CHAPTER NINETEEN

Succeeding with Requirements

A Drama in Three Acts

Karl E. Wiegers

The Setting

Autumn 1994. The research laboratories of a large company, Contoso Pharmaceuticals (a real company, but not its real name).

The Cast

Paul
: Manager of the Health and Safety Department at Contoso Pharmaceuticals

Dana
: Manager of Contoso's chemical stockroom

Sarah
: One of the chemists who works in the Contoso Research Laboratories

Jonathan
: A member of the Purchasing Department

Janet
: A project manager and analyst from the research labs' IT department

Devon

A programmer/analyst from the research labs' IT department

Karl

An internal consultant for the Contoso Research Laboratories

Prologue: Paul Is in a Pickle

Paul is feeling some heat and he isn't happy. New government regulations require Contoso Pharmaceuticals to supply specific quarterly reports describing how it acquires, stores, uses, and disposes of chemicals. Hundreds of Contoso scientists have vast arrays of chemicals in their labs and thousands of additional containers are stored in the chemical stockroom. The only way Paul can comply with these reporting regulations is to have a robust Chemical Tracking System (CTS) that can monitor the location and status of every chemical container in the company.

Paul has been aware of this need for some time, but now it has become critical. Two previous teams from a corporate IT department had taken a stab at the Chemical Tracking System. Each team sat down with Paul to discuss his requirements, but neither team ever produced a written requirements specification and eventually both efforts were abandoned without delivering anything. Now Paul is under increasing pressure to deliver these reports for regulatory compliance and he has nothing to show for the previous efforts. Knowing how critically this application is needed, the research labs' IT department charters a new team to beat the project into submission.

Act I: Girding Our Loins

Wherein we develop a strategy and assemble the cast.

The initial goal of the project team is to develop a requirements specification that is sufficiently accurate and complete to allow the software development to begin. The decision to build the system in-house or to outsource part or all of the development is deferred to a later date. Our requirements analysis team consists of three people:

- Janet is an experienced developer and analyst who is taking on her first significant project management role. She will plan the project activities, track our progress, and serve as one of the requirements analysts. Although fairly young, Janet is mature, doesn't get rattled easily, and has a calm demeanor.
- Devon is a more junior developer. Bright and energetic, he's looking forward to building some analyst skills. If the CTS software is implemented in-house, Devon will be one of the developers.
- I'm Karl. I've worked at Contoso for 15 years as a research scientist, software developer, software manager, and quality engineer. Having pursued ways to improve requirements engineering for some time, I frequently serve as an internal consultant. I will work half-time in an analyst role on the CTS project.

Janet meets with Paul and explains that our team of analysts is going to do our best to meet his needs. But first, of course, we need to understand his requirements for the CTS. Paul's frustration is evident. "I gave your predecessors on the two previous teams my requirements," he says. "I don't have time to talk about requirements anymore. Build me a system!" This becomes our mantra for the project: "Build me a system!"

Facing a hostile, disillusioned, and somewhat intimidating lead customer, our challenge is clear. We need to overcome the distasteful legacy left by the preceding groups and somehow engage Paul and other customers in an effective and collaborative requirements development process. It looks like an uphill struggle.

It's not unusual for the requirements development process to be strained and maybe even adversarial. The participants sometimes forget that they are (or at least ought to be) on the same side, working toward the common goal of successfully delivering a useful product. Our first step is to begin forging a collaborative partnership with Paul. He needs to have confidence that this time will be different, that we really will move him toward his goal of meeting the regulatory requirements. And we IT people must concoct a plan for delivering on that expectation.

Expectation Management

Too often, teams begin working on a project without having discussed just how they will collaborate. The team members make assumptions about the activities involved and how the participants will interact with each other. People have different communication styles, various understandings of the problem, diverse perceptions about just what "requirements" are, and so on. Many groups don't explicitly agree upon how they will make the myriad decisions that arise in the course of every project. Neglecting these issues can lead to mismatched expectations, ineffective collaboration, and hard feelings. It's well worth taking some time at the outset to discuss just how the team will operate.

