Editor: Forrest Shull ■ Fraunhofer Center for Experimental Software Engineering, Maryland ■ fshull@fc-md.umd.edu

Understanding the Customer: What Do We Know about Requirements Elicitation?

Oscar Dieste, Natalia Juristo, and Forrest Shull

I once worked in the IT department of a start-up company doing research in materials science (an area I had next-to-no experience in). I had just spent a few weeks developing a software application that was going to help the scientists make sense of the results of a round of experiments. I was sure that my latest prototype was going to elicit much gratitude for the way it would improve their job. But when I sat down with the user representative, we ended up with a long list of changes that were needed before the software would really support him in the job he wanted to do.

I thought I had listened to him with respect and taken all his comments with good humor. But when he got defensive toward the end of the meeting, I realized just how upset I was that the software I'd poured so much energy into still had so far to go. I heard the customer defending himself: "Well, I couldn't really understand all the fine points of what the software should do until I saw the basics in place!" And I thought, well, so much for the illusion that I have any kind of poker face.

I'd be willing to bet that most software developers have had at least one meeting like this one at some point in their careers. This is admittedly a small-scale example, but it points to something I've always found interesting. Of all the activities associated with software development, understanding the customer requirements is one of the most important. In my experience, however, it still relies mainly on the same old proven techniques: interviewing the customer and showing him or her prototypes to get feedback. It's interesting to consider whether research has provided any clues on how to better accomplish this.

—Forrest Shull

"Getting the requirements right" is one of the most important activities in software development. Making a crucial misstep at this phase can easily lead to large amounts of rework when the customer simply can't accept a system the way it was developed—and we all know that trying to change an existing system to incorporate new functionality is almost always harder than developing the system right the first time. When used correctly, approaches such as incremental development or agile methods can mitigate the risks of getting the requirements wrong by making sure that systems are developed in smaller chunks and that each chunk can be shown to the customer for approval. However, the best way to develop a high-quality system with minimal effort is still to get the requirements right the first time.

Have you got a better way?

We found two papers that attempted to summarize the state of the practice regarding requirements elicitation.[1,2] Both confirm the key role of unstructured interviews between the developers and the customer representatives. (To be precise, "unstructured interviews" in this context means any kind of unstructured interaction between the developers and the user representatives. Rather than having a planned set of concrete, focused questions, the interviewer brings general questions to help the user talk about the problem domain.) The data from Colin Neill and Phillip Laplante in particular also point out that developers use prototyping extensively.

Both papers point out several strategies that can help with brainstorming requirements in group scenarios, such as focus groups. To keep focused here, however, we limit our conversation to techniques for one-on-one scenarios. Even with this restriction, the number and range of proposed alternative techniques might surprise developers:

- In *protocol analysis,* the user representative thinks aloud while solving the problem that the software will support.
- In *card sorting*, the user puts similar concepts or entities from the problem domain into groups. Each concept is printed on a card so that the user can physically arrange or rearrange them to assist with categorizing.
- In *textual laddering*, the user starts from a single seed concept and answers short, directed questions, which help explicate the important concepts in the problem domain. These concepts are arranged into a hierarchy of problem domain elements.

All in all, our systematic literature search found 30 studies that compared at least two techniques to examine which are the most effective in different circumstances. (You can find the details of this review, including our specific search criteria and a list of the studies, in a Web appendix at www.computer.org/portal/pages/software/articles/s2voe.html.) Across these studies, we found 43 different elicitation techniques. Given the many different approaches, it's perhaps not surprising that experts have produced papers that try to guide developers to the approaches that would best apply to their situations.[3–6] However, these tend to be based either on theory or on one person's experience. It's not that such sources can't be useful, but we'd like to think that with 30 studies to draw from, we could provide a bit more comprehensive guidance.

These 30 studies took place from 1987 to 2005 (when we began our research). The studies are distributed fairly evenly over time, showing a constant level of interest in working out various approaches' strengths and weaknesses. We looked at studies from the software engineering literature as well as related fields, such as marketing, where eliciting a customer's needs is also important.

To arrive at useful guidelines for developers about what approaches work best when, it can be helpful to talk about general classes of elicitation approaches, rather than study each one individually. Doing so can help us reason about strategies that are successful or not, rather than get caught up in discussing relatively small differences from one technique to the next. So, for example, we can analyze the differences among

- interview-based methods, including 18 specific types of interviews, each with a different focus (for example, unstructured interviews, interviews to elicit critical success factors, and interviews to construct data-flow diagrams);
- questionnaires, which are fairly self-explanatory;
- introspective and observational methods, which elicit information about the users' tasks and values (such as researcher analysis of documentation regarding the task or process being followed, observation of the customer at work, and protocol analysis); and
- contrived techniques, which ask users to engage in some kind of artificial task to help elicit information such as priorities, domain concepts, or goals and subgoals (such as card-sorting and similar strategies for understanding the domain, decomposing goals into finer-grained tasks, or creating hierarchies as in textual laddering).

Each study has measured completely different effects of the elicitation approaches, making it hard to summarize multiple studies in a standardized way.

Why is answering the question so hard?

So, now that we've aggregated categories of elicitation approaches, we should just be able to tally up what all the studies in each category say, and then we'll have a sense of which categories are most effective, right? If only. Unfortunately, we quickly run into a problem that seems typical for empirical software engineering: when we try to abstract a general conclusion from multiple studies, we find that each study has measured completely different effects of the elicitation approaches. This makes it hard to summarize multiple studies in a straightforward way.

