Chapter 3

Requirements Elicitation

DOI: 10.1201/9781003129509-3

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 quite an equivalent term. Requirements are not like fallen fruit to be simply retrieved and placed in a bushel. 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. The following are common obstacles in the requirements elicitation process:

- **New Project Domain**: when requirements engineer doesn't possess enough knowledge on the industry or the developed solution. Engaging a domain expert can help alleviate this problem.
- **Unclear Project Vision**: when stakeholders don't have a clear understanding of what functionality their system needs. A clear mission statement or ConOps can be very helpful to avoid this problem.
- **Limited Access to Documentation**: when the requirements engineer can't access documentation or when evaluating the current state of the project takes too much time. Consistent and disciplined use of a good document/change management system is important to combat this problem.
- **Focus on the Solution Instead of Requirements**: when the customer focuses more on solutions or architectural tactics instead of requirements themselves. Active expectation management is very important in this regard.
- **Fixation of Specific Functionalities**: when stakeholders insist on designing certain features because they believe they will benefit their business if it is not. Again, expectation management and refocus on the mission statement is the key to addressing this issue.
- **Contradictory Requirements**: when a project includes a wide range of stakeholders, their requirements may contradict each other. We will discuss techniques for addressing this problem in a later chapter.

In addition to the specific suggestions above, there are many general techniques that you can choose to conduct requirements elicitation to overcome the above challenges, and you will probably need to use more than one and likely different ones for different classes of users/stakeholders. The techniques that we will discuss are as follows:

- Brainstorming

- Card sorting
- Crowdsourcing
- Designer as apprentice
- Domain analysis
- Ethnographic observation
- Goal-based approaches
- Group work
- Interviews
- Introspection
- Joint application development (JAD)
- Laddering
- Protocol analysis
- Prototyping
- Quality function deployment (QFD)
- Questionnaires
- Repertory grids
- Reverse engineering
- Scenarios
- Task analysis
- Use cases
- User stories
- Viewpoints
- Workshops

This list is partially adapted from one suggested by Zowghi and Coulin (**1998**).

Quick Access to a summary of the elicitation techniques

https://phil.laplante.io/requirements/elicit/summary.php

Requirements Elicitation - First Step

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 users for the pet store point of sale (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)

Each of these subgroups of users has different desiderata and these need to be determined.

The process, then, to prepare for elicitation is as follows:

- Identify all customers and stakeholders.
- Partition customers and other stakeholders groups 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 including:

- Travelers
- System maintenance personnel
- Baggage handlers
- Airline schedulers/dispatchers
- Airport personnel
- Airport managers and policymakers

But there are various kinds of travelers each with different needs. For example, consider the following subclasses:

- Children
- Senior citizens
- Business people
- Casual travelers
- Military personnel
- Civilians
- Casual travelers
- Frequent flyers

Each of these subclasses may need to be approached with different elicitation techniques. For example, surveys may not be appropriate for children, while 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 will 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, minute taking, and the use of formal structures (e.g., Robert's 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 embody some form of brainstorming implicitly. Brainstorming is also useful for general objective setting, such as mission or vision statement generation. Once brainstorming is selected as a technique, the following guidelines are recommended to get the most out of your brainstorming session:

- Select a facilitator, ideas recorder, and the participants, and reserve the proper place and time for the session.
- Make sure everyone is on the same page regarding the process.
- Give each participant a small amount of time to brainstorm on their own before bringing their ideas to the group.
- Once the brainstorming session has started, keep everyone on topic.
- Do not limit creativity, free association, or the number of ideas.
- If the session is long, build in some coffee break times.
- Write all ideas down in plain view of the entire group.

Once the brainstorming session is over, begin the refining process utilizing other elicitation techniques.

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 a ranking and rationale for each of the functionalities.

The time period to allow customers and stakeholders to complete the cards is an important decision. While 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 1 week (and no more than 2 weeks) be allowed for the completion of the cards. Another alternative is to have the customers complete

the cards in a 2-hour session and then return 1 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, collaboration) cards to determine program classes in the eventual code. Another technique to be 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, each card contains only a brief description of the functionality and a priority rating is included (for brevity, no rationale is shown).

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

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**.

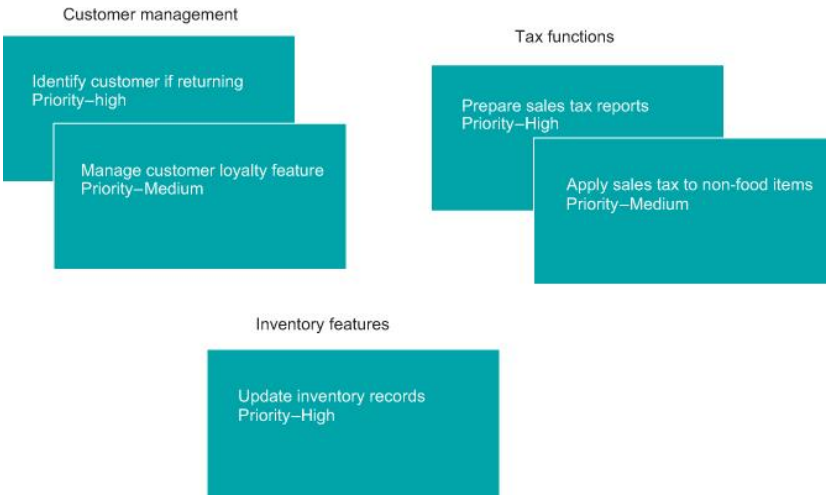

Figure 3.2 Sorted cards for the pet store POS system.

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.

Crowdsourcing

Crowdsourcing is a business model that harnesses the power of a large and diverse number of people to contribute knowledge and solve problems. The term "crowdsourcing" is a combination of crowd and outsourcing and was coined in 2006 by Wired magazine author Jeff Howe in his article "The Rise of Crowdsourcing" (**Howe 2006**).