Janet begins by promising Paul that this analyst team will use more effective approaches to understanding his requirements for this application. Furthermore, we'll document what we learn in a way that serves as a suitable foundation for the development work that will follow. Janet also sets expectations for what we need from Paul if this collaborative effort is to succeed. Notwithstanding his previous frustrating experiences, the fact is that we have to start over with the process of exploring requirements. So we're going to need time from Paul and perhaps his colleagues so that we can understand just what kind of a system would meet his needs. Paul reluctantly agrees to play along.

Classy Users

Paul is not the only stakeholder for the system, and the members of the Health and Safety Department won't be its sole users. We apply the concept of *user classes* to identify other groups of users who will have largely different sets of needs. Members of different user classes might need different functions or features, have various educational or skill levels, work in different locations, or have other distinguishing characteristics.

Based on input from Paul and the analyst team's understanding of Contoso's research environment, we identify the following four major user classes:

Health and Safety Department staff
: Led by Paul, these people are responsible for generating the necessary government reports that describe how Contoso Pharmaceuticals handles its chemicals.

Chemists
: Contoso employs hundreds of chemists in the research labs, product development areas, and manufacturing. These chemists acquire new chemicals, store them in their labs, use them in experiments, and dispose of leftover chemicals that are no longer fit for use.

Chemical stockroom staff
: Although few in number, the people who work in this stockroom are central to the chemical tracking process. They place requests for chemicals to be purchased from vendors, stock and dispense thousands of chemical containers, manage supplies of new chemicals invented by the research scientists themselves, and dispose of outdated chemicals.

Purchasing Department staff
: Like the Health and Safety Department staff, these people will never touch a chemical. They serve as the interface between Contoso employees who need to buy chemicals and the vendors who supply the chemicals.

These user classes have certain needs in common. For example, both chemists and the chemical stockroom staff will place requests for new chemicals periodically and dispose of chemicals. However, each user class also has a distinct set of requirements for services they expect from the CTS.

Our next analyst challenge is to find the right individuals with whom to explore these needs. We must then consolidate those needs into a cohesive software requirements specification. This requirements exploration will include resolving requirement conflicts and setting priorities to reach the best balance of timely and cost-effective delivery of a useful—and usable—system.

Who Ya Gonna Call?

In 1986, my small development group at Contoso realized we had to learn how to interact more closely with our users so that we could properly meet their software needs. We conceived the project role of *product champion*, a key user representative who works closely with the requirements analyst. The product champion serves as the literal voice of the customer for a particular user class. Product champions typically are experienced users and subject matter experts in their domain.

Our next step on the CTS project, therefore, is to identify champions for the four user classes. My previous group had created a list of possible product champion activities that the CTS analysts can present to each candidate champion so that they understand what they're

getting into. Negotiating the exact responsibilities that each champion is willing to accept is an important part of crafting that collaborative customer–development partnership.

As the key manager on the spot and a hands-on user of the future system, Paul is the obvious product champion for the Health and Safety Department. Our project manager and lead analyst, Janet, will work with Paul to define his requirements. A member of the Purchasing Department named Jonathan is willing to present requirements for the CTS from his community's perspective. Our second analyst, Devon, agrees to work with Jonathan on Purchasing's requirements. Dana manages the chemical stockroom. She's the natural product champion for that user class, although she'll obtain additional input from the members of her staff who perform the day-to-day operations. Janet will do double-duty to lead requirements elicitation with Dana.

Finding a product champion for the chemist group is more challenging. Dana tries to help. "Before I became the chemical stockroom manager, I was a laboratory chemist," she tells us. "Therefore, you don't need to talk to any other chemists about their requirements. I can tell you everything you need to know."

