Chapter 3

Requirements Elicitation

Introduction

In this chapter we explore the many ways that requirements can be found, discovered, captured, or coerced. In this context, all of these terms are synonymous with "elicitation." But "gathering" is not equivalent. Requirements are not like fallen fruit to be simply retrieved and placed in a bushel. No, requirements are usually not so easy to come by, at least not all of them. Many of the more subtle and complex ones have to be teased out through rigorous, if not dogged, processes.

There are many techniques that you can choose to conduct requirements elicitation, and you will probably need to use more than one, and likely different ones for different classes of users/stakeholders. The techniques that we discuss are:

- Brainstorming
- Card Sorting
- Designer as Apprentice
- Domain Analysis
- Ethnographic Observation
- Goal-Based Approaches
- Group Work
- Interviews
- Introspection
- Joint Application Design (JAD)
- Laddering
- Protocol Analysis
- Prototyping
- Quality Function Deployment (QFD)

- Questionnaires/surveys
- Repertory Grids
- Scenarios
- Task Analysis
- User Stories
- Viewpoints
- Workshops

This list is based on one suggested by Zowghi and Coulin (1998).

Preparing for Requirements Elicitation

Identifying all customers and stakeholders is the first step in preparing for requirements elicitation. But stakeholder groups, and especially customers, can be nonhomogeneous, and therefore you need to treat each subgroup differently. For example, the different subclasses of user for the pet store POS system include:

- Cashiers
- Managers
- System maintenance personnel
- Store customers
- Inventory/warehouse personnel
- Accountants (to enter tax information)
- Sales department (to enter pricing and discounting information)
- And possibly others

Each of these subgroups of users has different desiderata and these need to be determined. The process, then, to prepare for elicitation is:

- Identify all customers and stakeholders.
- Partition customers into classes according to interests, scope, authorization, or other discriminating factors (some classes may need multiple levels of partitioning).
- Select a champion or representative group for each user class and stakeholder group.
- Select the appropriate technique(s) to solicit initial inputs from each class or stakeholder group.

Here is another example of user class partitioning. There are many different stakeholders for the baggage handling system:

- Travelers
- System maintenance personnel
- Baggage handlers

- Airline schedulers/dispatchers
- Airport personnel
- Airport managers and policy makers
- And many others

In order to elicit requirements for the baggage handling system discussed in the text, consider the following partitioning:

- Children
- Senior citizens
- Business people
- Casual travelers
- Military personnel
- Civilians
- Frequent fliers
- And so on

Each of these subclasses may need to be approached with different elicitation techniques. For example, surveys may not be appropriate for children and focus groups may be less useful for military personnel. Many of these subclasses overlap: for example, a person can be a business traveler and also a casual traveler and these overlaps need to be taken into consideration when analyzing the data from the elicitation activities.

Elicitation Techniques Survey

Now it is time to begin examining the elicitation techniques. We offer these techniques in alphabetical order; no preference is implied. At the end of the chapter, we discuss the prevalence and suitability of these techniques in different situations.

Brainstorming

Brainstorming consists of informal sessions with customers and other stakeholders to generate overarching goals for the systems. Brainstorming can be formalized to include a set agenda, minutes taking, and the use of formal structures (e.g., Roberts Rules of Order). But the formality of a brainstorming meeting is probably inversely proportional to the creative level exhibited at the meeting. These kinds of meetings probably should be informal, even spontaneous, with the only structure embodying some recording of any major discoveries.

During brainstorming sessions, some preliminary requirements may be generated, but this aspect is secondary to the process. The JAD technique incorporates brainstorming (and a whole lot more), and it is likely that most

other group-oriented elicitation techniques implicitly embody some form of brainstorming. Brainstorming is also useful for general objective setting, such as mission or vision statement generation.

Card Sorting

This technique involves having stakeholders complete a set of cards that includes key information about functionality for the system/software product. It is also a good idea for the stakeholders/customers to include ranking and rationale for each of the functionalities.

The period of time to allow customers and stakeholders to complete the cards is an important decision. Although the exercise of card sorting can be completed in a few hours, rushing the stakeholders will likely lead to important missing functionalities. Giving stakeholders too much time, however, can slow the process unnecessarily. It is recommended that a minimum of one week (and no more than two weeks) be allowed for the completion of the cards. Another alternative is to have the customers complete the cards in a two-hour session, then return one week later for another session of card completion and review.

In any case, after each session of card generation the requirements engineer organizes these cards in some manner, generally clustering the functionalities logically. These clusters form the basis of the requirements set. The sorted cards can also be used as an input to the process to develop CRC (capability, responsibility, class) cards to determine program classes in the eventual code. Another technique (discussed shortly), QFD, includes a card-sorting activity.

To illustrate the process, Figure 3.1 depicts a tiny subset of cards generated by the customer for the pet store POS system, lying in an unsorted pile. In this case,

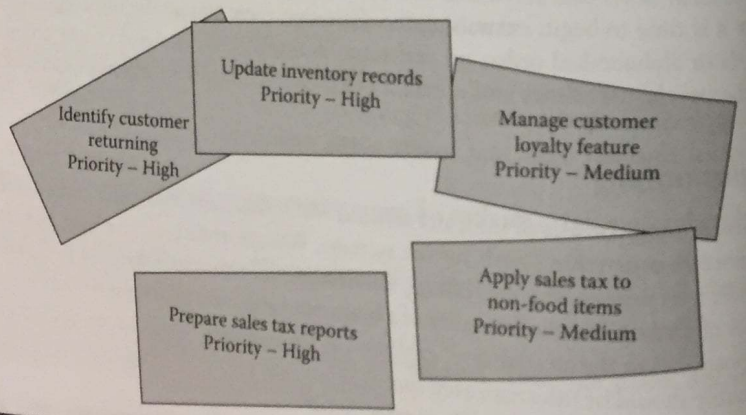

Figure 3.1 A tiny subset of the unsorted cards generated by customers for the pet store POS system.

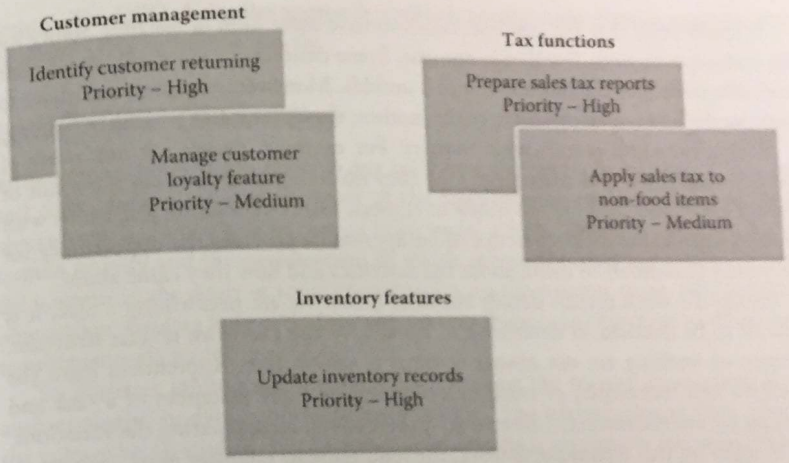

Figure 3.2 Sorted cards for the pet store POS system.

each card contains only a brief description of the functionality and a priority rating is included (for brevity, no rationale is shown).

