Qualitative Methods in Empirical Studies of Software Engineering

Carolyn B. Seaman, *Member*, *IEEE*

Abstract—While empirical studies in software engineering are beginning to gain recognition in the research community, this subarea is also entering a new level of maturity by beginning to address the human aspects of software development. This added focus has added a new layer of complexity to an already challenging area of research. Along with new research questions, new research methods are needed to study nontechnical aspects of software engineering. In many other disciplines, qualitative research methods have been developed and are commonly used to handle the complexity of issues involving human behavior. This paper presents several qualitative methods for data collection and analysis and describes them in terms of how they might be incorporated into empirical studies of software engineering, in particular how they might be combined with quantitative methods. To illustrate this use of qualitative methods, examples from real software engineering studies are used throughout.

Index Terms—Qualitative methods, data collection, data analysis, experimental design, empirical software engineering, participant observation, interviewing.

1 Introduction

The study of software engineering has always been complex and difficult. The complexity arises from technical issues, from the awkward intersection of machine and human capabilities, and from the central role of human behavior in software development. The first two aspects have provided more than enough complex and interesting problems to keep empirical software engineering researchers engaged up until now. But it is the last factor, human behavior, that software engineering empiricists are only recently beginning to address in a serious way.

Empirical studies have been conducted in software engineering for several decades, but have only relatively recently achieved significant recognition in the broader software engineering research community (as evidenced by this special issue). But this subarea has also reached a discernibly new level of maturity that is evidenced by the new types of questions and methods seen in more recent studies. In particular, software engineering empiricists are beginning to address the human role in software development. One indication of this broadening of focus is the nature of recent work in traditionally empirical software engineering research groups. For example, recent studies at the Software Engineering Laboratory[1] have concentrated on human aspects through observation of communication among developers [17] and the elicitation of the processes used to build systems based on COTS[2] components [15].

Part of the reason for this new interest among researchers actually comes from practitioners, many of whom have seen the advances gained by adapting research results in technical areas. But many in the industry recognize that software development also presents a number of unique management and organizational issues, or "people problems," that need to be addressed and solved in order for the field to progress. Calls to take "people problems" seriously were first made decades ago [4], [6], and continue to appear regularly in the literature [1], [5], [13]. Finally, they are starting to be heeded by researchers who are starting to study nontechnical issues and the intersection between the technical and nontechnical in software engineering.

Qualitative data are data represented as words and pictures, not numbers [8]. Qualitative research methods were designed, mostly by educational researchers and other social scientists [19], to study the complexities of human behavior (e.g., motivation, communication, understanding). It could be argued that human behavior is one of the few phenomena that is complex enough to require qualitative methods to study it. Anything else can be adequately described and explained through statistics and other quantitative methods. In software engineering, the blend of technical and human behavioral aspects lends itself to combining qualitative and quantitative methods, in order to take advantage of the strengths of both.

The focus of this paper is on showing how qualitative methods can be adapted and incorporated into the designs of empirical studies in software engineering. The principal advantage of using qualitative methods is that they force the researcher to delve into the complexity of the problem rather than abstract it away. Thus, the results are richer and

1. The Software Engineering Laboratory (SEL) is sponsored jointly by NASA/Goddard Space Flight Center, Computer Sciences Corporation, and the Empirical Software Engineering Group at the University of Maryland. The SEL has been conducting various types of empirical studies of diverse software engineering issues for more than two decades.

2. Commercial-Off-The-Shelf.

• *C.B. Seaman is with the Department of Information Systems, University of Maryland Baltimore County, Baltimore, MD 21250.*
E-mail: cseaman@umbc.edu.

Manuscript received 30 June 1998.
Recommended for acceptance by D. Ross Jeffery.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 109541.

more informative. There are drawbacks, however. Qualitative analysis is generally more labor-intensive and exhausting than quantitative analysis. Qualitative results often are considered "softer" or "fuzzier" than quantitative results, especially in technical communities like ours. They are more difficult to summarize or simplify. But then, so are the problems we study in software engineering.

The methods described here are described in terms of how they could be used in a study that mixes qualitative and quantitative methods. Purely qualitative studies, which are rare to say the least in software engineering, would probably employ these methods slightly differently, or at least more intensively. There are also many other, more sophisticated, qualitative analysis methods that are employed in purely qualitative studies. See [11], [12], [14], [19] for descriptions of other qualitative methods.

The presentation of this paper divides qualitative methods into those for collecting data and those for analyzing data. Examples of several methods are given for each, and the methods can be combined with each other, as well as with quantitative methods. Later, several example method combinations are discussed. Throughout the article, examples will be drawn from one particular study, described in detail in [17], of communication among developers during code inspection meetings.

2 Data Collection Methods

Two data collection methods, participant observation and interviewing, are presented in this section. These are useful ways of collecting firsthand information about software development efforts. Historical qualitative information can also be gained by examining documentation. Techniques for analyzing archival documents are discussed in [19]. Also in this section is a discussion of one type of "coding," which is a technique for preparing qualitative data to be analyzed quantitatively.

2.1 Participant Observation

Participant observation, as defined in [19], refers to "research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected." The idea is to capture firsthand behaviors and interactions that might not be noticed otherwise. Although the name is misleading, participant observation does not necessarily imply that the observer is engaged in the activity being observed (e.g., [2]), only that the observer is visibly present and is collecting data with the knowledge of those being observed.

Although a great deal of information can be gathered through observation, the parts of the software development process that can actually be observed are limited. Much of software development work takes place inside a person's head. Such activity is difficult to observe, although there are some techniques for doing so. *Think aloud protocols* [9] require the subject to verbalize his or her thought process so that the observer can understand the process going on. Such protocols are limited by the comfort level of the subject and their ability to articulate their thoughts. It might be possible, also, to capture some of the thought process of individual developers by logging their keystrokes and mouse movements as they work on a computer [18]. These techniques are often used in usability studies, where the subjects are software users, but they have not been widely employed in studies of software developers (an exception is the work of von Mayrhauser and Vans [20]).

Software developers reveal their thought processes most naturally when communicating with other software developers, so this communication offers the best opportunity for a researcher to observe the development process. One method is for the researcher to observe a software developer continuously, thus recording every communication that takes place with colleagues, either planned or unplanned. A good example of a study based on this type of observation is [16]. A less time-consuming approach is to observe meetings of various types. These could include inspection meetings, design meetings, status meetings, etc. By observing meetings, a researcher can gather data on the types of topics discussed, the terminology used, the technical information that was exchanged, and the dynamics of how different project members speak to each other.

There are a number of issues that a participant observer must be aware of. Many of these are presented here, based in part on the literature (in particular [19]) and partly on the particular experience of this researcher with studies of software engineering.

The participant observer must take measures to ensure that those being observed are not constantly thinking about being observed. This is to help ensure that the observed behavior is "normal," i.e., what usually happens in the environment being observed, and is not affected by the presence of the observer. For example, a researcher observing meetings should be as unobtrusive as possible (like a "fly on the wall"). Ideally, all those present should know beforehand that the observer will be at that meeting and why. This advance notice avoids having to do a lot of explaining during the meeting, which will only remind the subjects that they are being observed. The observer, although visible, should not be disruptive in any way. All of the observer's materials (pen, watch, paper, recording devices, etc.) should be ready and at hand before the meeting starts so that the observer doesn't have to hunt around for them during the meeting. If the meeting takes place at a table, the observer should probably not sit at the table, but back from it a little so that he or she can see everything that is going on at the table, but is not directly in everyone's line of sight. Again, these are all techniques to help ensure that the subjects are concentrating on the job at hand, not on being observed. The observer should always look for signs that their presence makes any of the participants nervous or self-conscious, which again may affect their behavior. Any such signs should be recorded in the notes that the observer takes, and will be considered in the analysis later.

The observer's notes should not be visible to any of the meeting participants. In fact, the notes should be kept confidential throughout the study. This gives the researcher complete freedom to write down any impressions, opinions, or thoughts without the fear that they may be read by someone who will be offended by them.

The data gathered during an observation is ultimately recorded in the form of *field notes*. These notes are begun during the actual observation, during which the observer writes what is necessary to fill in the details later. Then, as soon after the observation as possible, the notes are augmented with as many details as the observer can remember. The information contained in the field notes should include the place, time, and participants in the meeting, the discussions that took place, any other events that took place either as part of the meeting or that impacted the meeting, and the tone and mood of the meeting. The notes should also contain "observer's comments," marked "OC" in the text of the notes, which record the observer's impressions of some aspect of the meeting, which may not correspond directly to anything that was actually said or that occurred during the meeting. The level of detail in the notes depends on the objectives of the researcher. The most detailed are verbatim transcripts of everything said during the meeting, plus detailed descriptions of the setting and participants. Writing such detailed notes is extremely time-consuming. Often what are needed are summaries of the discussions and/or some details that are specific to the aims of the study. The more exploratory and open-ended the study, the more detailed the field notes should be, simply because in such a study anything could turn out to be relevant. In any study, the observer should begin with very detailed notes at least for the first few observations, until it is absolutely clear what the objectives of the study are and exactly what information is relevant.

