CHAPTER 7

# Requirements elicitation

*"Good morning, Maria. I'm Phil, the business analyst for the new employee information system we're going to build for you. Thanks for agreeing to be the product champion for this project. Your input will help us a lot. So, can you tell me what you want?"*

*"Hmmm, what do I want?" mused Maria. "I hardly know where to start. The new system should be a lot faster than the old one. And you know how the old system crashes if an employee has a really long name and we have to call the help desk and ask them to enter the name for us? The new system should take long names without crashing. Also, a new law says we can't use Social Security numbers for employee IDs anymore, so we'll have to change all of the IDs when the new system goes in. Oh, yes, it'd be great if I could get a report of how many hours of training each employee has had so far this year."*

*Phil dutifully wrote down everything Maria said, but his head was spinning. Maria's desires were so scattered that he wasn't sure he was getting all her requirements. He had no idea if Maria's needs aligned with the project's business objectives. And he didn't know exactly what to do with all these bits of information. Phil wasn't sure what to ask next.*

The heart of requirements development is *elicitation*, the process of identifying the needs and constraints of the various stakeholders for a software system. Elicitation is not the same as "gathering requirements." Nor is it a simple matter of transcribing exactly what users say. Elicitation is a collaborative and analytical process that includes activities to collect, discover, extract, and define requirements. Elicitation is used to discover business, user, functional, and nonfunctional requirements, along with other types of information. Requirements elicitation is perhaps the most challenging, critical, error-prone, and communication-intensive aspect of software development.

Engaging users in the elicitation process is a way to gain support and buy-in for the project. If you're the business analyst, try to understand the thought processes behind the requirements the users state. Walk through the processes that users follow to make decisions about their work, and extract the underlying logic. Make sure that everyone understands why the system must perform certain functions. Look for proposed requirements that reflect obsolete or ineffective business processes or rules that should *not* be incorporated into a new system.

The BA must create an environment conducive to a thorough exploration of the product being specified. To facilitate clear communication, use the vocabulary of the business domain instead of forcing customers to understand technical jargon. Record significant application domain terms in a glossary, rather than assuming that all participants share the same definitions. Customers must understand that a discussion about possible functionality is not a commitment to include it in the

product. Brainstorming and imagining the possibilities is a separate matter from analyzing priorities, feasibility, and the constraining realities. It's never too early for stakeholders to prioritize their blue-sky wish lists to avoid defining an enormous project that never delivers anything useful.

The output of requirements development is a common understanding of the needs held by the diverse project stakeholders. When the developers understand those needs, they can explore alternative solutions to address them. Elicitation participants should resist the temptation to design the system until they understand the problem. Otherwise, they can expect to do considerable design rework as the requirements become better defined. Emphasizing user tasks rather than user interfaces, and focusing on true needs more than on expressed desires, help keep the team from being sidetracked by prematurely specifying design details.

As Figure 7-1 shows, the nature of requirements development is cyclic. You will do some elicitation, study what you learned, write some requirements, perhaps determine that you are missing some information, perform additional elicitation, and so forth. Don't expect to just hold a couple of elicitation workshops and then declare victory and move on.

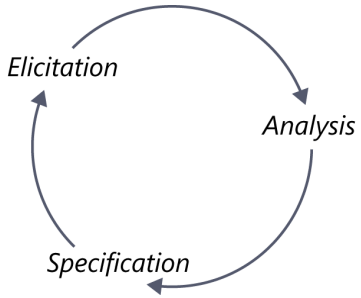


**FIGURE 7-1** The cyclic nature of requirements elicitation, analysis, and specification.

This chapter describes a variety of effective elicitation techniques, including when to use each one, as well as tips and challenges for each. The rest of the chapter describes the overall elicitation process, from planning elicitation activities to organizing the session outputs. Later in the chapter, we offer cautions about a few traps to watch out for during elicitation, and specific suggestions for identifying missing requirements. Figure 7-2 depicts the activities for a single requirements elicitation session. Before we walk through this process, though, let's explore some of the requirements elicitation techniques you might find valuable.

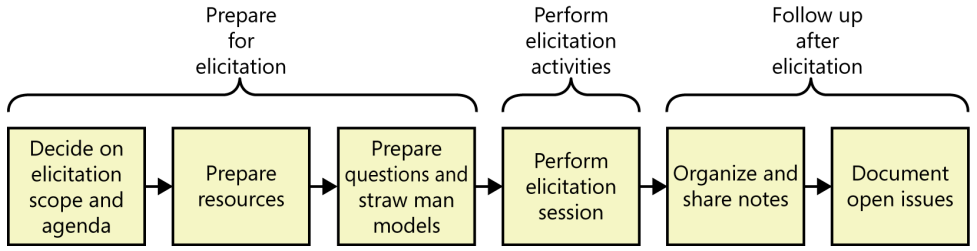


**FIGURE 7-2** Activities for a single requirements elicitation session.

# Requirements elicitation techniques

Numerous elicitation techniques can be employed on software projects. In fact, no project team should expect to use only one elicitation technique. There are always many types of information to be discovered, and different stakeholders will prefer different approaches. One user might be able to clearly articulate how he uses the system, whereas you might need to observe another performing her job to reach the same level of understanding.

Elicitation techniques include both facilitated activities, in which you interact with stakeholders to elicit requirements, and independent activities, in which you work on your own to discover information. Facilitated activities primarily focus on discovering business and user requirements. Working directly with users is necessary because user requirements encompass the tasks that users need to accomplish with the system. To elicit business requirements, you will need to work with people such as the project sponsor. The independent elicitation techniques supplement requirements that users present and reveal needed functionality that end users might not be aware of. Most projects will use a combination of both facilitated and independent elicitation activities. Each technique offers a different exploration of the requirements or might even reveal completely different requirements. The following sections describe several techniques commonly used to elicit requirements.

## Interviews

The most obvious way to find out what the users of a software system need is to ask them. Interviews are a traditional source of requirements input for both commercial products and information systems, across all software development approaches. Most BAs will facilitate some form of individual or small-group interviews to elicit requirements on their projects. Agile projects make extensive use of interviews as a mechanism to get direct user involvement. Interviews are easier to schedule and lead than large-group activities such as requirements workshops.

If you are new to an application domain, interviews with experts can help you get up to speed quickly. This will allow you to prepare draft requirements and models to use in other interviews or in workshops. If you can establish rapport with the interviewees, they will feel safer when sharing their thoughts one-on-one or in a small group than in a larger workshop, particularly about touchy topics. It's also easier to get user buy-in about participating in the project or reviewing existing requirements during a one-on-one or small-group interview than in a large group setting. Interviews are appropriate for eliciting business requirements from executives who do not have a lot of time to meet.

For guidance on how to conduct user interviews, see Ian Alexander and Ljerka Beus-Dukic (2009) and Howard Podeswa (2009). A few suggestions for conducting interviews follow. These are useful tips for conducting elicitation workshops as well.