Although Dana is sincere and means well, she is wrong. And it's really hard to convince her that she's wrong. The first problem is that she's no longer a member of the chemist user class. She is literally the best person in the world to describe the needs of the chemical stockroom staff. She isn't, however, a practicing chemist anymore. I learned long ago that it is best to have actual members of the user class participate in requirements development, rather than using surrogates or former members of that user class. In some cases, particularly on commercial product development projects, you don't have direct access to appropriate user representatives, so you must use surrogates, such as product managers or marketing staff. That isn't a problem for this internal corporate project, where we can arrange to work with actual users.

The second concern we have with Dana's offer is that she has a narrow and parochial view of the chemists' requirements. She is also very adamant about her opinions on such matters, and she is prone to crying when discussions get a bit tense or her opinions are not immediately accepted. (Dana isn't all that much fun to work with.) Relying on Dana's strong opinions alone would not provide the rich understanding of chemist needs that we need to get. So we politely decline Dana's offer and go searching for a suitable chemist representative.

We find one! A highly experienced and respected chemist named Sarah offers to serve as the product champion. Although she isn't terribly sophisticated when it comes to computers, Sarah recognizes the value that the Chemical Tracking System will provide to her and her colleagues. It's common that the people best suited as representatives for their user class are already overextended in their own work and are reluctant to commit much time to requirements-related activities. We really luck out with Sarah, though. She promises to create time in her busy schedule for requirements elicitation discussions and review sessions. Since my own educational background is in chemistry, I will be the analyst working with Sarah to understand the requirements that chemists have for CTS.

We realize it's just not realistic to expect even an experienced chemist like Sarah to know the full spectrum of requirements for all the chemists at Contoso. She needs some help. We establish a backup team of five additional chemists drawn from various departments across the company. They will have many requirements in common, but these representatives know about specific needs that pertain to the kinds of chemical work going on in their own areas.

We don't expect Sarah alone to produce the chemists' requirements, or even Sarah plus the backup team. Each of our product champions is responsible for interacting with other members of his or her user class to collect ideas and get feedback on proposals. The product champion also will resolve requirements conflicts that arise between individual members of the user class (which makes the analysts' lives much easier). So in this representative engagement model, the primary pipelines through which requirements flow are from each product champion to the corresponding analyst, with extensive interactions behind the scenes between the champion and other users. The analyst will need to work with the various product champions to resolve requirements conflicts that arise between the user classes.

Now we're ready to roll. Table 19-1 shows the complete cast of key participants in the requirements elicitation activities we are about to launch. But exactly how should we proceed?

TABLE 19-1. Requirements elicitation participants for the Chemical Tracking System project

User class	Requirements analyst	Product champion	Backup team?
Health and Safety Department	Janet	Paul	No
Chemical stockroom staff	Janet	Dana	No
Chemists	Karl	Sarah	Yes (5)
Purchasing Department	Devon	Jonathan	No

Act II: Use Cases, Schmuse Cases

Wherein we try some requirements development techniques and see what happens.

Traditionally, requirements analysts have often opened an interview or workshop by asking users, "What do you want?" This is the least useful question you can ask when exploring requirements. A related nearly useless question is "What are your requirements?" People aren't sure just how to answer these vague questions, and often they yield random bits of important, but unorganized, information. We don't want to fall into that same trap on this project, so we need a better way to hunt the elusive and secretive requirement.

The Case for Use Cases

Several members of our IT groups routinely attend technical conferences on software development. The conference attendees then present summaries of the talks they heard at our weekly group meeting and we all contemplate how we might apply new techniques to our own projects. I recall that one of my colleagues recently attended a conference presen-

tation on a requirements elicitation technique called *use cases*. This sounds like a potentially useful method, so I ask to borrow his copy of the presentation slides.

As I pore over his notes on the use case method, I get the feeling this just might work for the Chemical Tracking System. We don't have much information about use cases available and none of us has any experience with them, but the concept surely makes sense. The main theme of the use case technique is to focus requirements discussions on what the *user* needs to accomplish with the system, rather than on system features and functions. That is, we will take a usage-centric approach instead of a product-centric approach. So instead of asking our users "What do you want?" or even "What do you want the system to do?" we plan to ask them "What do you need to do with the system?" It's a small change in the question but a profound shift in perspective. Each of these "things the user needs to do with the system" is a potential use case, literally a case of usage.

