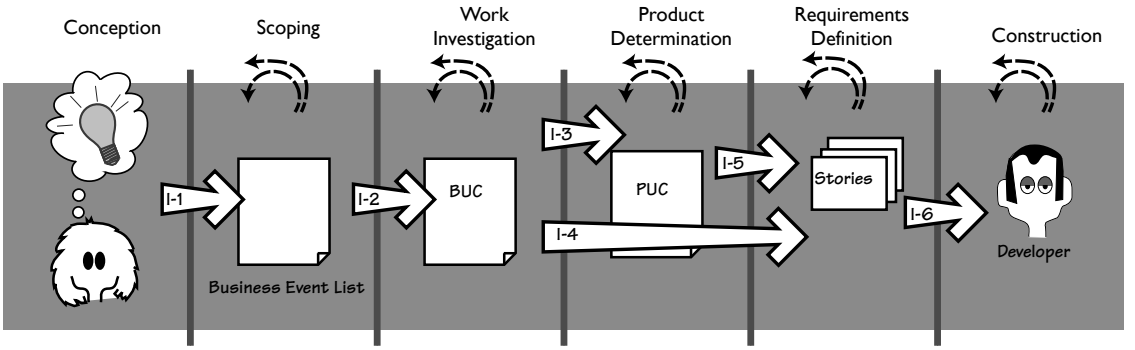
GETTING REQUIREMENTS RIGHT

SUZANNE ROBERTSON • JAMES ROBERTSON

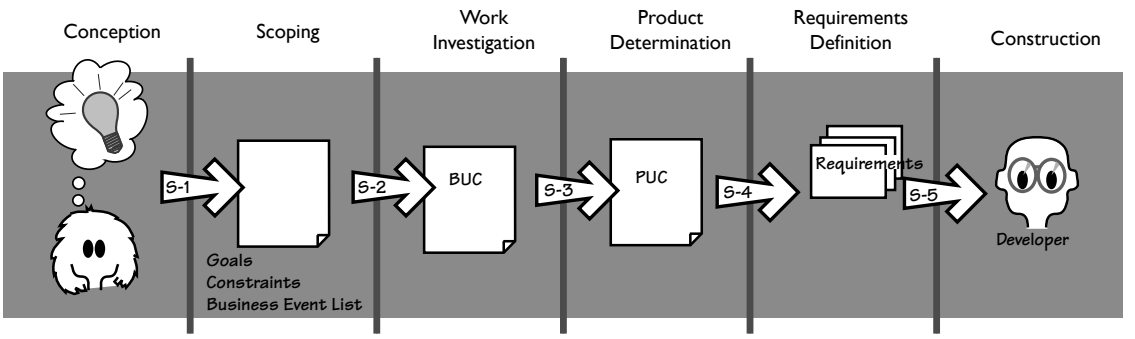
Requirements Strategy Maps



External Requirements Strategy



Iterative Requirements Strategy



Sequential Requirements Strategy

make mistakes: forget his password, choose the wrong option, get honey on the keys, or any of the many unfortunate things a Pooh can do.

Cast your protagonist as someone who is forgetful, slow, distracted, and not paying attention. (You are bound to know someone like that.) Go through the normal case scenario, and for each step, ask what can be done wrongly. It may be simpler to write a new scenario for each misuse, as the ramifications of a misstep may be complex enough to warrant their own separate story.

So much for the protagonist. The antagonist is the person who opposes the work, seeks to harm it, or wants to defraud it. Hackers are the most commonly thought-of example of antagonists. Examine all of the steps for the normal case and ask if there is a possibility of someone opposing or misusing that action. In this case, the ramifications upon discovering an antagonist's misuse may be to simply stop the business use case. For example:

Examine all of the steps for the normal case and ask if there is a possibility of someone opposing or misusing that action.

> *3. Check the passport is valid and belongs to the passenger.*
>
> *M3.1 The passenger produces a passport that is not his.*
>
> *M3.2 Call security.*
>
> *M3.3 Freeze the reservation.*

Whether you annotate the normal case with the misuse steps or write a separate misuse scenario depends on the complexity of the situation and the comfort level of the stakeholders.