**Establish rapport** To begin an interview, introduce yourself if the attendees don't already know you, review the agenda, remind attendees of the session objectives, and address any preliminary questions or concerns attendees have.

**Stay in scope** As with any elicitation session, keep the discussion focused on its objective. Even when you are talking with just one person or a small group, there's a chance the interview will go off topic.

**Prepare questions and straw man models ahead of time** Prepare for interviews by drafting any materials you can beforehand, such as a list of questions to guide the conversation. Draft materials will give your users a starting point to think from. People can often critique content more easily than they can create it. Preparing questions and drafting straw man models are described further in the "Preparing for elicitation" section later in this chapter.

**Suggest ideas** Rather than simply transcribing what customers say, a creative BA proposes ideas and alternatives during elicitation. Sometimes users don't realize the capabilities developers can provide; they might get excited when you suggest functionality that will make the system especially valuable. When users truly can't express what they need, perhaps you can watch them work and suggest ways to automate portions of the job (see the "Observations" section later in this chapter). BAs can think outside the mental box that limits people who are too close to the problem domain.

**Listen actively** Practice the techniques of active listening (leaning forward, showing patience, giving verbal feedback, and inquiring when something is unclear) and paraphrasing (restating the main idea of a speaker's message to show your understanding of that message).

## Workshops

Workshops encourage stakeholder collaboration in defining requirements. Ellen Gottesdiener (2002) defines a requirements workshop as "a structured meeting in which a carefully selected group of stakeholders and content experts work together to define, create, refine, and reach closure on deliverables (such as models and documents) that represent user requirements." Workshops are facilitated sessions with multiple stakeholders and formal roles, such as a facilitator and a scribe. Workshops often include several types of stakeholders, from users to developers to testers. They are used to elicit requirements from multiple stakeholders concurrently. Working in a group is more effective for resolving disagreements than is talking to people individually. Also, workshops are helpful when quick elicitation turnaround is needed because of schedule constraints.

According to one authority, "Facilitation is the art of leading people through processes toward agreed-upon objectives in a manner that encourages participation, ownership, and productivity from all involved" (Sibbet 1994). The facilitator plays a critical role in planning the workshop, selecting participants, and guiding them to a successful outcome. Business analysts frequently facilitate elicitation workshops. When a team is getting started with new approaches to requirements elicitation, consider having an outside facilitator or a second BA facilitate the initial workshops. This way the lead BA can devote his full attention to the discussion. If the sole BA is also acting as facilitator, she needs to be mindful of when she is speaking as a facilitator and when she is participating in the discussion. A scribe assists the facilitator by capturing the points that come up during the discussion. It's extremely challenging to facilitate, scribe, and participate simultaneously and do a good job on all three.

Workshops can be resource intensive, sometimes requiring numerous participants for several days at a time. They must be well planned to avoid wasting time. Minimize wasted time by coming into a workshop with drafts of materials prepared ahead of time. For example, you might draft use cases that can be reviewed as a group rather than having the entire group draft them together. Rarely does it make sense to start a workshop with a completely blank slate. Use other elicitation techniques prior to the workshops, and then bring the stakeholders together to work through only the necessary areas.

General facilitation practices apply to requirements elicitation (Schwarz 2002). A definitive resource specific to facilitating requirements elicitation workshops is Gottesdiener's *Requirements by Collaboration* (2002). She describes a wealth of techniques and tools for workshop facilitation. Following are a few tips for conducting effective elicitation workshops, many of which also apply to interviews.

**Establish and enforce ground rules** The workshop participants should agree on some basic operating principles. Examples include starting and ending on time; returning from breaks promptly; silencing electronic devices; holding one conversation at a time; expecting everyone to contribute; and focusing comments and criticisms on issues rather than individuals. After the rules are set, ensure that participants follow them.

**Fill all of the team roles** A facilitator must make sure that the following tasks are covered by people in the workshop: note taking, time keeping, scope management, ground rule management, and making sure everyone is heard. A scribe might record what's going on, while someone else watches the clock.

**Plan an agenda** Each workshop needs a clear plan, as discussed in the "Preparing for elicitation" section later in this chapter. Create the plan and workshop agenda ahead of time, and communicate them to participants so they know the objectives and what to expect and can prepare accordingly.

**Stay in scope** Refer to the business requirements to confirm whether proposed user requirements lie within the current project scope. Keep each workshop focused on the right level of abstraction for that session's objectives. Groups easily dive into distracting detail during requirements discussions. Those discussions consume time that the group should spend on developing a higher-level understanding of user requirements; the details will come later. The facilitator will have to reel in the elicitation participants periodically to keep them on topic.

> **Trap** Watch out for off-topic discussions, such as design explorations, during elicitation sessions. Keep the participants focused on the session's objectives, while assuring them that they'll have future opportunities to work through other issues that arise.

**Use parking lots to capture items for later consideration** An array of random but important information will surface during elicitation discussions: quality attributes, business rules, user interface ideas, and more. Organize this information on flipcharts—parking lots—so you don't lose it and to demonstrate respect for the participant who brought it up. Don't be distracted into discussing off-topic details unless they turn out to be showstoppers. Describe what will happen with the parking lot issues following the meeting.

**Timebox discussions** Consider allocating a fixed period of time to each discussion topic. The discussion might need to be completed later, but timeboxing helps avoid the trap of spending far more time than intended on the first topic and neglecting other important topics entirely. When closing a timeboxed discussion, summarize status and next steps before leaving the topic.

**Keep the team small but include the right stakeholders** Small groups can work much faster than larger teams. Elicitation workshops with more than five or six active participants can become mired in side trips, concurrent conversations, and bickering. Consider running multiple workshops in parallel to explore the requirements of different user classes. Workshop participants could include the product champion and other user representatives, perhaps a subject matter expert, a BA, a developer, and a tester. Knowledge, experience, and the authority to make decisions are qualifications for participating in elicitation workshops.

### Too many cooks

Requirements elicitation workshops that involve too many participants can slow to a contentious crawl. My colleague Debbie was frustrated at the sluggish progress of the first use case workshop she facilitated for a website project. The 12 participants held extended discussions of unnecessary details and couldn't agree on how each use case ought to work. The team's progress accelerated nicely when Debbie reduced the number of participants to about six who represented the key roles of analyst, customer, system architect, developer, and visual designer. The workshop lost some input by using the smaller team, but the rate of progress more than compensated for that loss. The workshop participants should exchange information off-line with colleagues who don't attend and then bring the collected input to the workshops.

**Keep everyone engaged** Sometimes certain participants will stop contributing to the discussion. These people might be frustrated for a variety of reasons. Perhaps their input isn't being taken seriously because other participants don't find their concerns interesting, or maybe they don't want to disrupt the work that the group has completed so far. Perhaps the stakeholder who has withdrawn is deferring to more aggressive participants or a domineering analyst. The facilitator must read the body language (lack of eye contact, fidgeting, sighing, checking the clock), understand why someone has tuned out of the process, and try to re-engage the person. Visual cues are absent when you are facilitating via a teleconference, so you have to listen carefully to learn who is not participating and the tones being used. You might ask these silent individuals directly if they have any thoughts about the discussion they'd like to share. The facilitator must ensure that everyone is heard.

