Introduction to Requirements Elicitation

1. Introduction: A Tale of Three Students

Once upon a time there were three students of computer science: Pat, Terry, and Chris. In their programming class, the professor gave this assignment:

> Write a program that will read in a list of 100 positive integers, sort them into ascending order, display the sorted list, and display the average of those values.

These are the *requirements* that the software must satisfy, and the three students had no difficulty in writing the program. Chris and Pat began with pencil and paper, sketching out the algorithm and writing a first draft of the code. Terry went immediately to the keyboard and started typing in the program.

Now our three students, with new computer science degrees in hand, are beginning their first jobs. Pat has gone to work for Consolidated Flange and Widget, a large manufacturing company. One day, Pat and the rest of the software engineering department are called to a meeting where the company's vice president for sales and marketing gives them this assignment:

> Develop an automated system that will allow us to process orders at least 24 hours sooner, on the average, and will allow us to ship our products to customers at least three days sooner than currently.

Terry has taken a job with Zooming Airplane Company and is assigned to the team developing the avionics software for the new Z-676 airliner. The team has just been given this task:

> Develop the software that will allow the Z-676 to land itself, without pilot intervention, at major airports.

Chris has gone to work for Megabuck Codemeisters, a company specializing in personal productivity software for small computers. The company president has called all the new software engineers together and given them this assignment:

> Develop a new product that will sell at least one million copies at a retail price of at least $200.

Unlike the situation in their programming class in school, neither Pat, Terry, nor Chris can head for the keyboard. They need a lot more information on what the software actually must do. How do they get that information? The answer is *requirements elicitation.*

To understand requirements elicitation, we first take a high-level look at the elicitation process: what terminology is used, who participates, and what the basic procedures are. We examine and compare the outcomes of a good elicitation process and a poor elicitation process. We then discuss the underlying difficulties of requirements elicitation. Finally, we sketch several different elicitation techniques that are currently in use by software engineers.

2. The Requirements Elicitation Process

Requirements elicitation is one of the most critical steps in a software engineering project. Experience over the last 30 years has shown that incorrect, incomplete, or misunderstood requirements are the most common causes of poor quality, cost overruns, and late delivery of software systems. The ability to employ a systematic process for requirements elicitation is therefore one of the fundamental skills of a good software engineer.

2.1. Terminology

There are many terms that are used in describing the process of understanding requirements for a software system. We use *requirements engineering* as a general term encompassing all the activities related to requirements. In particular, requirements engineering comprises four specific processes:

requirements elicitation	the process through which the customers, buyers, or users of a software system discover, reveal, articulate, and understand their requirements.
requirements analysis	the process of reasoning about the requirements that have been elicited; it involves activities such as examining requirements for conflicts or inconsistencies, combining related requirements, and identifying missing requirements.
requirements specification	the process of recording the requirements in one or more forms, including natural language and formal, symbolic, or graphical representations; also, the product that is the document produced by that process.
requirements validation	the process of confirming with the customer or user of the software that the specified requirements are valid, correct, and complete.

In an actual situation, these four processes cannot be strictly separated and performed sequentially. All four are intertwined and performed repeatedly. For example, the expression of requirements in a formal or graphical representation is often helpful in identifying conflicting or missing requirements, and the validation of some requirements often elicits requirements or details that the users had not previously recognized or stated.

We should note that the term *elicitation* is not universally accepted for the process described above. Some software engineers use terms such as requirements *identifying, gathering, determining, formulating, extracting,* or *exposing.* Each of these terms has different connotations. For example, *gathering* suggests that the requirements are already present somewhere and we need only bring them together; *formulating* suggests that we get to make them up; *extracting* and *exposing* suggest that the requirements are being hidden by the users. There is some truth to all of these connotations, as we will see in our discussion of requirements elicitation.

2.2. **A General Elicitation Procedure**

By far, the most common kind of requirements elicitation effort is one that gets information directly from the people who will use the system. In such cases, the elicitation procedure can be described in very general terms as five steps:

1. Identify relevant sources of requirements (the users).
2. Ask them appropriate questions to gain an understanding of their needs.
3. Analyze the gathered information, looking for implications, inconsistencies, or unresolved issues.
4. Confirm your understanding of the requirements with the users.
5. Synthesize appropriate statements of the requirements.

Specific elicitation techniques have evolved from this general procedure by defining detailed processes, specific questions or categories of questions to ask, structured meeting formats, specific individual or group behaviors, or templates for organizing and recording information. We sketch some of these techniques in section 5.

2.3. **Participants in Requirements Elicitation**

A requirements elicitation effort normally involves many people. The software engineer who is responsible for producing the requirements specification (sometimes designated a *software requirements engineer)* leads the effort. He or she is often supported by other software engineers, documentation specialists, or clerical staff.

The potential users of the software are also involved. In a typical information system project, such as that encountered by Pat at Consolidated Flange and Widget, there are many kinds of users who will use the system directly: sales representatives, order processing personnel, shipping department personnel, and accounting personnel. Department managers and company executives are also involved, especially those who have authorized the building of the new system.