The requirements engineer analyzes this pile of cards and decides that two of the cards pertain to "customer management" functions, two cards to "tax functions," and one card to "inventory features" or functions, and arranges the cards in appropriate piles as shown in Figure 3.2.

The customer can be shown this sorted list of functionalities for correction or missing features. Then, a new round of cards can be generated if necessary. The process continues until the requirements engineer and customer are satisfied that the system features are substantially captured.

Designer as Apprentice*

Designer as apprentice is a requirements discovery technique in which the requirements engineer "looks over the shoulder" of the customer in order to learn enough about the customer's work to understand his needs. The relationship between customer and designer is like that between a master craftsman and apprentice. That is, the apprentice learns a skill from the master just as the requirements engineer (the designer) learns about the customer's work from the customer. The apprentice is there to learns whatever the master knows (and therefore must guide the customer in talking about and demonstrating those parts of the work). The designer is there to address specific needs.

* This discussion is adapted from one found in Laplante (2006), with permission.

It might seem that the customer needs to have some kind of teaching ability for this technique to work, but that is not true. Some customers cannot talk about their work effectively, but can talk about it as it unfolds. Moreover, customers don't have to work out the best way to present it, or the motives; they just explain what they're doing.

Seeing the work reveals what matters. For example, people are not aware of everything they do and sometimes why they do it. Some actions are the result of years of experience and are too subtle to express. Other actions are just habits with no valid justification. The presence of an apprentice provides the opportunity for the master (customer) to think about the activities and how they came about.

Seeing the work reveals details because, unless we are performing a task, it is difficult to be detailed in describing it. Finally, seeing the work reveals structure. Patterns of working are not always obvious to the worker. Apprentices learn the strategies and techniques of work by observing multiple instances of a task and forming an understanding of how to do it themselves, incorporating the variations.

In order for this technique to work, the requirements engineer must understand the structure and implication of the work, including:

- The strategy to get work done
- Constraints that get in the way
- The structure of the physical environment as it supports work
- The way work is divided
- Recurring patterns of activity
- The implications these have on any potential system

The designer must demonstrate an understanding of the work to the customer so that any misunderstandings can be corrected.

Finally, using the designer as apprentice approach provides other project benefits beyond requirements discovery. For example, using this technique can help improve the process that is being modeled.

Both customer and designer learn during this process: the customer learns what may be possible and the designer expands his understanding of the work. If the designer has an idea for improving the process, however, this must be fed back to the customer immediately (at the time).

Domain Analysis

We have already emphasized the importance of having domain knowledge (whether it is had by the requirements engineer or the customer) in requirements engineering. Domain analysis involves any general approach to assessing the "landscape" of related and competing applications to the system being designed. Such an approach can be useful in identifying essential functionality and, later, missing functionality. Domain analysis can also be used downstream for identifying reusable components (such as open source software elements that can be incorporated into the final design). The QFD elicitation approach explicitly incorporates domain analysis, and we discuss this technique shortly.

Ethnographic Observation

Ethnographic observation refers to any technique in which observation of indirect and direct factors informs the work of the requirements engineer. Ethnographic observation is a technique borrowed from social science in which observations of human activity and the environment in which the work occurs are used to inform the scientist in the study of some phenomenon. In the strictest sense, ethnographic observation involves long periods of observation (hence, an objection to its use as a requirements elicitation technique).

To illustrate ethnographic observation, imagine the societal immersion of an anthropologist studying some different culture. The anthropologist lives among the culture being studied, but in a way in which she is minimally intrusive. While eating, sleeping, hunting, celebrating, mourning, and so on within the culture, all kinds of direct and indirect evidence of how that society functions and its belief systems are collected.

In applying ethnographic observation to requirements elicitation, the requirements engineer immerses herself in the workplace culture of the customer. Here, in addition to observing work or activity to be automated, the requirements engineer is also in a position to collect evidence of customer needs derived from the surroundings that may not be communicated directly. Designer as apprentice is one requirements elicitation technique that includes the activity of ethnographic observation.

To illustrate this technique in practice, consider this situation in which ethnographic observation occurs:

- You are gathering requirements for a smart home for some customer.
- You spend long periods of time interviewing the customer about what he wants.
- You spend time interacting with the customer as he goes about his day and ask questions ("Why are you running the dishwasher at night? Why not in the morning?").
- You spend long periods of time passively observing the customer "in action" in his current home to get nonverbal clues about his wants and desires.
- You gain other information from the home itself: the books on the bookshelf, paintings on the wall, furniture styles, evidence of hobbies, signs of wear and tear on various appliances, and so on.

Ethnographic observation can be very time consuming and requires substantial training of the observer to be useful. There is another objection, based on the intrusiveness of the process. There is a well-known principle in physics known as the Heisenberg uncertainty principle, which, in layperson's terms, means that you can't

precisely measure something without affecting that which you are measuring. So, for example, when you are observing the work environment for a client, processes and behaviors change because everyone is out to impress, so an incorrect picture of the situation is formed, leading to flawed decisions down the line.

Goal-Based Approaches

Goal-based approaches comprise any elicitation techniques in which requirements are recognized to emanate from the mission statement, through a set of goals that lead to requirements. That is, looking at the mission statement, a set of goals that fulfill that mission is generated. These goals may be subdivided one or more times to obtain lower-level goals. Then, the lower-level goals are branched out into specific high-level requirements. Finally, the high-level requirements are used to generate lower-level ones.

For example, consider the baggage handling system mission statement:

> To automate all aspects of baggage handling from passenger origin to destination.

The following goals might be considered to fulfill this mission:

- Goal 1: To completely automate the tracking of baggage from check-in to pick-up
- Goal 2: To completely automate the routing of baggage from check-in counter to plane
- Goal 3: To reduce the amount of lost luggage to 1%

These goals can then be decomposed into requirements using a structured approach such as the goal-question-metric (GQM). GQM is an important technique used in many aspects of systems engineering such as requirements engineering, architectural design, systems design, and project management. GQM incorporates three steps: state the system's objectives or goals; derive from each goal the questions that must be answered to determine if the goal is being met; and decide what must be measured in order to be able to answer the questions (Basili and Weiss 1984).