## Focus groups

A focus group is a representative group of users who convene in a facilitated elicitation activity to generate input and ideas on a product's functional and quality requirements. Focus group sessions must be interactive, allowing all users a chance to voice their thoughts. Focus groups are useful for exploring users' attitudes, impressions, preferences, and needs (IIBA 2009). They are particularly valuable if you are developing commercial products and don't have ready access to end users within your company.

### When conflicts erupt

Differing perspectives, priorities, and personalities can lead to conflict and even anger within a group. If this happens, deal with it immediately. Look for nonverbal clues showing conflict or anger and try to understand the cause. When the group is clear on the reason for the conflict, you might be able to find a solution to it (if one is needed).

If an individual simply will not participate in a productive way, talk with him privately to determine whether his presence will prevent the group from moving forward. If so, you might need to thank the person for his time and continue without him. Sometimes this will not be an option and you need to simply abandon the session or topic completely for now. Conflict management is a complex skill to develop and there are numerous resources on this (Fisher, Ury, and Patton 2011; Patterson et al. 2011).

I once scheduled a session to elicit business requirements from a new director of sales. He was known to have an antagonistic personality, so I came to the meeting prepared to really listen to and understand his desires. In the very first minute of the meeting, he started yelling at me, asking why we were holding this meeting at all. He said, "Who are you to think you have a right to ask me about my business objectives?" I took a deep breath and a long pause. Then I tried to explain why I needed to understand his business objectives—that without them, the team would be guessing at what we needed to develop to meet the customers' desires, and he would be sorely disappointed with the results. And as fast as he got mad, he got over it. Without hesitation, he started rattling off his business objectives. Thankfully my scribe was there to catch them because I was still a bit taken aback by the whole exchange.

Often, you will have a large and diverse user base to draw from, so select the focus group members carefully. Include users who have used previous versions or products similar to the one you're implementing. Either select a pool of users who are of the same type (and hold multiple focus groups for the different user classes) or select a pool representing the full spectrum of user classes so everyone is equally represented.

Focus groups must be facilitated. You will need to keep them on topic, but without influencing the opinions being expressed. You might want to record the session so you can go back and listen carefully to comments. Do not expect quantitative analysis from focus groups, but rather a lot of subjective feedback that can be further evaluated and prioritized as requirements are developed. Elicitation sessions with focus groups benefit from many of the same tips described previously for workshops. Participants in focus groups normally do not have decision-making authority for requirements.

## Observations

When you ask users to describe how they do their jobs, they will likely have a hard time being precise—details might be missing or incorrect. Often this is because tasks are complex and it's hard to remember every minute detail. In other cases, it is because users are so familiar with executing a

task that they can't articulate everything they do. Perhaps the task is so habitual that they don't even think about it. Sometimes you can learn a lot by observing exactly how users perform their tasks.

Observations are time consuming, so they aren't suitable for every user or every task. To avoid disrupting the users' regularly assigned work activities, limit each observation time to two hours or less. Select important or high-risk tasks and multiple user classes for observations. If you use observations in agile projects, have the user demonstrate only the specific tasks related to the forthcoming iteration.

Observing a user's workflow in the task environment allows the BA to validate information collected from other sources, to identify new topics for interviews, to see problems with the current system, and to identify ways that the new system can better support the workflow. The BA must abstract and generalize beyond the observed user's activities to ensure that the requirements captured apply to the user class as a whole, not just to that individual. A skillful BA can also often suggest ideas for improving the user's current business processes.

### Watch me bake a cake

To demonstrate the power of observations, tell some friends the steps to bake a cake from a mix. You'll likely remember the steps to turn on the oven, get out the necessary dishes and utensils, add each ingredient, mix the ingredients, prepare the pan, put the batter in the pan, bake it, and pull it out of the oven when done. But when you told your friends to add each ingredient, did you remember to say to open the bag with the mix in it? Did you remember to say to crack the eggshell, add only the contents of the egg, and discard the shell? These seemingly obvious steps might not be so obvious to someone who has never baked before.

Observations can be silent or interactive. Silent observations are appropriate when busy users cannot be interrupted. Interactive observations allow the BA to interrupt the user mid-task and ask a question. This is useful to understand immediately why a user made a choice or to ask him what he was thinking about when he took some action. Document what you observe for further analysis after the session. You might also consider video recording the session, if policies allow, so you can refresh your memory later.

I was developing a call-center application for customer service representatives (CSRs) who were used to having to page through printed catalogs to find products that customers wanted to order. The BA team met with several CSRs to elicit use cases for the new application. Each one said how difficult it was to have to flip through multiple catalogs to find exactly what product a customer was referring to. Each BA sat with a different CSR while the CSRs took orders over the phone. We saw the difficulty they faced by watching them first try to find the catalog by date, then try to locate the right product. The observation sessions helped us understand what features they would need in an online product catalog.

## Questionnaires

Questionnaires are a way to survey large groups of users to understand their needs. They are inexpensive, making them a logical choice for eliciting information from large user populations, and they can be administered easily across geographical boundaries. The analyzed results of questionnaires can be used as an input to other elicitation techniques. For example, you might use a questionnaire to identify users' biggest pain points with an existing system, then use the results to discuss prioritization with decision makers in a workshop. You can also use questionnaires to survey commercial product users for feedback.

Preparing well-written questions is the biggest challenge with questionnaires. Many tips are available for writing questionnaires (Colorado State University 2013), and we suggest the most important ones here:

- Provide answer options that cover the full set of possible responses.
- Make answer choices both mutually exclusive (no overlaps in numerical ranges) and exhaustive (list all possible choices and/or have a write-in spot for a choice you didn't think of).
- Don't phrase a question in a way that implies a "correct" answer.
- If you use scales, use them consistently throughout the questionnaire.
- Use closed questions with two or more specific choices if you want to use the questionnaire results for statistical analysis. Open-ended questions allows users to respond any way they want, so it's hard to look for commonalities in the results.
- Consider consulting with an expert in questionnaire design and administration to ensure that you ask the right questions of the right people.
- Always test a questionnaire before distributing it. It's frustrating to discover too late that a question was phrased ambiguously or to realize that an important question was omitted.
- Don't ask too many questions or people won't respond.

## System interface analysis

Interface analysis is an independent elicitation technique that entails examining the systems to which your system connects. System interface analysis reveals functional requirements regarding the exchange of data and services between systems (IIBA 2009). Context diagrams and ecosystem maps (see Chapter 5, "Establishing the business requirements") are an obvious choice to begin finding interfaces for further study. In fact, if you find an interface that has associated requirements and that is *not* represented in one of these diagrams, the diagrams are incomplete.