At Zooming Airplane Company, Terry sees a different kind of user. The engineers designing the Z-676 airliner know how the various subsystems of the aircraft work and how the avionics software interacts with those subsystems. They are the users who can answer questions about what the software must be able to do. In addition, because the U. S. Federal Aviation Administration (FAA) certifies civilian commercial aircraft and operates the air traffic control system, there are government regulations and standards that must be considered as software requirements. FAA representatives may need to be part of the requirements elicitation effort. Airline pilots also need to be involved, especially in the elicitation of user interface requirements.

Chris faces still different problems at Megabuck Codemeisters. If the new software package they decide to build is a "new and improved" word processor or spreadsheet, a representative sample of users of existing packages should participate in the requirements elicitation process. They can be asked about their likes and dislikes for the packages they now use, and about new features that they would like to have. On the other hand, if the new package is an unprecedented kind of system, it is more difficult to elicit detailed requirements. Market research may identify the need for the system, and hence identify very general requirements, but the detailed requirements may have to come from a series of prototypes and user tests.

The lesson to be learned is simple: no one person knows everything about what a software system should do. There are always many participants in a successful requirements elicitation effort.

3. Outcomes of Requirements Elicitation

The tangible result of requirements elicitation is a set of requirements that can be used by the software development team. However, there are many other intangible outcomes of the process that can affect the overall success of the project. Those outcomes differ, depending on whether the elicitation process was conducted well or poorly.

3.1. Outcomes of a Good Process

The buyers or users of a software system often come to the requirements elicitation process with only a vague idea of what they really need and with little idea of what software technology might offer. A good elicitation process helps them explore and fully understand their requirements, especially in the separation of what they *want* and what they *need.* Their interactions with the software engineer help them understand the constraints that might be imposed on the system by technology, organizational practices, or government regulations. They understand alternatives, both technological and procedural, that might be considered in the proposed system. They come to understand the tradeoffs that might need to be made when two requirements cannot both be satisfied fully.

Overall, the buyers and users have a good understanding of the implications of the decisions they have made in developing the requirements. This results in fewer surprises when the system is built and delivered. The buyers and users share with the software engineer a vision of the problems they are trying to solve and the kinds of solutions that are feasible. They feel a sense of ownership of the products of the elicitation process. They are satisfied with the process, feel informed and educated, believe their risk is minimized, and are committed to the success of the project.

Similarly, the software engineers and developers who have participated in the requirements elicitation process are solving the right problem for the users. This is obviously the most important result of a good process; otherwise the whole project will fail. The developers have clear, high-level specification of the system to be built.

The developers are also confident that they are solving a problem that is feasible from all perspectives, not only technical but human. They know that the customers will be able to use the system, like it, make effective use of it, and that the system will not have undesirable side effects. They have the trust and confidence of the customers; they know the customers will cooperate if clarifications are needed during development, but they also believe such interaction will be minimal.

The developers have gained knowledge of the domain of the system; they have a variety of peripheral or ancillary information about the system that will be useful later when making low-level tradeoffs and design decisions. However, they do not feel that the system is overly specified; they are comfortable that they have freedom to make implementation decisions.

3.2. Outcomes of a Poor Process

The most serious outcome of a poor requirements elicitation process is that the developers are solving the wrong problem. This guarantees the failure of the whole project. (Take another look at Figure 1 at the beginning of section 2.)

Even if the developers are solving essentially the right problem, a poor elicitation process can have other negative outcomes. The buyers and users can be dissatisfied; this often happens if the developers did not really listen to them, or if the developers dominated the process and tended to force their own views and interpretations on the buyers and users. Dissatisfaction may result in less effective participation by the buyers and users, resulting in less complete answers to the developer's questions. The dissatisfaction can continue to affect the project through development and delivery of the software.

A poor elicitation process often leads to a chaotic development process. The developers may discover that they are missing important information, resulting in additional meetings with the buyers and users. The developers may make the wrong decisions or tradeoffs because of a lack of understanding of the users' needs. Requirements may change more often, resulting in greater need for configuration management, or in delays or wasted effort in design and implementation. The result is cost and schedule overruns, and sometimes failed or canceled projects.

All of these effects can result in a loss of money for the company developing or buying the software, loss of reputation or credibility for the developers, and a decline in the developers' morale.

4. Underlying Difficulties of Requirements Elicitation

Requirements elicitation is an imprecise and difficult process. To do it successfully requires that we overcome the underlying difficulties. In this section we discuss those difficulties, and in the next section we see some of the elicitation techniques that have been created to overcome the difficulties.

Throughout this discussion, we use the term *user* to mean both the actual user of the software (in the case where there is a human user) and the buyer or customer. For example, at Consolidated Flange, the users of Pat's software are the sales staff and the clerical staff that process orders. Terry's "users" might be considered to be the pilots or passengers of the Z-676 airliner being flown by the software, but the "customers" are really the engineers designing the flight controls for the aircraft. At Megabuck Codemeisters, the ultimate users of the new package that Chris is developing are the unknown buyers of the package, but the customers who understand the requirements are the people within the company who have done the market research to determine what kind of package is likely to be a big seller and who have examined competitors' products to identify how the Codemeisters product can be better.

