Software Anthropology: Performing Field Studies in Software Companies

Timothy C. Lethbridge

University of Ottawa

Susan Elliott Sim

University of Toronto

Janice Singer[1]

National Research Council Canada

[1] The authors appear in alphabetical order and contributed equally.

Abstract[2]

There are many things to be learned about software engineering by working directly with software practitioners in companies. In this paper, we present an overview of some of the techniques for performing such field studies, focusing on such issues as: What kind of problems can be addressed by field studies? What techniques are available for gathering data and analysis of the results? In what situations are these techniques suitable? What difficulties might the software researcher encounter when performing field studies? We illustrate the discussion with examples from our own work as well as numerous studies reported in the literature.

Index Terms–Field Studies, Work Practices, Empirical Software Engineering

[2] This work is supported by NSERC and sponsored by the Consortium for Software Engineering Research (CSER).

I. INTRODUCTION

Software engineering is a labor-intensive activity. Thus to improve software engineering practice, a considerable portion of our research resources ought logically to be dedicated to the study of *humans* as they create and maintain software.

How can such studies be performed? In this paper we will look at one approach[3].: a set of techniques that we collectively call *software anthropology* because it involves observing and analyzing the day-to-day work of software engineers in the field.

We define software anthropology to be the use of field study techniques in industrial software settings whereby researchers study any aspect of software engineering. Conventional anthropology has two branches that look at different aspects of a group of subjects. Cultural anthropology seeks to understand how people live or lived by looking at their interactions and social systems. In the case of software engineering, this means studying the work practices of individuals and teams during such activities as development and maintenance. Physical anthropology, on the other hand, seeks to understand how people live or lived by looking at their artifacts, such as tools and crafts. In the case of software engineering, this means studying the tools used and the products

[3] See Zelkowitz & Wallace[39] for an overview of several different approaches.

of the those tools such as documentation, source code, etc. It is also possible to combine approaches and look at what are called socio-technical aspects of the group.

While the idea behind software anthropology is borrowed from anthropology, the techniques have been primarily adapted from fields such as sociology, psychology, and human-computer interaction. In particular, these methods rely heavily on the use of field study techniques. Field study techniques are a group of methods that can be used individually or in sets to understand different aspects of real world environments.

The results of software anthropology can be applied when one has several different types of research goals. Firstly, they can be used to derive requirements for software tools and environments. For example, we have performed field studies where not only have we learned about software design and maintenance, but also successfully applied our results to tool design and integration [29][30][31]. Secondly, the results can be used to improve software engineering work practices. For example one of us (Sim) was able to make useful recommendations following a study of how newcomers adapt to a software team [28]. Thirdly, analysis of the results can yield new theories or hypotheses that can then be subjected to formal experimental validation [24] [31].

This latter point illustrates the major difference between software anthropology and formal experimentation, the most important alternative, but complementary, approach. In formal experimentation, researchers define and test hypotheses that they desire to confirm or reject about software engineering processes and products. Field study techniques *can* be used to test hypotheses, however hypothesis testing is not a prerequisite to using field study techniques. A related distinction can be made with respect to the environment in which the techniques are used: Although field study techniques are usually used in the field (*in vivo*) and experimental techniques are normally employed in the laboratory (*in vitro*), this need not be the case. Certain techniques lend themselves to deployment in a particular location, but they are not restricted to these locations.

In this paper, we will discuss a number of data collection techniques suitable for use in software anthropology such as observation, interviews, questionnaires, logs of time and of tool usage, and archival research. These methods are described in Section II along with an explanation of their advantages and disadvantages. In Section III, we explain how the data collection techniques can be applied in software anthropology. Finally, some of the practical and logistical considerations surrounding a field study are described in Section IV.

II. Data Collection Methods

When studying work practices it is important to obtain accurate and reliable information about how work is actually done. Interviews and questionnaires are the most straightforward instruments, but the data they produce typically present an incomplete picture. For example, assume your goal is to assess which programming language features are most error-prone: A developer may be able to give you general opinions and anecdotal evidence about this; however the researcher would obtain far more accurate and quantitative information by recording and analyzing the developer's work practices: their efforts at repeatedly editing and compiling code. Methods such as think-aloud protocols and work diaries are used for this type of research. In fact, to learn about different aspects of a phenomenon, it is often necessary to use multiple data collection methods: In effect one uses different sources of data to *triangulate* the work practice.

