Chapter 2
Qualitative Methods[1]

Carolyn B. Seaman

Abstract Software engineering involves a blend of non-technical as well as technical issues that often have to be taken into account in the design of empirical studies. In particular, the behavior of people is an integral part of software development and maintenance. This aspect of our subject presents complexities and challenges for the empirical researcher. In many other disciplines, qualitative research methods have been developed and are commonly used to handle the complexity of issues involving people performing tasks in their workplace. This chapter 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.

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 the people performing software engineering tasks. The first two aspects provide more than enough complex problems to keep empirical software engineering researchers busy. But the last factor, the people themselves, introduces aspects that are especially difficult to capture. However, studies attempting to capture human behavior as it relates to software engineering are increasing and, not surprisingly, are increasingly employing qualitative methods (e.g. Lethbridge et al., 2005; Lutters and Seaman, 2007; Orlikowski, 1993; Parra et al., 1997; Rainer et al., 2003; Seaman and Basili, 1998; Singer, 1998; Sharp and Robinson, 2004).

Historically, qualitative research methods grew out of the interpretivist tradition in social science research. Interpretivism, in turn, arose as a reaction to positivism,

[1] Based on "Qualitative Methods in Empirical Studies of Software Engineering" by Carolyn B. Seaman, which appeared in *IEEE Transactions on Software Engineering*, *25*(4):557–572, July/August 1999. © 1999 IEEE.

F. Shull et al. (eds.), *Guide to Advanced Empirical Software Engineering*.

which was and continues to be the prevailing (if implicit) philosophical underpinning of research in the natural and physical sciences, including computer science and software engineering. The positivist researcher views objective truth as possible, i.e. that there exists some absolute truth about the issues of relevance, even if that truth is elusive, and that the role of research is to come ever closer to it. Interpretivism, on the other hand, posits that all truth is socially constructed, meaning that human beings create their own truth about the issues of relevance to them, and these socially constructed truths are valid and valuable. Qualitative methods, then, were required to capture and describe these socially constructed realities. See Creswell (1998) for a fuller explanation of positivism, interpretivism, other related philosophical frameworks, and the role of qualitative research methods in them. For many social science researchers, qualitative methods are reserved exclusively for use by interpretivist researchers, and are not to be mixed with quantitative methods or positivist points of view. However, in recent decades, researchers in information systems, human–computer interaction, and software engineering have begun using qualitative methods, even though the predominant, implicit philosophical stance of these research areas remains positivist (Orlikowski and Baroudi, 1991). Thus, the perspective of this chapter is that qualitative methods are appropriate for (even implicitly) positivist research in software engineering, and a researcher does not have to subscribe wholeheartedly to the interpretivist world view in order to apply them.

Qualitative data are data represented as text and pictures, not numbers (Gilgun, 1992). Qualitative research methods were designed, mostly by educational researchers and other social scientists (Taylor and Bogdan, 1984), to study the complexities of humans (e.g. motivation, communication, understanding). In software engineering, the blend of technical and human aspects lends itself to combining qualitative and quantitative methods, in order to take advantage of the strengths of both.

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 more informative. They help to answer questions that involve variables that are difficult to quantify (particularly human characteristics such as motivation, perception, and experience). They are also used to answer the "why" to questions already addressed by quantitative research. 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.

Methods are described here in terms of how they could be used in a study that mixes qualitative and quantitative methods, as they often are in studies of software engineering. The focus of this chapter is rather narrow, in that it concentrates on only a few techniques, and only a few of the possible research designs that are well suited to common software engineering research topics. See Judd et al. (1991), Lincoln and Guba (1985), Miles and Huberman (1994) and Taylor and Bogdan (1984) for descriptions of other qualitative methods.

The presentation of this chapter divides qualitative methods into those for collecting data and those for analysing data. Examples of several methods are given

for each, and the methods can be combined with each other, as well as with quantitative methods. Throughout this chapter, examples will be drawn from several software engineering studies, including (von Mayrhauser and Vans 1996; Guindon et al., 1987; Lethbridge et al., 2005; Perry et al. 1994; Lutters and Seaman, 2007; Singer, 1998; Orlikowski 1993). More detailed examples will also be used from studies described in Parra et al. (1997) and Seaman and Basili (1998) because they represent the author's experience (both positive and negative).

2. Data Collection Methods

Two data collection methods, direct 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 analysing archival documents are discussed in Taylor and Bogdan (1984). Another useful technique is focus groups, which are treated extensively in the chapter by Kontio et al. (2007, this volume).