For each system that interfaces with yours, identify functionality in the other system that might lead to requirements for your system. These requirements could describe what data to pass to the other system, what data is received from it, and rules about that data, such as validation criteria. You might also discover existing functionality that you do *not* need to implement in your system. Suppose you thought you needed to implement validation rules for a shopping-cart order in an e-commerce

website before passing it to an order-management system. Through system interface analysis, you might learn that multiple systems pass orders to the order-management system, which performs the validation, so you don't need to build this function.

## User interface analysis

User interface (UI) analysis is an independent elicitation technique in which you study existing systems to discover user and functional requirements. It's best to interact with the existing systems directly, but if necessary you can use screen shots. User manuals for purchased packaged-software implementations often contain screen shots that will work fine as a starting point. If there is no existing system, you might be able to look at user interfaces of similar products.

When working with packaged solutions or an existing system, UI analysis can help you identify a complete list of screens to help you discover potential features. By navigating the existing UI, you can learn about the common steps users take in the system and draft use cases to review with users. UI analysis can reveal pieces of data that users need to see. It's a great way to get up to speed on how an existing system works (unless you need a lot of training to do so). Instead of asking users how they interact with the system and what steps they take, perhaps you can reach an initial understanding yourself.

Do not assume that certain functionality is needed in the new system just because you found it in an existing one. Furthermore, do not assume that because the UI looks or flows a certain way in the current system that it must be implemented that way in the future system.

## Document analysis

Document analysis entails examining any existing documentation for potential software requirements. The most useful documentation includes requirements specifications, business processes, lessons-learned collections, and user manuals for existing or similar applications. Documents can describe corporate or industry standards that must be followed or regulations with which the product must comply. When replacing an existing system, past documentation can reveal functionality that might need to be retained, as well as obsolete functionality. For packaged-solution implementations, the vendor documentation mentions functionality that your users might need, but you might have to further explore just how to implement it in the target environment. Comparative reviews point out shortcomings in other products that you could address to gain a competitive advantage. Problem reports and enhancement requests collected from users by help desk and field support personnel can offer ideas for improving the system in future releases.

Document analysis is a way to get up to speed on an existing system or a new domain. Doing some research and drafting some requirements beforehand reduces the elicitation meeting time needed. Document analysis can reveal information people don't tell you, either because they don't think of it or because they aren't aware of it. For example, if you are building a new call-center application, you might find some complicated business logic described in the user manual for an existing application. Perhaps users don't even know about this logic. You can use the results of this analysis as input to user interviews.

A risk with this technique is that the available documents might not be up to date. Requirements might have changed without the specifications being updated, or functionality might be documented that is not needed in a new system.

## Planning elicitation on your project

Early in a project, the business analyst should plan the project's approach to requirements elicitation. Even a simple plan of action increases the chance of success and sets realistic expectations for the stakeholders. Only by gaining explicit commitment on elicitation resources, schedule, and deliverables can you avoid having participants pulled away to do other work. An elicitation plan includes the techniques you'll use, when you plan to use them, and for what purpose. As with any plan, use it as a guide and reminder throughout the project, but realize that you might need to change the plan throughout the project. Your plan should address the following items:

- **Elicitation objectives** Plan the elicitation objectives for the entire project and the objectives for each planned elicitation activity.
- **Elicitation strategy and planned techniques** Decide which techniques to use with different stakeholder groups. You might use some combination of questionnaires, workshops, customer visits, individual interviews, and other techniques, depending on the access you have to stakeholders, time constraints, and your knowledge of the existing system.
- **Schedule and resource estimates** Identify both customer and development participants for the various elicitation activities, along with estimates of the effort and calendar time required. You might only be able to identify the user classes and not specific individuals up front, but that will allow managers to begin planning for upcoming resource needs. Estimate the BA time, including time to prepare for elicitation and to perform follow-up analysis.
- **Documents and systems needed for independent elicitation** If you are conducting document, system interface, or user interface analysis, identify the materials needed to ensure that you have them when you need them.
- **Expected products of elicitation efforts** Knowing you are going to create a list of use cases, an SRS, an analysis of questionnaire results, or quality attribute specifications helps ensure that you target the right stakeholders, topics, and details during elicitation.
- **Elicitation risks** Identify factors that could impede your ability to complete the elicitation activities as intended, estimate the severity of each risk, and decide how you can mitigate or control it. See Chapter 32, "Software requirements and risk management," for more on risk management. See Appendix B, "Requirements troubleshooting guide," for symptoms, root causes, and possible solutions for common elicitation problems.

Many BAs have their "go-to" elicitation technique—commonly interviews and workshops—and do not think to use other techniques that might reduce resource needs or increase the quality of the information discovered. Rarely will a BA get the best results by using only one elicitation technique on

a project. Elicitation techniques apply across the spectrum of development approaches. The selection of elicitation techniques should be based on the characteristics of the project.

Figure 7-3 suggests the elicitation techniques that are most likely to be useful for various types of projects. Select the row or rows that represent characteristics of your project and read to the right to see which elicitation techniques are most likely to be helpful (marked with an X). For instance, if you're developing a new application, you're likely to get the best results with a combination of stakeholder interviews, workshops, and system interface analysis. Most projects can make use of interviews and workshops. Focus groups are more appropriate than workshops for mass-market software because you have a large external user base but limited access to representatives. These suggestions for elicitation techniques are just that—suggestions. For instance, you might conclude that you do want to apply user interface analysis on mass-market software projects.

| | Interviews | Workshops | Focus groups | Observations | Questionnaires | System interface analysis | User interface analysis | Document analysis |
|---|---|---|---|---|---|---|---|---|
| Mass-market software | x | | x | | x | | | |
| Internal corporate software | x | x | x | x | | x | | x |
| Replacing existing system | x | x | | x | | x | x | x |
| Enhancing existing system | x | x | | | | x | x | x |
| New application | x | x | | | | x | | |
| Packaged software implementation | x | x | | x | | x | | x |
| Embedded systems | x | x | | | | x | | x |
| Geographically distributed stakeholders | x | x | | | x | | | |

**FIGURE 7-3** Suggested elicitation techniques by project characteristic.

## Preparing for elicitation

Facilitated elicitation sessions require preparation to make the best use of everyone's time. The larger the group participating in the session, the more important preparation is. Figure 7-4 highlights the activities to prepare for a single requirements elicitation session.