Some professions have always made use of what if? scenarios. For instance, chess players routinely think, "What would Black do if I moved my knight to e4?", leading to the minimize–maximize algorithms for game-play. Our governments routinely generate what if? scenarios to plan policy: "What if the United States reestablished the gold standard?" "What if Switzerland elects a communist government?" "What if Canada closes the St. Lawrence Seaway?" While these examples are obviously fanciful, most governments generate thousands of likely and unlikely scenarios to explore their reactions to possible future events.

When gathering requirements, try generating several what if? scenarios to experiment with the unforeseen. The intention is to turn the unforeseen into the foreseen: The more you know about eventualities before you build the product, the more robust and long-lasting it will be.

> *"I don't always try to identify the predator flows. I use the same flows as other actors but ask the question, What if a hacker got here? What other precautions need to be implemented? What is the risk to the corporation if they get here? Can the corporation live with the risk of occurrence? How much are you willing to pay to avoid the risk?"*
>
> —Patricia Ferdinandi, *A Requirements Pattern: Succeeding in the Internet Economy,* Addison-Wesley, 200

Scenario Template

You may, of course, write your business use case scenarios in whatever form you and your stakeholders prefer. The template presented in this section is

one we have found useful on many assignments. We suggest it as a fairly good compromise between informality and an overly bureaucratic approach.

Business Event Name: The name of the business event to which the business use case responds.

Business Use Case Name and Number: Give each business use case a unique identifier and a name that communicates the functionality—for example, Record Library Loan, Register New Student Enrollment, Make Benefit Payment, Produce Sales Report. Ideally, the name should be an active verb plus a specific direct object.

Trigger: The data or request for a service that arrives from an external source and triggers a response from the work. The trigger may be the arrival of data from one of the adjacent systems—that is, from outside the work area that you are studying. Alternatively, the trigger may be the arrival of the temporal condition that causes the use case to become activated—for example, the end of the month.

Preconditions: Sometimes certain conditions must exist before the use case is valid. For example, a customer has to be registered before he can access his frequent-flyer statement. Note that another business use case usually takes care of the precondition. In the preceding example, the customer would have registered using the Register Passenger business use case.

Interested Stakeholders: The people, organizations, and/or representatives of computer systems that have knowledge necessary to specify this use case or that have an interest in this use case.

Active Stakeholders: The people, organizations, and/or computer systems that are doing the work of this use case. Don't think about users just yet; instead, think of the real people who are involved in the work of the business use case.

Normal Case Steps: The steps that this use case goes through to complete the desired course of its work. Write these steps as clear, natural-language statements that are understandable to business people related to the project. There are usually between three and ten steps.

Step 1 . . .
Step 2 . . .
Step 3 . . .

Note that a business use case can make use of the services or functionality of another business use case as part of its own processing. However, be careful not to start programming at this stage.

Alternatives: Alternatives are acceptable variations on the normal case of processing. For example, gold cardholders may be given an invitation to visit the lounge when they check in. Tell the story in the same way:

Alternative step 1 . . .
Alternative step 2 . . .
Alternative step 3 . . .

If the alternative action is simple, you can make it part of the normal case:

Step 4. Attach the frequent-flyer number to the reservation.
Alternative 4.1 Issue a lounge invitation if the passenger holds a gold card.

Exceptions: These cases are unwanted but inevitable variations. For example, a customer may have insufficient funds for a withdrawal at an ATM. In this case, the procedure has to offer a lower amount, or offer a loan, or do whatever the stakeholders decide is appropriate. Tag each exception to the appropriate step:

Exception 2.1 . . .
Exception 2.2 . . .
Exception 2.3 . . .

Outcome: The desired situation at the end of this use case. You might call this the "post condition." Think of it as the stakeholder's objective at the time when he triggers the use case. For example, the money has been dispensed and taken from the ATM, the customer's account has been debited, and the card has been extracted from the ATM.

Summary

A scenario is a natural language tool for telling a story. We have discussed how to write this story; its intention is to help you and your stakeholders come to a coherent understanding of the functionality of a business use case. Writing the BUC scenario tests whether sufficient study has been done or the business analyst needs to ask more questions and investigate further.

Once the BUC scenario is agreed—that is, you and your stakeholders have come to a consensus that it accurately represents the desired state of the work—it forms the basis for writing the requirements. We shall explore this progression, including the use of scenarios for product use cases, over the next few chapters.