2.1. Participant Observation

Participant observation, as defined in Taylor and Bogdan (1984), 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.

Definitions of participant observation differ as to whether it implies that the observer is engaged in the activity being observed (e.g. Barley, 1990), or only that the observer is visibly present and is collecting data with the knowledge of those being observed. To avoid this confusion in terminology, the term direct observation is more usefully used when the researcher is not actively involved in the work 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. For example, it is sometimes possible to capture some of the thought processes of individual developers by logging their keystrokes and mouse movements as they work on a computer (Shneiderman, 1998). This technique is sometimes used in usability studies, where the subjects are software users, but it has not been widely employed in studies of software developers.

Think aloud observation (Hackos and Redish, 1998) requires the subject to verbalize his or her thought process so that the observer can understand the mental process going on. Such protocols are limited by the comfort level of the subject and

their ability to articulate their thoughts. A good software engineering example of this technique is the work of von Mayrhauser and Vans (1996), in which software maintainers were asked to verbalize their thought processes while working on understanding source code. The data was collected by audio- and video-taping the sessions. Another example of a software engineering study based on thinking aloud observations is Guindon, Krasner, and Curtis's study of software designers (Guindon et al., 1987).

A variation on think aloud observation is synchronized shadowing, described in Lethbridge et al. (2005). With synchronized shadowing, two observers watch a subject perform some task while the subject is thinking aloud. Both observers record their notes on laptops whose clocks have been previously synchronized to the second. The two observers record different types of information. For example, one might concentrate on the subject's actions (keystrokes, commands, mouse clicks) while the other concentrates on the subject's goals and motivations (as evidenced by the subject thinking aloud). Both observers timestamp individual observations (using a macro in the word processor) so that the notes can later be synchronized. The end result is a detailed set of field notes that relates actions to goals.

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 Perry et al. (1994). 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 of which an observer must be aware. Many of these are presented here, based in part on the literature (in particular Taylor and Bogdan, 1984) and partly on the particular experience of this researcher with studies of software engineering.

The 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. that it is what usually happens in the environment being observed, and is not affected by the presence of the observer. For example, observers should strive for "fly on the wall" unobtrusiveness. Ideally, all those being observed should know beforehand that the observer will be observing and why. This advance notice avoids having to do a lot of explaining during the observation, which will only remind the subjects that they are being observed. The observer, although visible, should not be disruptive in any way, in particular avoiding making noise or movement that is distracting. 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 those being observed. 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 misinterpret 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 observation, the discussions that took place, any events that took place during the observation, and the tone and mood of the interactions. The notes can also contain observer's comments, marked "OC" in the text of the notes, which record the observer's impressions of some aspect of the activity observed, which may not correspond directly to anything that was actually said or that occurred. For example, impressions about the setting of the observation (e.g. quality of the light, temperature, noise level), the demeanor of the people observed (e.g. if someone appeared to be agitated, ill, or tired), or the internal state of the observer (e.g. if the observer is agitated, ill, or tired, or has some strong emotional reaction to what is being observed) could all be recorded in observer's comments. The level of detail in the notes depends on the objectives of the researcher. The most detailed are verbatim transcripts of everything said and done, 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 Seaman and Basili (1998), 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 Sect. 2.2. Information about communication effort was collected during the observations of code inspections.

Figure 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), the responsibilities of each inspector (which products each was responsible for inspecting), each preparation time, and who was present were all recorded on the data form either before or during the observed inspection. 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. 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.

In order to evaluate the validity and consistency of data collected during observations, rater agreement exercises (Judd et al., 1991) 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%), 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

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 not 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 excecution 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/D—— P—— A—— M———