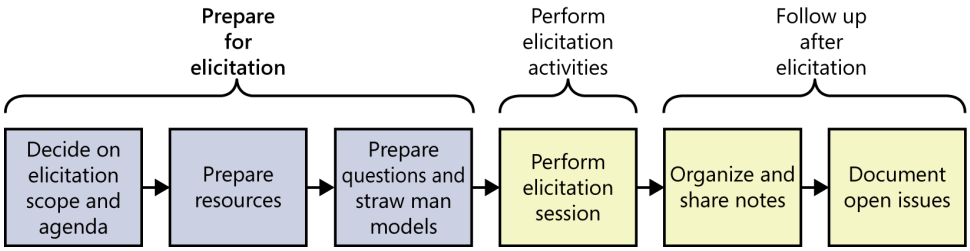


**FIGURE 7-4** Activities to prepare for a single elicitation session.

Prepare for each session by deciding on the scope of the session, communicating an agenda, preparing questions, and drafting materials that might be useful during the session. The following tips will help you prepare for elicitation.

**Plan session scope and agenda** Decide on the scope of the elicitation session, taking into account how much time is available. You might define the session scope by using a set of topics or questions, or you might list a specific set of process flows or use cases to be explored. Align the scope of the session with the overall project scope defined in the business requirements so you can keep the conversation on topic. The agenda should itemize what topics will be covered, the available time for each topic, and targeted objectives. Share the session agenda with stakeholders in advance.

**Prepare resources** Schedule the physical resources needed, such as rooms, projectors, teleconference numbers, and videoconferencing equipment. Also, schedule the participants, being sensitive to time zone differences if you are not all in the same location. For geographically dispersed groups, change the schedule each time you meet so the sessions do not always inconvenience the same people in a particular part of the world. Collect documentation from various sources. Gain access to systems as necessary. Take online training to learn about existing systems.

**Learn about the stakeholders** Identify the relevant stakeholders for the session (see Chapter 6, "Finding the voice of the user"). Learn about the stakeholders' cultural and regional preferences for meetings. If some of the participants are not native speakers of the language in which the session will be conducted, consider providing them with supporting documentation, such as slides, ahead of time so they can read ahead or follow along. The slides can list specific questions you will be asking or simply provide context for the session that you might also verbally explain. Avoid creating an "us" versus "them" tension.

**Prepare questions** Go into every facilitated elicitation session with a set of prepared questions. Use areas of uncertainty in straw man models (described in the next section) as a source of questions. If you are preparing for an interview or workshop, use results from other elicitation techniques to identify unresolved questions. There are many sources of suggested questions for elicitation (Wiegers 2006; Miller 2009).

Phrase your questions to avoid leading customers down an unintended path or toward a specific answer. As an analyst, you must probe beneath the surface of the requirements the customers present to understand their true needs. Simply asking users, "What do you want?" generates a mass of random information that leaves the analyst floundering. "What do you need to do?" is a much better question. Asking "why" several times can move the discussion from a presented solution to a solid understanding of the problem that needs to be solved. Ask open-ended questions to help you understand the users' current business processes and to see how the new system could improve their performance.

Imagine yourself learning the user's job, or actually do the job under the user's direction. What tasks would you perform? What questions would you have? Another approach is to play the role of an apprentice learning from a master user. The user you are interviewing then guides the discussion and describes what he views as the important topics for discussion.

Probe around the exceptions. What could prevent the user from successfully completing a task? How should the system respond to various error conditions? Ask questions that begin with "What else could . . . ," "What happens when . . . ," "Would you ever need to . . . ," "Where do you get . . . ," "Why

do you (or don't you) . . . ," and "Does anyone ever . . ." Document the source of each requirement so that you can obtain further clarification if needed and trace development activities back to specific customer origins.

As with any improvement activity, dissatisfaction with the current situation provides excellent fodder for the new and improved future state. When you're working on a replacement project for a legacy system, ask the users, "What three things annoy you the most about the existing system?" This question surfaces expectations that the users hold for the follow-on system.

You won't have—nor do you need—a perfect script going into an interview or a workshop. The prepared questions are to help you if you get stuck. The questions should seem natural and comfortable—like a conversation, not an interrogation. Five minutes into a session, you might realize that you missed an important area for discussion. Be ready to abandon your questions if needed. At the end of your session, ask "Is there anything else you expected me to ask?" to try to surface issues you just didn't think of.

**Prepare straw man models** Analysis models can be used during elicitation sessions to help users provide better requirements. Some of the most useful models are use cases and process flows because they closely align with how people think about doing their jobs. Create straw man, or draft, models ahead of your elicitation sessions. A straw man serves as a starting point that helps you learn about the topic and inspires your users to think of ideas. It is easier to revise a draft model than to create one from scratch.

If you are new to the project's domain, it might be hard to create a draft model on your own. Use other elicitation techniques to glean enough knowledge to work from. Read existing documents, examine existing systems for models you can reuse as a starting point, or hold a one-on-one interview with a subject matter expert to learn enough to get started. Then tell the group you're working with, "This model will probably be wrong. Please tear it apart and tell me how it should look. You won't hurt my feelings."

## Performing elicitation activities

Figure 7-5 highlights the activity to perform elicitation in a single requirements elicitation session.

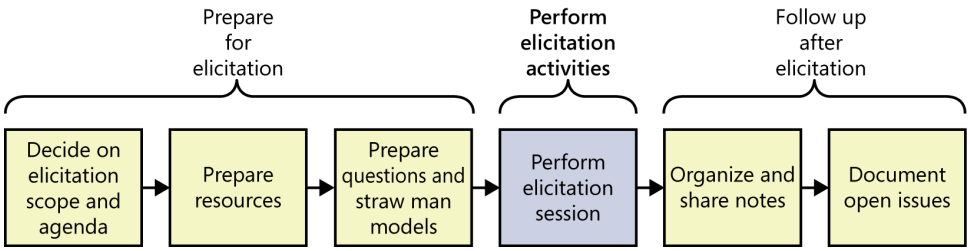


**FIGURE 7-5** The perform elicitation activities step for a single elicitation session.

Executing the elicitation activity itself is relatively obvious—if you are interviewing, you talk to people; if you are performing document analysis, you read the document. However, when facilitating an elicitation activity, the following tips might be useful.

**Educate stakeholders** Teach your stakeholders about your elicitation approach and why you chose it. Explain the exploration techniques you'll be using, such as use cases or process flows, and how they can help stakeholders provide better requirements. Also describe how you will capture their information and send them materials for review after the session.

**Take good notes** Assign someone who isn't actively participating in the discussion to be the scribe, responsible for taking accurate notes. Session notes should contain an attendee list, invitees who did not attend, decisions made, actions to be taken and who is responsible for each, outstanding issues, and the high points of key discussions. Unfortunately, BAs sometimes hold facilitated elicitation sessions without a dedicated scribe and have to fill the role themselves. If you're in this situation, be prepared to write shorthand, type fast, or use a recording device (if the participants agree). Audio pens can translate handwritten notes to electronic form and tie them to the recorded audio discussion. You can also use whiteboards and paper on the walls and photograph them.

Prepare questions ahead of time to eliminate some of the on-the-spot thinking necessary to keep the conversation going. Come up with a shorthand notation to capture a question that comes to mind while someone is talking, so you can quickly flip back to it when you have an opportunity. Don't try to capture diagrams in complicated diagramming software; just photograph sketched diagrams or draw quickly by hand.