Since we've never tried use cases before, we aren't sure how this is going to work and we face some learning curves. Use cases will also be unfamiliar to our product champions. We'll need to educate them about their role in this new approach and explain why we're trying this route rather than holding a more traditional requirements conversation.

Working in the Shop

Instead of conducting many individual interviews with user representatives, we decide to hold a series of facilitated requirements elicitation workshops. Each analyst will meet periodically with his or her user representatives to learn about the goals those users hope to accomplish with the help of the CTS. The analyst will serve as the facilitator and recorder for each workshop (although we soon learn that sometimes it's helpful to split these two roles). Having teams of users put their heads together saves the time often needed to resolve the conflicts that arise when individuals are interviewed sequentially. The synergy of a group discussion also generates new ideas and helps the participants reach a shared vision of the ultimate product.

We elect to hold workshops for the different user classes separately. We've subdivided our users into multiple user classes with largely different requirements, so it doesn't make sense to have them all participate in the same workshop sessions.

I lead the workshops with our chemist user representatives—the product champion, Sarah, and the five members of her backup team. In keeping with the spirit of having teams agree on how they're going to collaborate, we establish some ground rules for each workshop group. One of our ground rules is that the sessions will begin and end on time; we don't wait for people who straggle in late. Another rule is that the participants who show up to a workshop constitute a decision-making quorum. If someone is unable to attend, he's invited to submit his input beforehand or to send a delegate. He'll also be able to review the results of each workshop. But those who attend will make the necessary decisions so that we can move along expeditiously. Agreeing on these and other ground rules is an important part of defining an effective and efficient collaborative team experience.

Before we hold our first workshop, I invite the chemists to think of "things chemists would need to do with the CTS." Each of these "things to do" becomes a candidate use case that we will explore in the workshops. Some are big things (Request a Chemical); some are small (View a Stored Request). I schedule our first workshop, planning to start with what seem to be the most important use cases.

Not certain exactly how to proceed, I refer to the conference presentation notes I mentioned earlier. The speaker described a technique for using flip charts and sticky notes to capture the essential components of a use case. These components include:

- Use case name
- Name of the actor who will execute the use case (to a first approximation, an *actor* is a type of user)
- A brief description of the use case and its priority
- Preconditions that must be satisfied before the system can initiate the use case
- Postconditions that reflect the state of the universe at the successful completion of use case execution
- The normal flow, which describes the most typical set of interactions between an actor and the system that lead to a successful outcome
- Alternative flows, which describe other ways the actor might perform the use case (branching options and other variations)
- Exceptions, which are possible ways the use case might fail to complete execution successfully

I am pleased when this flip-chart approach appears to work well in our workshop. I'm further pleased when my chemists are receptive to the use case approach. Use cases are comfortable for many users because they relate directly to their business and how they anticipate using the system in that business. As we gain experience, I learn how to facilitate the workshops more effectively, to limit them to about two and a half hours in duration, and to time-box the discussion on each use case so that we don't get bogged down. I also learn that it's important to keep the use case discussion at the right level of abstraction. We avoid drilling down into excessive detail, such as user interface specifics, prematurely.

I am Lewis (or maybe Clark) in this use case exploration. We don't know anyone who has applied use cases before, so I do my best to blaze the trail. Once the chemists get into a rhythm and start making progress, the other analysts sit in on one of my workshops to observe our approach and determine how they want to adapt the method to their own user groups. Janet and Devon hold their own workshops in a similar style and also make good progress with their users. The three analysts work independently, but we share our experiences and insights so that we can all learn to do a better job. All software team members can work more effectively if they learn by looking over the shoulders of their colleagues.