For example, in the case of the baggage handling system, consider goal 3. Here the related question is "What percentage of luggage is lost for a given (airport/airline/flight/time period/etc.)?" This question suggests a requirement of the form:

> The percentage of luggage lost for a given (airport/airline/flight/time period/etc.) shall be not greater than 1%.

The associated metric for this requirement, then, is simply the percentage of luggage lost for a particular (airport/airline/flight/time period/etc.). Of course, we really need a definition for "lost luggage," inasmuch as so-called lost luggage often

reappears days or even months after it is declared lost. Also, reasonable assumptions need to be made in framing this requirement in terms of an airport's reported luggage losses over some time period, or for a particular airline at some terminal, and so forth.

In any case, we deliberately picked a simple example here; the appropriate question for some goal (requirement) is not always so obvious, nor is the associated metric so easily derived from the question. Here is where GQM really shows its strength.

For example, it is common to see requirements that are a variation of the following:

> The system shall be user friendly.

The problem with such a requirement is that there is no way to demonstrate its satisfaction. Any person on the acceptance testing team can declare that the system is not user friendly. But user friendliness is a reasonable goal for the system. So, following GQM, we create a series of questions that pertain to the goal of user friendliness, for example:

1. How easy is the system to learn to use?
2. How much help does a new user need?
3. How many errors does a user get?

Next, we define one or more metrics for each of these questions. Let's generate one for each:

1. The time it takes a user to learn how to perform certain functions
2. The number of times a user has to use the "help" feature in some period of time
3. The number of times a user sees an "error" message during certain operations in a period of time.

Finally, we work with the customer to set acceptable ranges for these metrics. After the system is built, requirements satisfaction can be demonstrated through testing trials with real users. The parameters of this testing, such as the characteristics and number of users, duration of testing, and so on, can be defined later and incorporated in the System Test Plan. In this way we can define an acceptable level of "user friendliness."

Group Work

Group work is a general term for any kind of group meeting that is used during the requirements discovery, analysis, and follow-up processes. The most celebrated of

group-oriented work for requirements elicitation is joint application design (JAD), which we discuss shortly.

Group activities can be very productive in terms of bringing together many stakeholders, but risk the potential for conflict and divisiveness. The key to success in any kind of group work is in the planning and execution of the group meetings. Here are the most important things to remember about group meetings:

- Do your homework: research all aspects of the organization, problems, politics, environment, and so on.
- Publish an agenda (with time allotted for each item) several days before the meeting occurs.
- Stay on the agenda throughout the meeting (no meeting scope creep).
- Have a dedicated note-taker (scribe) on hand.
- Do not allow personal issues to creep in.
- Allow all to have their voices heard.
- Look for consensus at the earliest opportunity.
- Do not leave until all items on the agenda have received sufficient discussion.
- Publish the minutes of the meeting within a couple of days of meeting close and allow attendees to suggest changes.

These principles will come into play for the JAD approach to requirements elicitation.

Group work of any kind has many drawbacks. First, group meetings can be difficult to organize and get the many stakeholders involved to focus on issues. Problems of openness and candor can occur as well because people are not always eager to express their true feelings in a public forum. Because everyone has a different personality, certain individuals can dominate the meeting (and these may not be the most "important" individuals). Allowing a few to own the meeting can lead to feelings of being "left out" for many of the other attendees.

Running effective meetings, and hence using group work, requires highly developed leadership, organizational, and interpersonal skills. Therefore, the requirements engineer should seek to develop these skills whenever possible.

Interviews

The "opposite" of group activities is the one-on-one (or small group) interview. This is an obvious and easy-to-use technique to extract system requirements from a customer.

There are three kinds of interviews that can be used in elicitation activities:

- Unstructured
- Structured
- Semi-structured

Unstructured interviews, which are probably the most common type, are conversational in nature and serve to relax the customer. Like a spontaneous "confession" these can occur any time and any place whenever the requirements engineer and customer are together, and the opportunity to capture information this way should never be lost. But depending on the skill of the interviewer, unstructured interviews can be hit or miss. Therefore, structured or semi-structured interviews are preferred.

Structured interviews are much more formal in nature, and they use predefined questions that have been rigorously planned. Templates are very helpful when employed with interviewing using the structured style. The main drawback to structured interviews is that some customers may withhold information because the format is too controlled.

Semi-structured interviews combine the best of structured and unstructured interviews. That is, the requirements engineer prepares a carefully thought-out list of questions, but then allows for spontaneous unstructured questions to creep in during the course of the interview.

Although structured interviews are preferred, the choice of which one to use is very much an opportunistic decision. For example, when the client's corporate culture is very informal and relaxed, and trust is high, then unstructured interviews might be preferred. In a stodgier, process-oriented organization, structured and semi-structured interviews are probably more desirable.

Here are some sample interview questions that can be used in any of the three interview types:

- Name an essential feature of the system.
- Why is this feature important?
- On a scale of one to five, five being most important, how would you rate this feature?
- How important is this feature with respect to other features?
- What other features are dependent on this feature?
- What other features must be independent of this feature?
- What other observations can you make about this feature?

Whatever interview technique is used, care must be taken to ensure that all of the right questions are asked. That is, leave out no important questions, and include no extraneous, offensive, or redundant questions. When absolutely necessary, interviews can be done via telephone, videoconference, or e-mail, but be aware that, in these modes of communication, certain important nuanced aspects to the responses may be lost.

Introspection

When a requirements engineer develops requirements based on what he "thinks" the customer wants, then he is conducting the process of introspection. In essence

the requirements engineer puts himself in the place of the customer and opines, "If I were the customer I would want the system to do this …."

An introspective approach is useful when the requirements engineer's domain knowledge far exceeds the customer's. Occasionally, the customer will ask the engineer questions similar to the following, "If you were me, what would you want?" Introspection will inform every aspect of the requirements engineer's interactions, but remember our admonition about not telling a customer what she ought to want.

Joint Application Design (JAD)*

Joint application design (JAD) involves highly structured group meetings or mini-retreats with system users, system owners, and analysts in a single venue for an extended period of time. These meetings occur four to eight hours per day and over a period lasting one day to a couple of weeks.

JAD and JAD-like techniques are becoming increasingly common in systems planning and systems analysis to obtain group consensus on problems, objectives, and requirements. Specifically, software engineers can use JAD for:

- Eliciting requirements and for the software requirements specification
- Design and software design description
- Code
- Tests and test plans
- Users' manuals

There can be multiple reviews for each of these artifacts, if necessary. But JAD reviews are especially important as a requirements elicitation tool.