4.1. Articulation Problems

The first class of difficulties includes those related to the articulation of the user's needs. These include problems both with the user's expression of needs and the developer's understanding.

1. The users of a proposed software system may be aware of their needs, but they are unable to articulate them appropriately. This is analogous to a situation where you recognize you are hungry and go into a restaurant. If you cannot decide what you want to eat, or if you cannot understand the menu, you cannot articulate your requirements. Telling the waiter "I'm hungry" is a statement of need but not a sufficiently articulate requirement to which the waiter can respond.

2. The users may not be aware of their needs. They may not understand how the technology may be able to help them. For example, the sales staff at Consolidated Flange may not know that with portable computers, modems, and appropriate software, they could send orders via telephone lines back to the main office during a sales trip, rather than waiting until they returned.

3. The user may be aware of a need but be afraid to articulate it. For example, a relatively new user at Consolidated Flange knows that he has trouble remembering all the part numbers when filling out customer order forms. He would like the system to display the part numbers in a menu, rather than having to type them in. However, he knows the other users don't have this problem, and he believes they would think him to be incompetent if he articulated his need. So he says nothing.

4. Users and developers misunderstand concepts or relationships because they have different meanings for common terms. Words like *system* and *integration* are widely used but understood differently by developers and users in many domains. To the developer, the word *implementation* means the writing of source code. To the user it

means the process of making the software system operational in an organization, including the associated changes in human behavior, management procedures, and accounting procedures.

5. Users cannot make up their minds on some issues because they don't understand the consequences of the decision or they don't understand the alternatives.

6. No single person has the complete picture. No matter how articulate a user may be in expressing needs, other users may have different or additional needs or different priorities. This is especially true for complex systems, where each individual user may have only a limited view or perspective of the system to be built. For example, some users of a word processing system may never have produced a document with an index and therefore will probably not ask for this feature. Only a few users might think of features like text change bars or switching from portrait to landscape mode in the middle of a document.

7. Developers may not really be listening to the users. The developers don't hear all the detailed information that the users are providing. This usually happens when the developers believe they already understand the user's needs, or when they begin to think abead to particular designs and implementations.

8. Developers may fail to understand, appreciate, or relate to the users. They may not empathize with the user's problems or be able to see the problems from the user's perspective. In such situations, the developers will not understand the users' context, **issues, or concerns.**

9. Developers tend to overrule or dominate the users. They may have an overly assertive style, projecting an image of knowing all about the technology and the buyers' domain. The users feel threatened and are unable to articulate their actual require-**ments.**

4.2. Communication Barriers

Many requirements elicitation difficulties are a direct result of differences in communications among users and developers.

1. Users and developers come from different worlds and have different professional vocabularies. The users may come from a financial, engineering, aeronautics, or manufacturing domain. Developers belong to the software domain. A term such as *process an order* might be well understood by the user but not by the developer.

At Zooming Airplane, Terry and the other developers discover that the users give them a blank stare when they start discussing class hierarchies and module cohesion. Terry has a similar reaction when the users mention VOR radials and RF interference.

2. The users have different concerns from those of developers; these are usually high-level attributes like usability and reliability. In contrast, developers are concerned with low-level technical issues, such as resource utilization, algorithms, and hardware/software tradeoffs.

3. Problems exist with each form or medium of communication. Natural languages,

such as English, are inherently ambiguous. This often proves useful in normal communication but it is a significant problem for requirements communication. So why would we choose natural language for requirements elicitation? Usually, it is the only common communication medium between developers and the users.

Other forms of communication, such as diagrams, charts, pictures, and artificial languages, can sometimes be used. However, every form has some things it communicates well and some that it communicates poorly. It is usually helpful to use several forms in order to cover all the blind spots.

4. Requirements elicitation, by its very nature, has significant social interaction, and the people involved are all different. Some are assertive, some are submissive; some deal with details and others with abstraction. Incompatible styles of interaction can lead to a breakdown of communication. The elicitor must try to recognize the incompatibilities and adjust the communication appropriately.

5. There are different personality types and different value systems among people. This can lead to unexpected difficulties in communication, as was discovered by a company that contracted to build an information system for a university. The project leader was a high-level person in the company, and he would only talk to comparably high-level people in the university-deans and vice presidents. The developers on the project would only talk to the lower level clerical staff in the university who would actually use system.

4.3. Knowledge and Cognitive Limitations

Buyers, users, and developers are human beings, and each brings some knowledge and cognitive limitations to the process. They vary from person to person.

1. The requirements elicitor must have adequate domain knowledge. A common error is that the team of users and the developers don't have adequate domain knowledge, so they make wrong decisions. Developers should not make domain tradeoffs, and the users should not make technical tradeoffs.

2. No person has perfect memory. The users and developers may not remember exactly what was said or decided. Furthermore, we all interpret oral and written communications differently. Even if we believe we are being careful to record what was decided, we may misinterpret that information later.

3. We often try to use quantitative information and statistics to express needs and requirements. However, informal or intuitive statistics are frequently interpreted differently by different people because of our own experiences and biases.

4. People sometimes have difficulty with scale and complexity. As problems become larger, we deal with them in different ways. Some people try to simplify the problem, but not always in a valid way. Some people simply ignore parts of the problem because they can't deal with them. Our perspective of the problem can become distorted.