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

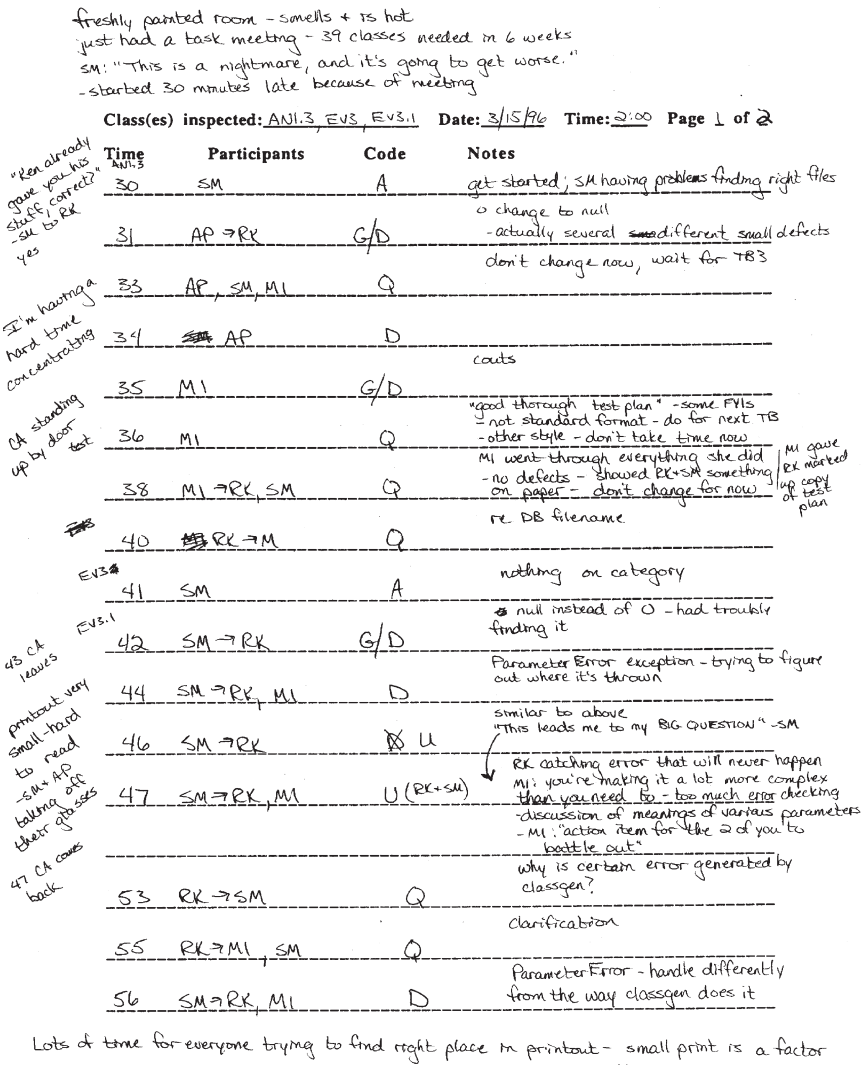

Fig. 2 Time log used to document discussions during inspection meetings

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. 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%. 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 (3 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 in the study design can be overcome if they are discovered early enough. 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 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.

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 (Lutters and Seaman, 2007), to collect opinions or impressions about something, or to help identify the terminology used in a particular setting. In software engineering, they are often used to elicit software processes (Parra et al., 1997). They are sometimes used in combination with observations 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 Lincoln and Guba (1985), 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 more structured an interview, the more likely it is to be focused on quantitative, rather than qualitative data. 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 semi-structured 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. A good example of a software engineering study based on semi-structured interviews is that conducted by Singer (1998), in which software maintainers were asked about their practices. Some of the more structured questions from this study include:

- How many years have you been programming?
- What languages have you had extensive experience programming in?
- How long have you worked on this project?

More open-ended questions included:

- When you get a maintenance request, how do you go about fulfilling it?
- What do you see as the biggest problem in maintaining programmes?

Again, as in the previous section on observation, the advice given here about interviewing is based in part on the literature [in particular Taylor and Bogdan (1984)] 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 judgement 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 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 that is very useful during an interview is an interview guide (Taylor and Bogdan, 1984). An interview guide is not as formal as a data form, but it helps the interviewer to organize the interview. It serves a purpose similar to a script. It usually consists of a list of 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. 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 study that we will use as a detailed example is Parra et al. (1997), a study of Commercial-Off-The-Shelf (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 at that time. 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 [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 licences, 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.

Semi-structured interviews were also used in the Inspection Study (Seaman and Basili, 1998). 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 audio taped 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.

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. The analysis of qualitative data, on the other hand, is not always as pleasant. Although the discovery of new knowledge is always motivating, the mechanics of

Interview Guide

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

Organization:

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

Have you work 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 been on?

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

Reviewed material:

How much was inspected?

How is that measure?

Were the inspected classes more or less complex then average?