Planning for a JAD review or audit session involves three steps:

1. Selecting participants
2. Preparing the agenda
3. Selecting a location

Great care must be taken in preparing each of these steps.

Reviews and audits may include some or all of the following participants:

- Sponsors (e.g., senior management)
- A team leader (facilitator, independent)
- Users and managers who have ownership of requirements and business rules
- Scribes
- Engineering staff

The sponsor, analysts, and managers select a leader. The leader may be in-house or a consultant. One or more scribes (note-takers) are selected, normally from the

* Ibid.

software development team. The analyst and managers must select individuals from the user community. These individuals should be knowledgeable and articulate in their business areas.

Before planning a session, the analyst and sponsor must determine the scope of the project and set the high-level requirements and expectations of each session. The session leader must also ensure that the sponsor is willing to commit people, time, and other resources to the effort. The agenda depends greatly on the type of review to be conducted and should be constructed to allow for sufficient time. The agenda, code, and documentation must also be sent to all participants well in advance of the meeting so that they have sufficient time to review them, make comments, and prepare to ask questions.

The following are some rules for conducting software requirements, design audits, or code walkthrough. The session leader must make every effort to ensure these practices are implemented:

- Stick to agenda.
- Stay on schedule (agenda topics are allotted specific time).
- Ensure that the scribe is able to take notes.
- Avoid technical jargon (if the review involves nontechnical personnel).
- Resolve conflicts (try not to defer them).
- Encourage group consensus.
- Encourage user and management participation without allowing individuals to dominate the session.
- Keep the meeting impersonal.
- Allow the meetings to take as long as necessary.

The end product of any review session is typically a formal written document providing a summary of the items (specifications, design changes, code changes, and action items) agreed upon during the session. The content and organization of the document obviously depend on the nature and objectives of the session. In the case of requirements elicitation, however, the main artifact could be a first draft of the SRS.

Laddering

In laddering, the requirements engineer asks the customer short prompting questions ("probes") to elicit requirements. Follow-up questions are then posed to dig deeper below the surface. The resultant information from the responses is then organized into a treelike structure.

To illustrate the technique, consider the following sequence of laddering questions and responses for the pet store POS system. "RE" refers to the requirements engineer:

RE: Name a key feature of the system.
Customer: Customer identification.

RE: How do you identify a customer?
Customer: They can swipe their loyalty card.
RE: What if a customer forgets the card?
Customer: They can be looked up by phone number.
RE: When do you get the customer's phone number?
Customer: When customers complete the application for the loyalty card.
RE: How do customers complete the applications?

And so on. Figure 3.3 shows how the responses to the questions are then organized in a ladder or hierarchical diagram. The laddering technique assumes that information can be arranged in a hierarchical fashion, or, at least, it causes the information to be arranged hierarchically.

Protocol Analysis

A protocol analysis is a process where customers, together with the requirements engineers, walk through the procedures they are going to automate. During such a walk-through, the customers explicitly state the rationale for each step that is being taken.

As shown shortly, this technique is very similar to designer as apprentice, however, there are subtle differences. These differences lie in the role of the requirements engineer who is more passive in protocol analysis than in designer as apprentice.

Prototyping

Prototyping involves construction of models of the system in order to discover new features. Prototypes can involve working models and nonworking models. Working models can include working code in the case of software systems and simulations

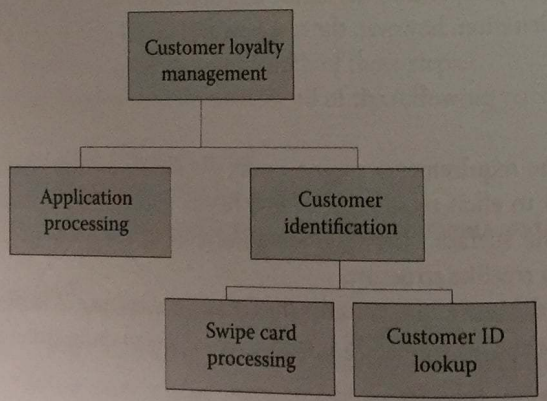

Figure 3.3 Laddering diagram for the pet store POS system.

or temporary or to-scale prototypes for nonsoftware systems. Nonworking models can include storyboards and mock-ups of user interfaces. Building architects use prototypes regularly (e.g., scale drawings, cardboard models, 3-D computer animations) to help uncover and confirm customer requirements. Systems engineers use prototypes for the same reasons.

In the case of working code prototypes, the code can be deliberately designed to be throwaway or it can be deliberately designed to be reused (nonthrowaway). For example, graphical user interface code mock-ups can be useful for requirements elicitation and the code can be reused. And agile software development methodologies incorporate a process of continuously evolving nonthrowaway prototypes.

In unfortunate cases, prototypes that were not intended to be kept, are in fact kept because of schedule pressures. This situation is potentially dangerous, because the code was likely not designed using the most rigorous techniques. The unintended reuse of throwaway prototypes occurs often in industry.

Prototyping is a particularly important technique for requirements elicitation. It is used extensively, for example, in the spiral software development model, and agile methodologies consist essentially of a series of increasingly functional nonthrowaway prototypes.

Both the Neill and Laplante (2003) and Marinelli (2008) surveys found that prototypes were widely used in industry, in 60% and 76.4% of reported projects, respectively. There are a number of different ways to use prototyping—for example, within a fourth-generation environment (i.e., a simulator), throw-away prototyping, evolutionary prototyping (where the prototype evolves into the final system), or user interface prototyping—and many organizations use more than one type. The aforementioned surveys' relative selection frequency for these different techniques is shown in Figure 3.4.

It is interesting to note that the use of evolutionary prototyping reported is high and increasing from the 2003 to 2008 surveys. Yet the 2008 survey does not show a corresponding increase in the use of agile software development methodologies, which incorporate evolutionary prototyping. The explanation is likely that software developers are using certain agile practices, such as evolutionary prototyping and test-driven development, but not strictly adopting agile process models.

Quality Function Deployment*

Quality function deployment (QFD) is a technique for discovering customer requirements and defining major quality assurance points to be used throughout the production phase. QFD provides a structure for ensuring that customers' needs and desires are carefully heard, then directly translated into a company's internal technical requirements, from analysis through implementation to deployment. The

* Ibid.