5. We often have a preconceived approach to the solution of a problem that affects our ability to state the problem clearly. We tend to state the problem in terms of the favored solution.

6. Some people develop a kind of "tunnel vision" when discussing a problem-they quickly focus all their attention on a few narrow aspects of the problem, usually those aspects that they believe they understand best or that affect them most directly.

7. On large systems, we usually need to explore a variety of novel formulations of the problem before reaching consensus on the nature of the problem. Some people are uncomfortable or impatient with this kind of exploration.

4.4. Human Behavior Issues

Requirements elicitation is a social process, so human behavior issues are involved.

1. There are sometimes conflicts and ambiguities in the roles that the users and developers play in the requirements elicitation process. Each user may assume that it is some other user's responsibility to tell the developers a particular aspect of the requirements, with the result being that no one tells the developer. The developer may assume that the user is a domain expert and will give all the needed domain information, and the user may assume that the developer will ask appropriate questions to get the domain information. This misunderstanding often leads to gaps in the requirements.

2. The development of a software system to support an organization usually results in an expectation or fear that installation of the software will necessitate all kinds of changes in behavior of individuals and groups (including the potential loss of jobs). This can cause individuals to withhold information from the developers or, in extreme cases, actively sabotage the development effort.

4.5. Technical issues

There are many other difficulties that we might characterize as *technical* that must be overcome by the requirements elicitation process if it is to be successful. Some of the more important of these are summarized below.

1. Problems to be solved by software systems are becoming increasingly complex. The requirements of these systems are based on increasingly detailed knowledge of the user's domain. The impact of the systems on society must be considered, but neither the users nor the developers may be skilled at identifying that impact.

2. Requirements change over time. The requirements elicitation process itself is a learning experience for users, and ideas discussed at one point may cause them to change their minds about prior decisions. We must be careful to avoid having a set of requirements that is obsolete by the time the elicitation process is completed.

3. Software and hardware technologies are changing rapidly. A technological advance may make feasible a requirement that was unacceptably complex or expensive yesterday.

4. There are many sources of requirements. The users of a system are not necessarily aware of all the requirements that the system must satisfy. There may be requirements best elicited from computer operators or users' support personnel. Corporate management may have guidelines for performing certain tasks or constraints that must be satisfied. There may be government regulations or industry standards for particular

aspects of a system. The marketing and sales departments may have requirements that would help improve the commercial viability of a product, especially when there are already similar competitors' products on the market.

5. The nature or novelty of the system often imposes constraints on the elicitation process. A new system that is very similar to several other systems previously built by the development team may be able to benefit from previous requirements elicitation efforts and feedback from users of the previous systems. An unprecedented system requires a much more substantial requirements elicitation effort.

Requirements elicitation for a one-of-a-kind system built for a specific customer can normally assume that the customer is the ultimate authority on what is needed. On the other hand, if the system will be offered for sale to customers other than the original buyer, the developers should look also at competing systems and additional or different requirements from those other customers.

Requirements elicitation for a typical shrink-wrapped, personal productivity software package depends heavily on market research, examination of competing products, and some kind of communication with a sample of typical users. A software system that goes through many versions over many years needs a continuing elicitation process to identify defects in the current version and to track users' requests for enhancements.

For a real-time control system, requirements elicitation often includes detailed collaboration with hardware and systems engineers to decide what functionality will be implemented in hardware (computer or otherwise) and what in software.

5. Overview of Requirements Elicitation Techniques

The requirements elicitation techniques that have been developed and used by software engineers have usually been designed to overcome one or more of the underlying difficulties. Some address communications difficulties, while others address human behavior or technical difficulties. Some are high-level, in that they are broad frameworks for a process that elicits general requirements; some are low-level, in that they provide specific tactics for eliciting details about a particular part of the system or from a particular user.

We can, to some extent, describe requirements elicitation techniques in broad, generic categories:

Asking. Identify the appropriate person, such as the buyer or user of the software, and ask what the requirements are.

Observing and inferring. Observe the behavior of users of an existing system (whether manual or automated), and then infer their needs from that behavior.

Discussing and formulating. Discuss with users their needs and jointly formulate a common understanding of the requirements.

Negotiating with respect to a standard set. Beginning with an existing or standard set of requirements or features, negotiate with users which of those features will be included, excluded, or modified.

Studying and identifying problems. Perform investigations of problems to identify requirements for improving a system. For example, if a system is too slow, it may require complex performance monitoring to identify the requirements to change the system. For a system with thousands or millions of users, a statistically valid survey using questionnaires may be needed to identify significant problems with the system.

Discovering through creative processes. For very complex problems with no obvious solutions, employ creative processes involving developers and users.

Postulating. When there is no access to the user or customer, or for the creation of an unprecedented product, use creative processes or intuition to identify features or capabilities that the user might want.

To illustrate these generic techniques, let's reconsider the software engineering projects described in section 1 and ask which of these techniques are Pat, Terry, and Chris likely to find most useful or least useful.