Use Cases Aren't Enough

As I examine the products from a use case workshop, I realize that we've done a good job of describing how users would interact with the CTS to achieve various goals. That is, the use cases are a good way to describe *user requirements*. However, they don't seem to provide all of the information a developer would need to implement those capabilities in the system. Also, the way information is packaged into small chunks in the use case isn't ideal for handing off to a developer. We conclude that use cases alone will not be a sufficient deliverable from our requirements elicitation activities.

To take it to the next level, we need to derive the associated *functional requirements* from each use case. As a former developer myself, I understand that developers don't implement use cases. Developers implement specific bits of system behavior—functional requirements—that, in the aggregate, allow a user to perform a use case. So after each use case workshop, I analyze the information we collected and begin growing a software requirements specification (SRS). For each use case I write a set of functional requirements that, if implemented, will enable a user to perform the use case.

Some of this functionality is straightforward, particularly regarding the dialog of interactions between the user and the system. However, I can derive additional functionality that is not so evident from the use case description. For example, the use case preconditions give no clue as to what the system should do if a precondition is *not* satisfied. Some of the actions the system must perform aren't visible to the user, so the use case doesn't describe them. These functional specifications provide a richer description of expected system behavior than developers would get if we simply handed them a stack of use cases and said, "Call me when you're done."

Beyond Functionality

People seem to learn best from painful experience. I reflect back on a previous project for which we did an excellent job of specifying functionality but did not explicitly explore the users' quality expectations. As a consequence, the newly released system met serious resistance because of less-than-ideal trade-offs between system efficiency, usability, and other characteristics. Not wanting to repeat that unhappy experience, we elect to also discuss *quality attribute* requirements as part of specifying the CTS requirements.

Quality attributes describe not *what* the system does—that's the functionality—but rather *how well* it does the things that it does. Quality attributes include usability, portability, maintainability, installability, availability, performance, efficiency, robustness, and security. Users have implicit expectations for certain of these quality characteristics. Unfortunately, they often do not spontaneously share those expectations. Users have difficulty articulating their quality expectations in a way that provides helpful guidance for architects and developers.

Simply asking a user, "What are your robustness requirements?" isn't likely to yield useful information. So we take a different approach. We think about what "robustness" might mean to our users and then we write some questions to help the user think through some

aspects of robustness. An example is "What do you think the system should do if the network connection fails partway through submitting a new chemical order?" We do the same for other quality attributes that are likely to be important to the success of the system. We analysts ask our user teams to answer these questions and to rate the relative importance of each attribute from their perspective. Patterns emerge that help us determine which characteristics of the system are critical and which are less important. Those quality attribute requirements go into our SRS; they are every bit as important as the functionality descriptions when it comes to creating customer satisfaction.

The Rules of the Game

During one elicitation workshop the chemists discuss a use case called View a Stored Request. A member of the backup team says, "I don't want to see Sarah's chemical requests, and Sarah shouldn't be able to see my requests." The chemists agree that this is a good policy. We record it as being a *business rule*. Business rules can come from various sources:

- Corporate, organizational, or project policies, such as security policies for accessing information systems
- Laws and regulations, like those for generating reports on chemical handling and disposal
- Industry standards, such as file formats for importing and exporting chemical structures
- Computational algorithms, perhaps to determine price discounts on large orders purchased from a single vendor

Business rules are not in themselves software requirements. However, a rule can serve as the source of functionality that must be implemented so that the system complies with or enforces the policy. The rule that constrains who may view which orders implies the need for functionality to authenticate each user's identity before he can view a request. During the elicitation workshops each user class identifies (or invents) numerous business rules. These rules help to establish a framework to make sure the software developers color inside the lines.

Test Before Coding? Are You Mad?

Requirements are all about describing what we're going to have when we're done. But as I'm working with the flip charts from one workshop, I have a revelation. The use case thought process appears to flow naturally into identifying tests we can use to tell whether we have in fact built what we intended to build. I had never heard about this connection between use cases and tests before, but it certainly makes sense to me.