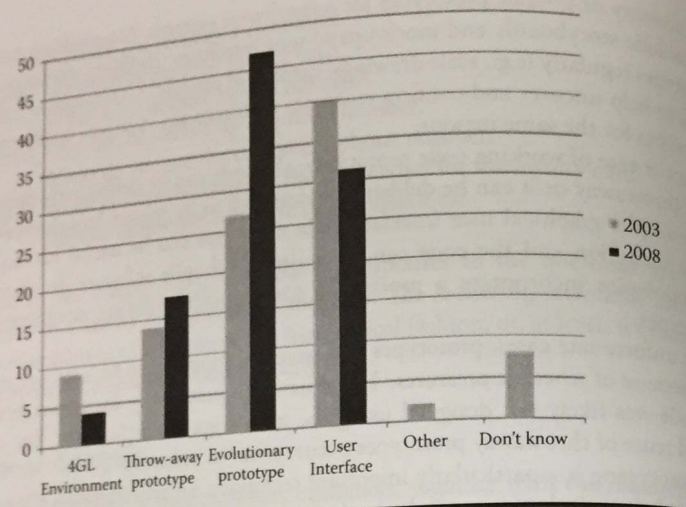

Figure 3.4 Type of prototyping performed [Neill and Laplante (2003) and Marinelli (2008) surveys].

basic idea of QFD is to construct relationship matrices among customer needs, technical requirements, priorities, and (if needed) competitor assessment. In essence, QFD incorporates card sorting and laddering and domain analysis.

Because these relationship matrices are often represented as the roof, ceiling, and sides of a house, QFD is sometimes referred to as the "house of quality" (Figure 3.5; Akao 1990). QFD was introduced by Yoji Akao in 1966 for use in manufacturing, heavy industry, and systems engineering. It has also been applied to software systems by IBM, DEC, HP, AT&T, Texas Instruments, and others.

When we refer to the "voice of the customer" we mean that the requirements engineer must empathically listen to customers to understand what they need from the product, as expressed by the customer in his words. The voice of the customer forms the basis for all analysis, design, and development activities, to ensure that products are not developed from only "the voice of the engineer." This approach embodies the essence of requirements elicitation.

The following requirements engineering process is prescribed by QFD:

- Identify stakeholder's attributes or requirements.
- Identify technical features of the requirements.
- Relate the requirements to the technical features.
- Conduct an evaluation of competing products.
- Evaluate technical features and specify a target value for each feature.
- Prioritize technical features for development effort.

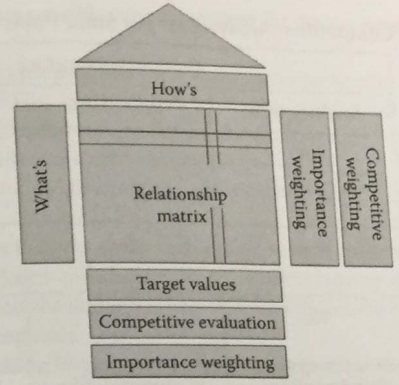

Figure 3.5 QFD's "house of quality." (Reprinted from Y. Akao, *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Cambridge, MA: Productivity Press, 1990.)

QFD uses a structured approach to competitive analysis. That is, a feature list is created from the union of all relevant features for competitive products. These features comprise the columns of a competition matrix. The rows represent the competing products and the corresponding cells are populated for those features included in each product. The matrix can be then used to formulate a starter set of requirements for the new or revised product. The matrix also helps to ensure that key features are not omitted from the new system and can contribute to improving the desirable quality of requirements completeness.

To illustrate, a partial competitive analysis for the pet store POS system is shown in Table 3.1. Notice that only very high level features are shown, although we could drill down to whatever level of detail is desired, greatly expanding the matrix. The matrix gives us a starter set of mandatory and optional features. For example, noting that wireless support is found in all these products might indicate that such a feature is mandatory in the new pet store POS system.

Because it incorporates a total life-cycle approach to requirements engineering, QFD has several advantages over other standalone elicitation techniques. QFD improves the involvement of users and managers. It shortens the development life cycle and improves overall project development. QFD supports team involvement by structuring communication processes. Finally, it provides a preventive tool that avoids the loss of information.

There are some drawbacks to QFD, however. For example, there may be difficulties in expressing temporal requirements. And QFD is difficult to use with an entirely new project type: how do you discover customer requirements for

Table 3.1 Partial Competitive Analysis for Pet Store Point of Sale System

	Competing Product		
Feature	*MyFavoritePet*	*BestFriends*	*Fido-2.0*
Maximum simultaneous users supported	100	250	Unlimited
Wireless device support	Yes	Yes	Yes
Business analytics features	Yes	Yes	No
Operating system support	Windows/Mac/ Linux	Windows/Linux	Windows/Mac
Cost (base system) ($K)	50	110	75

something that does not exist and how do you build and analyze the competitive products? In these cases the solution is to look at similar or related products, but still there is apt to be a cognitive gap.

Sometimes it is hard to find measurements for certain functions and to keep the level of abstraction uniform. And, the less we know, the less we document. Finally, as the feature list grows uncontrollably, the house of quality can become a "mansion."

Even if QFD is not used as the primary requirements elicitation approach, its approach to competitive systems analysis should be employed wherever possible. The structured nature of the QFD competitive analysis is an effective way to ensure that no important requirements are missing, leading to a more complete requirements set.

Questionnaires/Surveys

Requirements engineers often use questionnaires and other survey instruments to reach large groups of stakeholders. Surveys are generally used at early stages of the elicitation process to define the scope boundaries quickly.

Survey questions of any type can be used. For example, questions can be closed (e.g., multiple choice, true–false) or open-ended, involving free-form responses. Closed questions have the advantage of easier coding for analysis, and they help to bound the scope of the system. Open questions allow for more freedom and innovation, but can be harder to analyze and can encourage scope creep.

For example, some possible survey questions for the pet store POS system are:

- How many unique products (SKUs) do you carry in your inventory?
  - (a) 0–1,000 (b) 1,001–10,000 (c) 10,001–100,000 (d) >100,000
- How many different warehouse sites do you have? _____
- How many different store locations do you have? _____
- How many unique customers do you currently have? _____

There is a danger in overscoping and underscoping if questions are not adequately framed, even for closed-ended questions. Therefore, survey elicitation techniques are most useful when the domain is very well understood by both stakeholders and the requirements engineer.

Before undertaking large-scale surveys, it is important to conduct a pilot study with a small subset of the intended survey population. The results are analyzed and the survey participants interviewed for the purpose of identifying confusing, missing, or extraneous questions. Then the instrument can be refined before administering the survey to the greater population.

In analyzing survey data, particularly when asking participants to identify and rank desirable features, be careful of the following effect.

> When given a set of choices that do not have to be realized, a person will tend to desire a much larger number of options then if the decision were actually to be made.