Pat faces a relatively common requirements elicitation task. The best technique is probably discussing and formulating requirements with the users. Joint Application Design, described below, is this kind of technique, and it is widely used for information systems. Postulating requirements would probably be the least useful technique in this situation, especially since Pat is new to the company.

Terry will certainly have to discuss and formulate requirements with the hardware engineers who understand the flight characteristics and controls of the aircraft. Observing pilots landing might also be helpful. Because this kind of software is almost unique and unprecedented, negotiating requirements with respect to a standard set is not possible, nor is studying and identifying problems with an existing system.

Chris may have the most difficult task, although the resulting requirements may not be as complex as those of Terry's project. Postulating the requirements may be necessary if Codemeisters decides to create an unprecedented product. If they instead choose to build a product that will compete head-to-head with those of competitors, it will be useful to study the existing systems to identify their weaknesses. The least useful techniques might be asking and discussing with users, because the users have not been identified.

We should note that no one technique is sufficient for realistic projects. A software engineer must be able to choose an assortment of techniques that best fit the kind of system being built.

We take a brief look at several techniques below. For each, we try to identify some of the underlying difficulties of requirements elicitation that are addressed by the technique.

<<N.B. *Some sections omitted*>>

5.1.1. Prototyping

In some situations, users may be better able to understand and express their needs by comparing those needs to an existing or reference system. When there is no similar existing system, prototyping can be used to create a system that illustrates the relevant features. By examining prototypes, the users can learn what their needs really are.

The prototyping process begins with a preliminary study of user requirements. Next comes an iterative process of building a prototype and evaluating it with the users. Each iteration allows the users to understand their requirements better, including understanding the implications of the requirements articulated in previous iterations. Eventually, a final set of requirements can be formulated and the prototypes discarded.

We sometimes distinguish the terms *prototype* and *mock-up,* with the former being something that demonstrates *behavior* of a part of the desired system, and the latter being something that demonstrates the *appearance* of the desired system. Mock-ups of user interfaces are especially common.

Clearly, prototyping of a system is beneficial only if the prototype can be built substantially faster than the actual system. For this reason, the process has sometimes been called *rapid prototyping.* Many software tools have been developed to facilitate building prototypes and mock-ups.

We also note that prototyping should *not* be viewed as a euphemism for trial-and-error programming or "hacking." These are wasteful practices. Prototyping is properly used to elicit and understand requirements; it is followed by a structured and managed process to build the actual system. Software engineers need to be careful to avoid making an inappropriate commitment to any prototype as the basis for full development.

When properly used, prototyping can be remarkable in overcoming articulation problems and communication barriers. At one time or another, we have all had experiences that cause us to think "I don't know what I want, but I'll know it when I see it," or "I didn't know I wanted one of those until I saw one." Prototyping provides this kind of experience during requirements elicitation.

<<NB some sections omitted>>

5.2. Detailed Techniques

The detailed techniques for requirements elicitation generally provide operational-level tactics and guidelines. They usually focus narrowly on specific aspects of the elicitation process.

5.2.1. Brainstorming

Brainstorming is a simple group technique for generating ideas. It allows people to suggest and explore ideas in an atmosphere free of criticism or judgment.

A brainstorming session works best with four to ten people. One person is the leader, but the role of the leader is more to get the session started than to constrain it.

The session consists of two phases. In the *generation* phase, participants are encouraged to offer as many ideas as possible, without discussion of the merits of the ideas. In the *consolidation* phase, the ideas are discussed, revised, and organized.

For purposes of software requirements elicitation, brainstorming can be helpful in generating a wide variety of views of the problem and in formulating the problem in different ways. It is especially useful very early in the elicitation process.

Good brainstorming sessions are very helpful in overcoming some of the cognitive limitations of participants by allowing (or forcing) them to expand their thinking. The lack of criticism and judgment during the generation phase also helps overcome some of the communication barriers of requirements elicitation.

5.2.2. Interviewing

Interviewing is an important technique for eliciting detailed information from an individual. It is commonly used in requirements elicitation for large systems as part of some of the high-level elicitation techniques. It can also be used for small projects as the only requirements elicitation technique

Interviewing is not simply a matter of asking question. It is a more structured technique that can be learned, and software engineers can gain proficiency with training and practice. It requires the development of some general social skills, the ability to listen, and knowledge of a variety of interviewing tactics.

A skilled interviewer can help the user to understand and explore software requirements, thus overcoming many of the articulation problems and communications barriers.

Requirements Elicitation by Brainstorming

Brainstorming is a simple group technique for generating ideas. It allows people to suggest and explore ideas in an atmosphere free of criticism or judgment.

A brainstorming session works best with four to ten people. One person is the leader, but the role of the leader is more to get the session started than to constrain it.

The session consists of two phases. In the *generation* phase, participants are encouraged to offer as many ideas as possible, without discussion of the merits of the ideas. In the *consolidation* phase, the ideas are discussed, revised, and organized.

For purposes of software requirements elicitation, brainstorming can be helpful in generating a wide variety of views of the problem and in formulating the problem in different ways. It is especially useful very early in the elicitation process. When used correctly, it can help overcome some of the underlying difficulties of requirements elicitation:

- It stimulates imaginative thinking to help users become aware of their needs.
- It helps build a more complete picture of the users' needs.
- It can avoid the tendency to focus too narrowly too soon.
- For some personality types, it provides a more comfortable social setting than some of the more structured group techniques.