So I begin writing tests from the use case description, thinking about not how I might implement the use case, but rather how I might tell whether I had implemented it correctly. These are conceptual tests, free from implementation and user interface specifics. In the process of writing these tests, I discover some errors in my use case. It's another "aha!" moment. Perhaps I can use these conceptual tests, developed early in the requirements process, to find errors long before any design or construction takes place!

I build on this theme. As I mentioned earlier, I had been deriving functional requirements from the use case description. In principle, if a developer implements that functionality, the user can perform the use case. Now I'm deriving tests from that same use case, using a different thought process. I compare those tests to my functional requirements, looking for two things:

- Is each of my functional requirements covered by a test?
- Could each of my tests be "executed" by firing off a particular series of functional requirements?

As I go through this analysis, I find missing, incorrect, and unnecessary requirements. Some of these indicate problems with the use case and some with my decomposition of the use case into functionality fragments. I also find missing, incorrect, and unnecessary tests. After I fix these problems, I have a lot more confidence that my requirements are correct.

I also realize it would be even better if different people, using entirely separate brains, were to derive the functional requirements and the tests from each use case. Disconnects between their results could indicate ambiguities and omissions in the use case that lead to different interpretations. I now know that I can begin testing a software application immediately after I've written its first requirement. This surely beats testing at the end and then spending a lot of time and money to fix the errors that originated in requirements.

I invite my product champion, Sarah, to review the tests I wrote for the chemists' use cases. She offers additional corrections and improvements. In a final workshop, all six chemist representatives walk through the tests to make sure we all share a common mental image of how the CTS would work. We all have different ideas of what the screens might look like, and we'll work through that when we get further into design. For now, though, we feel confident that we all have the same expectations and understanding of what the CTS will do for the chemists.

Act III: Look Over My Shoulder

Wherein we do lots of peer reviews to find as many requirement errors as we can.

Simply documenting the requirements an analyst hears during interviews or workshops isn't sufficient to give confidence that the requirements are correct. There are so many opportunities for miscommunications and misunderstandings that validation is an essential step in the requirements development process. *Peer reviews* constitute one of the most powerful mechanisms for finding errors in requirements. During a peer review, someone other than the author of a work product examines that work product for possible defects and improvement opportunities.

Peer reviews are a type of static testing, a way to filter out requirement problems before writing the first line of code. Reviews provide a way for users to confirm that their input has been interpreted and recorded properly. They also provide a way to detect ambiguous

requirements, which helps all stakeholders reach a common understanding of what the requirements are trying to say. If I could perform only one quality practice on a software project, that practice would be formal peer review (also called *inspection*) of all requirements information. Given this appreciation of the power of peer reviews, we build them into our requirements activities in two ways, informally and formally.

The Casual Style

Following each use case elicitation workshop, the analyst supplies the documents he or she creates to the workshop participants. Such documents include use cases, portions of a growing SRS, graphical analysis models, data definitions, and other knowledge acquired during the workshop. The workshop participants then examine these materials informally on their own, looking for omissions, errors, misinterpretations, and any other issues.

And, boy, do we find a lot of problems this way! Sometimes the analyst misunderstood something or put his own twist of interpretation on the requirements, which the users catch and correct. In other cases, a user realizes we overlooked an alternative flow for a use case. Or perhaps he concludes that an error condition should be handled in some different way than we originally envisioned. Sometimes the act of reviewing the requirements triggers an idea for some additional functionality that users would find helpful. These individual, incremental, informal, and inexpensive reviews greatly improve the quality of our growing body of requirements information. The types of problems we find also give us insights into how to improve our future requirements elicitation and specification activities. In addition, the reviews provide clear evidence to Paul and the other project stakeholders that we are making real progress on understanding and recording their requirements for the CTS.

The analysts on the team try different review approaches. I am holding workshops once per week with my chemists. Within two days following each workshop I deliver my write-up of the workshop outputs to the product champion and members of the backup team. Reviewing these materials helps anyone who missed a workshop get caught up. My users do a great job of finding errors, which they share with me at the beginning of the next workshop.