We call this effect the "Ice Cream Store Effect" because of the following example. Consider an entrepreneur who decides to open a handmade ice cream shop. As part of her product research, she surveys a number of people with an instrument in which the respondents check off the flavors of ice cream they would purchase. She finds that of the 30 different flavors listed in the survey, 20 of the flavors are selected by 50% of the respondents or more. Thus, she decides to produce and stock these 20 flavors roughly in proportion to the demand indicated by the survey results. Yet after one week of opening her ice cream store she discovers that 90% of her business is due to the top three flavors: chocolate, vanilla, and strawberry. Of the 17 other flavors from the survey she keeps in her inventory, several have never even been purchased. She realizes that even though customers said they would buy these flavors in the survey, when it came time to exercise their choice, they behaved differently. Therefore, remember the Ice Cream Store Effect when giving customers choices about feature sets; they will say one thing and do another.

Surveys can be conducted via telephone, e-mail, in person, and using web-based technologies. There are a variety of commercial tools and open source solutions that are available to simplify the process of building surveys and collecting and analyzing results that should be employed.

Repertory Grids

Repertory grids incorporate a structured ranking system for various features of the different entities in the system and are typically used when the customers are domain experts. Repertory grids are particularly useful for identification of agreement and disagreement within stakeholder groups.

The grids look like a feature or quality matrix in which rows represent system entities and desirable qualities and columns represent rankings based on each of the stakeholders. Although the grids can incorporate both qualities and features, it is usually the case that the grids have all features or all qualities to provide for consistency of analysis and dispute resolution.

To illustrate the technique, Figure 3.6 represents a repertory grid for various qualities of the baggage handling system. Here we see that for the airport operations manager, all qualities are essentially of highest importance (safety is rated as slightly lower, at 4). But for the Airline Worker's Union representative, safety is the most important (after all, his union membership has to interact with the system on a daily basis).

In essence, these ratings reflect the agendas or differing viewpoints of the stakeholders. Therefore, it is easy to see why the use of repertory grids can be very helpful in confronting disputes involving stakeholder objectives early. In addition, the grids can provide valuable documentation for dealing with disagreements later in the development of the system because they capture the attitudes of the stakeholders about qualities and features in a way that is hard to dismiss. Still, when using repertory grids remember the Ice Cream Store Effect: stakeholders will say one thing in a public setting and then act differently later.

Scenarios

Scenarios are informal descriptions of the system in use that provide a high-level description of system operation, classes of users, and exceptional situations.

Baggage handling speed	1	1	5
Fault-tolerance	4	5	5
Safety	5	4	4
Reliability	3	5	5
Ease of maintenance	3	5	5

1 = lowest importance

Airline workers union rep

Maintenance engineer

Airport operations manager

Figure 3.6 Partial repertory grid for the baggage handling system.

Here is a sample scenario for the pet store POS system.

> A customer walks into the pet store and fills his cart with a variety of items. When he checks out, the cashier asks if the customer has a loyalty card. If he does, she swipes the card, authenticating the customer. If he does not, then she offers to complete one for him on the spot.
>
> After the loyalty card activity, the cashier scans products using a bar code reader. As each item is scanned, the sale is appropriately totaled and the inventory is appropriately updated. Upon completion of product scanning a subtotal is computed. Then any coupons and discounts are entered. A new subtotal is computed and applicable taxes are added. A receipt is printed and the customer pays using cash, credit card, debit card, or check. All appropriate totals (sales, tax, discounts, rebates, etc.) are computed and recorded.

Scenarios are quite useful when the domain is novel (consider a scenario for the Space Station, e.g.). User stories are, in fact, a form of scenario.

Task Analysis

As do many of the hierarchically oriented techniques that we have studied already, task analysis involves a functional decomposition of tasks to be performed by the system. That is, starting at the highest level of abstraction, the designer and customers elicit further levels of detail. This detailed decomposition continues until the lowest level of functionality (single task) is achieved.

As an example, consider the partial task analysis for the pet store POS system shown in Figure 3.7. Here the overarching "pet store POS system" is deemed to consist of three main tasks: inventory control, sales, and customer management. Drilling down under the sales functions, we see that these consist of the tasks of tax functions and purchase transactions. Next proceeding to the purchase transaction function, we decompose these tasks into sales, refunds, and discounts and coupons.

The task analysis and decomposition continues until a sufficient level of granularity is reached (typically, to the level of a method or nondecomposable procedure) and the diagram is completed.

User Stories*

User stories are short conversational texts that are used for initial requirements discovery and project planning. User stories are widely employed in conjunction with agile methodologies. User stories are closely related to use cases.

User stories are written by the customers in terms of what the system needs to do for them and in their own "voice." User stories usually consist of two to

* Ibid.

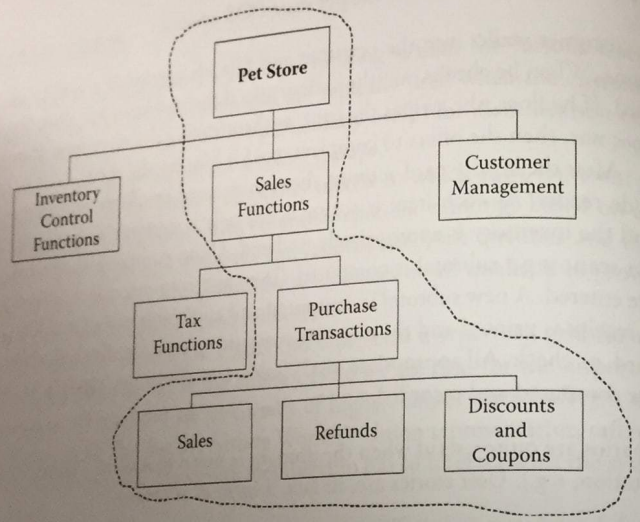

Figure 3.7 Partial task analysis for the pet store POS system.

four sentences written on a three-by-five-inch card. About 80 user stories is usually appropriate for one system increment or evolution, but the appropriate number will vary widely depending on the application size and scope and development methodology to be used (e.g., agile versus incremental).

An example of a user story for the pet store POS system is as follows:

- Each customer should be able to check out at a register easily.
- Self-service shall be supported.
- All coupons, discounts, and refunds should be handled this way.

User stories should only provide enough detail to make a reasonably low-risk estimate of how long the story will take to implement. When the time comes to implement, the story developers will meet with the customer to flesh out the details.

User stories also form the basis of acceptance testing. For example, one or more automated acceptance tests can be created to verify the user story has been correctly implemented. User stories are discussed further in Chapter 7. Use cases are introduced in Chapter 4.

Viewpoints

Viewpoints are a way to organize information from the (point of view of) different constituencies. For example, in the baggage handling system, there are differing perspectives of the system for each of the following stakeholders:

- Baggage handling personnel
- Travelers
- Maintenance engineers
- Airport managers
- Regulatory agencies

By recognizing the needs of each of these stakeholders and the contradictions raised by these viewpoints, conflicts can be reconciled using various approaches.