In many studies, there are very specific pieces of information that are expected to be collected during an observation. This is often true in studies that combine qualitative and quantitative methods, in which qualitative information from an observation will later be coded into quantitative variables, e.g., the length of a meeting in minutes, the number of people present, etc. When this is the case, forms will be designed ahead of time that the observer will fill in during the course of the observation. This will ensure that specific details will be recorded. These forms are used in addition to, not instead of, field notes.

An example of a study based largely on observation data is [17], a study of code inspection meetings (hereafter referred to as the Inspection Study). Most of the data for this study was collected during direct observation of 23 inspections of C++ classes. The objective of the study was to investigate the relationship between the amount of effort developers spend in technical communication (e.g., the amount of time spent discussing various issues in inspection meetings) and the organizational relationships between them (e.g., how much a group of inspection participants have worked together in the past). Information about organizational relationships was collected during interviews with inspection participants, described in Section 2.2. Information about communication effort was collected during the observations of code inspections. This study serves as a good example of employing a variety of qualitative methods, along with quantitative methods, to investigate an issue in software engineering. The findings of this study were deeper and more illustrative than would have been gained using a more restricted set of research methods. Also, this author learned a great many lessons (through both success and failure) while conducting this study. For all these reasons, this study will serve as an example throughout this article to illustrate the methods presented. An additional example study will be presented in the next section.

Fig. 1 shows a form that was filled out by the observer for each observed meeting in the Inspection Study. The administrative information (classes inspected, date, time, names of participants) were all provided in the announcement of each inspection. The responsibilities of each inspector (which products each was responsible for inspecting) were either stated in the inspection announcement, became obvious during the meeting, or were related during interviews. At some point during each inspection meeting, each inspector reported his or her preparation time to the moderator, and the observer also recorded it. Whether or not each was present at the meeting was also recorded on the data form. The amount and complexity of the code inspected was addressed during interviews later.

Another form filled out during observations was a time log, an example of which is shown in Fig. 2. At the top of each page of the log is recorded basic identifying information. For each discussion that took place during the meeting, the observer recorded the time (to the closest minute) it started, the initials of the participants in that discussion, a code corresponding to the type of discussion, and some notes indicating the topic of discussion, the tone of the discussion, and any other relevant information. The arrows in some of the lists of participants' initials indicate that a comment or question was made by one participant, specifically targeted to another participant. In the margins of the time log, the observer also recorded other relevant information about the participants, the setting of the meeting, and other activities taking place. The number of minutes spent in each discussion category was calculated from the time logs after the meeting.

Extensive field notes were also written immediately after each meeting observed in the Inspection Study. These notes contained broader descriptions of observations noted on the inspection data forms. Below is a sanitized excerpt from these field notes:

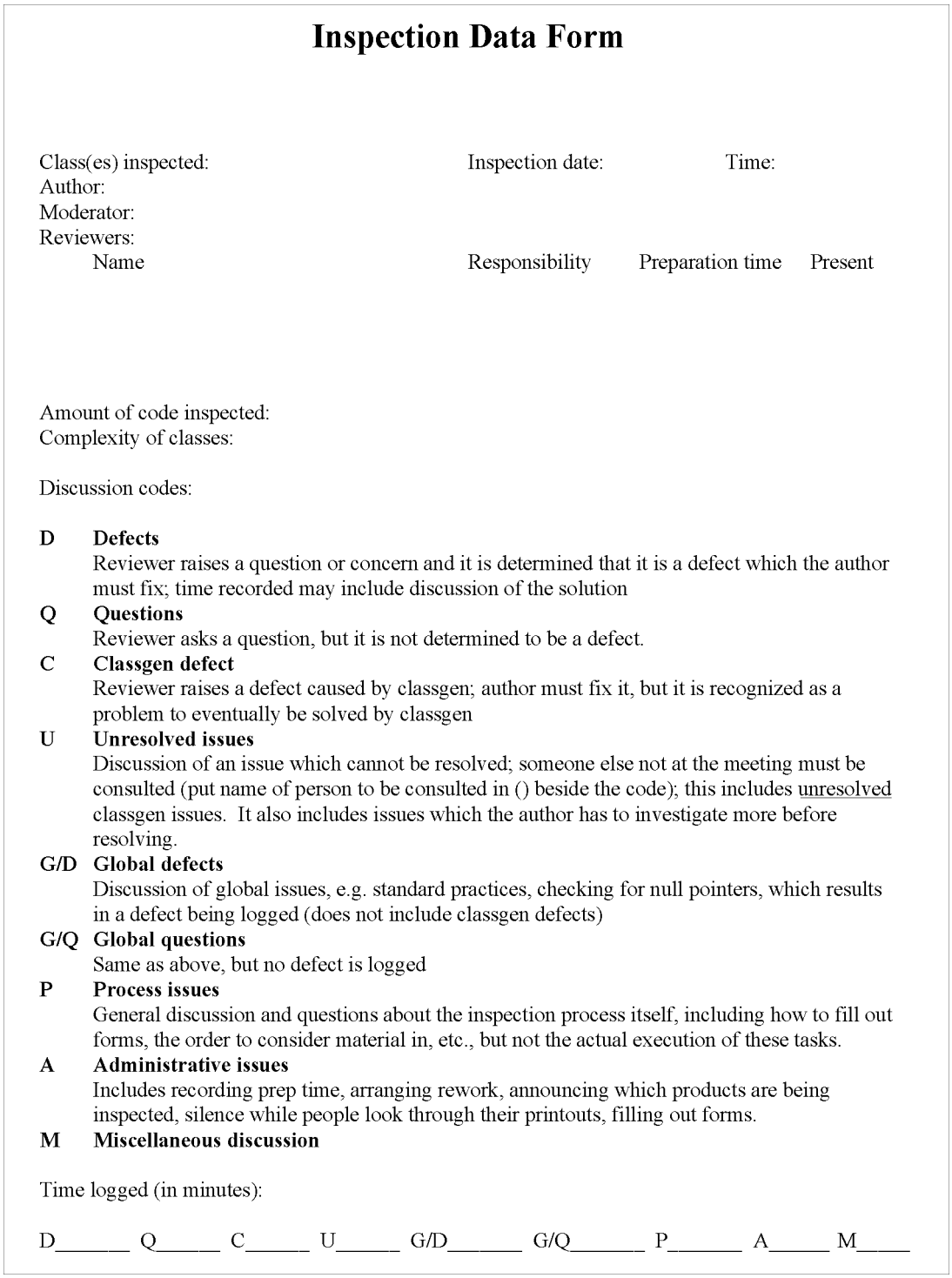
[Inspector1] raised a bunch of defects all together, all concerning checking for certain error conditions (unset dependencies, negative time, and null pointers).

[Inspector2] raised a defect which was a typo in a comment. She seemed slightly sheepish about raising it, but she did nevertheless.

OC: [Inspector2] seemed more harsh on [Author] than I had ever seen her on any of the [subcontractor] authors. My impression of her is that she would never raise a typo as a defect with anyone else. Does she have something against [government agency] folks?

[Inspector2] raised a defect concerning the wrong name of a constant.

[Inspector3] raised a defect having to do with the previous single dependency issue. In particular, dereferencing would have to be done differently, although there were several ways to fix it. [Inspector3] recommended using the dot instead of the arrow.



Inspection Data Form

Class(es) inspected: Inspection date: Time:
Author:
Moderator:
Reviewers:
Name Responsibility Preparation time Present

Amount of code inspected:
Complexity of classes:

Discussion codes:

D **Defects**
Reviewer raises a question or concern and it is determined that it is a defect which the author must fix; time recorded may include discussion of the solution

Q **Questions**
Reviewer asks a question, but it is not determined to be a defect.

C **Classgen defect**
Reviewer raises a defect caused by classgen; author must fix it, but it is recognized as a problem to eventually be solved by classgen

U **Unresolved issues**
Discussion of an issue which cannot be resolved; someone else not at the meeting must be consulted (put name of person to be consulted in () beside the code); this includes unresolved classgen issues. It also includes issues which the author has to investigate more before resolving.

G/D **Global defects**
Discussion of global issues, e.g. standard practices, checking for null pointers, which results in a defect being logged (does not include classgen defects)

G/Q **Global questions**
Same as above, but no defect is logged

P **Process issues**
General discussion and questions about the inspection process itself, including how to fill out forms, the order to consider material in, etc., but not the actual execution of these tasks.