Brainstorming also has the advantage that it is easy to learn and requires very little overhead With practice, the participants can become very good at it. On the other hand, because it is an unfacilitated and relatively unstructured process, it may not produce the same quality or level of detail of some other processes.

Conducting a Brainstorming Session

Preparation for a brainstorming session requires identifying the participants, designating the leader, scheduling the session with all participants, and preparing the meeting **room.**

The participants are those who normally participate in requirements elicitation: customers, buyers, and users who need the software, and the software engineers who will

This document is taken from the SEI educational materials package "Lecture Notes on Requirements Elicitation" by Sridhar Raghavan, Gregory Zelesnik, and Gary Ford, document number CMU/SEI-94-EM-**l0, copyright 1994 by Carnegie Mellon University. Permission is granted to make and distribute copies for noncommercial purposes.**

develop the software. The outcome of the session depends on the ideas generated by the participants, so it is essential to include people with knowledge and expertise appropriate to the system being built.

The leader opens the session by expressing a general statement of the problem. This *seed* expression should be general, but still sufficiently focused to put the session on the right track.

The participants are then free to generate new ideas relevant to the problem expression. Some leaders prefer to give each participant in turn an opportunity to express one idea, going around the table as many times as necessary. Other leaders take ideas from participants in any order, selecting them on the basis of a raised hand. The process continues as long as ideas are being generated.

Alex F. Osborn, a researcher and writer on creating thinking, offers four rules for the generation phase of the session:

1. Criticism of ideas is absolutely forbidden. Participants must feel totally free to express any idea.
2. Wild, offbeat, or unconventional ideas are encouraged. Such ideas often stimulate the thinking of participants in unintended and unpredictable directions, which can lead to really creative approaches to the problem.
3. The number of ideas generated should be very large. The more ideas proposed, the more good ones are likely to be present.
4. In addition to suggesting totally new ideas, participants should be encouraged to combine or embellish ideas of others.

To facilitate this last rule, it is necessary for all ideas to remain visible to the participants. Several techniques can be used to do this; the technique used may depend on the equipment available in the meeting room.

- One person, either the leader or a *scribe,* is designated to record all ideas on a whiteboard or large sheets of paper. Unless the meeting room has wall-to-wall whiteboards, flip chart pads are probably better. As each sheet is filled, it is posted in view of all participants.
- Participants step to the whiteboard or flip chart to record their own ideas.
- Several smaller sheets of paper are used, and they are placed in the middle of the table where all participants can reach them. When an idea is proposed, it is added to any of the sheets.

The generation phase can conclude in either of two ways. If the leader believes that not enough ideas are being generated, the meeting can be stopped. The group reconvenes and continues at another time when people (and their ideas) are fresh. If enough ideas have been generated and recorded, the leader can move the meeting to the next phase.

The consolidation phase permits the group to organize the ideas in ways that they can best be used. It is in this phase that evaluation of ideas takes place.

The first step is usually to review the ideas for the purpose of clarification. It may be necessary to reword some of the ideas so that they are better understood by all participants. During this step, it is also common for two or more ideas to be recognized as being essentially the same, so they may be combined and reworded to capture the sense of the originals.

Next, the participants can usually agree that some of the ideas are too wild to be usable. These are discarded.

The remaining ideas are then discussed with a goal of ranking or prioritizing them. In the case of software requirements, it is often necessary to identify those that are absolutely essential, those that would be nice but not essential, and those that might be appropriate for a second or subsequent release of the system.

After the session, the leader or other designated person produces a record of all the remaining ideas, along with their priorities or other relevant comments from the consolidation phase.

Tools to Support Brainstorming

There is an area of research called *computer-supported cooperative work* (CSCW) that is developing tools and techniques by which people can work together without necessarily being located in the same room or building. A few tools are starting to appear that could be applied to brainstorming.

Videoconferencing tools are an example. With appropriately configured and networked workstations, the participants in a brainstorming session could remain in their offices and still be seen and heard by all other participants. The ideas could be entered by the individual participants or by a scribe, with each participant seeing the ideas immediately on the workstation screen.

The effectiveness of these tools is still uncertain. Some people believe the tools may first be useful in the consolidation phase, which involves editing and reordering the statements of the ideas. Doing this online provides the group an opportunity to evolve the final idea list during the session.

Suggested Reading

These books contain detailed discussions of brainstorming, although not in the context of software requirements elicitation.

Clark, C. H. *Brainstorming.* Garden City, N. Y.: Doubleday & Company, Inc., 1958.

Osborn, Alex F. *Applied Imagination, Principles and Procedures of Creative Thinking.* New York: Charles Scribner's Sons, 1953.

Requirements Elicitation by Interviewing

Interviewing is an important technique for eliciting detailed information from an individual. As a software engineer, you will use it in requirements elicitation for large systems as part of some of the high-level elicitation techniques. For small projects, you may also use interviewing as your only requirements elicitation technique.