The actual viewpoints incorporate a variety of information from the business domain, process models, functional requirements specs, organizational models, and the like.

Sommerville and Sawyer (1997) suggested the following components should be in each viewpoint:

- A representation style, which defines the notation used in the specification
- A domain, which is defined as "the area of concern addressed by the viewpoint"
- A specification, which is a model of a system expressed in the defined style
- A work plan, with a process model, which defines how to build and check the specification
- A work record, which is a trace of the actions taken in building, checking, and modifying the specification

Viewpoint analysis is typically used for prioritization, agreement, and ordering of requirements.

Workshops

On a most general level, workshops are any gathering of stakeholders to resolve requirements issues. We can distinguish workshops as being of two types, formal and informal.

Formal workshops are well-planned meetings and are often "deliverable" events that are mandated by contract. For example, DOD MIL STD 2167 incorporated multiple required and optional workshops (critical reviews). A good example of a formal workshop style is embodied in JAD.

Informal workshops are usually less boring than highly structured meetings. But informal meetings tend to be too sloppy and may lead to a sense of false security and lost information. If some form of workshop is needed it is recommended that a formal one be held using the parameters for successful meetings previously discussed.

Elicitation Summary

This tour has included many elicitation techniques, and each has its advantages and disadvantages, which were discussed along the way. Clearly, some of these

techniques are too general, some too specific, some rely too much on stakeholder knowledge, some not enough, and so on. Therefore, it is clear that some combination of techniques is needed to address the requirements elicitation challenge successfully.

Which Combination of Requirements Elicitation Techniques Should Be Used?

In order to facilitate the discussion about appropriate elicitation techniques, we can roughly cluster the techniques previously discussed into categories or equivalence classes (interviews, domain-oriented, group work, ethnography, prototyping, goals, scenarios, viewpoints) as shown in Table 3.2.

Now we can summarize how effective various techniques are in dealing with various aspects of the elicitation process as shown in Table 3.3 (based on work by Zowghi and Coulin 1998). For example, interview-based techniques are useful for all aspects of requirements elicitation (but are very time consuming). On the other hand, prototyping techniques are best used to analyze stakeholders and to elicit the requirements. Ethnographic techniques are good for understanding the problem domain, analyzing stakeholders, soliciting requirements, and so on.

Finally, there is clearly overlap between these elicitation techniques (clusters) in that some accomplish the same thing and, hence, are alternatives to each other. In other cases, these techniques complement each another. In Table 3.4 alternative (A) and complementary (C) elicitation groupings are shown.

Clearly, in selecting a set of techniques to be used, the requirements engineer would look for a set of complementary techniques. For example, a combination of viewpoint analysis and some form of prototyping would be desirable. On the other hand, using both viewpoint analysis and scenario generation would probably yield excessively redundant information.

There is no "silver bullet" combination of elicitation techniques. The right mix will depend on the application domain, the culture of the customer organization and that of the requirements engineer, the size of the project, and many other factors. You can use Tables 3.2 through 3.4 to guide you in selecting an appropriate set of elicitation techniques.

Prevalence of Requirements Elicitation Techniques

Before we conclude this discussion, let's get an idea of how various elicitation techniques are commonly used in industry. To do so, we return to the surveys of 2003 (Neill and Laplante) and 2008 (Marinelli). A summary of the answers to the question, "Which requirements elicitation technique(s) do you use?" is shown in Figure 3.8.

The data revealed that in both cases about 50% surveyed used scenarios or use cases in the requirements phase. Other popular approaches to requirements elicitation reported in 2008 included group-consensus-type techniques such as use

Table 3.2 Organizing Various Elicitation Techniques Roughly by Type

Technique Type	*Techniques*
Domain-oriented	Card sorting Designer as apprentice Domain analysis Laddering Protocol analysis Task analysis
Ethnography	Ethnographic observation
Goals	Goal-based approaches QFD
Group work	Brainstorming Group work JAD Workshops
Interviews	Interviews Introspection Questionnaires
Prototyping	Prototyping
Scenarios	Scenarios User stories
Viewpoints	Viewpoints Repertory grids

Source: Zowghi and Coulin (1998).

case diagrams, interviews, storyboarding/white-boarding, and focus groups (Neill and Laplante 2003) and (Marinelli 2008).

Eliciting Hazards

We previously noted that "shall not" behaviors are the set of output behaviors that are undesired and that hazards were a subset of those behaviors that tended to cause serious or catastrophic failures. The terms "serious" and "catastrophic" are subjective.

Table 3.3 Techniques and Approaches for Elicitation Activities

	Interviews	*Domain*	*Group Work*	*Ethnography*	*Prototyping*	*Goals*	*Scenarios*	*Viewpoints*
Understanding the domain	•	•	•	•		•	•	•
Identifying sources of requirements	•	•	•			•	•	•
Analyzing the stakeholders	•	•	•	•	•	•	•	•
Selecting techniques and approaches	•	•	•					
Eliciting the requirements	•	•	•	•	•	•	•	•

Source: Zowghi and Coulin (1998).

Table 3.4 Complementary and Alternative Techniques

	Interviews	*Domain*	*Group Work*	*Ethnography*	*Prototyping*	*Goals*	*Scenarios*	*Viewpoints*
Interviews		C	A	A	A	C	C	C
Domain	C		C	A	A	A	A	A
Group Work	A	C		A	C	C	C	C
Ethnography	A	A	A		C	C	A	A
Prototyping	A	A	C	C		C	C	C
Goals	C	A	C	C	C		C	C
Scenarios	C	A	C	A	C	C		A
Viewpoints	C	A	C	A	C	C	A	

Source: Zowghi and Coulin (1998).

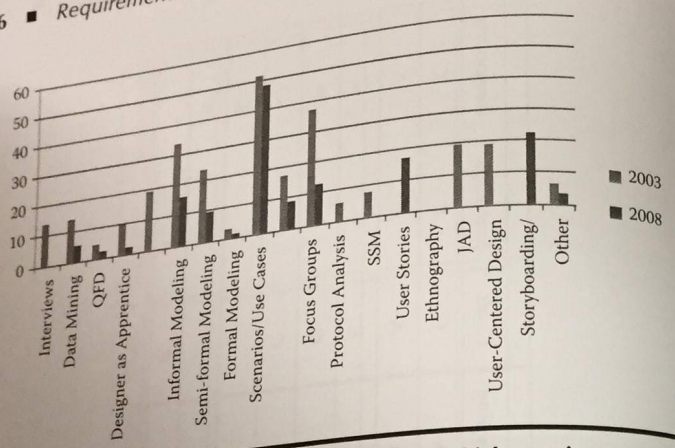

Figure 3.8 Summary of answers to the question "Which requirements elicitation technique(s) do you use?" (Adapted from C.J. Neill and P.A. Laplante, *Software*, 20(6): 40–45, 2003 and V. Marinelli, *An Analysis of Current Trends in Requirements Engineering Practice*, Master of Software Engineering Professional Paper, Penn State University, Great Valley School of Graduate Professional Studies, Malvern, PA, 2008.)