A **Administrative issues**
Includes recording prep time, arranging rework, announcing which products are being inspected, silence while people look through their printouts, filling out forms.

M **Miscellaneous discussion**

Time logged (in minutes):

D_____ Q_____ C_____ U_____ G/D_____ G/Q_____ P_____ A_____ M_____

Fig. 1. Form used to collect data during observation of inspection meetings.

In order to evaluate the validity and consistency of data collected during participant observations, *rater agreement exercises* [11] are often conducted. The basic idea is to ensure not only that the data being recorded are accurate, but also that the observer is not recording data in a form that is understandable only to him or her. During three of the inspection meetings observed in the Inspection Study (about 15 percent), a second observer was present to record data. The same second observer was used all three times. All three were among the first half of meetings observed, i.e., they occurred fairly early in the study. This was intentional, in order to get the greatest advantage from improvements made to data collection procedures as a result of the exercise.

Before the observations in which she participated, the second observer was instructed by the principal observer in the forms used for data collection, the codes used to categorize discussions, the procedure used to time discussions, and some background on the development project and developers.

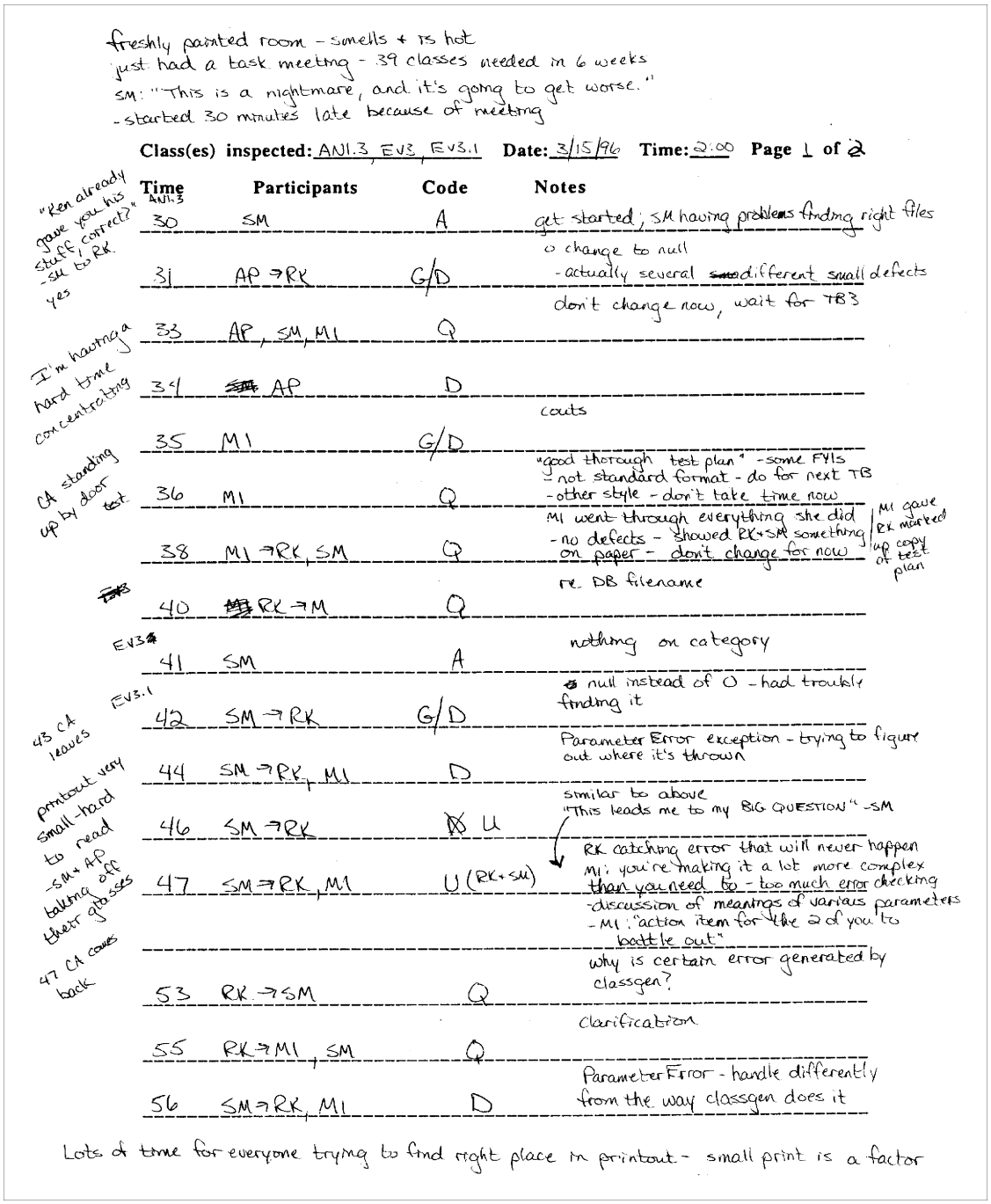


Fig. 2. Time log used to document discussions during inspection meetings.

A total of 42 discussions were recorded during the three doubly-observed meetings. Out of those, both observers agreed on the coding for 26, or 62 percent. Although, to our knowledge, there is no standard acceptable threshold for this agreement percentage, we had hoped to obtain a higher value. However, the two observers were later able to come to an agreement on coding for all discussions on which they initially disagreed. The observers generally agreed on the length of each discussion.

Many of the coding discrepancies were due to the second observer's lack of familiarity with the project and the developers. Others arose from the second observer's lack of experience with the instrument (the form and coding categories), and the subjectivity of the categories. The coding scheme was actually modified slightly due to the problems the second observer had. It should be noted that some of the discrepancies over coding (three out of 26 discrepancies) were eventually resolved in the second observer's favor. That is, the principal observer had made an error. Another troubling result of this exercise was the number of discussions (five) that one observer had completely missed, but had been recorded by the other. Both the principal and second observers missed discussions. This would imply that a single observer will usually miss some interaction.

The results of a rater agreement exercise, ideally, should confirm that the data collection techniques being used are robust. However, as in the Inspection Study, the exercise often reveals the limitations of the study. This is valuable, however, as many of the limitations revealed can be overcome if they are discovered early enough, and even if they are not surmountable, they can be reported along with the results and can inform the design of future studies. For example, in the Inspection Study, the results of the rater agreement exercise indicated that the data collected during observations would have been more accurate if more observers had been used for all observations, or if the meetings had been recorded. These procedural changes would have either required prohibitive amounts of effort, or stretched the goodwill of the study's subjects beyond its limits. However, these should be taken into consideration in the design of future studies.

Recording of observations, either with audio or video, is another issue to be considered when planning a study involving participant observation. The main advantage of electronically recording observations is in ensuring accuracy of the data. Usually, the field notes are written after the observation while listening to or watching the recording. In this way, the notes are much less likely to introduce inaccuracies due to the observer's faulty memory or even bias. In the Inspection Study, it was decided not to audio- or videotape observed meetings for the reasons mentioned in the previous paragraph. However, it is done in many studies [19].

2.2 Interviewing

Another commonly used technique for collecting qualitative data is the interview. Interviews are conducted with a variety of objectives. Often they are used to collect historical data from the memories of interviewees. In other studies they are used to collect opinions or impressions about something. In others, interviews are conducted to help identify the terminology used in a particular setting. They are sometimes used in combination with observations. In this case, they serve to clarify things that happened or were said during an observation, to elicit impressions of the meeting or other event that was observed, or to collect information on relevant events that were not observed.

Interviews come in several types. In [12], a *structured interview* is described as one in which "the questions are in the hands of the interviewer and the response rests with the interviewee," as opposed to an *unstructured interview* in which the interviewee is the source of both questions and answers. In an unstructured interview, the object is to elicit as much information as possible on a broadly defined topic. The interviewer does not know the form of this information ahead of time, so the questions asked must be as open-ended as possible. In the extreme, the interviewer doesn't even ask questions, but just mentions the topic to be discussed and allows the interviewee to expound. In a structured interview, on the other hand, the interviewer has very specific objectives for the type of information sought in the interview, so the questions can be fairly specific. The extreme of a structured interview is one in which no qualitative information is gained at all, i.e., all responses can be quantified (e.g., yes/no, high/medium/low, etc.). If the study is qualitative, however, the interview must be flexible enough to allow unforeseen types of information to be recorded. A purely unstructured interview is often too costly to be used extensively. Therefore, many studies employ *semistructured interviews*. These interviews include a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information.

Again, as in the previous section on participant observation, the advice given here about interviewing is based in part on the literature (in particular [19]) and partly on the experience and reflection of this author.

