CHAPTER FOUR

Understanding Software Engineering Through Qualitative Methods

Andrew Ko

People trust numbers. They are the core of computation, the fundamentals of finance, and an essential part of human progress in the past century. And for the majority of modern societies, numbers are how we know things: we use them to study which drugs are safe, what policies work, and how our universe evolves. They are, perhaps, one of the most powerful and potent of human tools.

But like any tool, numbers aren't good for everything. There are some kinds of questions for which numbers answer little. For example, public health researchers in the 1980s wanted to explain epileptic patients' difficulties with taking their medication regularly. Researchers measured how many people failed to comply; they looked for statistical differences between those who complied and those who didn't; they even ran elaborate longitudinal controlled experiments to measure the consequences of noncompliance. But none of these approaches explained the lack of compliance; they just described it in rigorous but shallow ways.

Then, a groundbreaking study [Conrad 1985] took a different approach. Instead of trying to measure compliance quantitatively, the researchers performed 80 interviews of people with epilepsy, focusing on the situations in which their informants were expected to take their medications but did not. The researchers found that the lack of compliance was due not to irrational, erratic behavior, but to patients' deliberate choice. For example, some patients were aware of the potential of becoming chemically dependent on the drugs and carefully regulated their use of the drug to avoid tolerance. These qualitative findings, one of the first of their kind

in public health research, became a critical part of redefining how epilepsy medication is delivered to society.

What does all of this have to do with software engineering? Like public health, software engineering is full of *why* and *how* questions for which numbers and statistics are of little help. For example, if you've managed a software team, you've probably asked yourself a number of questions. Why won't my developers write unit tests? Why do users keep filling out this form incorrectly? Why are some of my developers 10 times as productive as the others? These questions can't be answered with numbers, but they can with the careful application of *qualitative methods*.

But using qualitative methods isn't as simple as asking people a few questions, nor is reading reports of qualitative studies as simple as reading the abstract. This chapter explains what qualitative methods are, how to interpret the results of qualitative studies, and how you might use them in your own work to improve software process and quality.

What Are Qualitative Methods?

Put simply, qualitative methods entail the *systematic gathering and interpretation of nonnumerical data* (including words, pictures, etc.). Like quantitative methods, qualitative methods can be used to gather data to confirm or reject beliefs (deductive reasoning). However, qualitative methods can also be used to support inductive reasoning: gather data to arrive at new explanations. Because qualitative methods gather nonnumerical data, they also lend themselves to being used in more natural settings.

To illustrate, let's focus on a specific scenario and discuss how qualitative methods might be used to understand it. Imagine you've just started managing a software team. You're leading a new project and have just adopted a new bug tracking system with some great features your team has been dying to have. Over the next few months, you see your team patching a lot of code and making great progress, but every time you check the bug list in the new system, there are only a few reports from the same few testers and they never seem to change. Then, one morning you walk by your best developer's desk and see the old bug tracker up on his screen. Your team has been secretly using the old system for weeks! After all of that training, the careful transition, and the costly licensing, *why aren't they using the new tracker*?

One obvious thing to do is simply ask. For example, you could call a meeting and just ask the team to explain why they're avoiding the new system. You'd get some explanation, but it may not be entirely trustworthy, especially if the developers find themselves opposing your decision to adopt the new tracker. Moreover, the fact that they're in a social context will also make the less talkative members of your team less likely to speak, biasing the explanations you get to your most vocal employees. A biased, warped explanation won't be helpful to anyone.

To take yourself out of the equation, perhaps you could ask a trustworthy friend to ask around during coffee breaks, conducting brief, informal interviews. That would give each person a

chance to state his opinion outside of a group context, perhaps freeing him to be more vocal and more honest. But you'll still be limited to the views of your friend's friends. Even worse, your friend might not be impartial; maybe he particularly dislikes the new tracker and unintentionally biases his report back to you.

The issue with the approaches I've just suggested is that they are secondhand or even thirdhand accounts. Ideally, you would be able to observe the moment of interest. For example, when people on the team decided to use the old tracker instead of the new one, what was going on in their heads? What were the other constraints on their time? Who were they collaborating with? What data were they entering? To get at these questions, you might sit next to developers for a day, watching what they do. This would allow you to directly observe the moments when they decide to use the old tracker instead of the new one.

Of course, directly observing people in these moments is often unfeasible. People don't like being watched, and often adapt their behavior to preserve their privacy or avoid embarrassment or anxiety. Moreover, since you'd be the sole observer, you're likely to bring biases to your observations.