Interviewing is not simply a matter of asking questions. It is a more structured technique that you can learn, and you can gain proficiency with training and practice. It requires the development of some general social skills, the ability to listen, and knowledge of a variety of interviewing tactics.

Interviewing has four phases: identifying candidates, preparing, conducting the interview, and following up. We discuss these phases in detail below.

Identifying Candidates for Interviewing

Requirements elicitation by interviewing begins with identifying the people to be interviewed. You usually start with the person who has authorized or is sponsoring the project to build the software system; this is often a manager or executive. The organization chart for a company helps identify other relevant people-those who report to that manager. These are the people who know why the system is being built and who will use it.

A requirements elicitation effort may involve interviewing many people, but it is not necessary to identify all of them before starting the interviews. One line of inquiry in each interview is the determination of other people who should be interviewed. This is done with questions such as:

- "Who else should I talk to?"
- "Who else may use the system?"
- **"Who will agree with you on this?"**
- "Who will disagree with you on this?"

You should also consider people who may not be actual users of the system to be built, but who interact with the users. Those interactions may be changed or disrupted after the system is installed, and you want to minimize these negative effects. You can ask:

This document is taken from the SEI educational materials package "Lecture Notes on Requirements Elicitation" by Sridhar Raghavan, Gregory Zelesnik, and Gary Ford, document number CMU/SEI-94-EM-**l0, copyright 1994 by Carnegie Mellon University. Permission is granted to make and distribute copies for noncommercial purposes.**

- **"Who else interacts with you?"**

Preparing for an Interview

There are two major activities in preparing for an interview: making arrangements with the people to be interviewed and preparing a list of questions.

Interviews must always be scheduled in advance, both as a matter of courtesy and to allow the interviewees to be prepared. You should make them aware of the goals of the interview, agree on the length of the interview, and give them any relevant materials they will need in order to prepare. You should also remind them of the interviews a day or two in advance; this can help ensure that they do the preparation.

Interviews are sometimes recorded on audio or video tape. Because taping makes some people nervous and thus affects the quality of the information gained from the interview, you should secure permission of the interviewees in advance.

Prepare in advance a list of questions to be asked at the interviews. Because interviewing is used to elicit detailed software requirements, you already have general ideas of the kind of system to be built. These general ideas will guide you in the preparation of questions. On the other hand, you cannot prepare *all* questions in advance. Information that you get during the interview will open new areas of inquiry, and you will need to create additional questions as you go.

Organize the list of questions into a logical order and arrange it as groups of questions about related issues. Finally, decide how much time to devote to each issue.

Interview Process Protocol

Beginning the interview. To get the interview started, introduce yourself (assuming you do not already know the interviewee). Next, review the goals of the interview: why you are here, what will be done with the information collected, the kinds of issues that will be covered, and the time allocation among the issues. During this review you can assess the extent to which the interviewee is prepared. In rare instances, the lack of preparation by the interviewee will necessitate stopping and rescheduling the interview at a later time.

Software requirements are often expressed in mathematical or graphical notations, such as data flow diagrams or state transition diagrams. If you are using any such notations, you should explain them at the start of the interview in order to be sure they are understood.

General guidelines. During the interview, you of course ask your prepared questions. However, there are oral communication skills and strategies that you can use to increase the quality of the information received.

First, you should keep in mind the fact that a person's first answer to a question will not necessarily be complete and correct, nor will it necessarily be expressed in language that you understand as clearly as the interviewee. You will need to explore most answers to improve your understanding. Some of the best ways to do this are to summarize, rephrase, and show implications of what you hear, so that the interviewee can confirm your understanding.

Summarizing is useful throughout the interview, not just at the end. It helps confirm understanding and it can elicit useful generalizations and higher level abstractions.

Rephrasing answers-stating information in your own words-is an important strategy for dealing with ambiguity in language. It helps you understand an issue by forcing you to translate the understanding into words. Rephrasing also helps uncover misunderstandings of specialized terminology.

For interviews in the context of software requirements elicitation, you as the software professional bring a range of technical knowledge to the interview that the interviewee does not have. This often gives you insight into the implications of a particular user requirement. It is helpful to explain those implications to the user, who may then decide that is not what was wanted after all.

Be an active listener during the interview. Look at the interviewee when he or she is speaking. When making notes, avoid the tendency to stop listening. If necessary, you can ask the interviewee to pause while you are writing.

Be courteous during the interview and try to keep the interviewee at ease. Avoid questions that might seem threatening, such as "I want an answer! Yes or no?"

You should allow the interviewee the opportunity to answer questions fully. Sometimes this results in wandering from topic to topic. This is acceptable, but you then must choose your next questions carefully to bring the interview back on track. You must remain in control of the interview.

You can also make use of some non-verbal communication techniques during interviews. In particular, body language can be an important indication of the mood of the interviewee. If body language suggests that he or she is becoming closed or less receptive to questions, you may need to move the discussion to a different issue or take other action to reduce the stress.

Keeping the process visible. From time to time it is useful to make comments or ask questions about the interview itself, in addition to the questions about the software requirements. Questions such as these help ensure that the process is going well:

- "Are we doing all right?"
- **"Have we ignored anything?"**
- "Did we spend enough time on this issue?"