Here are some details to help you get a sense of the problem's magnitude. The 30 studies together measured 50 variables in trying to describe the effects of various elicitation approaches. These variables included things such as the number of software requirements that were elicited, the time spent on various parts of the task, the technique's efficiency, the type of knowledge gained, and the correctness and importance of the requirements produced. The studies by and large all looked at different subsets of these response variables, and even when two studies focused on the same variable, they tended to use different measures. (The only exception was when the studies relied on measuring natural phenomena, such as time in minutes.) So perhaps a more natural question is, how does anyone make sense of this mess of data?

Our approach was to aggregate variables together in the same way that we just talked about aggregating techniques. For example, we can say something about whether a given type of technique tends to produce more requirements without worrying about whether the quantity of output is measured by

- the number of requirements produced,
- the number of customer activities identified for automation,
- the number of system attributes identified, or

- some more abstruse measure of information units in the output document.

Our methodology isn't perfect. When we talk about the time required for a technique, for example, we might be comparing a technique that measured only preparation time with one that measured the time spent doing the elicitation. Neither measure gives a complete picture, and the techniques in question might minimize the time spent in one activity but shift effort to other types of activities. Also, by comparing across so many different variables and measures, we lose the ability to combine the data statistically and hence lose our most rigorous approach for examining what the data say. However, to get some broad conclusions that can give us a sense of whether we see a recurring pattern in all this data, this approach is useful. We're essentially reducing a 43 × 50 matrix of results to a 7 × 8 one. Like any other map that represents a complex landscape, some simplifications are necessary to get a handle on the rough contours.

So, what have we learned?

Rather than talk about each of those 56 categories of conclusions individually, we present here some of the overall patterns we've seen. The names of studies in this section refer to the list in the Web appendix.

Regardless of their experience, developers should avoid interviewing a customer without preparing a few questions beforehand—in other words, interviews with some structure are preferable to completely unstructured ones. Two studies took on the question of structured versus unstructured interview effectiveness (S02 and S25); both pointed to more customer needs being identified with a structured approach. Two studies (S02 and S05) also found that the experience of the developers doing the interviewing didn't affect the number of requirements identified using this approach. So, even if you're just starting the requirements process and know little about the domain, a few general questions can make interviewing more effective.

Introspective techniques such as protocol analysis aren't as useful as other types of techniques. Although we found six introspective techniques, most of the data concerned protocol analysis. The studies in our set provided no evidence that protocol analysis was any more productive than techniques such as interviews, sorting, and laddering (S06, S08, S17, and S18). These other techniques provided more complete information in the same or less time and were usually simpler to apply.

To elicit specialized types of knowledge, you should complement interviews with contrived techniques and specialized models. The studies that addressed this topic (S06, S08, S17, and S18) found that laddering was as effective and efficient as unstructured interviews. So, overall, structured interviews were still more useful than laddering. However, the contrived techniques produce intermediate output—namely, some kind of model of the domain. For this reason, especially if hierarchies are an important part of the problem domain, laddering might help make elicitation interviews more effective. Although there really isn't sufficient evidence to say with certainty, we hypothesize that, for the same reason, other contrived techniques such as mock-ups, scenarios, and storyboarding can also be worthwhile for augmenting structured interviews.

The studies strongly supported the conclusion that sorting techniques are overall less effective than interviews (S03, S04, S06, S08, and S17). However, as with contrived techniques, these techniques might be useful when combined with structured interviews.

Although interviews were more effective than other techniques in most cases, they weren't always the most efficient. Card sorting tended to be less time-consuming than interviews (S03, S08, and S17), whereas laddering was generally equally time-consuming (S06 and S17) but could be faster (S08). The explanation might be that the noninterview techniques are more focused: the interaction between developer and customer is better defined, and the questions to ask are somewhat prespecified. Planning the interview with some minimal structure and keeping the interview focused might be ways of incorporating some of these benefits into the interview technique without giving up the effectiveness.

As always, the best results come from having more tools in the toolbox—and knowing which to use together for the job at hand.

In short, from aggregating across all these studies, we've found that interviews continue to be the primary technique for requirements elicitation for good reason. However, certain good practices (such as providing some structure in the interview) have been shown to improve the results. Although not most effective when used on their own, contrived techniques can augment interviews effectively when specific types of information are required. As always, the best results come from having more tools in the toolbox—and knowing which to use together for the job at hand.

References

1. E. Kamsties, K. Hormann, and M. Schlich, "Requirements Engineering in Small and Medium Enterprises: State-of-the-Practice, Problems, Solutions, and Technology Transfer," *Proc. Conf. European Industrial Requirements Eng.* (CEIRE 98), British Computer Soc., 1998, pp. 40–50.
2. C.J. Neill and P.A. Laplante, "Requirements Engineering: The State of the Practice," *IEEE Software*, vol. 20, no. 6, 2003, pp. 40–45.
3. E.F. Fern, "Focus Groups: A Review of Some Contradictory Evidence, Implications, and Suggestions for Future Research," *Advances in Consumer Research*, vol. 10, no. 1, 1983, pp. 121–126.
4. B. Fischhoff, "Eliciting Knowledge for Analytical Representation," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 19, no. 3, 1989, pp. 448–461.
5. S. Frankel, "NGT+MDS: An Adaptation of the Nominal Group Technique for Ill-Structured Problems," *J. Applied Behavioral Science*, vol. 23, no. 4, 1987, pp. 543–551.
6. N.A.M. Maiden and G. Rugg, "ACRE: Selecting Methods for Requirements Acquisition," *Software Eng. J.*, vol. 11, no. 3, 1996, pp. 183–192.

Oscar Dieste is a researcher at Madrid Technical University. Contact him at odieste@fi.upm.es.

Natalia Juristo is a professor at Madrid Technical University. Contact her at natalia@fi.upm.es.

Forrest Shull is a senior scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland, and director of its Measurement and Knowledge Management Division. Contact him at fshull@fc-md.umd.edu.