In the remainder of this section, we discuss the field study techniques listed in Table 1. Each technique is categorized according to the degree of human contact it requires. First degree contact requires direct access to a subject population. Second degree contact requires access to subjects' environment as they work, but not necessarily subjects. Finally, third degree contact requires access to work artifacts, such as source code or documentation[4].

Each technique we present is followed by a list of advantages and disadvantages, and is illustrated, where possible, using an example taken from the software engineering

[4] Second degree contact is distinguished from third degree contact in that second degree contact requires data acquisition during work, while third degree contact requires only work artifacts and has no requirements with respect to when data is acquired.

literature. The example is given in the context of the questions the researchers were trying to answer and how the particular technique allowed them to do so.

First Degree	Brainstorming Surveys Interviews Questionnaires System Illustration Work Diaries Think-aloud protocols Shadowing Synchronized Shadowing Participant Observation
Second Degree	Instrumenting Systems Fly on the Wall
Third Degree	Problem Report Analysis Documentation Analysis Analysis of Tool Logs

Table 1: Data Collection Techniques Suitable for Field Studies of Software Engineering

All the methods, from the simplest to the most complex, need to be applied with care and respect. First, questions and forms must be crafted carefully to ensure that the data collected is meaningful. A poorly-worded question results in ambiguous responses that cannot be interpreted or analyzed. Second, subjects in the study must be treated with respect because they are, first and foremost, human beings in social situations. Researchers need to be sensitive to the disruptions their presence can cause. These “researchers effects” are further discussed in subsection A on First Degree Techniques.

The set of available methods is constantly evolving. A good place to look for new data gathering and analysis methods is in the human-computer interaction literature (e.g. [14][32]); there is much in common between trying to improve a software system so users can use it more effectively, and trying to improve software engineering practices.

A. First Degree Techniques: Direct Contact

The first five techniques listed in Table 1 are what we call *inquisitive* techniques, while the remaining three are *observational*. Each type is appropriate for gathering different types of information from software engineers.

Inquisitive techniques allow the experimenter to obtain a general understanding of the maintenance process. They are probably the only way to gauge how enjoyable or motivating certain tools are to use or certain activities to perform. However, they are often subjective, and additionally do not allow for accurate time measurements.

Observational techniques provide a real-time portrayal of the studied phenomena. However, it is more difficult to analyze the data, either because it is dense, or because it requires considerable knowledge to interpret correctly. Observational techniques can be used at randomly chosen times or when a software engineer is engaged in a specific type of activity (such as whenever she is using a debugger). Observational techniques always run the risk of changing the process simply by observing it; the Hawthorne effect was first identified when a group of researchers found that output was not related to environmental conditions as expected, but rather to whether or not workers were being observed [23]. Careful consideration of this effect is therefore warranted in implementing the research and explaining its purpose and protocol to the research participants.

1) Brainstorming

In a brainstorming session, several people get together and focus on a particular issue. The idea is to ensure that discussion is not limited only to 'good' ideas or ideas that make immediate sense; i.e. to work together to uncover as many ideas as possible. This technique usually works best with a moderator because the moderator can focus the group when it is drifting and additionally motivate the group when it is floundering. Two important side benefits of brainstorming are that it can introduce the researchers and participants to each other and give the participants more of a sense of being involved. Conducting research in field environments is often stressful to the research participants. They are more likely to be willing subjects if they feel comfortable with the researchers and feel they are partners in research that focuses on issues that they consider to be important.

Bellotti, and Bly [1] used brainstorming during an initial meeting with a product design group. The goal of their research was to study the process of mobile collaboration among design team members. The brainstorming meeting was held to identify problems and possible solutions as seen by the team. This meeting gave the researchers an initial understanding of the team's work and additionally let the researchers know how existing technology was either supporting or inhibiting the work. A nice side effect of the meeting was that it gave the researchers an entry point for communication about the design process with their colleagues in the design department at Apple.

Advantages: Brainstorming generates ideas that can lead to testable hypotheses. It is an excellent method for finding out what is important or particularly salient to your subject population. Brainstorming works best when you are starting a project and looking for ideas.

Disadvantages: Even with a moderator, brainstorming can be too unfocused. People can be shy in a group and not say what they really think. Just because a subject population focuses on a particular issue, it does not mean that it occurs frequently or even that it is a part of their daily work. It is often hard to schedule a brainstorming session with the busy schedules of software engineers.

2) *Interviews and Questionnaires*

In this subsection, the strengths and drawbacks that are common to interviews and questionnaires will be discussed, followed by an examination of the characteristics of each method. Each technique will be described in the following two subsections.