Make sure the interviewee understands the rationale for your questions. If asked, you should explain the purpose of a question.

You should take care, however, to remain in control of the interview. Don't accept too many questions, and if the discussion moves away from the subject of the interview, be prepared to point it out to the interviewee.

Types of questions. There are a few general types of questions that you will almost always use in interviews. Protocol questions address the context for the software system rather than the behavior of the system itself.

- "Why are we building this system?"
- "What do you expect from it?"
- **"Who are other users of this system?"**

Open-ended questions encourage unconstrained answers and can elicit a large amount of information. They can be very useful when you don't yet know enough about the system to ask more detailed questions.

- "Tell me what you do."
- "What aspects of your job are tedious?"

Closed-ended questions are useful when you need to educate the interviewee about a particular issue and force a precise or detailed answer.

You should be careful when asking some kinds of leading questions, depending on the personality and mind-set of the interviewee. For example, compare these two questions:

- "Should the sales report be produced weekly?"
- "How often should the sales report be produced?"

A "yes or no" question allows the interviewee to make a complete response without giving the question much thought. If you use too many such questions with a passive user, you may end up with your own view of the requirements instead of those of the **user.**

Avoid the tendency to anticipate an answer. When you have asked your question, stop talking. For example, if you ask, "How often should the sales report be produced?" don't follow immediately with "Daily? Weekly? Monthly?"

Software requirements are often complex, and the user may not have a fully developed understanding of his or her needs. This normally means that a single question about an issue may not elicit a complete or meaningful response. You should explore issues with questions that approach the issue from different directions, or that are at different levels of abstraction.

You should also ask questions to raise the level when the interview begins to get too detailed or too focused on a single solution to the problem. When the user says that a specific function is needed, you can ask a series of *laddering* questions to raise the level:

- "What is the goal of that?"
- **"What is its purpose?"**
- "By what means will that be accomplished?"

You may need to ask these questions two or three times, each time forcing the answer to be at a higher level.

Putting questions in context. During the course of the interview, you will switch topics or question contexts from time to time. Make sure the interviewee understands the context in which you are asking each question. You can often depend on the context of previous question, but after changing topics, you should explicitly state the new context. Otherwise you may get unreliable details.

For example, if you pose a question about the format of particular data items, the answer may depend on whether the context is a discussion of input data or output data.

Avoid switching context too often, because this prolongs the interview and increases confusion.

Checking for errors. During the interview, you must be sensitive to communication errors, check for them periodically, recognize when they occur, and correct them. Some of the most common kinds of errors are:

- Observational errors: when viewing a phenomenon, different people focus on different aspects and may "see" different things.
- Recall errors: the interviewee may be relying on recall of specific information, and human memory is fallible.
- Interpretation errors: you and the interviewee may have different interpretations of common words, such as *"small* amount of data" or *"special* characters."
- Focus errors: you may be thinking broadly, while the interviewee is thinking narrowly about an issue (or vice versa), which affects the level of abstraction in the discussion of that issue.
- Ambiguities: there are inherent ambiguities in most forms of communication, and especially in natural language.
- Conflicts: you and the interviewee may have conflicting opinions on an issue, resulting in a tendency to record your own view rather than what the interviewee is saying.
- Facts that are simply not true: the interviewee may give information as fact that is really judgment or opinion; you should check facts with other sources, especially those facts on which you will base significant decisions.

With experience, you can learn to recognize when errors like these might have occurred. You can then ask a question to confirm the error, and ask additional questions to correct the error.

Ending the interview. The interview can end when all the questions have been asked and answered, when the allotted time has been exhausted, or when you sense that the interviewee is becoming too fatigued or "drained" to continue.

Be sure to leave five to ten minutes for summarizing and consolidating the information you have received. Describe briefly the major issues that you believe you have adequately explored and those, if any, that you believe require additional information. Explain the follow-up actions that will be taken, including an opportunity for the interviewee to review a written summary of the interview. Solicit and answer questions about the interview, the follow-up actions, or what will happen to the information collected. Finally, thank the interviewee for the time and effort he or she has given.

Follow-up Activities

After conducting an interview, there are a few activities that you should perform. As a courtesy, it is usually appropriate to send the interviewee a written expression of thanks.

The most significant post-interview activity is to produce a written summary of the interview. The process of writing the summary provides an opportunity to reorganize or reorder the topics discussed and to consolidate related information. It may also help you uncover ambiguities, conflicting information, or missing information.

Give the interviewee a copy of the summary and request confirmation that the summary accurately reflects the information exchanged in the interview.

If the interview produced statistical or other factual information that depended solely on the memory of the interviewee, you should confirm that information with reliable sources.

Finally, you should review the procedures you used to prepare for and conduct the interview, with the goal of finding ways to improve the process in the future. You may want to pay particular attention to the kinds of questions that you found most or least successful in eliciting useful information. If you will conduct interviews with several potential users of a new software system, you can revise your prepared questions before the next interview.

Suggested Reading

This book contains a variety of information about interviewing, including an especially helpful section titled "General Suggestions for Beginners."

Bingham, W. V. D.; & Moore, B. V. *How To Interview, 4th Revised Edition.* New York: Harper & Brothers Publishers, 1959.