This page intentionally left blank

Understanding the Real Problem

7

in which we "think above the line" to find the true essence of the business, and so deliver the right product—one that solves the right problem

I know that's what I asked for, but it's not what I need. You do not have to be part of IT very long to hear that statement and to see the expectant smiles dashed from the developers' faces—it happens all the time. So what is going wrong here? The developers have delivered exactly what the business stakeholders asked for, but it turns out not to solve their business problem. Why not? *Because the real problem was never stated, and so was never correctly understood.*

When stakeholders ask for some feature or capability, they quite often state their request as an implementation. "I want an external disk to back up my laptop's drive." This is a business stakeholder asking for a solution to a problem, but not really saying what that problem is. How can you know if it is the correct solution? You can't; and until you know what the real problem is there is little point in thinking about solutions.

So what is this user's problem? What does he really need?

If our stakeholder is concerned that his laptop's hard drive will crash, then perhaps the correct solution is to replace the hard drive with something more reliable. If the user is concerned that his computer will be stolen from the office, then the thieves will most probably cart away the backup drive as well. And if there is a fire, then the backup drive will melt down at about the same time as this guy's computer.

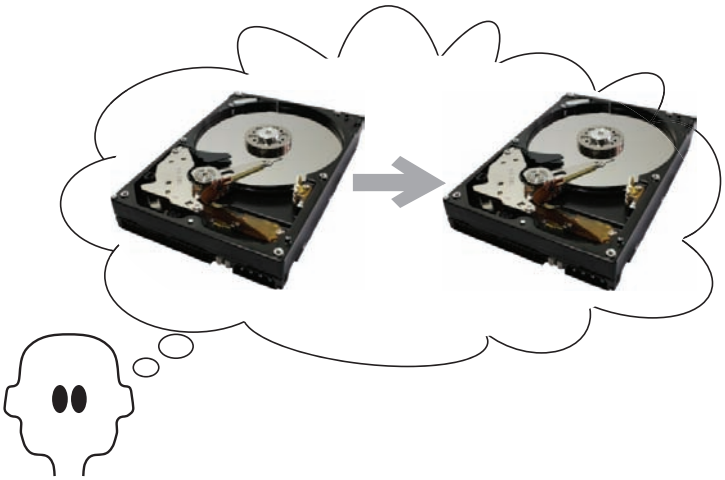
There's more: The user wants to back up his hard drive, which suggests that at some stage every day or every week he will boot up the backup program and do the backup. But what if he forgets? Is the real business need still being met if the solution relies on fallible human memory?

In this chapter we talk about how to get to the real problem by using *abstraction*—focusing on ideas rather than solutions. To put it another way,

> *At the end of the day, if the software doesn't meet the user's needs, it is still lousy software regardless of how it was created.*
>
> —Pete McBreen, *Questioning Extreme Programming*

Figure 7.1

The problem is to back up the hard drive—or is it?



abstraction involves thinking about the *essence* of the subject by discarding the technological and physical components.

Thus, instead of the suggested physical implementation "Back up the hard drive," you look at the essence and say, "Eliminate loss of data" or "Guard the data against theft" or "Guard the data against fire." Once you understand the real problem—the underlying business need—then you are much better placed to find the optimal solution for it. So, dear reader, before you leap to a solution for the backup problem—which can range from frequent, automated uploads to a cloud service, to attaching the laptop to a well-trained attack Rottweiler and giving it an exit from the building in case of fire—we want you to consider the idea of abstraction, and how it can work for you.

The physical reality was that the company rented DVDs. But if you abstract from that, you see that Netflix is in the business of renting out movies.

Let's look at an example of abstraction. Netflix, an American company, had a successful business renting out DVDs though the mail. In return for a monthly fee, customers could have several Netflix DVDs at home, and when they had watched one and returned it, another was sent. At the time when Netflix was founded, this was a new business model for renting movies. But at some point Netflix looked at the abstraction of its business. The physical reality was that the company rented DVDs, but if you abstract from that, you see that Netflix is in the business of *renting out movies*.

Now that we see the real business is about renting movies, we ask if there is a different, more convenient way to do so? Well, yes. Netflix found it and switched its business to providing *downloadable* movies. This model put the customers in direct contact with the Netflix website, and gave the customers even better service by providing a wide choice of instantly available movies, along with the convenience of not having to return their watched DVDs through the mail. By adopting this approach, Netflix changed its operation to be closer to the essence of the problem.