Capturing the requirements directly from the crowd (the bigger the crowd, the better) gives the requirements engineers access to a wide diversity of actual and potential users which has the potential to increase the comprehensiveness and quality of the captured requirements. Crowd-based requirements engineering as a term was coined by Groen et al. (**2015**) to be a requirements engineering approach for acquiring and analyzing any kind of users' feedback from the crowd, with the aim of seeking validated user requirements. It was further elaborated by Groen and Koch (**2016**) be "the combined set of techniques for analyzing data from the crowd using text and usage mining, motivational techniques for stimulating the further generation of data, and crowdsourcing to validate requirements."

Crowdsourcing in requirements elicitation has the potential to increase the comprehensiveness and quality of the captured requirements by giving the requirements engineers access to a wide diversity of actual and potential users. The captured requirements from the crowd (the bigger the crowd, the better) will almost always be superior to products developed only with input from experts. But who is "the crowd" for your system? It could be system users in the company, it could be the sales representatives, or it could be millions of customers worldwide, it all depends on the system that is being developed.

Social media (e.g., LinkedIn, Facebook, Twitter) provide the best-developed mechanisms to efficiently get feedback from crowds. The more efficient the communication mechanism to the crowd, the more people can be included in the process. A typical process to capture the requirements via crowdsourcing includes the following steps:

1. Choosing the potential crowd to be targeted.
2. Choosing the proper social media tool that is both popular with that crowd and that will allow the requirements engineer to ask questions of the crowd.
3. Creating a community with the crowd (perhaps by sharing useful information and news that's relevant to the crowd).
4. Once the "crowdsourced elicitation process" is set up, it comes down to asking the right questions when opportunities arise. Asking ques-

> tions to the crowd will be only efficient if the right questions are asked. That's where expertise comes in, along with other requirements elicitation discussed in this chapter. Since most people don't know what they want, it is the requirements engineer's job to present the right questions in an effective manner in order to trigger useful insights from the crowd. The same types of questions that don't work in one-on-one elicitation will not work in crowd elicitation as well (see the questionnaires technique).

The elicitation process via crowdsourcing is often carried on with the support of tools. Some developed crowd-based tools for requirements elicitation include CrowdREquire (**Adepetu et al. 2012**), Refine (**Snijders et al. 2015**), StakeRare (**Lim and Finkelstein 2011**), and Requirements Bazaar (**Renzel and Klamma 2014**).

Groen et al. (**2015**) discussed the concept of crowd-based requirements engineering and its landscape and challenges to emphasize its use. The highlighted challenges are as follows:

- Challenges related to largeness
- Challenges related to diversity
- Challenges related to anonymity
- Challenges related to competence
- Challenges related to collaboration
- Challenges related to intrinsic motivations
- Challenges related to volunteering
- Challenges related to extrinsic incentives
- Challenges related to opt-out opportunity
- Challenges related to feedback

In most cases, crowdsourcing every aspect of the system is not necessary or beneficial, but in almost all cases the feedback from the crowd on a myriad of questions about the requirements can be of great benefit.

Designer as Apprentice[1]

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 their 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 we want the requirements engineer (the designer) to learn about the customer's work from the customer. The apprentice is there to learn 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.

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 also 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. Think about an expert system that automates cake decoration based on customer preferences. 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 since, 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. An apprentice learns 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 their 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 and/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 later 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 will discuss this technique shortly.

Ethnographic Observation

Ethnographic observation refers to any technique in which observation of indirect and direct factors inform 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 different cultures. The anthropologist lives among the culture being studied, but in a way that 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.

As another example, imagine that you are leading the requirements elicitation activities for a new home security system that includes alarms, window and door sensors, cameras, and motion sensors. The system interacts with the security monitoring company via a dedicated telephone line and with the home residents via one or more keypads (as well as through the sensors). You may choose ethnographic observation to capture the requirements related to how children (age 5–12) will interact directly or indirectly with the system.

In applying ethnographic observation to requirements elicitation, the requirements engineer immerses himself 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 a customer.
- You spend long periods of time interviewing the customer about what they want.
- You spend time interacting with the customer as they go about their 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 the current home to get nonverbal clues about 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, etc.

Ethnographic observation can be very time-consuming and requires substantial training of the observer. 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.

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 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, since 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 over some period of time
3. The number of times a user sees an error message during certain operations over 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 into 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 meetings that are used during the requirements discovery, analysis, and follow-up processes. The most celebrated group-oriented work for requirements elicitation is JAD, which we will 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

Elicitation through interviews involves in-person communication between two individual stakeholders or a small group of stakeholders (sometimes called a focus group). Interviews are an easy-to-use technique to extract system-level requirements from stakeholders, especially usability requirements.

Three kinds of interviews can be used in elicitation activities, and they can be applied to individuals or focus groups:

- Unstructured
- Structured
- Semi-structured

Unstructured interviews, which are probably the most common type, are conversational in nature and serve to relax the participants. Like a spontaneous "confession," these can occur any time and any place whenever the requirements engineer and stakeholder 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 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.

While 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.

Hickey and Davis (**2003**) conducted in-depth interviews with some of the world's most experienced analysts in requirements elicitations. Their findings on using the "interviews" technique in practice revealed that experts use it in the following situations:

- Gathering initial background information when working on new projects in new domains.
- Whenever heavy politics are present to ensure that the group session does not self-destruct.
- When there is a need to isolate and show conflicts among stakeholders.
- When senior management has an idea, but the employees consider this idea to be unreasonable, the problem can be addressed by interviewing subject matter experts and visionaries.
- Using it with subject matter experts is essential when the users and customers are inaccessible.

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 email, 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 that of the customer. Occasionally, the customer will ask the engineer questions similar to the following: "if you were me, what would you want?" While introspection will inform every aspect of the requirements engineer's interactions, remember our admonition about not telling a customer what he ought to want. Introspection is also a great way to gather requirements when users are too busy to be involved in interviews, group sessions, or questionnaires. Introspection can be used to assess political and power relationships when working in new organizations.

Joint Application Design

JAD involves highly structured group meetings (sometimes called "mini-retreats") with system users, system owners, and analysts focused on a specific set of problems for an extended period of time. These meetings occur 4–8 hours per day and over a period lasting 1 day to a couple of weeks. JAD has even been adapted for multisite implementation when participants are not colocated (**Cleland-Huang and Laurent 2014**). While traditionally associated with large, government systems projects, the technique can be used in industrial settings on systems of all sizes.