It might also be possible to take people out of the picture altogether and just study the documents. Which bugs are getting reported in the old system, and which are reported in the new system? This might uncover differences in the type of reports that the team hasn't been filing in the new system. This might give you a hunch about the reasons for using the old tracker, but it still wouldn't tell you the actual reasons inside the minds of your team.

All of these approaches are qualitative methods and all of them have limitations. The solution to getting around these limitations is to *embrace* them: no one approach will reveal the whole unbiased truth. Instead, good qualitative research combines multiple methods, allowing one to triangulate evidence from multiple sources. For example, suppose you interviewed individual employees about the tracker but also studied which bugs were being filed in the new tracker. Each approach will give you a distinct story, more or less consistent with the stories from other approaches. By comparing and contrasting these stories, one can uncover the ground truth behind a question.

Aside from gaining understanding, qualitative methods are also good for gaining *empathy*. More often than not, the cause of communication breakdowns, broken processes, and user frustration is people's inability to see the world from another person's perspective. And this is precisely what qualitative methods are designed to correct. This perspective-taking is often precisely the goal of qualitative methods. For example, suppose you manage a team and have noticed that every Friday the build breaks, people spend the whole day trying to fix it, and everyone goes home frustrated. Using a qualitative approach to diagnose the cause of these broken builds might lead you to discover that Thursday nights are a big night for check-ins because you've decided Friday is meeting day. Knowledge like this can help you see the world from the developers' perspective and find solutions that make everyone happy.

Reading Qualitative Research

Having described what qualitative methods are, we now turn to a discussion of how to read qualitative studies like the ones that appear throughout this book. For example, what does a particular study teach? When can you trust a study's results? When can you generalize a study's results to the larger world? To discuss these issues, let's consider *The Errors of TeX*, published in 1989 by the Turing Award winner Donald Knuth [Knuth 1989].

In this classic article, Knuth analyzes more than 850 errors he logged while writing the TeX software. The study, as Knuth described it, was "to present a list of all the errors that were corrected in TeX while it was being developed, and to attempt to analyse those errors." Knuth describes the rationale for his approach as overcoming the limitations of quantitative methods:

> The concept of scale cannot easily be communicated by means of numerical data alone; I believe that a detailed list gives important insights that cannot be gained from statistical summaries.

What did Knuth discover in this study? He presents 15 categories of errors, gleaned from a much larger catalog, and then describes them with examples from his log. For example, Knuth describes the "blunder or blotch" category, which included program statements that were syntactically correct but semantically wrong. The root cause of these errors was variable names that were closely related conceptually but led to very different program semantics (e.g., reversing variables named *before* and *after*, or *next_line* and *new_line*). Knuth goes on to describe the other error categories, the history behind the TeX software project, his personal experiences in writing the software, and how he recorded errors in his log.

At the end of the article, he concludes:

> What have I really learned then? I think I have learned, primarily, to have a better sense of balance and proportion. I now understand the complexities of a medium-size software system, and the ways in which it can be expected to evolve. I now understand that there are so many kinds of errors, we cannot stamp them out by systematically eliminating everything that might be 'considered harmful.' I now understand enough about my propensity to err that I can accept it as an act of life; I can now be convinced more easily of my fallacy when I have made a mistake.

Now let us step back and reflect on the merits of this work: what did we learn, as readers? The first time I read this article in the mid-1990s, I learned a great deal: I had never written a medium-sized software system, and the rich details, both contextual and historical, helped me understand the experience of undertaking such a large system by one's self. I recognized many of the error categories that Knuth described in my own programming, but also learned to spot new ones, which helped me become better at thinking of possible explanations for why my code wasn't working. It also taught me, as a researcher, that the human factors behind software development—how we think, how our memory works, how we plan and reason—are powerful forces behind software quality. This was one of just a few articles that compelled me to a career in understanding these human factors and exploiting them to improve software quality through better languages, tools, and processes.

But few of these lessons came immediately after reading. I only started to notice Knuth's categories in my own work over a period of months, and the article was just one of many articles that inspired my interests in research. And this is a key point in how to read reports on qualitative research critically: not only do the implications of their results take time to set in, but you have to be open to reflecting on them. If you dismiss an article entirely because of some flaw you notice or a conclusion you disagree with, you'll miss out on all of the other insights you might gain through careful, sustained reflection on the study's results.