The interviewer should begin each interview with a short explanation of the research being conducted. Just how much information the interviewer should give about the study should be carefully considered. Interviewees may be less likely to fully participate if they do not understand the goals of the study or agree that they are worthy. However, if interviewees are told too much about it, they may filter their responses, leaving out information that they think the interviewer is not interested in. Another judgment that the interviewer must often make is when to cut off the interviewee when the conversation has wandered too far. On one hand, interview time is usually valuable and shouldn't be wasted. However, in a qualitative study, all data is potentially useful and the usefulness of a particular piece of data often is not known until long after it is collected. Of course, interviewees should never be cut off abruptly or rudely. Steering them back to the subject at hand must be done gently. In general, it is better to err on the side of letting the interviewee ramble. Often the ramblings make more sense in hindsight. The opposite problem, of course, is that of an interviewee who says the barest minimum. One strategy is to ask questions that cannot possibly be answered with a "yes" or a "no." Another is to feign ignorance, i.e., to ask for details that are already well known to the interviewer. This may get the interviewee talking, as well as help dispel any perception they might have of the interviewer as an "expert." It is also important to make it clear that there are no "right" answers. Software developers sometimes mistakenly believe that anyone coming to interview them (or observe them) is really there to evaluate them.

Like observational data, interview data are ultimately recorded in field notes, which are governed by the same guidelines as described in the previous section. Also as described earlier, forms can be used and filled out by the interviewer in order to facilitate the gathering of specific pieces of information. Another tool which can be useful during an interview is an *interview guide* [19]. An interview guide is not as formal as a data form, but it helps the interviewer to organize the interview. It usually consists of a list of open-ended questions, possibly with some notes about the direction in which to steer the interview under different circumstances. In a structured interview, the questions are fairly straightforward, and they might be arranged in an "if-then" structure that leads the interviewer along one of several paths depending on the answers to previous questions. In an unstructured interview, there might not be an interview guide, or it may simply be a short

list of topics to be touched on. Interview guides are purely for the use of the interviewer; they are never shown to the interviewee.

The interviewer may make some notes on the guide to help him or her remember how to steer the interview, but the guide should not be used for taking notes of the interview. In general, it is difficult for an interviewer to take notes and conduct the interview at the same time, unless the interviewer is very skilled. It is useful, if the interviewee consents, to audiotape the interview. The tape can then be used to aid the writing of the field notes later. Recording has the added advantage that the interviewer can hear him/herself on the tape and assess his or her interviewing skills. This is particularly useful in discovering one's own annoying conversational habits (e.g., interrupting, overuse of "um" etc.). Another way to facilitate the taking of notes is to use a scribe. A scribe is present at the interview only to take notes and does not normally participate in any other way. Using a scribe takes the note-writing responsibilities from the interviewer completely, which can be an advantage for the researcher. However, verbatim notes are not possible this way, and the scribe does not always share the interviewer's ideas about what is important to record. The use of a scribe is also often prohibitively expensive or intimidating to the interviewee.

Another example study that will be used in this article is [15], a study of COTS integration (hereafter referred to as the COTS Study). The objective of the study was to document the process that NASA software project teams were following to produce software systems largely constructed from COTS components. This type of system development, or "integration," was fairly new in the NASA group studied. Consequently, there was no documented process for it and it was suspected that a number of different processes were being followed. The COTS Study team was tasked with building a process model general enough to apply to all of the different ways that COTS integration was being done. The model would then be used as a baseline to design process measures, to plan improvements to the process, and to make recommendations for process support. Interviews with developers on projects that involved a large amount of COTS integration provided the bulk of the data used to build the process model. Scribes, as described above, were used to record these interviews. Many interviewees were interviewed multiple times, at increasing levels of detail. These interviews were semi-structured because each interview started with a specific set of questions, the answers to which were the objective of the interview. However, many of these questions were open-ended and were intended for (and successful in) soliciting other information not foreseen by the interviewer. For example, one question on the COTS Study interview guide was:

What are the disadvantages of Package-Based Development (i.e., COTS integration) in comparison with traditional development?

The study team had expected that answers to this question would describe technical difficulties such as incompatible file formats, interface problems, or low COTS product quality. However, much of the data gathered through this question had to do with the administrative difficulties of COTS integration, e.g., procurement, finding information on current licenses, negotiating maintenance agreements, etc. As a result, a major portion of the study's recommendations to NASA had to do with more administrative support of various kinds for COTS integration projects.

Semistructured interviews were also used in the Inspection Study. After each inspection meeting, an interview guide was constructed to include the information missing from the data form for that inspection, as well as several questions that were asked of all interviewees. The questions asked also varied somewhat depending on the role that the interviewee played in the inspection. An example of such a form is shown in Fig. 3. Most interviews in this study were audiotaped in their entirety. Extensive field notes were written immediately after each interview. The tapes were used during the writing of field notes, but they were not transcribed verbatim.

2.3 Coding

Most empirical software engineering studies employ a combination of qualitative and quantitative methods and data. There are a number of ways to combine such methods. One commonly used strategy is to extract values for quantitative variables from qualitative data (often collected from observations or interviews) in order to perform some type of quantitative or statistical analysis. This process is called *coding*.

To understand the data transformation that takes place during coding, we need to address a common misconception about the difference between quantitative and qualitative data. As defined earlier, qualitative data is information expressed as words or pictures, while quantitative data is represented as numbers or other discrete categories. In other words, the distinction between qualitative and quantitative data has to do with how the information is represented, not whether it is subjective or objective. Qualitative data is often assumed to be subjective, but that is not necessarily the case. On the other hand, quantitative data is often assumed to be objective, but neither is that necessarily the case. In fact, the objectivity or subjectivity of data is completely orthogonal to whether it is qualitative or quantitative. The process of coding transforms qualitative data into quantitative data, but it does not affect its subjectivity or objectivity. For example, consider the following text, which constitutes a fragment of qualitative data:

> Tom, Shirley, and Fred were the only participants in the meeting.

Now consider the following quantitative data, which was generated by coding the above qualitative data:

> num_participants = 3

The fact that the information is objective was not changed by the coding process. Note also that the process of coding has resulted in some lost information (the names of the participants). This is frequently the case, as qualitative information often carries more content than is easily quantified. Consider another example:

Interview Guide

Logistical info: record name, office#, date, time

Organization:

How long have you worked on [project]? At [company]?

Have you worked with any of the [project] members before on other projects?

Who on the [project] team do you interact with most?

To whom do you report?

To whom are you responsible for your progress on [project]?

Inspection process:

Who chose the inspectors?

How long did it take?

Why were those ones chosen in particular?

Which inspectors inspected what?

Who took care of scheduling?

Was it done via email or face-to-face?

How much time did it take?

What steps were involved in putting together the inspection package?

How much time did that take?

How are [project] inspections different from inspections in other [company] projects you□ve bee on?

How was this inspection different from other [project] inspections you□ve been involved with?

Reviewed material:

How much was inspected?

How is that measured?

Were the inspected classes more or less complex than average?

Fig. 3. An interview guide used in the Inspection Study.

> Susan said that this particular C++ class was really very easy to understand, and not very complex at all, especially compared to other classes in the system.

And the resulting coded quantitative data:

> complexity = low

Again, the process of coding this information did not make it more objective, although the quantitative form may appear less subjective.

Coding results in more reliably accurate quantitative data when it is restricted to straightforward, objective information, as in the first example above. However, it is often desirable to quantify subjective information as well in order to perform statistical analysis. This must be done with

care in order to minimize the amount of information lost in the transformation and to ensure the accuracy of the resulting quantitative data as much as possible. Often subjects use different words to describe the same phenomenon, and the same words to describe different phenomena. In describing a subjective concept (e.g., the complexity of a C++ class), a subject may use straightforward words (e.g., low, medium, high), that mask underlying ambiguities. For example, if a subject says that a particular class has "low complexity," does that mean that it was easy to read and understand, or easy to write, or unlikely to contain defects, or just small? In coding such concepts, the researcher must pay close attention to the words and meanings of the subjects (as contained in the field notes) to make sure that they are being interpreted accurately.

In the Inspection Study, much of the qualitative data was coded into quantitative variables to be used in statistical analyses. Much of the coding was straightforward extraction of objective data (e.g., number of participants, meeting length, etc.). But some thorny coding issues came up with subjective data. In particular, the coding of complexity (alluded to above) turned out to be rather complicated.

Complexity information was gathered by asking at least two of the participants in each inspection to comment on the complexity of the material being inspected. "Complexity" was intentionally left undefined in an effort to collect qualitative information about developers' views of complexity. If the developer requested a clarification, the interviewer used the term "how difficult it was to inspect." The complexity variable was coded into five levels (very low, low, average, high, very high), which in some cases was not difficult to do from the developers' comments (comments like "Class from Hell" and "piece of cake" were helpful). However, in many cases, developers went beyond these categories to explain the complexity of the inspected material in more detail.