JAD and JAD-like techniques are commonly used in systems planning and systems analysis activities to obtain group consensus on problems, objectives, and requirements. Specifically, the requirements engineer can use JAD sessions for the concept of operation definition, system goal definition, requirements elicitation, requirements analysis, requirements document review, and more.

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 (i.e., meeting minutes and note-takers)
- 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 software development team. The analyst and managers must select individuals from the user community. These individuals should be knowledgeable and articulate in their business area.

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 walkthroughs. The session leader must make every effort to ensure these practices are implemented.

- Stick to the 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 tree-like 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 their 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.

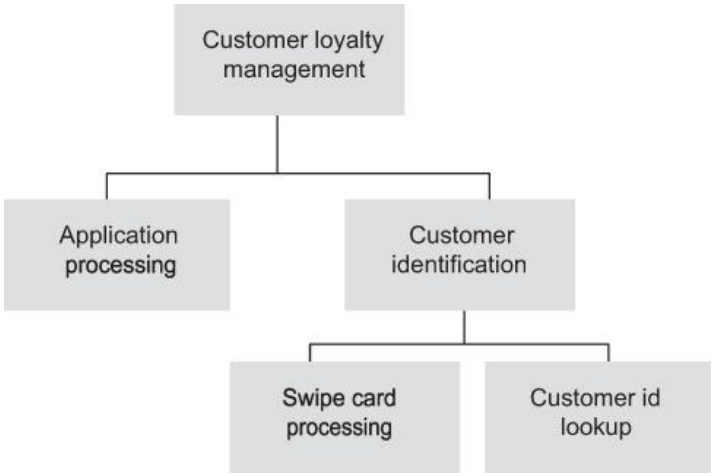

Figure 3.3 Laddering diagram for the pet store POS system.

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 that they are going to automate. During such a walkthrough, the customers explicitly state the rationale for each step that is being taken.

For example, for the major package delivery company discussed in the section of domain vocabulary understanding in **Chapter 1**, it was the practice to have engineers and other support professionals ride with the regular delivery personnel during the busy winter holiday season. This practice not only addressed the surge in packages to be delivered but also reacquainted the engineers with the processes and procedures associated with the company's services as they were actually applied. Observations made by the engineers in the field often led to processes optimization and other innovations.

While you will see shortly that protocol analysis is very similar to designer as apprentice, 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 the construction of models of the system in order to discover new features, particularly usability requirements. 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 non-throwaway prototypes.

Suppose a company is developing a new online commerce website for their products. A new customer support application is proposed to allow customers to view the status of their orders online. The company needs to determine the best "look and feel" of the user interface. Developing different prototypes of the user interface and showing it to users followed by interviewing them about these interfaces can be the appropriate approach to follow.

Prototypes can involve working models and non-working models. Working models can include executable code in the case of software systems and simulations, or temporary or to-scale prototypes for non-software systems. Non-working 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 software prototypes, the code can be deliberately designed to be throwaway or it can be deliberately designed to be reused (non- throwaway). 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 non-throwaway prototypes.

Recently, 3-D printing has become an important tool in building physical models of certain systems. 3-D printing has two important advantages over other rapid prototyping technologies. The first is cost—industrial quality 3-D printers can be purchased for a few thousand dollars, while rapid prototyping machines using traditional computer numerical control (CNC) can cost several hundreds of thousands. The second advantage is that 3-D printers can take as input standard format files produced by commonly used computer-aided design (CAD) programs (**Berman 2012**).

There are a number of different ways to use prototyping—for example, within a fourth-generation environment (i.e., a simulator), throwaway prototyping, evolutionary prototyping (where the prototype evolves into the final system), or user interface prototyping. Some organizations may use more than one type of prototyping. The results from the requirements engineering state of practices conducted in 2020 (**Kassab and Laplante 2022**) pertaining to the selection frequency for these different prototyping techniques in practice are shown in **Figure 3.4**.

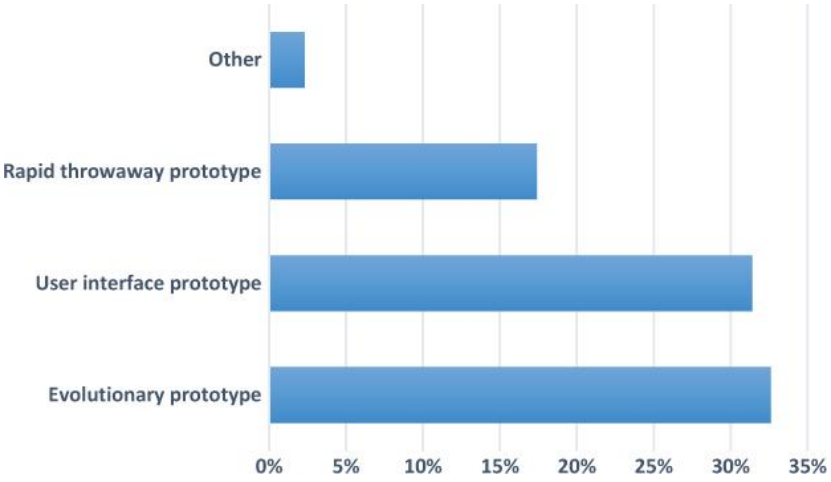

Figure 3.4 Prototype methods selection across software development life cycle methodology. (Kassab and Laplante 2022).

There are at least three dangers to consider when using prototyping for requirements elicitation.

Stay Updated: For the up-to-date data from the RE state of practice survey.

https://phil.laplante.io/requirements/updates/survey.php

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

The second problem is that prototyping is not always effective in discovering certain nonfunctional requirements (NFRs). Suppose that you are conducting requirements elicitation activities for a new smart washing machine/dryer combination. While prototyping can be good for understanding existing functionalities, revealing missing functionalities, and identifying unwanted functionalities, many NFRs (e.g., security, safety) won't be easy to identify. Prototyping is particularly not suitable for those requirements that can only be derived by an analysis of prevailing standards and laws (**Kassab and Ormandjieva 2014**).