Both interviews and questionnaires are centered around asking a series of questions and the resulting data is respondent's answers to these inquiries. The questions can be closed-ended, i.e. multiple-choice, or they can be open-ended, i.e. long-answer. Therefore, to implement this method correctly, questions and forms must be crafted carefully to ensure that the data collected is meaningful [8]. A poorly worded question results in ambiguous responses that cannot be interpreted or analyzed. If time permits, it makes sense to pilot test the questions or forms and then redesign them as you see which questions really attack the pertinent issues.

In order to generate good statistical results from a survey, a sample representative of the population of interest is needed. This requirement is particularly difficult in software engineering because we lack good demographic information about the population of developers and maintainers. However, this drawback should not prevent us from using surveys to study work practices, particularly when the problem or population is small and well-defined.

Advantages: People are familiar with answering questions, either verbally or on paper, and as a result they tend to be comfortable and familiar with this data collection method. Participants also enjoy the opportunity to talk about their work. Surveys tend to be relatively quick and cost-efficient. Questionnaires, in particular, can easily collect data from a large number of respondents in geographically diverse locations.

Disadvantages: Surveys, in any form, rely on respondents' self reports of their behaviors or attitudes. This dependency can bias the results in a number of ways. People are not perfect recorders of events around them; they remember events that are meaningful to them. For instance in one of our questionnaires, subjects reported that reading documentation was a time-consuming aspect of their job, but in 40 hours of observation, we hardly saw anyone doing so.

3) Interviews

Face-to-face interviews involve at least one researcher talking, in person, to at least one respondent at a time. An interview with a number of respondents is called a focus group. Normally, a fixed list of carefully worded questions forms the basis of the interview. Depending on the goal of the study, respondents may be encouraged to elaborate on areas and deviate slightly from the script.

Telephone interviews are the middle ground between face-to-face interviews and questionnaires. You have the interactivity of an interview, but at the cost of a phone call rather than traveling to another site. They are not as personal as a face-to-face interview, yet they still provide the researcher with opportunities to clarify questions and further probe interesting responses. Although this technique is popular in opinion polling and market research, it is little used in empirical software engineering.

Interviews have been used in many studies because they fit well with many types of inquiries. We have used interviews in longitudinal studies as an aid in understanding how newcomers adapt to a development team and software system [28]. We interviewed newcomers once every three weeks over a number of months to track their progress as maintenance team members. Since this was an exploratory study, the questions were open-ended

Curtis et al. [4]used interviews to study the design process used on nineteen different projects at various organizations. Their research design was predicated on a layered model of behaviour that takes into account an individual's context when analyzing their behaviour. Consequently, they interviewed personnel from three different levels of the projects, systems engineers, senior software designers and the project manager. The researchers conducted 97 interviews, which resulted in over 3000 pages of transcripts of the audio recordings. They found three key problems common to the design processes: communication and coordination breakdowns, fluctuating and conflicting product requirements, and the tendency for application domain knowledge to be located in individuals across the company. They characterized the problems at each level of the model.

Advantages: Interviews are highly interactive. Researchers can clarify questions for respondents and probe unexpected responses. Interviewers can also build rapport with a respondent to improve the quality of responses.

Disadvantages: Interviews are time and cost inefficient. Contact with the respondent needs to be scheduled and at least one person, usually the researcher, needs to travel to the meeting. Data from interviews tend to be qualitative and as a result, needs to be transcribed and/or coded.

4) Questionnaires

Questionnaires are sets of questions administered in a written format. These are the most common field method because they can be administered quickly and easily. Unfortunately, many poorly-designed questionnaires have been created. Many people think they know how to design a questionnaire because they have filled out (poorly-designed) questionnaires in the past. Attention needs to be paid to the wording of the questions, the layout of the forms, and the ordering of the questions.

Iivari used a paper-based survey to test nine hypotheses about factors affecting CASE tool adoption in 52 organizations in Finland [9]. The author contacted organizations who had purchased CASE tools and surveyed key information systems personnel about the use of the tool. Companies and individuals were more likely to use CASE tools when adoption was voluntary, the tool was perceived to be superior to its predecessor(s) and there was management support.

Sim et al. used a web-based questionnaire to study source code searching behaviors [27]. We solicited respondents from seven USENET newsgroups from the `comp.*` hierarchy to complete a questionnaire at a given web address. The questionnaire used a two-page format. The first page informed the participants of the goals of the study and their rights and the second page displayed the questions. Using a combination of open- and closed-ended questions, we identified a set of eleven search archetypes.