Fig. 3 An interview guide used in the Inspection Study

qualitative analysis are sometimes boring, often tedious, and always more time-consuming than expected. It is tempting to take shortcuts in the analysis process, but rigorous analysis is necessary for the integrity of the research, and results in more insightful, useful, and valid conclusions.

As in quantitative studies, data analysis should be planned up front, before data collection begins. However, the difference is that qualitative researchers collect and analyse data nearly in parallel, or at least alternate between the two. Qualitative analysis begins as soon as some significant amount of data has been collected. Preliminary analysis results also can modify subsequent data collection.

In the next two sections, we present several analysis techniques, roughly divided into two categories, although the line between them is not well delineated. The first set of methods (Sect. 3.1) is used to generate hypotheses that fit the data (or are "grounded" in the data), normally used in exploratory, or grounded theory studies (Glaser and Strauss, 1967). Section 3.2 describes some methods used to build up the "weight of evidence" necessary to confirm hypotheses in confirmatory studies. Following, in Sect. 3.3, we discuss the use of visualization of qualitative data, which is useful in conjunction with any analysis approach, and for presenting results. Finally, Sect. 3.4 presents some basic techniques for transforming qualitative data for subsequent quantitative analysis. The methods presented in these sections represent only a small sample of the methods, techniques, and approaches available for analysing qualitative data. Yin (1994) and Miles and Huberman (1994) are excellent sources for other data analysis approaches.

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 (Glaser and Strauss, 1967). Two grounded theory techniques, the constant comparison method and cross-case analysis, are briefly described below. See Seaman (1999) for a fuller description of these techniques as applied to software engineering studies.

3.1.1. Constant Comparison Method

There are a number of methods for conducting and analysing single case studies. An excellent reference for this type of research design is Yin (1994). Here, we will

explore a classic theory generation method, the constant comparison method. This method was originally presented by Glaser and Strauss (1967), but has been more clearly and practically explained by others since (e.g. Miles and Huberman, 1994).

The process begins with open coding of the field notes, which involves attaching codes, or labels, to pieces of text that are relevant to a particular theme or idea of interest in the study. 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" (Miles and Huberman, 1994)] can be constructed before data collection begins and then used to code the data. Postformed codes (codes created during the coding process) are used when the study objectives are very open and unfocused. In either case, the set of codes often develops a structure, with subcodes and categories emerging as the analysis proceeds. 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"

(*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.

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 in the next step of the process, called axial coding. Axial coding can be thought of as the process of reassembling the data that was broken up into parts (chunks) in open coding. One way to do this is 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. This is where the intensive, or "constant" comparison comes in. The coded data is reviewed and re-reviewed in order to identify relationships among categories and codes. The focus is on unifying explanations of underlying phenomenon, in particular the how's and why's.

The next step, selective coding or "sense making," culminates in the writing of a field memo that articulates a proposition (a preliminary hypothesis to be considered) or an observation synthesized from the coded data. Because qualitative data collection and analysis occur concurrently, the feasibility of the new proposition is then checked in the next round of data collection. 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. Field memos also provide a way to capture 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, field memos are "one of the most useful and powerful sense-making tools at hand." (Miles and Huberman, 1994, p. 72)

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. In this way, opportunities may arise for refining the proposition 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 Miles and Huberman (1994) for a good explanation of its use and justification.

3.1.2. Cross-Case Analysis

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, some cross-case analysis was also performed by treating each inspection as a "case."

Eisenhardt (1989) 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 (Seaman and Basili, 1998), we used a comparison method that progressed as follows. The field notes corresponding to the first two inspections observed were reviewed and a list of short descriptors (e.g. aggressive author; discussion dominated by one inspector; really long meeting, etc.) was compiled for each inspection. 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 (beginning with the first two inspections compared). Then the third inspection was examined, a list of its descriptors was compiled, and 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. 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.

A different approach to cross-case analysis was used in the COTS Study (Parra et al., 1997). 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 accommodate 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 by development project. Then, for each project, the notes were used to build a preliminary process model 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 Sect. 3.2) was conducted through individual interviews with project members, a large group interview with a number of project personnel, and some email reviews of the model. The resulting model can be found in Parra et al. (1997).

Cross-case analysis was also used in the Orlikowski study of CASE tool adoption (Orlikowski, 1993). Data from the first case was collected and coded, then the second case's data was collected and an attempt was made to use the same set of codes to analyse it. Of course, some codes were inappropriate or inadequate and so new or modified codes resulted. These were then taken back to the first case, whose data was re-sorted and re-analysed to incorporate the new concepts. This type of back-and-forth analysis [sometimes referred to as "controlled opportunism" (Eisenhardt, 1989)] is a unique and valuable property of grounded theory research.