Finally, problems can occur when using prototypes to discover the ways in which users interact with the system. The main concern is that users interact differently with a prototype (in which the consequences of behavior are not real) vs. the actual system. Consider, for example, how users might drive in a vehicle simulator, where there is no real injury or damage from a crash. The drivers may behave much more aggressively in

the simulator than they would in a real vehicle, leading to possibly erroneous requirements discovery.

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

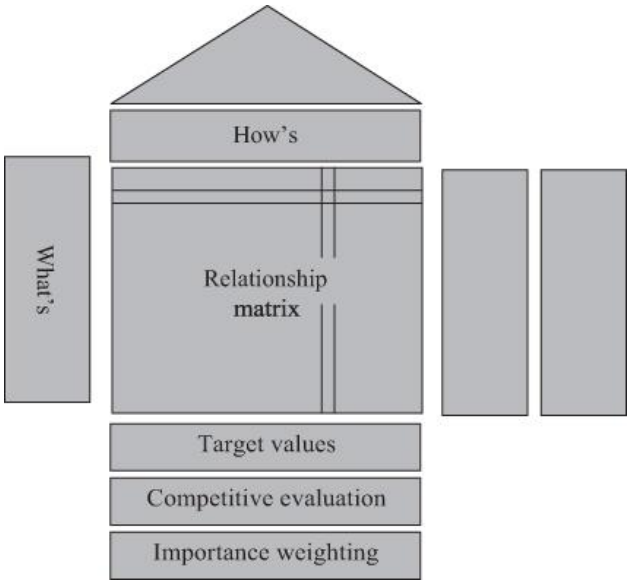

Figure 3.5 QFD's "house of quality" (Akao 1990).

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 their 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 the development effort.

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**.

Table 3.1 Partial Competitive Analysis for Pet Store POS System

Feature	***Competing Product***		
	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

Notice that only very high-level features are shown, though 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.

Since it incorporates a total life cycle approach to requirements engineering, QFD has several advantages over other stand-alone elicitation techniques. QFD improves the involvement of users and managers. It shortens the development lifecycle 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 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 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 the early stages of the elicitation process to quickly define the scope boundaries.

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 bind 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 as follows:

- 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 overscoring and underscoring 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 requirements engineers.

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 are 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 which 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 their product research, they survey a number of people with an instrument in which the respondents check off the flavors of ice cream they would purchase. They find that of the 30 different flavors listed in the survey, 20 of the flavors are selected by 50% of the respondents or more. Thus, they decide to produce and stock these 20 flavors roughly in proportion to the demand indicated by the survey results. Yet, after 1 week of opening their ice cream store, they discover that 90% of business is due to the top three flavors—chocolate, vanilla, and strawberry. Of the 17 other flavors in the survey they keep in inventory, several have never even been purchased. They realize 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, email, 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 the 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. While 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.

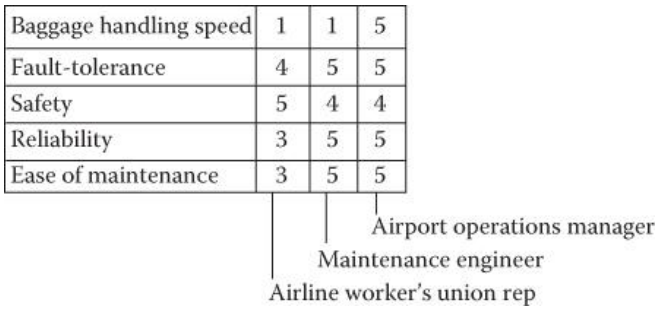

Figure 3.6 Partial repertory grid for the baggage handling system.

Reverse Engineering

If an existing system has outdated documentation (or even nonexistent documentation), then reverse engineering can be applied to the system to extract the requirements from the system to understand what the system does. This elicitation technique can be particularly useful for migration projects when dealing with legacy systems. Generally, there are two types of reverse engineering techniques:

- **Black-Box Reverse Engineering**: the system is studied without examining its internal structure (function and composition of software).
- **White-Box Reverse Engineering**: The inner workings of the system are studied (analyzing and understanding of software code).

Several methods are proposed in the literature for reverse engineering as requirements elicitation techniques. For example, **Yu et al. (2005)** proposed a methodology to extract stakeholder goal models from both struc-

tured and unstructured legacy code. The methodology consists of the four major steps: (i) refactoring source code by extracting methods based on comments; (ii) converting the refactored code into an abstract structured program through state chart refactoring and hammock graph construction; (iii) extracting a goal model from the structured programs abstract syntax tree; and (iv) identifying NFRs and deriving soft goals based on the traceability between the code and the goal model. Other existing studies include Hassan et al. (**2015**), Fahmi and Choi (**2007**), and Alderson and Liu (**2012**).

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.

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

> *A customer walks into the pet store and fills the cart with a variety of items. When checking out, the cashier asks if the customer has a loyalty card. If so, the cashier swipes the card, authenticating the customer. If not, then the cashier offers to complete one 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 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, for example). User stories are, in fact, a form of scenario.

Task Analysis

Like 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**.

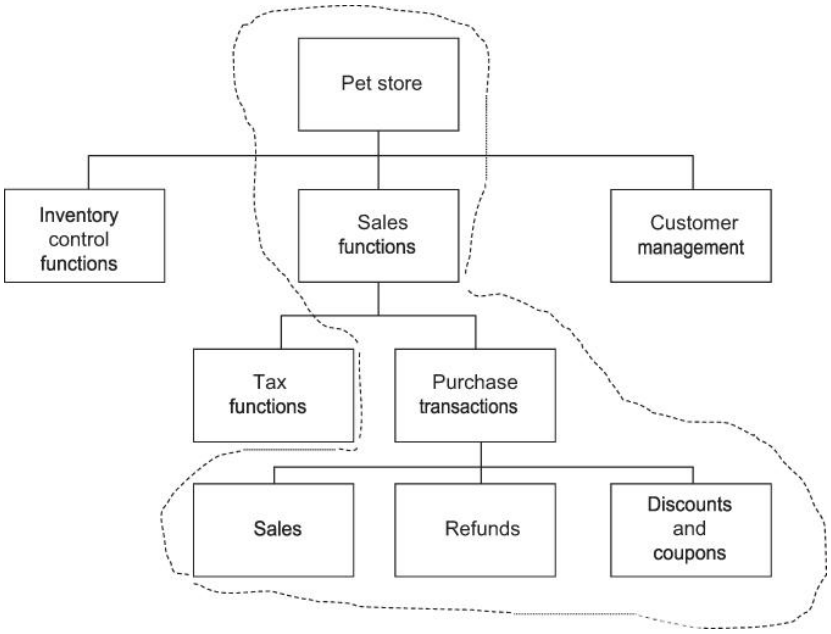

Figure 3.7 Partial task analysis for the pet store POS system.