Of course, that's not to say you should trust Knuth's results in their entirety. But rather than just reacting to studies emotionally, it's important to read them in a more systematic way. I usually focus on three things about a study: its *inputs*, its *execution*, and its *outputs*. (Sounds like software testing, doesn't it?) Let's discuss these in the context of Knuth's study.

First, do you trust the inputs into Knuth's study? For example, do you think TeX is a representative program? Do you think Knuth is a representative programmer? Do you trust Knuth himself? All of these factors might affect whether you think Knuth's 15 categories are comprehensive and representative, and whether they still occur in practice, decades after his report. If you think that Knuth isn't a representative programmer, how might the results have changed if someone else did this? For example, let's imagine that Knuth, like many academics, was an absent-minded professor. Perhaps that would explain why so many of the categories have to do with forgetting or lack of foresight (such as the categories *a forgotten function*, *a mismatch between modules*, *a surprising scenario*, etc.). Maybe a more disciplined individual, or one working in a context where code was the sole focus, would not have had these issues. None of these potential confounding factors are damning to the study's results, but they ought to be considered carefully before generalizing from them.

Do you trust Knuth's *execution* of his study? In other words, did Knuth follow the method that he described, and when he did not, how might these deviations have affected the results? Knuth used a *diary study* methodology, which is often used today to understand people's experiences over long periods of time without the direct observation of a researcher. One key to a good diary study is that you don't tell the participants of the study what you expect to find, lest you bias what they write and how they write it. But Knuth was both the experimenter and the participant in his study. What kinds of expectations did he have about the results? Did he already have categories in mind before starting the log? Did he categorize the errors throughout the development of TeX, or retrospectively after TeX was done? He doesn't describe any of these details in his report, but the answers to these questions could significantly change how we interpret the results.

Diary studies also have inherent limitations. For example, they can invoke a Heisenberg-style problem, where the process of observing may compel the diary writer to reflect on the work being captured to such a degree that the nature of the work itself changes. In Knuth's study, this might have meant that by logging errors, Knuth was reflecting so much about the causes of errors that he subconsciously averted whole classes of errors, and thus never observed them. Diary studies can also be difficult for participants to work on consistently over time. For

example, there was a period where Knuth halted his study temporarily, noting, "I did not keep any record of errors removed during the hectic period when TeX82 was being debugged...." What kinds of errors would Knuth have found had he logged during this period? Would they be different from those he found in less stressful, hectic periods?

Finally, do you trust the *outputs* of the study, its implications? It is standard practice in academic writing to separate the discussion of results and implications, to enable readers to decide whether they would draw the same conclusions from the evidence that the authors did. But Knuth combines these two throughout his article, providing both rich descriptions of the faults in TeX and the implications of his observations. For example, after a series of fascinating stories about errors in his *Surprises* category (which Knuth describes as global misunderstandings), he reflects:

> This experience suggests that all software systems be subjected to the meanest, nastiest torture tests imaginable; otherwise they will almost certainly continue to exhibit bugs for years after they have begun to produce satisfactory results in large applications.

When results and implications appear side-by-side, it can be easy to forget that they are two separate things, to be evaluated independently. I trust Knuth's memory of the stories that inspired the implication quoted here because he explained his process for recording these stories. However, I think Knuth over-interpreted his stories in forming his recommendation. Would Knuth have finished TeX if he spent so much time on torture tests? I trust his diary, but I'm skeptical about his resulting advice.

Of course, it's important to reiterate that *every* qualitative study has limitations, but most studies have valuable insights. To be an objective reader of qualitative research, one has to accept this fact and meticulously identify the two. A good report will do this for you, as do the chapters in this book.

Using Qualitative Methods in Practice

While qualitative methods are usually applied in research settings, they can be incredibly useful in practice. In fact, they are useful in any setting where you don't know the entire universe of possible answers to a question. And in software engineering, when is that *not* the case? Software testers might use qualitative methods to answer questions about inefficiencies in the human aspects of a testing procedure; project managers can use qualitative methods to understand how the social dimensions of their team might be impacting productivity. Designers, developers, and requirements engineers can use qualitative methods to get a deeper understanding of the users they serve, ensuring a closer fit between user needs and the feature list. Any time you need to analyze the dynamics between *who*, *what*, *when*, *where*, *how* and *why*, qualitative methods can help.

Using qualitative methods is a lot like being a good detective or journalist: the point is to uncover truth, and tell a story people can trust—but also realize that there are usually many

truths and many perspectives. How you go about uncovering these perspectives depends a lot on the situation. In particular, you need an instinct for the social context of what you want to understand, so you know whom you can trust, what their biases are, and how their motives might affect your sleuthing. Only with this knowledge can you decide what combination of direct observation, shadowing, interviews, document analysis, diary studies, or other approaches you might take.