**Exploit the physical space** Most rooms have four walls, so use them during facilitation to draw diagrams or create lists. If there aren't whiteboards available, attach big sheets of paper to the walls. Have sticky notes and markers available. Invite other participants to get up and contribute to the wall as well; moving around helps to keep people engaged. Gottesdiener (2002) refers to this technique as the "Wall of Wonder" collaboration pattern. If there are existing artifacts to look at (such as straw man models, existing requirements, or existing systems), project them on the wall.

Facilitating collaborative sessions with participants in multiple locations requires more creativity. You can use online conferencing tools to share slides and permit interactions. If several participants are in the same room, use videoconferencing tools to show remote participants what's on the walls and whiteboards.

### Stakeholders on the move

I once facilitated a workshop to elicit process flows for a semiconductor fabrication plant with a dozen engineers. I started out by working at the whiteboard, drawing the flows as we talked. Each time we completed a flow, I'd stop to photograph it before moving on to the next. Half a day into the first session, one of the engineers asked if he could have a turn at the whiteboard. I happily handed him the marker. He had learned the flowchart notation, and since he was already an expert in the system, he could easily draw the flow on the board. He then walked us through it, asking his peers at each step to validate or correct it. He was leading the process, which allowed me to focus on asking probing questions and taking notes. Soon, all the engineers were passing the marker around, so everyone got a turn.

If it's culturally appropriate, use toys to stimulate participants' minds or give them something to do with their hands. Simple toys can help inspire ideas. One team held a brainstorming session to establish the business objectives for their project. To start the day, I gave every participant some modeling clay and asked them to model their product vision using the clay—with no more instruction than that. It woke them up, got them thinking creatively, and they had some fun with it. We transitioned that energy into actually writing down a real vision for the product.

## Following up after elicitation

After each elicitation activity is complete, there's still a lot to do. You need to organize and share your notes, document open issues, and classify the newly gathered information. Figure 7-6 highlights the activities to follow up after a single requirements elicitation session.

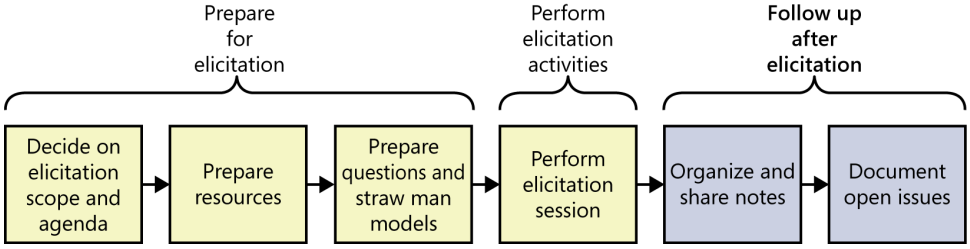


**FIGURE 7-6** Activities to follow up after an elicitation session.

### Organizing and sharing the notes

If you led an interview or workshop, organizing your notes probably requires more effort than if you organized information as you encountered it during an independent elicitation activity. Consolidate your input from multiple sources. Review and update your notes soon after the session is complete, while the content is still fresh in your mind.

Editing the elicitation notes is a risk. You might be incorrectly remembering what something meant, thereby unknowingly changing the meaning. Keep a set of the raw notes to refer to later if necessary. Soon after each interview or workshop, share the consolidated notes with the participants and ask them to review them to ensure that they accurately represent the session. Early review is essential to successful requirements development because only those people who supplied the requirements can judge whether they were captured correctly. Hold additional discussions to resolve any inconsistencies and to fill in any blanks. Consider sharing the consolidated notes with other project stakeholders who weren't present in the session, so that they are aware of progress. This gives them the opportunity to flag any issues or concerns immediately.

## Documenting open issues

During elicitation activities, you might have encountered items that need to be further explored at a later date or knowledge gaps you need to close. Or you might have identified new questions while reviewing your notes. Examine any parking lots from elicitation sessions for issues that are still open and record them in an issue-tracking tool. For each issue, record any relevant notes related to resolving the issues, progress already made, an owner, and a due date. Consider using the same issue-tracking tool that the development and testing teams use.

# Classifying customer input

Don't expect your customers to present a succinct, complete, and well-organized list of their needs. Analysts must classify the myriad bits of requirements information they hear into various categories so that they can document and use it appropriately. Figure 7-7 illustrates nine such categories. During elicitation activities, make quick notations in your notes if you detect that some bit of information is one of these types. For example, write "DD" in a little circle if you recognize a data definition.

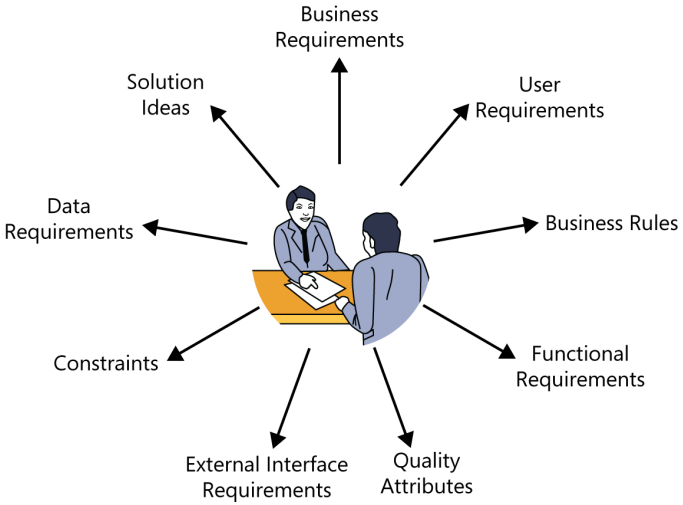


**FIGURE 7-7** Classifying customer input.

As with many categorizations, the information gathered might not fit precisely into these nine buckets. You will probably have pieces of information left over after this classification. Anything that doesn't fit into one of these categories might be:

- A project requirement not related to the software development, such as the need to train users on the new system.
- A project constraint, such as a cost or schedule restriction (as opposed to the design or implementation constraints described in this chapter).
- An assumption or a dependency.

- Additional information of a historical, context-setting, or descriptive nature.
- Extraneous information that does not add value.

Elicitation participants won't simply tell you, "Here comes a business requirement." As an analyst, you need to determine what type of information each provided statement you hear represents. The following discussion suggests some phrases to listen for that will help you in this classification process.

**Business requirements** Anything that describes the financial, marketplace, or other business benefit that either customers or the developing organization wish to gain from the product is a business requirement (see Chapter 5). Listen for statements about the value that buyers or users of the software will receive, such as these:

- "Increase market share in region X by Y percent within Z months."
- "Save $X per year on electricity now wasted by inefficient units."