For example, here are some "shall not" requirements for the pet store POS:

- The system shall not expose customer information to external systems.
- The system shall not allow unauthorized access.
- The system shall not allow customers to overdraw store credit.

In these examples, the first two requirements might be considered hazards because the potential for financial damage to the company is far greater than for the third requirement.

Hazards are a function of input anomalies that are either naturally occurring (such as hardware failures) or artificially occurring (such as attacks from intruders; Voas and Laplante 2010). These anomalous input events need to be identified and their resultant failure modes and criticality need to be determined during the requirements elicitation phase in order to develop an appropriate set of "shall not" requirements for the system. As with any other requirements, "shall not" requirements need to be prioritized.

Typical techniques for hazard determination include the traditional development of misuse cases, antimodeling, and formal methods (Robinson 2010). Prevailing standards and regulations may also include specific "shall not" requirements; for example, in the United States the Health Insurance Portability and Accountability

Act of 1996 prohibits the release of certain personal information to unauthorized parties and standard construction codes in all jurisdictions include numerous prohibitions regarding certain construction practices.

Misuse Cases

Use cases are structured brief descriptions of desired behavior. Just as there are use cases describing desired behavior, there are misuse cases (or abuse cases) describing undesired behavior. Misuse cases are structured in exactly the same way as use cases and we explore these further in Chapter 4.

Antimodels

Another way of deriving unwanted behavior is to create antimodels for the system. Antimodels are related to fault-trees; that is, the model is derived by creating a cause and effect hierarchy for unwanted behaviors leading to system failure. Then the causes of the system failure are used to create the "shall not" requirements. For example, consider the security functionality for the baggage handling system involving the unwanted outcome of damaged baggage shown in Figure 3.9.

The figure leads us to write the following raw requirements:

- If a baggage jam is sensed, then the conveyor shall not move.
- If the baggage feeder is stuck, then the conveyor shall not move.

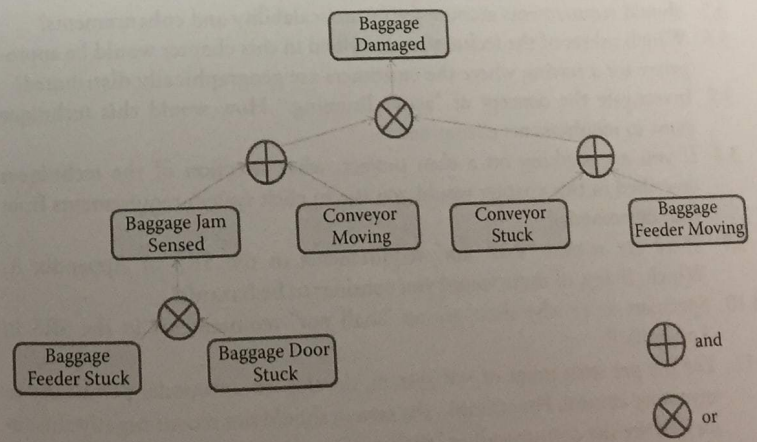

Figure 3.9 Partial antimodel for baggage handling system.

- If the baggage door is stuck, then the conveyor shall not move.
- If the conveyor is stuck, then the baggage feeder should not be moving.

These requirements need further analysis and possibly simplification, but the antimodel helped us to derive these raw requirements in a systematic way.

Formal Methods

Discussed in Chapter 6, mathematical formalisms can be used to create a model of the system and its environment as related to their goals, operations, requirements, and constraints. These formalisms can then be used in conjunction with automated model checkers to examine various properties of the system and ensure that unwanted ones are not present.

Exercises

3.1 What are some different user classes for the smart home system described in Appendix A?

3.2 What are some difficulties that may be encountered in attempting to elicit requirements without face-to-face interaction?

3.3 Does the Heisenberg uncertainty principle apply to techniques other than ethnographic observation? What are some of the ways to alleviate the Heisenberg uncertainty principle?

3.4 During ethnographic observation what is the purpose of recording the time and day the observation was made?

3.5 Should requirements account for future scalability and enhancements?

3.6 Which subset of the techniques described in this chapter would be appropriate for a setting where the customers are geographically distributed?

3.7 Investigate the concept of "active listening." How would this technique assist in requirements elicitation?

3.8 If you are working on a class project, what selection of the techniques described in this chapter would you use to elicit system requirements from your customer(s)?

3.9 There are several "shall not" requirements in the SRS of Appendix A. Which, if any, of these would you consider to be hazards?

3.10 Speculate as to why there are no "shall not" requirements in the SRS in Appendix B.

*3.11 For the pet store point of sale system, develop an antimodel pertaining to inventory control. For example, the system should not record negative inventory. Write the corresponding "shall not" requirements for this antimodel.

* This exercise is suitable for a small research assignment.

References

Akao, Y. (1990). *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Cambridge, MA: Productivity Press.

Aurum, A. and Wohlin, C. (Eds.) (2005). *Engineering and Managing Software Requirements*, New York: Springer.

Basili, V.R. and Weiss, D. (1984). A methodology for collecting valid software engineering data, *IEEE Transactions on Software Engineering*, Nov., 728–738.

FitNesse project, www.fitnesse.org, last accessed January 23, 2013.

Laplante, P.A. (2006). *What Every Engineer Needs to Know About Software Engineering*, Boca Raton, FL: CRC/Taylor & Francis.

Marinelli, V. (2008). An Analysis of Current Trends in Requirements Engineering Practice, Master of Software Engineering Professional Paper, Penn State University, Great Valley School of Graduate Professional Studies, Malvern, PA.

Neill, C.J. and Laplante, P.A. (2003). Requirements engineering: The state of the practice, *Software*, 20(6): 40–45.

Robinson, W.N. (2010). A roadmap for comprehensive requirements modeling. *Computer*, 43(5): 64–72.

Sommerville, I. and Sawyer, P. (1997). Viewpoints for requirements engineering, *Software Quality Journal*, 3: 101–130.

Voas, J. and Laplante, P. (2010). Effectively defining "shall not" requirements. *IT Professional*, 12(3): 46–53.

Zowghi, D. and Coulin, C. (1998). Requirements elicitation: A survey of techniques, approaches, and tools. In A. Aurum and C. Wohlin (Eds.) (2005) *Engineering and Managing Software Requirements*, New York: Springer, pp. 19–46.