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 following tasks: tax functions and purchase transactions. Next, proceeding to the purchase transaction function, we decompose these tasks into sales, refunds, discounts, and coupons tasks.

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

Use Cases[2]

Use cases are a way for more sophisticated customers and stakeholders to describe their desiderata. Use cases depict the interactions between the system and the environment around the system, in particular, human users and other systems. They can be used to model the behavior of pure software or hybrid hardware-software systems.

Use cases describe scenarios of operation of the system from the designer's (as opposed to customers') perspective. Use cases are typically represented using a use case diagram, which depicts the interactions of the system with its external environment. In a use case diagram, the box represents the system itself. The stick figures represent "actors" that designate external entities that interact with the system. The actors can be humans, other systems, or device inputs. Internal ellipses represent each activity of use for each of the actors (use cases). The solid lines associate actors with each use. **Figure 3.8** shows a use case diagram for the baggage inspection system.

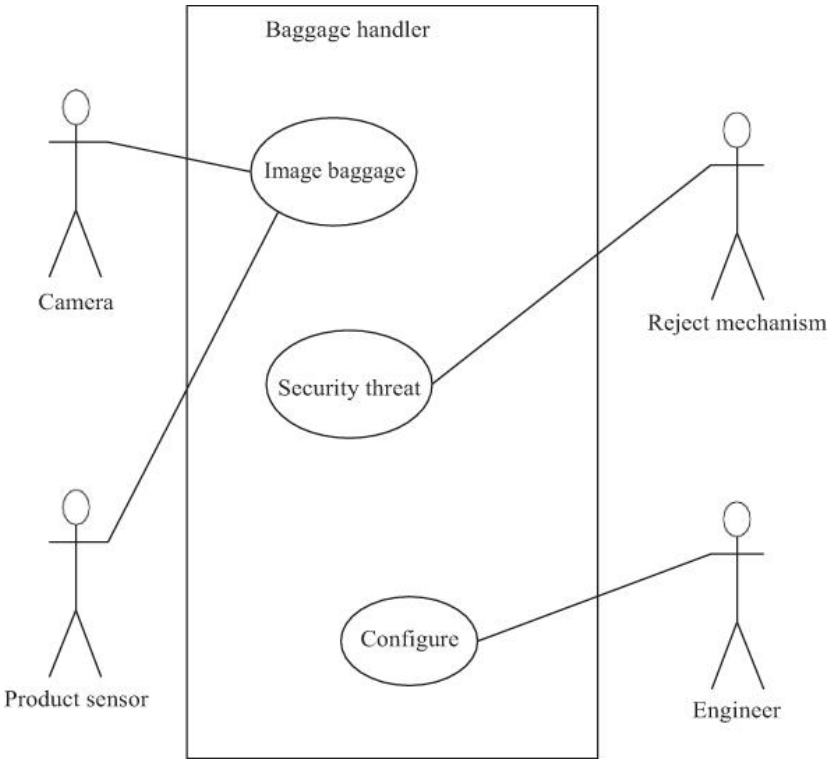

Figure 3.8 Use case diagram of baggage inspection system.

Three uses are shown—capturing an image of the baggage ("image baggage"), the detection of a security threat (in which case the bag is rejected from the conveyor for offline processing), and then configuration by the systems engineer. Notice that the imaging camera, product sensor, and reject mechanism are represented by a human-like stick figure—this is typical—the stick figure represents a system "actor" whether human or not.

Appendix B, **Figure B.4** provides another example—a use case diagram for the wet well.

Each use case is a form of documentation that describes scenarios of operation of the system under consideration as well as pre- and postconditions and exceptions. In an iterative development life cycle, these use cases will become increasingly refined and detailed as the analysis and design workflows progress.

Interaction diagrams are then created to describe the behaviors defined by each use case. In the first iteration, these diagrams depict the system as a "black box," but once domain modeling has been completed, the black box is transformed into a collaboration of objects as will be seen later.

If well developed, sometimes, the use cases can be used to form a pattern language, and these patterns and the derived design elements can be reused in related systems (**Issa and Al-Ali 2010**). Using patterns when specifying requirements ensures a greater level of consistency and can reduce errors in automated measurement of certain requirements properties.

Because they have become such an important tool in requirements discovery and in modeling requirements in the requirements specification document, a comprehensive discussion of use cases, with many examples, can be found in Appendix E.

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

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

- Each customer should be able to easily check out at a register.
- 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 whether the user story has been correctly implemented.

Surprisingly, in their 2020 survey on requirements engineering state of practices, Kassab and Lapante (2022) found that even though user stories were developed specifically for Agile methodologies, the technique is being used with the Waterfall model—28% of Waterfall projects reported using it. According to that survey, "User stories" witnessed the biggest jump in overall usage from 2013 when a similar survey was conducted (43% in 2020 from 14% in 2013 (**Kassab et al. 2014**)).

User stories are discussed further in **Chapter 8** and in Appendix D.

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 different 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 specifications, organizational models, etc.

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 the workshop is needed, it is recommended that a formal one be held using the parameters for successful meetings previously discussed.

Eliciting Nonfunctional Requirements

Nonfunctional requirement (NFR) elicitation techniques differ from functional requirements elicitation techniques. NFRs are generally stated informally during the requirements analysis, are often contradictory, and are difficult to enforce and validate during the software development process. Borg et al. (**2003**) carried out interviews aimed at identifying the roots of NFR-related problems in different organizations. The conclusion was that NFR-related problems occur at four stages of the development process: elicitation, documentation, management, and test; elicitation being flagged as the main source of potential NFR-related problems as NFR omission at the elicitation stage propagates through the entire development process. The reasons are, for instance, that (i) certain constraints are unknown at the requirements stage, (ii) NFRs tend to conflict with each other, and (iii) separating FRs and NFRs makes it difficult to trace dependencies between them, whereas functional and nonfunctional considerations are difficult to separate if all requirements are mixed together.