In most cases, complexity was concentrated in the source code of the class or classes being inspected. Some interviewees compared the class in question with other classes in the system. In most of these cases, the class was compared to the developer's idea of "average." The field notes contain phrases like "much less complex than average," "more complex than average," "a little easier than average," "simpler than average," and "harder than average." In some cases, the comparisons were absolute: "the most complex he's looked at," or "the most complex in the system," or "It's a harder class than the other ones we've inspected," or "the simplest class we've got." Such characterizations were also fairly easy to code on a five-point scale, where anything described as "average" fell in the middle category, classes described as differing from average (e.g., "more complex than average") were rated in the category just above or below "average," and the classes described as extreme in some way (e.g., "the simplest class we've got") were put in the extreme categories.

Many classes were difficult for developers to rate because they contained one small piece that was difficult to inspect, but the class as a whole was not particularly complex. Some developers specifically referred to the mathematics or computational nature of a class in defining its complexity. Some functions or classes had "tons of complex mathematical formulas," or "ugly mathematical equations." In some cases, the code for the class itself was not so difficult, but inspecting the test plan and results was. And in other cases, it was the complexity of the functional specifications that was a problem. In some cases, there was some specific characteristic of the class that developers cited that was strongly related to its complexity. There was a large number of such characteristics, including things inherent to the class (multiple parameters, time tags), programming style (a proliferation of temporary variables), the overhead associated with C++, and general tedium. While information like this is what makes qualitative data so rich and informative, it does make coding difficult. In such situations, it is important to keep in mind the goals and objectives of the study. In this example, "complexity" was interesting only in how it affected the inspection. Thus, the coding process concentrated on those cited characteristics that would logically affect how well or how efficiently the material could be inspected (e.g., complex mathematics). When a particular class exhibited one or more of these characteristics, it was put into the "high" complexity category. If many of these characteristics were used to describe a particular class, or if the subject described it as very complex in other ways, then it was rated "very high."

Another situation that complicates coding is when something is rated differently by different subjects. There were eight inspections in the Inspection Study in which the complexity of the inspected material was rated differently by different participants in the inspection. In all but one of these cases, the ratings differed by only one level (e.g., "average" and "high," or "high" and "very high," etc.). In half of the eight cases, the author of the inspected class rated the material more complex than did the inspectors. One way to resolve such discrepancies is to decide that one subject (or data source) is more reliable than another. Miles and Huberman [14] discuss a number of factors that affect the reliability of one data source as compared with another, and the process of weighting data with respect to its source. In the Inspection Study, it was decided that an inspector was a more reliable judge of the complexity of the code than the author, since we were interested in how complexity might affect the inspection of that code.

In summary, coding qualitative information into quantitative data is often useful and even necessary, but must be done carefully. It should be remembered that coding adds neither objectivity nor accuracy to data, although it may appear that way. Coding is especially difficult when the concept to be coded is subjective in nature, when the terminology used to describe it varies and is difficult to interpret, and when different data sources disagree.

3 Data Analysis Methods

Collection of qualitative data is often a very satisfying experience for the researcher. Although it is often more labor-intensive, it is also more enjoyable to collect than quantitative data. It is interesting and engaging and it often gives the researcher the sense that they are closer to reality than when dealing with quantitative abstractions. Many researchers wish that their work could end there. The

analysis of qualitative data is, in this researcher's experience, not nearly as inspiring as its collection. It is sometimes boring, often tedious, and always more time-consuming than expected. However, the alternative to data analysis (which, unfortunately, is sometimes practiced even in published work) is to simply write down all the researcher's beliefs and impressions based on the time they have spent in the field collecting data. This alternative pseudoanalysis method is attractive because it is certainly easier than rigorous analysis, and most researchers feel that they "know" a great deal about the setting they have studied. But it is neither scientific nor reliable, and this practice is largely responsible for the skepticism about qualitative methods that is so prevalent in our field.

In the following sections, I have divided analysis methods roughly into two categories, although the line between them is not well delineated. The first set of methods is used to generate hypotheses that fit the data (or are "grounded" in the data). The second set of methods is used to build up the "weight of evidence" necessary to confirm hypotheses. In most studies, methods from both groups are used and combined in order to produce results that are both grounded and supported by a body of evidence.

3.1 Generation of Theory

Theory generation methods are generally used to extract from a set of field notes a statement or proposition that is supported in multiple ways by the data. The statement or proposition is first constructed from some passage in the notes, and then refined, modified, and elaborated upon as other related passages are found and incorporated. The end result is a statement or proposition that insightfully and richly describes a phenomenon. Often these propositions are used as hypotheses to be tested in a future study or in some later stage of the same study. These methods are often referred to as "grounded theory" methods because the theories, or propositions, are "grounded" in the data [9].

3.1.1 Constant Comparison Method

The classic theory generation method in the qualitative literature is the *constant comparison method*. This method was originally presented by Glaser and Strauss [9], but has been more clearly and practically explained by others since (e.g., [14]). The process begins with coding the field notes, but this is a different type of coding than that described earlier. Coding in this context means attaching codes, or labels, to pieces of text which are relevant to a particular theme or idea that is of interest in the study. Then passages of text are grouped into patterns according to the codes and subcodes they've been assigned. These groupings are examined for underlying themes and explanations of phenomena. The next step is the writing of a field memo that articulates a proposition (a preliminary hypothesis to be considered) or an observation synthesized from the coded data. The field memo is meant to be an informal way to record the researcher's discoveries quickly before they are lost. Because qualitative data collection and analysis occur concurrently, the feasibility of the new proposition is then checked in the next round of data collection.

There are several ways to go about coding qualitative data. Codes can be either preformed or postformed. When the objectives of the study are clear ahead of time, a set of preformed codes (a "start list" [14]) can be constructed before data collection begins and then used to code the data. This initial set of codes comes from the goals of the study, the research questions, and predefined variables of interest. Of course, codes can be added, deleted, merged, subdivided, or modified during the course of the study. Having a preformed set of codes, however, helps the process get started. There are a number of high-level coding taxonomies suggested in the literature (see [14] for some examples), but they are more appropriate for use by sociologists and anthropologists and are not very useful to software engineering researchers. Postformed codes (codes created during the coding process) are used when the study objectives are very open and unfocused.

The set of codes often has a structure to it. That is, there are categories of codes as well as subcodes. For example, in the study of inspections, one of the categories of codes used was "variables," which indicated passages in the field notes that helped to determine values for the quantitative study variables. One code in the "variables" category was "complexity" (described in detail in Section 2.3), which attempted to capture the complexity of the inspection. Subcodes included "specification complexity," "mathematical complexity," "relative complexity," etc. Notice that none of the codes relates a value, just a concept. For example, there are no codes like "high complexity" or "low complexity."

Field notes should be coded periodically, i.e., it's not wise to wait until all data have been collected and then try to code all the field notes at once. Coding a section of notes involves reading through it once, then going back and assigning codes to "chunks" of text (which vary widely in size) and then reading through it again to make sure that the codes are being used consistently. Not everything in the notes needs to be assigned a code, and differently coded chunks often overlap. In the section of coded notes from the Inspection Study, below, the codes **T**, **CG**, and **S** correspond to passages about testing, the core group, and functional specifications, respectively. The numbers simply number the passages chronologically within each code.

(T4) These classes had already been extensively tested, and this was cited as the reason that very few defects were found. [Moderator] said: "...must have done some really exhaustive testing on this class" (EV2.2)

(CG18) [Inspector2] said very little in the inspection, despite the fact that twice [Moderator] asked him specifically if he had any questions or issues. Once he said that he had had a whole bunch of questions, but he had already talked to [Author] and resolved them all.

OC: Find out how much time was spent when [Author] and [Inspector2] met.

(S4) Several discussions had to do with the fact that the specs had not been updated. [Author] had worked from a set of updated specs that she had gotten from her officemate (who is not on the [project] team, as far as I know). I think these were updated [previous project]

specs. The [project] specs did not reflect the updates. [Team lead] was given an action item to work with [Spec guru] to make sure that the specs were updated.

It is good practice to read through the field notes written thus far from time to time, even after they have been coded. Each reading often brings new insight and keeps the relevant issues fresh in the researcher's mind. It also helps to review the codes being used to determine if they still capture the relevant ideas present in the data. Reviewing coded notes often brings out opportunities for refining, aggregating, or augmenting the set of codes. When the set of codes change, the codes in the text should also be updated.