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. The emphasis is on addressing various threats to the validity of the proposition. 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. Qualitative methods have the added advantage of providing more explanatory information, and help in refining a proposition to better fit the data.

Negative case analysis (Judd et al., 1991) is a very important 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 Sect. 3.1. When performed rigorously, the process involves an exhaustive search for evidence that might logically contradict a generated proposition, revision of the proposition to cover the negative evidence, re-checking 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 earlier, as well as seeking new sources and types of data to help triangulate the findings.

Triangulation (Jick, 1979) 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 analysed 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 (Seaman and Basili, 1998), 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.

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 (1994) 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 as an outlier in terms of both its long duration and the high number of defects reported in the meeting. This case also involved a set of organizationally distant inspection participants. The unusual values for meeting 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.

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, is expected to employ to some degree the same instruments, measures, and procedures as the original study [see the discussion by Andy Brooks et al. (2007), this volume], 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 analysed 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* (Lincoln and Guba, 1985). 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 (1994) 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 unresolved issues in inspection meetings, i.e. issues that eventually had to be referred to someone not at 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.

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 Sect. 3.1, 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 Sect. 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 (Whyte, 1984) 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.

In summary, many qualitative methods for confirming theory are also employed during theory generation. 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. Ideally, any proposition, no matter how generated, is best supported by both qualitative and quantitative evidence.

3.3. Data Modelling and Visualization

In theory, qualitative data can take a number of forms, including pictures and images. However, in practice, most raw qualitative data is in the form of text. While

text has the advantage of being able to fully capture the richness and complexity of the phenomena being studied, it also has some drawbacks. First, text is linear in the sense that only one passage can be read at a time, so concepts that are non-linear or spatial can be difficult, cognitively, to capture by reading. Second, text is often more voluminous than is necessary to express a concept. "A picture is worth a thousand words" is sometimes very, very true. Finally, it can be difficult to visually identify what parts of a textual dataset might be related to other parts without some visual clues.

For all these reasons, visual modelling is often used in qualitative analysis for several purposes. Diagrams of different types are often used as a mechanism for presenting and explaining findings. In writing up qualitative work, using a diagram can often save a lot of space when a concept is more succinctly summarized graphically than textually. But diagrams also serve as a useful mechanism for the analysis task itself. Graphical representations of data often help the researcher to organize concepts and to reveal relationships and patterns that are obscured by volumes of textual data. This is similar and analogous to the use of graphs and charts when presenting quantitative results and data. Although there are numerous types of diagrams that can be useful in various ways in qualitative analysis, we will discuss two: matrices and maps (Dey, 1993) [called "networks" in Miles and Huberman (1994)].

Matrices are especially useful when the data comes from a series of distinct cases (i.e. sites, interviewees, episodes, etc.). In such a study, the researcher creates a matrix in which the rows are cases and the columns are variables of interest. For example, suppose a study has been conducted consisting of interviews with managers of a variety of software development projects. One useful technique to check the representativeness of the data is to create a matrix of characterization information on the cases from which data has been collected. The columns of the matrix would include such characteristics as project size, application domain, experience of the development team, etc. Filling in the cells of such a matrix for each case studied is a useful exercise and gives the reader feedback on what background information is missing, and what types of projects are missing from the sample.

Augmenting such a matrix with more columns representing emerging constructs (i.e. codes or categories) is also a useful analysis technique. For example, suppose in the previous example that many of the interviewees talked about development team meetings, and this topic emerged as an important issue in the study. In the (very simplified) matrix excerpt shown in Fig. 4 (from a fictitious study), we see that the first few columns contain characterizing information on the cases, while the last column contains passages that have been coded under "meetings." Organizing the data in this way clearly shows that the implications of development meetings are very different for small projects than for medium projects. This insight might not have been evident if the data analysis had relied solely on coding the textual data. It's usually advisable to use an electronic spreadsheet to create analysis matrices in order to take advantage of searching and sorting capabilities.