**User requirements** General statements of user goals or business tasks that users need to perform are user requirements, most typically represented as use cases, scenarios, or user stories (see Chapter 8, "Understanding user requirements"). A user who says, "I need to <do something>" is probably describing a user requirement, as in the following examples:

- "I need to print a mailing label for a package."
- "As the lead machine operator, I need to calibrate the pump controller first thing every morning."

**Business rules** When a customer says that only certain users can perform an activity under specific conditions, he might be presenting a business rule (see Chapter 9, "Playing by the rules"). These aren't software requirements as they stand, but you might derive some functional requirements to enforce the rules. Phrases such as "Must comply with . . . ," "If <some condition is true>, then <something happens>," or "Must be calculated according to . . . " suggest that the user is describing a business rule. Here are some examples:

- "A new client must pay 30 percent of the estimated consulting fee and travel expenses in advance."
- "Time-off approvals must comply with the company's HR vacation policy."

**Functional requirements** Functional requirements describe the observable behaviors the system will exhibit under certain conditions and the actions the system will let users take. Here are some examples of functional requirements as you might hear them from users:

- "If the pressure exceeds 40.0 psi, the high-pressure warning light should come on."
- "The user must be able to sort the project list in forward and reverse alphabetical order."

These statements illustrate how users typically present functional requirements, but they don't represent good ways to write functional requirements. The BA will need to craft these into more precise specifications. See Chapter 11, "Writing excellent requirements," for guidance on writing good functional requirements.

**Quality attributes** Statements that describe how well the system does something are quality attributes (see Chapter 14, "Beyond functionality"). Listen for words that describe desirable system characteristics: fast, easy, user-friendly, reliable, secure. You'll need to work with the users to understand just what they mean by these ambiguous and subjective terms so that you can write clear, verifiable quality goals. The following examples suggest what quality attributes might sound like when described by users:

- "The mobile software must respond quickly to touch commands."
- "The shopping cart mechanism has to be simple to use so my new customers don't abandon the purchase."

**External interface requirements** Requirements in this category describe the connections between your system and the rest of the universe. The SRS template in Chapter 10, "Documenting the requirements," includes sections for interfaces to users, hardware, and other software systems. Phrases such as "Must read signals from . . . ," "Must send messages to . . . ," "Must be able to read files in <format>," and "User interface elements must conform to <a standard>" indicate that the customer is describing an external interface requirement. Following are some examples:

- "The manufacturing execution system must control the wafer sorter."
- "The mobile app should send the check image to the bank after I photograph the check I'm depositing."

**Constraints** Design and implementation constraints legitimately restrict the options available to the developer (see Chapter 14). Devices with embedded software often must respect physical constraints such as size, weight, and interface connections. Phrases that indicate that the customer is describing a design or implementation constraint include: "Must be written in <a specific programming language>," "Cannot exceed <some limit>," and "Must use <a specific user interface control>." The following are examples of constraints that a customer might present:

- "Files submitted electronically cannot exceed 10 MB in size."
- "The browser must use 256-bit encryption for all secure transactions."

As with functional requirements, don't just transcribe the user's statement of a constraint. Ask why the constraint exists, confirm its validity, and record the rationale for including it as a requirement.

**Data requirements** Customers are presenting a data requirement whenever they describe the format, data type, allowed values, or default value for a data element; the composition of a complex business data structure; or a report to be generated (see Chapter 13, "Specifying data requirements"). Some examples of data requirements are as follows:

- "The ZIP code has five digits, followed by an optional hyphen and four digits that default to 0000."
- "An order consists of the customer's identity, shipping information, and one or more products, each of which includes the product number, number of units, unit price, and total price."

**Solution ideas** Many "requirements" from users are really solution ideas. Someone who describes a specific way to interact with the system to perform some action is suggesting a solution. The business analyst needs to probe below the surface of a solution idea to get to the real requirement. Repeatedly asking "why" the user needs it to work this way will likely reveal the true need (Wiegers 2006). For instance, passwords are just one of several possible ways to implement a security requirement. Two other examples of solution ideas are the following:

- "Then I select the state where I want to send the package from a drop-down list."
- "The phone has to allow the user to swipe with a finger to navigate between screens."

In the first example, the phrase *from a drop-down list* indicates that this is a solution idea because it's describing a specific user interface control. The prudent BA will ask, "Why from a drop-down list?" If the user replies, "That just seemed like a good way to do it," then the real requirement is something like, "The system shall permit the user to specify the state where he wants to send the package." But maybe the user says, "We do the same thing in several other places and I want it to be consistent. Also, the drop-down list prevents the user from entering invalid data." These are legitimate reasons to specify a specific solution. Recognize, though, that embedding a solution in a requirement imposes a design constraint on that requirement: it limits the requirement to being implemented in only one way. This isn't necessarily wrong or bad; just make sure the constraint is there for a good reason.

Classifying the customer input is just the beginning of the process to create requirements specifications. You still need to assemble the information into clearly stated and well-organized requirements collections. As you work through the information, craft clear individual requirements and store them in the appropriate sections of the team's document templates or repository. Make additional passes through this information to ensure that each statement demonstrates the characteristics of high-quality requirements as described in Chapter 11. As you process your elicitation notes, mark the items complete as you store them in the right place.

## How do you know when you're done?

No simple signal will indicate when you've completed requirements elicitation. In fact, you'll never be entirely done, particularly if you are deliberately implementing a system incrementally, as on agile projects. As people muse in the shower each morning and talk with their colleagues, they'll generate ideas for additional requirements and want to change some of the ones they already have. The following cues suggest that you're reaching the point of diminishing returns on requirements elicitation, at least for now. Perhaps you are done if:

- The users can't think of any more use cases or user stories. Users tend to identify user requirements in sequence of decreasing importance.
- Users propose new scenarios, but they don't lead to any new functional requirements. A "new" use case might really be an alternative flow for a use case you've already captured.
- Users repeat issues they already covered in previous discussions.

- Suggested new features, user requirements, or functional requirements are all deemed to be out of scope.
- Proposed new requirements are all low priority.
- The users are proposing capabilities that might be included "sometime in the lifetime of the product" rather than "in the specific product we're talking about right now."
- Developers and testers who review the requirements for an area raise few questions.

Amalgamating requirements input from numerous users is difficult without using a structured organizing scheme, such as use cases or the sections in an SRS template. Despite your best efforts to discover *all* the requirements, you won't, so expect to make changes as construction proceeds. Remember, your goal is to accumulate a shared understanding of requirements that is good enough to let construction of the next release or increment proceed at an acceptable level of risk.

## Some cautions about elicitation

Skill in conducting elicitation discussions comes with experience and builds on training in interviewing, group facilitation, conflict resolution, and similar activities. However, a few cautions will decrease the learning curve.

**Balance stakeholder representation** Collecting input from too few representatives or hearing the voice of only the loudest, most opinionated customer is a problem. It can lead to overlooking requirements that are important to certain user classes or to including requirements that don't represent the needs of a majority of the users. The best balance involves a few product champions who can speak for their respective user classes, with each champion backed up by other representatives from the same user class.