Advantages: Questionnaires are time and cost effective. Researchers do not need to schedule sessions with the SEs to administer them. They can be filled out when an SE has time between tasks, for example, waiting for information or during compilation. Paper form-based questionnaires can be transported to the respondent for little more than the cost of postage. Web-based questionnaires cost even less since the paper forms are eliminated and the data are received in electronic form.

Disadvantages: Since there is no interviewer, ambiguous and poorly-worded questions are problematic. Even though it is relatively easy for SEs to fill out questionnaires, they still must do so on their own and may not find the time. Thus, return rates can be relatively low which can adversely affect the representativeness of the sample. Responses tend to be more terse than with interviews.

5) System Illustration

During system illustration, subjects illustrate, through drawing, some aspect of their work. For instance, SEs may be asked to draw the important architectural clusters of their system. As another example, SEs may be asked to draw a data flow or control flow diagram. As an orthogonal usage, SEs may be asked to draw a physical map of their environment, pointing out who they talk to and how often.

Another usage of system illustration involves researchers creating the diagrams and then asking subjects to either confirm or disconfirm the information contained therein.

In one of our pilot trials, we collected system maps from all members of the researched group. Additionally, as we followed two newcomers to a system, we had them update their original system maps on a weekly basis. We gave them a photocopy of the previous week's map, and asked them to either update it or draw a new one. The newcomers almost exclusively updated the last week's map.

Originally, our instructions to the study participants were to ‘draw their understanding of the system.’ These instructions turned out to be too vague. Some participants drew data flow diagrams, some drew architectural clusters, others listed the important data structures and variables, etc. Not surprisingly, the manager of the group subsequently noted that the system illustrations reflected the current problems on which the various software engineers were working.

We learned from this exercise that for illustration data to be useful, it is important to specify to the greatest extent possible the type of diagram required. It is next to impossible to compare diagrams from different members of a group if they are not drawing the same type of diagram. Of course, this limits researchers in the sense that they will not be getting unbiased representations of a system. Specifying that data-flow diagrams are required means that SEs must then think of their system in terms of data-flow.

Advantages: System illustrations provide an accurate portrayal of the user’s conception of his mental model of the system. System illustrations are easy to collect and require only low-tech aids (pen and paper).

Disadvantages: System illustrations are hard to interpret, especially if you do not have domain knowledge about the system. Some SEs are reluctant to draw.

6) Work Diaries

Work diaries require respondents to record various events during the day. It may involve filling out a form at the end of the day, recording specific activities as they occur, or noting the current task at a pre-selected time. These diaries may be kept on paper or on the computer. Paper forms are adequate for recording information at the end of the day. A computer application can be used to prompt users for input at random times.

Over a series of studies, Perry et al. developed a paper-based time diary designed to record how SEs' spent their time over the course of a day [19]. The final form reflected the researchers' model of the SEs' work patterns and was designed to facilitate coding of the data. Developers tended to spend only about half their time coding, the remaining time was spent on design, re-work, administrative tasks, etc.

Jørgensen randomly selected software maintainers and asked them to complete a form to describe their next task [12]. These reports were used to profile the frequency distribution of maintenance tasks. Thirty-three hypotheses were tested and a number of them were supported. For example, programmer productivity (lines of code / unit time) is predicted by the size of the task, type of the change, but it is not predicted by maintainer experience, application age, nor application size.

Advantages: Work diaries obtain better self-reports of events because they record activities on an ongoing basis rather than in retrospect. Random sampling of events gives researchers a way of understanding how SEs spend their day without a great deal of observation or shadowing.

Disadvantages: Work diaries still rely on self-reports and ones that require participants to recall events still have problems with accuracy. Another problem with work diaries is that they can interfere with respondents as they work. For instance, if software engineers have to record each time they go and consult a colleague, they may consult less often. They may also forget or neglect to record some events and may not record at the expected level of detail.

7) Think-aloud protocols

In think-aloud protocol analysis [7] [16], experimenters ask participants to think out loud while performing a task. The task can occur naturally at work or be predetermined by the experimenter. As SEs sometimes forget to verbalize, experimenters may occasionally remind them to continue thinking out loud. Other than these occasional reminders, experimenters do not interfere in the problem solving process. Think-aloud sessions are generally two hours or less.