There are software packages on the market that facilitate coding and other types of qualitative analysis (see [14], appendix, for an overview of qualitative analysis software). However, this author found a word processor to be adequate for this purpose. Codes can be placed in the text in ways that separate them from the text (e.g., delimited by special characters or in capital letters) so that the search facility of the word processor can be used to find them easily. This can facilitate grouping of coded chunks, as well as modifying codes according to changes in the coding scheme.

The next step in the process is to look at groups of coded passages to find patterns and trends. One way to do this is to use the search facility of the word processor to search for a particular code, moving to each passage assigned that code and reading it in context. It is not recommended to cut and paste similarly coded passages into one long passage so that they can be read together. The context of each passage is important and must be included in consideration of each group of passages.

There is little guidance in the literature for the intellectual process of finding patterns and trends in qualitative data. Coding helps a great deal in organizing and breaking up what is usually a very large amount of data. However, beyond the mechanics of coding and grouping, the process is largely creative. That is not to say that it is purely subjective, however. Any proposition that the researcher synthesizes must be clearly and strongly supported by the data. Data analysis is not a process of writing down "impressions" or "hunches." There is a great temptation to simply write, however, because the researcher has by this time spent a great deal of time in the study setting and may believe that he or she has a deep intuitive understanding of what is going on in that setting. Such intuition may help guide the process of analyzing the data, but it does not constitute conclusions unless it is clearly supported by the data.

Field memos are the vehicle by which the researcher first articulates the findings. Field memos can take a number of forms, from a bulleted list of related themes, to a reminder to go back to check a particular idea later, to several pages outlining a more complex proposition. They can be very informal but they must be clear enough to express the idea being presented, either to other researchers or to the principal researcher later, when the idea is not so fresh. The point is that, during qualitative data analysis, ideas sometimes form very quickly and it is easy to jump from topic to topic without forming complete propositions. Field memos provide a way to capture some of those possibly incomplete thoughts before they get lost in the next interesting idea. More detailed memos can also show how strong or weak the support for a particular proposition is thus far. According to Miles and Huberman, they are "one of the most useful and powerful sense-making tools at hand." [14, p. 72]. Fig. 4 shows an example of a short field memo from the Inspection Study on the subject of the role of functional specifications in inspection meetings. Interspersed throughout the memo are references to coded segments in the field notes.

The actual development of propositions can be done through memos, as described above, or they can be documented more directly. That is, they can be listed as they are discovered in the coded data. This straightforward approach is more efficient when the propositions are simpler and more obviously supported by the data. In addition to listing them, their supporting and refuting texts must be documented as well.

Ideally, after every round of coding and analysis, there is more data collection to be done which provides an opportunity to check any propositions that have been formed. This can happen in several ways. In particular, intermediate propositions can be checked by focusing the next round of data collection in an effort to collect data that might support or refute the proposition. For example, if the proposition had to do with the amount of time spent in preparation for code inspections where the author is inexperienced, then an effort might be made in the next round of data collection to observe as many inspections as possible with inexperienced authors but which vary in other ways. In this way, opportunities may arise for refining the proposition (e.g., we may find that it holds only when the material to be inspected is particularly large). Also, if the proposition holds in different situations, then further evidence is gathered to support its representativeness. This approach may offend the sensibilities of researchers who are accustomed to performing quantitative analyses that rely on random sampling to help ensure representativeness. The qualitative researcher, on the other hand, typically uses methods to ensure representativeness later in the study by choosing cases accordingly during the course of the study. This is sometimes called theoretical sampling, which we will not discuss in detail here, but the reader is referred to [14] for a good explanation of its use and justification.

3.1.2 Cross-Case Analysis

Glaser and Strauss's constant comparison method can be used on any set of field notes, whether they all come from the same "case" or setting, or whether they constitute the data collected from a number of settings. This is one reason that keeping chunks in their context is so important. In many software engineering studies, the data can be divided into "cases," which in quantitative studies might be referred to as "data points" or "trials." When this is possible, *cross-case analysis* is appropriate. For example, in the Inspection study, all data were collected from the same development project, so they could be viewed as a single case study. Some of the analysis was done with this perspective (e.g., the analysis described in the previous section). However,

Specifications played an important role in the inspection process. Many developers said that they relied heavily on the specs as they reviewed the material prior to the inspection meeting. One developer, when asked how he spent his preparation time before an inspection, said that the first thing he did was to print out the specs and read them "to get a feel for what the class is supposed to do." (S1) Another developer described her preparation activities as going through the specs and checking to make sure that everything in the spec is represented in the code. At the same time, she tries to find any extraneous code that implements features not in the spec. (S3) A new member of the team spent 14 hours inspecting a class, primarily going through the code units, comparing them to the specs, in an effort to understand each line of code. (S6) Another developer explained the advantages of the specs used on the project: "I like our specs; I like having that kind of information at that level...Most projects end up without specs." (S8)

However, occasional inadequacies or errors in the specifications often caused problems during inspections, especially those involving highly complex code. Many times the problem was one of interpretation; there was difficulty in understanding some aspect of the spec which became relevant during the inspection meeting. These cases all involved highly complex code (S2, S5, S12) and most (S2, S5) occurred early in the project. In several inspections, however, actual errors were found in the specification (S2, S10, S13, S14, S11). Again, most of these cases involved highly complex code. Other problems included incomplete and outdated specs. Incompleteness was a problem when the spec did not specify all functions or interfaces that were necessary for a class to be implemented correctly (S9). Because the specs were being updated during development, problems sometimes occurred when the latest version was not placed in the central specification repository (S4, S7). This happened even with fairly simple classes (S7).

Most issues that arose concerning specifications could not be resolved during the inspection meeting. The analyst who had major responsibility for specifications did not attend inspection meetings. So most specs questions were left as open issues and referred to this analyst. His name was mentioned frequently. In one case, it was even suggested that he be invited to inspections. Discussion of specs issues consumed a total of 40 minutes of meeting time over the 23 inspection meetings observed. Discussion of specs issues constituted more than a quarter of meeting time in only two cases, both of which involved highly complex classes. Most of this time was spent attempting to understand the complexity of the spec.

Fig. 4. An example field memo.

some crosscase analysis was also performed by treating each inspection as a "case."

Eisenhardt [7] suggests several useful strategies for cross-case analysis, all based on the goal of looking at the data in many different ways. For example, the cases can be partitioned into two groups based on some attribute (e.g., number of people involved, type of product, etc.), and then examined to see what similarities hold within each group, and what differences exist between the two groups. Another strategy is to compare pairs of cases to determine variations and similarities. A third strategy presented by Eisenhardt is to divide the data based on data source (e.g., interviews, observations, etc.).

In the Inspection Study, a comparison method was used that combined Glaser and Strauss's method and the Eisenhardt approach and was further modified for the purposes of the study. The main purpose of this part of the data analysis was to generate hypotheses that could be tested in the quantitative stage of data analysis. Therefore, the emphasis was on identifying both the relevant variables and the possible relationships between them.

Our comparison method progressed as follows. The field notes corresponding to the first two inspections observed were reviewed. For each of these two inspections, a list was compiled of short phrases that described each inspection (e.g., aggressive author; a lot of discussion of the code generator; discussion dominated by one inspector; really long meeting, etc.). Then these two lists were compared to determine the similarities and differences. The next step was to list, in the form of propositions, conclusions one would draw if these two inspections were the only two in the data set (e.g., really long meetings are generally dominated by one inspector). Each proposition had associated with it a list of inspections that supported it (this list began with the first two inspections compared). After analyzing the first two inspections in this way, the third inspection was examined and a list of its characteristics was compiled. Then it was determined whether this third inspection supported or refuted any of the propositions formulated from the first two. If a proposition was supported, then this third inspection was added to its list of supporting evidence. If it contradicted a proposition then either the proposition was modified (e.g., really long meetings are generally dominated by one inspector when the other inspectors are inexperienced) or the inspection was noted as refuting that proposition. And then any additional propositions suggested by the third inspection were added to the list. This process was repeated with each

subsequent inspection. The end result was a list of propositions (most very rich in detail), each with a set of supporting and refuting evidence (inspections).

A different approach to cross-case analysis was used in the COTS Study. Each development project that was studied was treated as a separate case. The objective of the analysis was to document the COTS integration process by building an abstraction, or model, of the process that was flexible enough to accomodate all of the different variations that existed in the different projects. This model-building exercise was carried out iteratively by a team of researchers. The first step was to group all of the field notes according to the development project that the interviewee was working on. Then, for each project, the notes were read carefully and a preliminary process model was built for that project's COTS integration process. These preliminary models were built by different researchers. Then the study team came together to study the models, identify similarities and differences, and resolve discrepancies in terminology. From this, one single model was built that encompassed the models for the different projects. This aggregate model went through numerous cycles of review and modification by different members of the study team. Finally, an extensive member checking process (see Section 3.2) was conducted through individual interviews with project members, a large group interview with a number of project personnel, and some e-mail reviews of the model. The resulting model can be found in [15].