It is not easy to choose a method for eliciting, detailing, and documenting NFRs among the variety of existing methods. The following are the desirable characteristics of NFR elicitation methods (**Herrmann et al. 2007**):

1. A guided process to ease the method usage by less experienced personnel and to support repeatability of the results
2. Derivation of measurable NFRs to ease quality assurance
3. Reuse of artifacts to support completeness of the derived NFRs to support learning and to avoid rework
4. Intuitive and creative elicitation of quality to capture also the hidden requirements and thus support completeness
5. Focused effort for efficient elicitation and NFR prioritization to support trade-off decisions
6. Handling dependencies between NFRs to support trade-off decisions
7. Integration of NFRs with functional requirements

Common ways to discover NFRs include competitive analysis of system qualities: NFRs can be discovered by analyzing the qualities for competing products in the market. For example, what is the response time for a competing product? And do we need to do better?

Another technique for discovering NFS is to use a pre-established questionnaire where a requirements engineer develops a questionnaire to be asked of the stakeholders and development team. For example: “How should the system respond to input errors? What parts of the system are likely candidates for later modification? What data of the system must be secure?” These questions can be prepared while following some template or standard (e.g., ISO 9126) in order to focus and ask questions about each type of NFR in the standard.

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, etc. Therefore, it is clear that some combination of techniques is needed to successfully address the requirements elicitation challenge.

Which Combination of Requirements Elicitation Techniques Should Be Used?

There is scant research to provide guidance on selecting an appropriate mix of requirements elicitation techniques. One notable exception is the knowledge-based approach for the selection of requirements engineering techniques (KASRET), which guides users to select a combination of elicitation from a library (**Eberlein and Jiang 2011**). The library of techniques and assignment algorithms are based on a literature review and survey of industrial and academic experts. The technique has not yet been widely used, however.

In order to provide some guidance on appropriate elicitation techniques, we first cluster the techniques previously discussed into categories or equivalence classes based on the kinds of information the techniques are likely to uncover. The classes (interviews, domain-oriented, group work, ethnography, prototyping, goals, scenarios, viewpoints) and the included elicitation techniques are shown in **Table 3.2**.

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 Use cases User stories

Technique Type	*Techniques*
Viewpoints	Viewpoints Repertory grids

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

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**).

Table 3.3 Techniques and Approaches for Elicitation Activities

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

Source: Zowghi, D., & Coulin, C., Requirements elicitation: A survey of techniques, approaches, and tools, in A. Aurum & C. Wohlin (Eds.), (2005). *Engineering and Managing Software Requirements*, pp. 19–46, Springer, 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 a clear 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 one another. In **Table 3.4**, alternative (A) and complementary (C) elicitation groupings are shown.

Table 3.4 Complementary and Alternative Techniques

	Interviews	*Domain*	**Group Work**	*Ethnography*	*Prototyping*	**Goals**	**Scenarios**	**V**
Interviews		C	A	A	A	C	C	C
Domain	C		C	A	A	A	A	A
Groupwork	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, D., & Coulin, C., Requirements elicitation: A survey of techniques, approaches, and tools, in A. Aurum & C. Wohlin (Eds.), (2005). *Engineering and Managing Software Requirements*, pp. 19–46, Springer, 1998.

You can use **Tables 3.2**–3.4 to guide you through selecting an appropriate set of elicitation techniques. In selecting a set of techniques to be used, you would select a set of complementary techniques. For example, a combination of viewpoint analysis and some form of prototyping would be desirable. Conversely, using both viewpoint analysis and scenario generation would probably yield excessively redundant information.

As an example, consider the case of an IoT healthcare system to track humans in a hospital. Here it would be appropriate to start with an initiative to define system overall goals and desired outcomes. Next, we would use a domain analysis to explore the laws, regulations, and standards that apply to the system. User stories, scenarios, and interviews would be appropriate for elicitation of requirements from users of the system. There would undoubtedly be some prototyping throughout the requirements discovery and refinement process. Each of these techniques would move from informal to more rigorous as requirements discovery proceeded. This approach is often used for federal, state, and municipal governments, and even for large industrial projects.

There is no “silver bullet” combination of elicitation techniques, however. 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. Learning from experience is very important in this regard. Finally, Hickey and Davis (2003) provide further insights on selecting the appropriate combinations of elicitation techniques from the viewpoint of a number of experts who were presented with various requirements elicitation scenarios and asked which approaches they would use.

Prevalence of Requirements Elicitation Techniques

Before we conclude this discussion, let’s get an idea of how various elicitation techniques are commonly used in the industry. Hickey and Davis (2003) found that less experienced analysts often select a technique based on one of two reasons: (i) it is the only one they know, or (ii) they think that a technique that worked well last time must surely be appropriate this time. If we return to the 2020 survey of requirements engineering state of practices (Kassab and Laplante 2022), then a summary of the answers to the question “which requirements elicitation technique(s) do you use?” is shown in Figure 3.9. The responses revealed that brainstorming, use cases, interviews, prototyping, and user stories were the top five frequently used elicitation techniques. On average, a participant selected three elicitation techniques.