Our least experienced analyst, Devon, takes a different tack. He's holding daily workshops with his users, so he must quickly write up the workshop information and get it to the representatives to examine before they meet again the next day. This puts a lot of time pressure on the participants. More significantly, Devon reports that his users aren't finding many problems. Unfortunately, the problems are there; the reviewers just aren't catching many of them.

The insight here is that when you review a document shortly after thinking intensely about it (either while creating it as an author or while contributing to it as a workshop participant), you don't really review the document—you mentally recite it. You're less likely to find problems than if you get some mental "settling time" before revisiting it. In contrast, when you come back to a document after a day or two, during which your short-

term memory has faded, you look at it with a fresh perspective. You're more likely to spot disconnects, omissions, and other problems. Perhaps you've been mulling over the matter in the back of your brain for the past couple of days and you have some ideas for how to improve on the original work. Ever since I had this realization I've always tried to set my own writing aside for at least 24 hours before I review it myself. This helps me look at it with a sharper eye than if I review it immediately after writing.

The Formal Style

The type of individual, informal peer review I described in the preceding section is called a *passaround*. Our passarounds yield many improvements, but they aren't a substitute for the more rigorous type of team peer review, called an *inspection*. In an inspection, several participants examine the work product on their own and then pool their observations and questions in a meeting. The interactions that take place during the inspection meeting often result in discovering problems that no inspectors found during their individual preparation.

For requirements specifications, inspections provide another advantage: the ability to catch ambiguities. An ambiguous requirement is one that can be interpreted in multiple ways by different readers (or sometimes even by the same reader). Suppose each inspector reads an ambiguous requirement on his own during a passaround review. The requirement makes sense to each of them, but it means something different to each of them. Each reviewer will say, "This is fine," and the ambiguity goes undetected. During an inspection, though, one member of the team called the reader describes in his own words what he thinks each requirement means. The other inspectors can compare the reader's interpretation against their own understanding. Sometimes this reveals a difference of interpretation—an ambiguity—that could cause big problems if not caught until much later.

Given these powerful benefits, we decide to inspect our compiled SRS. Janet takes responsibility for editing together the partial requirements specifications and associated materials that the three analysts have developed. We end up with a 50-page SRS, with about another fifty pages of back matter, including analysis models, definitions, and other supporting information.

We invite our four product champions, our three analysts, and one additional project stakeholder to participate in the inspection. Eight is a fairly large inspection team. I have performed inspections for several years, so I serve as the moderator, which helps keep us on track. Devon serves as the recorder, logging the issues as they come up in the discussion. We realize that we can't possibly cover this volume of material in a single session, at least not if we're serious about scrutinizing it closely for problems. So we schedule three inspection meetings on a Monday, Wednesday, and Friday in the same week, with a maximum duration of two and a half hours per meeting (this is still a fairly rapid inspection rate). Marathon review meetings are useless because after a couple of hours, tired eyes contribute little additional insight.

Our inspection is a success, at least if one measures success by the number of errors detected. Every error we fix now is one we won't have to fix later on at considerably greater cost and aggravation. Devon keeps running out of the room for more blank copies of the inspection issues log. It's a bit discouraging to find so many defects because we've all worked hard on the requirements. But we also realize that each problem found at this stage is a "good catch."

The Outcome

Our inspection identifies no fewer than 223 defects and issues. Most are minor, but some would have had a major impact on the project had we not found them at this early stage. Finding so many errors makes us glad we took the time to look. It's clear that the cost of performing the inspection, although not trivial, greatly outweighs the potential loss had those defects lingered into the final requirements we will present to developers and testers.

Following the inspection meetings we correct the many errors the inspectors found. Paul and the other product champions agree that the revised requirements documents accurately state their requirements for the Chemical Tracking System, so far as those requirements are known today. Next we define a *baseline* for our SRS. A baseline doesn't mean that the specification is frozen or that changes can't be made in the future. The reality is that requirements *will* change, for many reasons. Customers might think of things they forgot, analysts might get bright ideas for useful new functionality, we might spot more requirement errors during design and coding, and the business itself can change during the development period. But our baseline serves as an agreed-upon foundation for the subsequent project work. Even though we acknowledge that the requirements are not perfect, achieving this baseline milestone gives all the participants a good feeling that we have accomplished what we set out to do on this important project. The requirements are "good enough" for development of the Chemical Tracking System to continue.