Maps, or basic shapes-and-lines diagrams, are also useful for sorting out concepts and relationships during qualitative analysis (Dey, 1993). Such maps are

Case	Project Size	Application Domain	Experience of Developers	Meetings
1	huge	banquing	mixed	"We spend way too much time in meetings"
2	small	banquing		"We try to touch base with the whole team as often as we can"
3	small	aerospace	low	"The daily briefings are really useful, although some people say it interrupts their 'real' work"
4	large		high	"We would all be so much more productive if we could somehow get rid of meetings"
5	medium	communications	high	"People don't like to come to meetings, but I guess most of them are useful"

Fig. 4 An example matrix

particularly effective at expressing complex concepts in much less space than one is able to do with text alone. The format and symbols used in maps are limited only by imagination; there are no rules governing them. There are, however, a few guidelines that help make maps meaningful to the reader and useful to the researcher. First, maps quickly lose their effectiveness if they become too complicated. If it takes more space to explain how to read and interpret the map than it would have to textually explain the concept depicted in the map, then the map is not useful. While shapes and lines can be uninspiring, their simplicity makes them ideal as a tool to illuminate complex concepts. On the other hand, the researcher must take care to clearly and consistently define the meanings of both the shapes and lines (and any other symbols used in the map). Because these symbols are so simple, they can also be used in multiple ways, and it is tempting to use them in multiple ways in the same diagram. So one must define, for a particular map, whether the lines connecting shapes (i.e. concepts) signify causal relationships (e.g. the presence of one concept causes the presence of the other), or temporal relationships (e.g. one concept precedes another), or contextual relationships (e.g. the two concepts tend to occur in similar contexts), etc.

Despite the need for simplicity, it is possible to include more than simple shapes and lines in a map. Of course, different shapes can be used to denote different types of concepts (e.g. aggregate concepts) (Dey, 1993). The thickness of a line can denote the strength of a relationship, or the weight of evidence supporting it. Colours and patterns can also be used to convey different meanings. Textual annotations, within reason, are also usually needed to label elements on a map.

Miles and Huberman (1994) devote much of their book on analysis to the development of different types of diagrams, and a very large number of examples and variations are explained there. Many of them are similar in appearance and concept

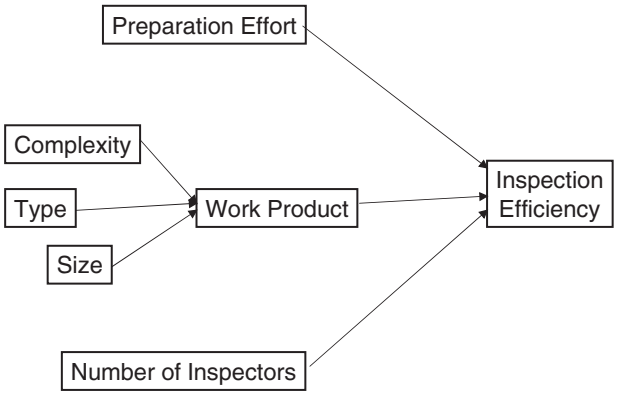

Fig. 5 A causal network showing hypothesized causal relationships

to diagramming techniques used in software development (e.g. control flow diagrams, statecharts, process models, class diagrams). These are especially appealing for software engineering studies because they are already familiar to our community.

While maps can be used for a variety of analysis tasks, one specific use is particularly handy when the qualitative work is exploratory, and intended to lay the groundwork for further empirical work. A good map of concepts and relationships can serve as a research plan for follow-up studies by defining the concepts (i.e. shapes) that need to be developed in further exploratory studies, and the hypotheses (i.e. relationships represented as lines) upon which further confirmatory work can be based. One version of this type of map is the causal network (Miles and Huberman, 1994), a simple example of which is shown in Fig. 5, which identifies factors affecting the efficiency of a software inspection. Such a map can be annotated to show the hypothesized (or tested) strength of the relationships and references to supporting evidence (e.g. identifiers for informants or coded segments).

Creating visual models of qualitative data, and the findings resulting from that data, is a very useful tool for qualitative researchers. Modelling is useful in two ways: during analysis to sort out ideas and relationships; and during presentation as a way to convey findings to the reader. Modelling can be seen as a form of data reduction because diagrams simply take up less space, and are more quickly scanned and digested, than text. They also depict insights arising from the data that are difficult to express succinctly in words.

3.4. Quantification of Qualitative Data

In many studies, it is appropriate to allow the analysis to iterate between quantitative and qualitative approaches. There are several ways to quantify some parts of a body of qualitative data. Such quantification is usually preceded by some preliminary

qualitative analysis in order to make sense of the main categories in the data. It is often also followed by further qualitative analysis to make sense of the quantitative findings, which then leads to further quantitative analysis or re-analysis, and so on.