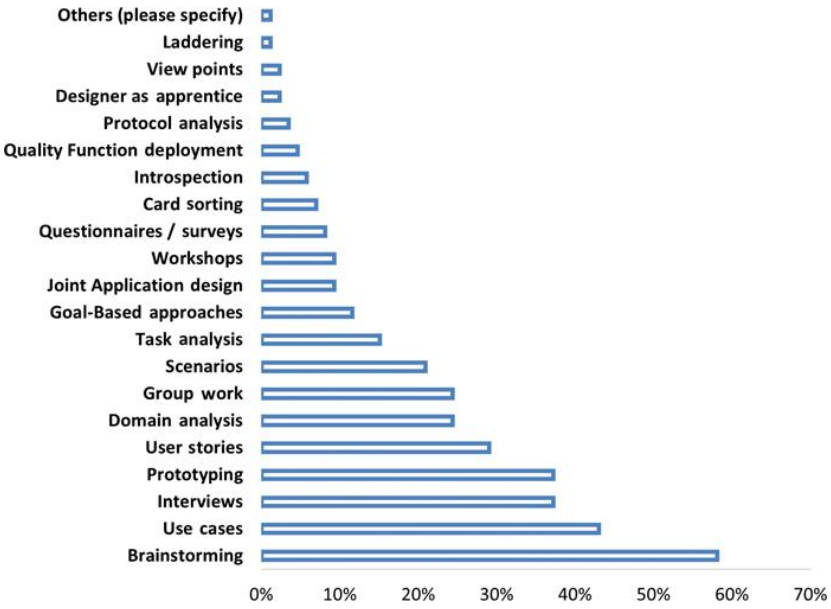

Figure 3.9 Summary of answers to the question “which requirements elicitation technique(s) do you use?” (Kassab and Laplante 2022).

Stay Updated: For the up-to-date data from the RE state of practice survey.

https://phil.laplante.io/requirements/updates/survey.php

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 but generally involve loss of life, serious injury, major infrastructure damage, or great financial loss.

For example, some “shall not” requirements for the pet store POS include the following:

- 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, anti-modeling, and formal methods (**Robinson 2010**). Checklists of unwanted behavior that have been created from previous versions of the system or related systems are also helpful in identifying unwanted behavior. Prevailing standards and regulations may also include specific “shall not” requirements; for example, in the United States, the Health Insurance Portability and Accountability Act (HIPPA 1996) prohibits the release of certain personal information to unauthorized parties, and standard construction codes in all jurisdictions include numerous prohibitions against 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. Typical misuses for most systems include security breaches and other malicious behaviors as well as abuse by untrained, disoriented, or incapable users. Cleland-Huang et al. (**2016**) describe several ways of identifying misuse cases based on threat modeling and brainstorming.

An easy way to create use cases is to assume the role of a persona non grata, that is, an unwanted user of the system, and then model the behaviors of such a person (Cleland-Huang 2014). Personae non gratae can include hackers, intruders, spies, and even well-meaning, but bumbling users.

To see how identifying these persons helps in creating hazard requirements, consider the following examples. In the pet store POS system, it would be appropriate to consider how a hacker would infiltrate this system and then create requirements that would thwart the hacker's intentions. In the baggage handling system, a requirements engineer could assume the role of a clumsy or absent-minded traveler and then prescribe requirements that would ensure the safety of such persons. The need to create misuse cases is a reason to completely identify all negative stakeholders since these persons are likely to comprise large many personae non gratae.

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 3.10.

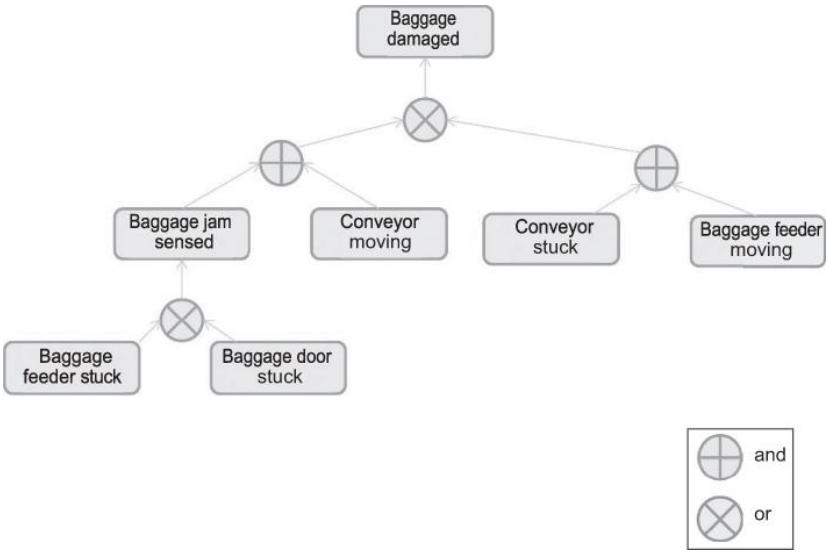

Figure 3.10 Partial antimodel for baggage handling system.

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

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

Formal Methods

To be discussed in **Chapter 7**, 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. For example, if using UML/SysML to model behavior, formal activity diagrams could be annotated with security concerns. Activity diagrams are described in Appendix C, UML.

VIGNETTE 3.1 Requirements Engineering for Safety-Critical Systems

Safety-critical systems are those in which failure can result in loss of human life or significant injury. These kinds of systems include public infrastructure for power and water, medical systems, transportation systems, and much more. With the growth of smart and autonomous systems, for example, smart highways, driverless vehicles, and robotic surgery, the focus on safety requirements has become paramount.

There are some domain-specific standards (e.g., nuclear, medical devices, aviation) and general standards that provide guidance for developing requirements for safety-critical systems. One example of a general standard is NASA-STD-8719.13, which provides guidance for safety considerations in software-intensive systems (**NASA 2013**).

This standard uses a risk assessment matrix table to show the probability and severity of a particular hazard (**Figure 3.11**).

Systems severity levels	System hazard likelihood of occurrence				
	Improbable	Unlikely	Possible	Probable	Likely
Catastrophic	4	3	2	1	1
Critical	5	4	3	2	1
Moderate	6	5	4	3	2
Negligible	7	6	5	4	3

1= Highest priority (highest system risk), 7= Lowest priority (lowes system risk).

Figure 3.11 The system risk index matrix of NASA-STD-8719.13C. (From NASA [National Aeronautics and Space Administration], NASA-STD 8719.13, Software Safety Standard, 2013, http://NASA.gov [accessed January 2017].)

The standard stipulates that safety requirements be tagged for traceability through the requirements engineering life cycle needed in order to enable the assessment of impacts and changes to the requirements. The uniquely identified hazards can be listed in a special section in the re-

quirements document, or be designated by a flag beside the requirement, or be tagged within a requirements management tool.