Epilogue: Let's Eat!

As time goes on during requirements elicitation, we detect a thawing of the ice we encountered in our early conversations with Paul. He sees that we're making real progress in understanding both his needs and the needs of other CTS stakeholders. Paul observes that the new techniques we're using do a better job of eliciting the right requirements than did the approach the earlier teams had taken. The result is that Paul believes the final set of requirements really will, if properly implemented, let him comply with the government reporting mandates, as well as providing many other valuable services for a wide variety of users.

We also experience a clear sign of a culture change as a result of our team approach to requirements on the CTS. Shortly after baselining the SRS, Paul throws a lunch bash for the analyst team, the product champions, and other key project participants. It's quite a spread. A good time is had by all and nobody goes home hungry. Paul's mood is much improved compared to that of four months earlier.

Coda: Then What Happened?

Since I became an independent consultant in 1998, I have primarily worked with clients on an intervention basis. I provide training, coaching, or assistance with a particular set of requirements-related problems a client is experiencing. I often do not get to see how the project turns out as a result of my involvement. That was true with the CTS project—at first.

Following our successful requirements development effort, I moved on to other activities and shortly thereafter moved to an entirely separate division within Contoso Pharmaceuticals. Two years later I left Contoso and started my own software training and consulting company. I use several examples from the Chemical Tracking System in my training courses to illustrate techniques and insights for requirements development.

In 1999, I was presenting a requirements course at a client site. When I shared some of our experiences on CTS, one student in the class said, "Hey, I recognize that project!" This student happened to work in marketing. His previous employer was the company that Contoso approached for outsourcing the implementation of part of the CTS. Intrigued by this coincidence, I asked him how the implementation went and what the developers at his former company thought of the CTS requirements. I was relieved when he said that the project went well and that the requirements were a major factor in that success.

Despite its rocky beginning, the Chemical Tracking System turned out to be an illuminating case study. We learned how to selectively and thoughtfully apply several new requirements development methods. We found ways to effectively engage diverse customer representatives in the process. We enjoyed some cultural benefits through our collaborative approach, as Paul and the other lead customers realized that it was indeed possible—nay, essential—to work closely with IT representatives to make sure that the right product comes out the other end of the process. Best of all, this application is still in use at Contoso Pharmaceuticals, 13 years after the requirements development experience described here.

Useful References

1. Ambler, Scott. 1995. "Reduce Development Costs with Use-Case Scenario Testing." *Software Development* 3(7):53–61.
2. Collard, Ross. 1999. "Test Design." *Software Testing and Quality Engineering* 1(4):30–37.
3. Gottesdiener, Ellen. 2002. *Requirements by Collaboration: Workshops for Defining Needs*. Boston: Addison-Wesley.
4. Kulak, Daryl, and Eamonn Guiney. 2004. *Use Cases: Requirements in Context, Second Edition*. Boston: Addison-Wesley.
5. Wiegers, Karl E. 1996. *Creating a Software Engineering Culture*. New York: Dorset House Publishing.
6. Wiegers, Karl E. 2002. *Peer Reviews in Software: A Practical Guide*. Boston: Addison-Wesley.

7. Wiegers, Karl E. 2003. *Software Requirements, Second Edition*. Redmond, WA: Microsoft Press.

8. Wiegers, Karl E. 2006. *More About Software Requirements: Thorny Issues and Practical Advice*. Redmond, WA: Microsoft Press.

Acknowledgments

I appreciate the many valuable review comments provided by Jim Brosseau, Barb Cardenuto, Chris Fahlbusch, Kathy Getz, Andre Gous, Shannon Jackson, Lori Schirmer, and Moe Stankay.