The most straightforward way to quantify qualitative data is simply to extract quantifiable pieces of information from the text. This is often also called coding, but must be distinguished from the types of coding related to the grounded theory approach, discussed in Sect. 3.1.

To understand the data transformation that takes place during this type of coding, we need to address a common misconception about the difference between quantitative and qualitative data. 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 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:

[Respondent] 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 subjective data did not make it more objective, although the quantitative form may appear less subjective.

When coding is performed on a set of qualitative data, the measurement scale of the resulting quantitative data is determined by the nature of the data itself, and is not restricted by the fact that it was derived from qualitative data. For example, in the "num_participants" example, above, the quantitative variable turned out to be on an absolute scale. But in the "complexity" example, the variable is ordinal.

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? This is why, as mentioned earlier, preliminary qualitative analysis of the data to be coded is important in order to sort out the use of language and the nuances of the concept being described.

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.). One way to resolve such discrepancies is to decide that one subject (or data source) is more reliable than another. Miles and Huberman (1994) 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. This assumption was used to resolve most of the discrepancies.

Another approach to quantification of qualitative data is content analysis (Holsti, 1969). Content analysis, originally developed for the analysis of human communication in the social sciences, is defined in various ways, but for our purposes can be described as an analysis method based on counting the frequency of occurrence of some meaningful lexical phenomenon in a textual data set. This technique is applicable when the textual data can be divided into cases along some criteria (e.g. different sites or respondents). In any particular application of content analysis, counting rules must be defined that make sense given the nature of the data and the research goals. This is why preliminary qualitative analysis is necessary, to determine the "nature of the data." Counting rules can take several forms, e.g.:

- Counting the occurrence of particular keywords in each case and then correlating (statistically or more informally) the counts with other attributes of the cases
- Counting the number of cases in which certain keywords occur and then comparing the counts of different keywords, or comparing the set of cases containing the keyword to those that do not
- Counting the occurrence of one keyword in proximity to a second keyword, and then comparing that count to the number of occurrences of the first keyword without the second keyword

There are numerous other variations on this theme. Note that the first example above only yields meaningful results if one can assume that the frequency of use of a particular word or phrase somehow indicates its importance, or the strength of opinion about it or some other relevant characteristic. This is often not a reasonable assumption because it depends too much on the speaking and writing style of the

sources of the case data. A good example of the use of content analysis is Hall and Rainer's work (with others), in particular (Rainer et al., 2003) and (Rainer and Hall, 2003). Holsti (1969) provides a good reference on content analysis as used in the social sciences.

4. Conclusions

The focus of this chapter 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. Nearly any software engineering issue is best investigated using a combination of qualitative and quantitative methods. Some of the more common mixed method research designs include the following:

- Qualitative data can be used to illuminate the statistical results employed to test a hypothesis. This allows the researcher to go beyond the statistics to help explain the causal relationships revealed by the quantitative results.
- When differences between subjects are an important part of the study design, quantitative measures of individual performance can be augmented with qualitative interview data that helps explain differences in performance, as well as may identify other relevant differences that were not measured.
- In studying a new process or technique, qualitative data from an early observation study of groups using the technique can be used to identify relevant variables to be measured in a subsequent experiment to evaluate the performance of the process or technique.
- Initial qualitative data, from interviews or document analysis, can serve as a starting point for a case study by both setting the context for the researchers as well as identifying important issues and variables for the study.

Finally, it should be noted that there are software packages on the market that facilitate coding and other types of qualitative analysis [see Miles and Huberman (1994), appendix, for an overview of qualitative analysis software]. Space does not permit a full discussion of software tools, but one commonly used application is NVivo™ from QSR International.[2] NVivo aids the researcher in organizing, coding, and grouping textual data, in defining and maintaining links between different pieces of data, and in developing visual models of the data and of findings.

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 analysing as much data of as many different types as possible. Qualitative data is richer than quantitative data, so using qualitative methods increases the

[2] http://www.qsrinternational.com/

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.

References

Barley SR (1990) The Alignment of Technology and Structure through Roles and Networks. Administrative Science Quarterly 35:61–103.

Brooks A, Roper M, Wood M, Daly J, Miller J (2007) Replication's Role in Software Engineering, this volume.

Creswell JW (1998) Qualitative Inquiry and Research Design: Choosing Among Five Traditions. Sage Publications, Thousand Oaks.