To illustrate, let's go back to our bug tracker example. One of the most important factors in choosing which methods to use is *you*, and so there are a number of things to consider:

- Do your employees like you?
- Do they respect you?
- Do they trust you?

If the answer to any of these questions is no, you're probably not the one to do the sleuthing. Instead, you might need to find a more impartial party. For example, a perfect role for sleuthing is an *ombudsperson*. Her job is to be a neutral party, a person who can see multiple perspectives to support effective communication and problem solving. If your organization has an ombudsperson, she would be a great candidate for a class on qualitative methods and could play an instrumental role in improving your workplace.

If your employees do like you, the next most important factor is your employees. Are they good communicators? Are they honest? Do they hide things? Are there rival factions on your team? Is there a culture of process improvement, or is the environment rigid and conservative? All of these social factors are going to determine the viability of applying any particular qualitative method. For example, direct observation is out of the question if your employees like to hide things, because they'll know they're being observed. Document analysis might work in this case, but what will your team think about their privacy? Interviews work quite well when the *interviewer* can establish rapport, but otherwise they're a garbage-in, garbage-out process. The goal of a good qualitative approach to answering a question is to find ways of probing that minimize bias and maximize objectivity.

Regardless of whether you or someone else is doing the sleuthing, another important consideration is how you explain to your team what you're sleuthing about. In all of the cases I can imagine, keeping the sleuthing a secret is a terrible idea. You're probably asking a question because you want to solve a problem, and you're managing a team of problem solvers: get them to help! It's important to point out to them, however, that your agenda is to understand, not to dictate. It's also important to say that there's probably not a single explanation and everybody's perspective will differ at least a little. Communicating these points creates an expectation that you'll be seeking your informants' perspectives and empathizing with them, which makes them more likely to reflect honestly.

Finally, qualitative methods can feel loose and arbitrary at times. How can you really trust the results of a process without structure? The trick is to realize that bias is all around you and

embrace it. You as the researcher are biased, your informants are biased, and the different methods you use to understand a context are biased toward revealing certain phenomena. The more you can explicitly identify the bias around you and understand how it affects your interpretations of what you see, the more objective your findings.

Generalizing from Qualitative Results

Whether you find qualitative results in a research paper or you get them yourself, one issue that always comes up is how much you can generalize from it.

For example, qualitative methods are can identify *common* behavior, but they cannot identify *average* behavior. Averages and other aggregate statistics require some ability to count, and that's not what qualitative methods are about. Moreover, because the gathering of nonnumerical data can be more labor-intensive, qualitative methods often end up with smaller sample sizes, making it difficult to generalize to the broader population of situations under study.

Nevertheless, qualitative studies can and do generalize. For example, they can demonstrate that the cause and effect relationships present in one context are similar to those in another context. Suppose you were part of a user interface team for a web application, trying to understand why code reviews were always taking so long. Although your findings might not generalize to the team managing the database backend, they might generalize to other frontend teams for web applications in a similar domain. Knowing when the study generalizes is no less challenging than for quantitative studies; in both cases, it's a subjective matter of knowing which assumptions still apply in new contexts.

Qualitative Methods Are Systematic

Qualitative methods are increasingly important in research on software engineering, and they can be quite important in software engineering practice as well. But there's a significant difference between simply understanding problems and understanding problems *systematically*. Therefore, the next time you read the results of a qualitative study or set out to do your own, make sure you or the article are following a process like this:

Formulate the question
: What knowledge is desired and why? How will the knowledge be used?

Consider viable sources of objective evidence
: What data can be gathered *objectively*, with as little bias as possible? Can you gather from multiple sources to account for biases in individual sources?

Interpret patterns in the evidence

Are the sources of evidence consistent with one another or do they conflict? What new questions does the evidence data raise? Do they lead to a hypothesis that can be tested with further data?

Although this cycle of question formulation, evidence gathering, and interpretation can be a long process, each iteration can lead to more confidence in your understanding, which means better decisions and better outcomes. Moreover, when you combine qualitative methods with quantitative tools, you'll have a powerful toolbox with which to understand and improve software engineering practice.

References

[Conrad 1985] Conrad, P. 1985. The meaning of medications: another look at compliance. *Social Science and Medicine* 20:29–37.

[Knuth 1989] Knuth, D. 1989. The Errors of TeX. *Software Practice and Experience* 19(7):607–685.