Software safety requirements can do more than protect against unsafe system behavior. The software can be used proactively to monitor the system, analyze critical data, look for trends, and signal or act when events occur that may be precursors to a hazardous state. Once such an indicator is detected, the software can be used to avoid or mitigate the effects of that hazard. Avoidance or mitigation could include restoring the system fully or partially or putting the system into a safe state.

Share your Opinion: What kinds of requirements engineering elicitation techniques does your organization use?

https://phil.laplante.io/requirements/opinion.php

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 of the observation 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 Which elicitation techniques would you use to elicit system requirements from the following stakeholder groups?
  - 3.8.1 Passengers in the baggage handling system
  - 3.8.2 Cashiers in the pet store POS system
  - 3.8.3 Doctors in the IoT healthcare system described at the end of **Chapter 2**
- 3.9 If you are working on a course project, list the elicitation techniques that you would you use to elicit system requirements from each of the stakeholder groups.

- 3.10 There are several “shall not” requirements in the SRS of Appendix A. Which, if any, of these would you consider being hazards?
- 3.11 Speculate as to why there are no “shall not” requirements in the SRS in Appendix B.
- *3.12 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.

Note

1, 2 This discussion is adapted from one found in Laplante (**2006**), with permission.

References

Adepetu, A., Khaja, A. A., Al Abd, Y., Al Zaabi, A., & Svetinovic, D. (2012, March). CrowdREquire: A requirements engineering crowdsourcing platform. In *2012 AAAI Spring Symposium Series*, Palo Alto, California.

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

Alderson, A., & Liu, K. (2012). Reverse requirements engineering: The AMBOLS approach. Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme (pp. 196–208). Springer, London.

Basili, V. R., & Weiss, D. (1984). A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering, 10: 728–738.

Berman, B. (2012). 3-D printing: The new industrial revolution. Business Horizons, 55(2): 155–162.

Borg, A., Yong, A., Carlshamre, P., & Sandahl, K. (2003). The bad conscience of requirements engineering: An investigation in real-world treatment of non-functional requirements. In Third Conference on Software Engineering Research and Practice in Sweden (SERPS’03) (pp. 1–8). Lund.

Centers for Medicare & Medicaid Services. (1996). The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at **http://www.hhs.gov/hipaa** (accessed June 2017).

Cleland-Huang, J. (2014). How well do you know your personae non gratae? IEEE Software, 31(4): 28–31.

Cleland-Huang, J., Denning, T., Kohno, T., Shull, F., & Weber, S. (2016). Keeping ahead of our adversaries. IEEE Software, 33(3): 24–28.

Cleland-Huang, J., & Laurent, P. (2014). Requirements in a global world. IEEE Software, 31(6): 34–37.

Eberlein, A., & Jiang, L. (2011). Selecting requirements engineering techniques. In P. Laplante (Ed.), Encyclopedia of Software Engineering (pp. 962–978). Taylor & Francis. Boca Raton, FL, Published online.

Fahmi, S. A., & Choi, H. J. (2007, November). Software reverse engineering to requirements. In *2007 International Conference on*

Convergence Information Technology (ICCIT 2007), (pp. 2199–2204). IEEE, Gwangju, South Korea.

Groen, E. C., Doerr, J., & Adam, S. (2015, March). Towards crowd-based requirements engineering a research preview. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 247–253). Springer, Cham.

Groen, E. C., & Koch, M. (2016). How requirements engineering can benefit from crowds. Requirements Engineering Magazine, 8: 10.

Hassan, S., Qamar, U., Hassan, T., & Waqas, M. (2015, August). Software reverse engineering to requirement engineering for evolution of legacy system. In *2015 5th International Conference on IT Convergence and Security (ICITCS)* (pp. 1–4). IEEE, Kuala Lumpur, Malaysia.

Herrmann, A., Kerkow, D., & Doerr, J. (2007). Exploring the characteristics of NFR methods – A dialogue about two approaches. In REFSQ 2007, LNCS 4542 (pp. 320–334). Trondheim, Norway.

Hickey, A. M., & Davis, A. M. (2003). Elicitation technique selection: How do experts do it? In Proceedings 11th IEEE International Requirements Engineering Conference, 2003 (pp. 169–178). IEEE, Monterey Bay, CA, USA.

Howe, J. (2006). The rise of crowdsourcing. Wired Magazine, 14(6): 1–4.

Issa, A. A., & Al-Ali, A. (2010). Use case patterns driven requirements engineering. In Proceedings Second International Conference on Computer Research and Development (pp. 307–313). Kuala Lumpur, Malaysia.

Kassab, M., & Laplante, P. (2022). The current and evolving landscape of requirements engineering state of practice. IEEE Software. DOI: **10.1109/MS.2022.3147692**.

Kassab, M., Neill, C., & Laplante, P. (2014). State of practice in requirements engineering: Contemporary data. Innovations in Systems and Software Engineering, 10(4): 235–241.

Kassab, M., & Ormandjieva, O. (2014). Non-functional requirements in process-oriented approaches. In Phillip A. Laplante (Ed.), Encyclopedia of Software Engineering (pp. 1–11). Taylor & Francis, Boca Raton, FL.

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

Lim, S. L., & Finkelstein, A. (2011). StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Transactions on Software Engineering, 38(3): 707–735.

NASA (National Aeronautics and Space Administration). (2013). NASA-STD 8719.13, Software Safety Standard. **http://NASA.gov** (accessed January 2017).

Renzel, D., & Klamma, R. (2014). Requirements bazaar: Open-source large scale social requirements engineering in the long tail. IEEE Computer Society Special Technical Community on Social Networking E-Letter, Vol. 2 No. 3.

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

Snijders, R., Dalpiaz, F., Brinkkemper, S., Hosseini, M., Ali, R., & Ozum, A. (2015, August). REfine: A gamified platform for participatory re-

quirements engineering. In *2015 IEEE 1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE)* (pp. 1–6). IEEE, Ottawa, Canada.

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

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

Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., & do Prado Leite, J. C. S. (2005, September). Reverse engineering goal models from legacy code. In 13th IEEE International Conference on Requirements Engineering (RE'05) (pp. 363–372). IEEE, Paris, France.

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