3.2 Confirmation of Theory

Most qualitative data analysis methods are aimed at generating theory, as described in the previous section, but there are a number of methods and approaches to strengthening, or "confirming" a proposition after it has been generated from the data. The goal is to build up the "weight of evidence" in support of a particular proposition, not to prove it. Although quantitative hypothesis testing methods seem more conclusive than the methods we will present in this section, they really do not provide any stronger evidence of a proposition's truth. A hypothesis cannot be proven, it can only be supported or refuted, and this is true using either quantitative or qualitative evidence, or both. However, software engineers are apt to attribute more significance to a single statistically significant finding in support of a hypothesis than is appropriate, simply because empirical findings are so scarce in our field. In short, the best we can hope for is to build a convincing body of evidence to support any proposition we are trying to confirm. This can be done either qualitatively or quantitatively, but is best done with a combination of methods. Qualitative methods have the added advantage of providing more explanatory information, and help in refining a proposition to better fit the data.

One of the most important ways to help confirm a qualitatively generated proposition is to ensure the *validity* of the methods used to generate it. In previous sections, we have briefly addressed some of the validity concerns in qualitative studies. One is representativeness, which has to do with the people and events chosen to be interviewed or observed. In Section 3.1.1, there is a discussion of how, after initial propositions are generated, cases for further study can be specifically chosen to increase or ensure representativeness. Another validity concern is the possibility of researcher effects on the study. Miles and Huberman warn of two types of researcher effects and present some techniques for countering them. The first is that the presence of the researcher may affect the behavior of the subjects. This type of effect is discussed earlier in Section 2.1. The second is that the researchers may lose their objectivity by becoming too close to the setting being observed. A quote from one researcher [21] illustrates the second type of bias: "I began as a nonparticipating observer and ended up as a nonobserving participant." In studies of software engineering, it is unlikely that the researcher will be permitted to become involved technically in the work being studied, unless that was part of the study plan from the beginning, but it is possible for the researcher to become part of the political and organizational context of the project without realizing it.

Triangulation is another important tool for confirming the validity of conclusions. The concept is not limited to qualitative studies. The basic idea is to gather different types of evidence to support a proposition. The evidence might come from different sources, be collected using different methods, be analyzed using different methods, have different forms (interviews, observations, documents, etc.), or come from a different study altogether. This last point means that triangulation also includes what we normally call replication. It also includes the combining of quantitative and qualitative methods. A classic combination is the statistical testing of a hypothesis that has been generated qualitatively. In the Inspection Study, triangulation occurred at the data source level. Certain types of data (e.g., size and complexity of the code inspected, the roles of different participants, etc.) were gathered multiple times, from observations, from interviews, and from the inspection data forms that each inspection moderator filled out. For example, the size of the code inspected was listed on the data form for each inspection, but it was also asked of each author when they were interviewed. For each inspection, the complexity of the code was determined by asking the author and at least one inspector to rate the code.

Anomalies in the data (including outliers, extreme cases, and surprises) are treated very differently in qualitative research than in quantitative research. In quantitative analysis, there are statistical methods for identifying and eliminating outliers from the analysis. Extreme cases can be effectively ignored in statistical tests if they are outweighed by more average cases. But in qualitative analysis, these anomalies play an important role in explaining, shaping, and even supporting a proposition. As Miles and Huberman explain, "the outlier is your friend." The Inspection Study has a good outlier example. There were few cases in the study that illustrated what happens when the group of inspection participants is organizationally distant (i.e., include members from disparate parts of the organization). However, one case could easily be identified in terms of both its length and the number of defects reported in the meeting. This case also involved a set of organizationally distant inspection participants. The unusual values for length and number of defects could not be explained by any

of the other variables that had been determined to affect these factors. Thus, we could hypothesize that organizational distance had an effect on length and number of defects. In addition, the case provided a lot of explanatory data on why that effect existed.

Negative case analysis [11] is another qualitative tool for helping to confirm hypotheses. Judd et al. even go so far as to say that "negative case analysis is what the field-worker uses in place of statistical analysis." The idea is incorporated into each of the analysis methods described in Section 3.1. When performed rigorously, the process involves an exhaustive search for evidence that might contradict a generated proposition, revision of the proposition to cover the negative evidence, rechecking the new proposition against existing and newly collected data, and then continuing the search for contradictory evidence. The search for contradictory evidence can include purposely selecting new cases for study that increase representativeness, as explained above, as well as seeking new sources and types of data to help triangulate the findings.

Replication, as with quantitative studies, is a powerful but expensive tool for confirming findings. Replication in the qualitative arena, however, has a slightly looser meaning than in quantitative research. While a quantitative study, to be called a replication of another study, generally is expected to employ the same instruments, measures, and procedures as the original study (although debate continues as to what extent this must be true), a qualitative replication must only preserve the conditions set forth in the theory being tested. That is, if the proposition to be tested is something like

Gilb-type inspections of C++ code involving two inspectors and a moderator will take longer but reveal more defects if the inspection participants have not worked together before.

then the replicating study must be of Gilb-type inspections of C++ code involving two inspectors and a moderator, some of which have participants who have worked together before and some who have participants who have not worked together before. Data do not necessarily have to be collected or analyzed in the same way that they were in the original study.

One last method for helping to confirm findings, which is particularly well suited to most studies of software engineering, is getting feedback on the findings from the subjects who provided the data in the first place. This strategy is sometimes called *member checking* [12]. Presenting findings to subjects, either formally or informally, has the added benefits of making subjects feel part of the process, helping them to understand how the results were derived, and gaining their support for final conclusions. This is especially important when the results of the study may change the way the subjects will be expected to do their jobs. This is usually what we, as empirical software engineering researchers, hope will happen. Researchers in our area often have a marketing role as well, trying to promote the importance and usefulness of empirical study in software engineering. Member checking helps to accomplish this at the grass roots. Miles and Huberman give several guidelines on how and when to best present intermediate findings to subjects, including taking care that the results presented are couched in local terminology, explaining the findings from the raw data up, and taking into account a subject's possible personal reaction to a finding (e.g., if it is threatening or critical).

Member checking was used extensively in the Inspection Study. An entire round of scheduled interviews was devoted to this exercise, and it yielded a great deal of insight. For example, a finding emerged that indicated that, as the project progressed, inspection participants were spending less and less time discussing issues in inspection meetings that eventually had to be referred to someone not at the meeting, i.e., issues that were not resolved in the meeting. One subject, when presented with this finding, explained that this was because developers were getting better at recognizing issues and problems that were best referred to others, and were less likely now than at the beginning of the project to waste time trying to resolve any issues they were not equipped to resolve. This was an important insight, and in particular one that had not occurred to the researcher.

In summary, many qualitative methods for confirming theory are also employed during the theory generation stage of a study. That is, as propositions are being generated, they are immediately subjected to some testing before they are even reported as findings. The idea is to build up a "weight of evidence" that supports the hypothesis, where the evidence is as diverse as possible. This is not so different from the aim of quantitative research, in which a hypothesis is never "proven," but evidence, in the form of statistically significant results from different settings and different researchers, is built up to support it. It could be said that some qualitative methods used to test propositions are actually stronger than statistical tests because they do not allow any contradictory evidence. Any data that contradict the proposition are used to modify it so that the resulting proposition fits all the data. However, ideally, any proposition, no matter how generated, is best supported by both qualitative and quantitative evidence.

4 Experimental Design

The focus of this article has been to provide guidance on using qualitative research methods, particularly in studies in which they are combined with quantitative methods, in empirical studies of software engineering. The combination of quantitative and qualitative methods is usually more fruitful than either in isolation. This section explores in a bit more detail how such combinations can be designed.

Empirical studies come in a wide variety of types, employing a variety of designs. A large number of them, however, fall into one of the following set of categories, described by [3]:

- *Blocked subject-project study.* In this design, several different development projects, or applications, are studied, with several different subjects or teams of subjects working on each application. Using multiple applications and subjects helps to reduce bias, but increases the cost of the experiment.

- *Replicated project study.* Studies of this type employ multiple subjects (or teams of subjects), all working on the same project or application. Keeping the application constant isolates the effect of differences between subjects, especially, it is hoped, the treatment effect.
- *Multiproject variation.* One use of this study design is to observe the performance of a single subject or team of subjects on a project before some treatment is applied (e.g., training in a new technique) and then after that treatment is applied, on a different project.
- *Single project study.* Similar to the common notion of a case study, this approach usually involves an in-depth study of a single instance of a project, in which certain attributes are examined and possibly compared to some organizational baseline.