Dey I (1993) Qualitative Analysis: A User-Friendly Guide. Routledge, New York.

Eisenhardt KM (1989) Building Theories from Case Study Research. Academy of Management Review 14:532–550.

Gilgun JF (1992) Definitions, Methodologies, and Methods in Qualitative Family Research, in *Qualitative Methods in Family Research*. Sage Publications, Thousand Oaks.

Glaser BG, Strauss AL (1967) The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine Publishing Company, Somerset, NJ, USA.

Guindon R, Krasner H, Curtis B (1987) Breakdowns and Processes During the Early Activities of Software Design by Professionals, in *Empirical Studies of Programmers*, second workshop, Gary Olsen, Sylvia Sheppard, and Elliot Soloway, eds., 65–82, Ablex Publishing, Greenwich, CT, USA.

Hackos JT, Redish JD (1998) User and Task Analysis for Interface Design. Wiley, New York.

Holsti OR (1969) Content Analysis for the Social Sciences and Humanities. Addison-Wesley, Menlo Park.

Jick T (1979) Mixing Qualitative and Quantitative Methods: Triangulation in Action. Administrative Science Quarterly 24(4):602–611.

Judd CM, Smith ER, Kidder LH (1991) Research Methods in Social Relations, sixth edition. Harcourt Brace Jovanovich, Fort Worth.

Kontio J, Bragge J, Lehtola L (2007) The Focus Group Method as an Empirical Tool in Software Engineering, this volume.

Lethbridge T, Sim SE, Singer J (2005) Studying Software Engineers: Data Collection Techniques for Software Field Studies. Empirical Software Engineering: An International Journal 10(3):311–341.

Lincoln YS, Guba EG (1985) Naturalistic Inquiry. Sage Publishing, Thousand Oaks.

Lutters WG, Seaman CB (2007) The Value of War Stories in Debunking the Myths of Documentation in Software Maintenance. Information and Software Technology 49(6): 576–587.

Miles MB, Huberman AM (1994) Qualitative Data Analysis: An Expanded Sourcebook, second edition. Sage Publishing, Thousand Oaks.

Orlikowski WJ (1993) CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development. MIS Quarterly 17(3):309–340.

Orlikowski WJ, Baroudi JJ (1991) Studying Information Technology in Organizations: Research Approaches and Assumptions. Information Systems Research 2(1):1–28.

Parra A, Seaman C, Basili V, Kraft S, Condon S, Burke S, Yakimovich D (1997) The Package-Based Development Process in the Flight Dynamics Division. Proceedings of the Twenty-second Software Engineering Workshop, NASA/Goddard Space Flight Center Software Engineering Laboratory (SEL), Greenbelt, MD, USA.

Perry DE, Staudenmayer NA, Votta LG (1994) People, Organizations, and Process Improvement. IEEE Software 11(July): 36–45.

Rainer A, Hall T (2003) A Quantitative and Qualitative Analysis of Factors Affecting Software Processes. Journal of Systems and Software 66:7–21.

Rainer A, Hall T, Baddoo N (2003) Persuading Developers to 'Buy Into' Software Process Improvement: Local Opinion and Empirical Evidence. Proceedings of the International Symposium on Empirical Software Engineering (ISESE), IEEE, Los Alamitos, CA, USA.

Seaman CB (1999) Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering 25(4):557–572.

Seaman CB, Basili VR (1998) Communication and Organization: An Empirical Study of Discussion in Inspection Meetings. IEEE Transactions on Software Engineering 24(7):559–572.

Sharp H, Robinson H (2004) An Ethnographic Study of XP Practice. Empirical Software Engineering 9:353–375.

Shneiderman B (1998) Designing the User Interface: Strategies for Effective Human-Computer Interaction, third edition. Addison-Wesley, Reading, MA, USA.

Singer J (1998) Practices of Software Maintenance. Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, pp. 139–145.

Taylor SJ, Bogdan R (1984) Introduction to Qualitative Research Methods. Wiley, New York.

von Mayrhauser A, Vans AM (1996) Identification of Dynamic Comprehension Processes During Large Scale Maintenance. IEEE Transactions on Software Engineering 22(6):424–437.

Whyte WF (1984) Learning from the Field: A Guide from Experience. Sage Publications, Beverly Hills.

Yin RK (1994) Case Study Research: Design and Methods. Sage Publications, Newbury Park, CA, USA.