**Define scope appropriately** During requirements elicitation, you might find that the project scope is improperly defined, being either too large or too small. If the scope is too large, you'll accumulate more requirements than are needed to deliver adequate business and customer value, and the elicitation process will drag on. If the project is scoped too small, customers will present needs that are clearly important yet just as clearly lie beyond the limited scope currently established for the project. The current scope could be too small to yield a satisfactory product. Eliciting user requirements therefore can lead to modifying the product vision or the project scope.

**Avoid the requirements-versus-design argument** It's often stated that requirements are about *what* the system has to do, whereas *how* the solution will be implemented is the realm of design. Although attractively concise, this is an oversimplification. Requirements elicitation should indeed focus on the *what*, but there's a gray area—not a sharp line—between analysis and design (Wiegers 2006). Hypothetical *how*s help to clarify and refine the understanding of what users need. Analysis models, screen sketches, and prototypes help to make the needs expressed during elicitation more tangible and to reveal errors and omissions. Make it clear to users that these screens and prototypes are illustrative only, not necessarily the ultimate solution.

**Research within reason** The need to do exploratory research sometimes disrupts elicitation. An idea or a suggestion arises, but extensive research is required to assess whether it should even be considered for the product. Treat these explorations of feasibility or value as project tasks in their own right. Prototyping is one way to explore such issues. If your project requires extensive research, use an incremental development approach to explore the requirements in small, low-risk portions.

## Assumed and implied requirements

You will never document 100 percent of the requirements for a system. But the requirements you *don't* specify pose a risk that the project might deliver a solution different from what stakeholders expect. Two likely culprits behind missed expectations are assumed and implied requirements:

- *Assumed requirements* are those that people expect without having explicitly expressed them. What you assume as being obvious might not be the same as assumptions that various developers make.
- *Implied requirements* are necessary because of another requirement but aren't explicitly stated. Developers can't implement functionality they don't know about.

To reduce these risks, try to identify knowledge gaps waiting to be filled with implied and assumed requirements. Ask, "What are we assuming?" during elicitation sessions to try to surface those hidden thoughts. If you come across an assumption during requirements discussions, record it and confirm its validity. People often assume that things have to be the way they've always been because they're so familiar with an existing system or business process. If you're developing a replacement system, review the previous system's features to determine whether they're truly required in the replacement.

To identify implied requirements, study the results of initial elicitation sessions to identify areas of incompleteness. Does a vague, high-level requirement need to be fleshed out so the stakeholders all understand it? Is a requirement that might be part of a logical set (say, saving an incomplete web form) lacking its counterpart (retrieving a saved form for further work)? You might need to re-interview some of the same stakeholders to have them look for missing requirements (Rose-Coutré 2007). Also, think of new stakeholders who know the topic and can spot gaps.

Read between the lines to identify features or characteristics the customers expect to be included without having said so. Ask *context-free questions*, high-level and open-ended questions that elicit information about global characteristics of both the business problem and the potential solution (Gause and Weinberg 1989). The customer's response to questions such as "What kind of precision is required in the product?" or "Can you help me understand why you don't agree with Miguel's reply?" can lead to insights that questions with standard yes/no or A/B/C answers do not.

### No assumed requirements

I once encountered a development team that was implementing a content portal that was intended to do many things, including upload, edit, and publish content to a website. There were approximately 1,000 pieces of existing content, organized in a hierarchy. The content management team assumed that users would be able to navigate the hierarchy to quickly find a specific piece of content. They did not specify requirements regarding the user interface navigation. However, when the developers implemented the user interface to navigate to content, they organized all of the content in a single level, not hierarchically, and showed only 20 items per screen. To find a specific piece of content, a user might have to navigate through as many as 50 screens. A little more specification and dialogue between developers and the content management team could have avoided considerable rework.

## Finding missing requirements

Missing requirements constitute a common type of requirement defect. Missing requirements are hard to spot because they're invisible! The following techniques will help you detect previously undiscovered requirements:

- Decompose high-level requirements into enough detail to reveal exactly what is being requested. A vague, high-level requirement that leaves much to the reader's interpretation will lead to a gap between what the requester has in mind and what the developer builds.
- Ensure that all user classes have provided input. Make sure that each user requirement has at least one identified user class who will receive value from the requirement.
- Trace system requirements, user requirements, event-response lists, and business rules to their corresponding functional requirements to make sure that all the necessary functionality was derived.
- Check boundary values for missing requirements. Suppose that one requirement states, "If the price of the order is less than $100, the shipping charge is $5.95" and another says, "If the price of the order is more than $100, the shipping charge is 6 percent of the total order price." But what's the shipping charge for an order with a price of exactly $100? It's not specified, so a requirement is missing, or at least poorly written.
- Represent requirements information in more than one way. It's difficult to read a mass of text and notice the item that's absent. Some analysis models visually represent requirements at a high level of abstraction—the forest, not the trees. You might study a model and realize that there should be an arrow from one box to another; that missing arrow represents a missing requirement. Analysis models are described in Chapter 12, "A picture is worth 1024 words."

- Sets of requirements with complex Boolean logic (ANDs, ORs, and NOTs) often are incomplete. If a combination of logical conditions has no corresponding functional requirement, the developer has to deduce what the system should do or chase down an answer. "Else" conditions frequently are overlooked. Represent complex logic by using decision tables or decision trees to cover all the possible situations, as described in Chapter 12.
- Create a checklist of common functional areas to consider for your projects. Examples include error logging, backup and restore, access security, reporting, printing, preview capabilities, and configuring user preferences. Periodically compare this list with the functions you've already specified to look for gaps.
- A data model can reveal missing functionality. All data entities that the system will manipulate must have corresponding functionality to create them, read them from an external source, update current values, and/or delete them. The acronym CRUD is often used to refer to these four common operations. Make sure you can identify functionality in your application to perform these operations on all of your entities that need them (see Chapter 13).

**Trap** Watch out for the dreaded *analysis paralysis*, spending too much time on requirements elicitation in an attempt to avoid missing any requirements.

You'll likely never discover all of the requirements for your product, but nearly every software team can do a better job of requirements elicitation by applying the practices described in this chapter.


## Next steps

- Think about requirements that were found late on your last project. Why were they overlooked during elicitation? How could you have discovered each of these requirements earlier? What would that have been worth to your organization?
- Select a portion of any documented customer input on your project or a section from the SRS. Classify every item in that requirements fragment into the categories shown in Figure 7-7. If you find items that were organized incorrectly, move them to the correct place in your requirements documentation.
- List the requirements elicitation techniques used on your previous or current project. Which ones worked well? Why? Which ones did not work so well? Why not? Identify elicitation techniques that you think would work better and decide how you'd apply them next time. Identify any barriers you might encounter to making those techniques work, and brainstorm ways to overcome those barriers.