Quantitative and qualitative methods of data collection and analysis can be combined in any of these types of study designs. In a blocked subject-project study, for example, one way to incorporate qualitative data is to use it to illuminate the statistical results. These types of studies are often aimed at testing hypotheses and finding causal relationships between variables. Qualitative data can be used to go beyond the statistics and help explain the reasons behind the hypotheses and relationships. For example, in a study evaluating a new software engineering technique (e.g., a testing technique), a blocked subject-project design may be chosen so that the technique can be tried on a variety of different applications that vary in different ways. The quantitative results from such a study might show that the new technique was effective on some applications but not on others. If, however, qualitative data was also collected, say from follow-up interviews with the subjects (Section 2.2), then the researchers may find the reasons for the differences in effectiveness.

As a more specific example, consider a replicated project study designed, as above, to test the effectiveness of a new testing technique. This design was chosen in order to concentrate on differences between subjects, not differences between applications. Quantitative measures are defined and initial data are collected using quantitative methods from groups of developers using and not using the new technique. The quantitative data are analyzed and the predefined hypotheses are tested. At the same time, the subjects are interviewed, as described in Section 2.2. The field notes from the interviews are analyzed using the constant comparison method to elicit trends and patterns in how the subjects describe their use of the technique and what they liked and disliked about it. The result is a set of propositions that illuminate the quantitative findings. For example, the quantitative analysis may show that testing effort increases for very experienced developers using the new technique. The interview data may explain this quantitative result by revealing that experienced developers were frustrated with the overhead of the new technique, which slowed down their testing progress.

Suppose that a multiproject variation study was planned, aimed at understanding and documenting a new collaborative design process. One team of subjects is identified and trained in the process, and their experience using it on different projects is studied. Qualitative data are first gathered through participant observation of the design meetings (as in Section 2.1). The extensive field notes are analyzed using both constant comparison and cross-case analysis (see Section 3). The propositions generated by this process then are used to design a study of the resulting designs and code. For example, suppose a proposition is generated from the qualitative data that says that, under certain conditions, the design seemed to be generated by one designer with little input from the others, the result being a simpler design but less confidence in the quality of that design. This proposition would suggest a quantitative investigation of designs created under those conditions, comparing their complexity and the number of defects found later. The qualitative and quantitative aspects of the study would then proceed in parallel. The qualitative analysis would concentrate on revealing new issues and tracking changes relative to other issues, while the quantitative analysis would focus on looking more closely at the issues suggested by the qualitative analysis. At the end of the series of projects, the result would be a very multifaceted view of the effectiveness of the new design process.

In single project studies, the process is often begun with qualitative methods. Suppose an organization wanted to investigate the types of errors made by developers with different types of experience and training. It is decided to concentrate on one particular project that is representative of the organization and that includes a wide variety of developers. First, data are collected qualitatively through interviews with testers (to get information on the types of defects found in the code) and developers (to get information on experience and training, and on the errors that led to the code defects). Part of the interview data are coded (see Section 2.3) to yield quantitative variables describing numbers of defects and years of training and/or experience in different areas. A taxonomy of error types and the types of defects they cause is generated qualitatively from the raw interview data, using cross-case analysis and some of the display techniques described in [14]. Any statistical relationships found between the quantitative variables are also checked against the qualitative data. The result is a set of well-triangulated, grounded, hypotheses, and a set of well-defined quantitative measures that can then be used to collect and analyze quantitative data for further investigation.

The previous examples present only a few ideas about how qualitative and quantitative methods can be used to complement each other. In each example, important information is gained that could not be gathered with only one type of method.

5 Conclusions

This article has reviewed a number of different methods for the collection and analysis of qualitative data. These methods are described in terms of how they might be applied to the empirical study of software engineering. It is also argued that nearly any software engineering issue is best investigated using a combination of qualitative and

quantitative methods. Several scenarios are described that illustrate different ways of combining these research methods.

Empiricists in software engineering often complain about the lack of opportunities to study software development and maintenance in real settings. This really implies that we must exploit to the fullest every opportunity we do have, by collecting and analyzing as much data of as many different types as possible. Qualitative data is richer than quantitative data, so using qualitative methods increases the amount of information contained in the data collected. It also increases the diversity of the data and thus increases confidence in the results through triangulation, multiple analyses, and greater interpretive ability.

ACKNOWLEDGMENTS

This work was supported in part by NASA grant 01-5-26393 and by IBM Canada Ltd.'s Centre for Advanced Studies.

REFERENCES

[1] T. Athey, "Leadership Challenges for the Future," *IEEE Software,* vol. 15, no. 3, pp. 72–77, May 1998.
[2] S.R. Barley, "The Alignment of Technology and Structure through Roles and Networks," *Administrative Science Quarterly,* vol. 35, pp. 61–103, 1990.
[3] V.R. Basili, R.W. Selby, and D.H. Hutchens, "Experimentation in Software Engineering," *IEEE Trans. Software Eng.,* vol. 12, no. 7, pp. 733–743, July 1986.
[4] F. Brooks Jr., *The Mythical Man-Month.* Reading, Mass.: Addison-Wesley, 1975.
[5] N. Brown, "Industrial-Strength Management Strategies," *IEEE Software,* vol. 13, no. 4, pp. 94–103, July 1996.
[6] W. Curtis, "By the Way, Did Anyone Study Any Real Programmers?" *Empirical Studies of Programmers,* R. Soloway and S. Iyengar, eds., pp. 256–262, Norwood, N.J.: Ablex Publishing, 1986.
[7] K.M. Eisenhardt, "Building Theories from Case Study Research," *Academy of Management Review,* vol. 14, pp. 532–550, 1989.
[8] J.F. Gilgun, "Definitions, Methodologies, and Methods in Qualitative Family Research," *Qualitative Methods in Family Research.* Thousand Oaks: Sage, 1992.
[9] B.G. Glaser and A.L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine Publishing, 1967.
[10] J.T. Hackos and J.D. Redish, *User and Task Analysis for Interface Design.* pp. 258–259, New York: John Wiley & Sons, ch. 9, 1998.
[11] C.M. Judd, E.R. Smith, and L.H. Kidder, *Research Methods in Social Relations,* sixth ed., ch. 13, pp. 298–320, Ft. Worth: Harcourt Brace Jovanovich, 1991.
[12] Y.S. Lincoln and E.G. Guba, *Naturalistic Inquiry.* Thousand Oaks Calif.: Sage, 1985.
[13] S. McConnell, *Rapid Development: Taming Wild Software Schedules.* ch. 11, pp. 249–272, Redmond, Washington: Microsoft Press, 1996.
[14] M.B. Miles and A.M. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook,* second ed. Thousand Oaks, Calif.: Sage, 1994.
[15] A. Parra, C. Seaman, V.R. Basili, S. Kraft, S. Condon, S. Burke, and D. Yakimovich, "The Package-Based Development Process in the Flight Dynamics Division," *Proc. 22nd Software Eng. Workshop,* pp. 21–56, NASA/Goddard Space Flight Center Software Eng. Laboratory (SEL), Dec. 1997.
[16] D.E. Perry, N.A. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software,* vol. 11, no. 4 pp. 36–45, July 1994.
[17] C.B. Seaman and V.R. Basili, "Communication and Organization: An Empirical Study of Discussion in Inspection Meetings," *IEEE Trans. Software Eng.,* vol. 24, no. 7, pp. 559–572, July 1998.
[18] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction,* third ed., ch. 4, pp. 146–47, Reading, Mass.: Addison-Wesley, 1998.
[19] S.J. Taylor and R. Bogdan, *Introduction to Qualitative Research Methods.* New York: John Wiley & Sons, 1984.
[20] A. von Mayrhauser and A.M. Vans, "Identification of Dynamic Comprehension Processes during Large Scale Maintenance," *IEEE Trans. Software Eng.,* vol. 22, no. 6, pp. 424–437, June 1996.
[21] W.F. Whyte, *Learning from the Field: A Guide from Experience.* Beverly Hills, Calif.: Sage, 1984.

Carolyn B. Seaman holds a BA degree in computer science and mathematics from the College of Wooster, Ohio, an MS degree in information and computer science from the Georgia Institute of Technology, and a PhD degree in computer science from the University of Maryland Baltimore County, College Park. Dr. Seaman is currently an assistant professor of information systems at the University of Maryland Baltimore County. Her research generally falls under the umbrella of empirical studies of software engineering, with particular emphases on maintenance, organizational structure, communication, organizational experience, measurement, COTS-based development, and qualitative research methods. She has worked in the software industry as a software engineer and consultant, and has conducted most of her research in industrial and governmental settings, e.g., IBM Canada Ltd., NASA. She is a member of the IEEE and the IEEE Computer Society.