Think-aloud protocol analysis is most often used for determining or validating a cognitive model as SEs do some programming task. For a good review of this literature, see Von Mayrhauser and Vans [34].

Von Mayrhauser and Vans [32] asked professional SEs to think aloud as they performed a maintenance task for their job that necessitated program comprehension. Both SEs involved in the experiment chose debugging sessions. The think-aloud protocols were coded to determine if subjects were adhering to the Integrated meta-model of program comprehension (defined by von Mayrhauser & Vans). They found evidence for usage of this model. Because the model usage was confirmed, von Mayrhauser and Vans were able to suggest tool requirements for software maintenance environments.

As another example of think-aloud protocol analysis, Sen [26] asked student subjects to think out loud while they drew Entity-Relationship Diagrams during a problem solving session. Before the experimental session, Sen had participants practice thinking out loud by having them solve simple math problems. Sen used the think-aloud protocols to draw cognitive maps for each of his nine subjects. These cognitive maps supported Sen's theory of the use of opportunism in the design reuse process. The results from this experiment were used to define an architecture to support reuse in design.

Advantages: Relatively easy to implement. Additionally, it is possible to implement think-aloud protocol analysis with manual record keeping [16] obliterating the need for transcription. This technique gives a unique view on the problem solving process and additionally gives access to mental model. It is an established technique.

Disadvantages: Think aloud protocol analysis was developed for use in situations where an experimenter could map out the entire problem space. It's not clear how this method translates to other domains where it is impossible to know a priori what the problem space is. Even using manual record keeping, it is difficult to analyze think-aloud data. Additionally, it can be extremely time intensive.

8) Shadowing

In shadowing, the experimenter follows the participant around and records their activities. Shadowing can occur for an unlimited time period, as long as there is a willing participant.

We have implemented shadowing in our work in two ways [30]. First, one experimenter took paper and pencil notes to indicate what the subject was doing and for approximately how long. This information gave us a good general picture of the work habits of the SEs. We also developed a method we call *synchronized shadowing*. Here we used two experimenters and, instead of pencil and paper, used two laptop computers to record the SEs actions. One of us was responsible for ascertaining the subjects' high level goals, while the other was responsible for recording their low-level actions. We used pre-defined categories (Microsoft Word macros) to make recording easier. We are currently analyzing this data to see if certain goals are followed by certain actions – we term such sequences *work patterns*.

To understand the build process at a large telecommunications company, Wolf & Rosenblum [37]also implemented a shadowing procedure. They observed an SE as he went about building the software. Wolf and Rosenblum began by defining a taxonomy of events that could occur during the build process. This taxonomy was then translated into specific categories of possible events. Then each time an event occurred, the observer used a simple log sheet to give the event an identifier, note the type of event, the date and time the event occurred, the names of the subsystems affected, and the names of the people involved. The data was analyzed by entering it into a relational database and subsequently performing queries on it. For example, Wolf & Rosenblum were able to identify possible problem processes by looking at the lag time between when the Build SE contacted a developer and when the developer answered the query.

Perry, Staudenmayer, & Votta [19] also shadowed SEs as they went about their work. They recorded continuous real-time non-verbal behavior in small spiral notebooks. Additionally, at timed intervals they asked the SEs "What are you doing now?" At the end of each day, they converted the notebook observations to computer files. The direct observations contributed to Perry, et al.'s understanding of the software process. In particular, shadowing was good for observing informal communication in the group setting.

Advantages: Shadowing is easy to implement, gives fast results, and requires no special equipment.

Disadvantages: It is often hard to see what the SE is doing, especially when they are using macros and working quickly. However, for the general picture, e.g., they are now debugging, shadowing does work well. Observers need to have a fairly good understanding of the environment to interpret the SEs' behavior. This can sometimes be offset by predefining a set of categories or looked-for behaviors. Of course, again, this limits the type of data that will be collected.

9) Participant Observervation

In the Participant-Observer method, the researcher essentially becomes part of the team and participates in key activities. Participating in the software development process provides the researcher with a high level of familiarity with the team members and the tasks they perform. As a result, SEs are comfortable with the researcher's presence and tend not to notice being observed.

Participant-Observer was one of the methods used by Seaman and Basili in their studies of how communication and organization factors affect the quality of software inspections [24]. The authors were integrated into a newly-formed development team and the process document for the group included Seaman as a participant in the inspection meetings. Over seventeen months, she participated in twenty-three inspection meetings. From this study, they developed a series of hypotheses on how factors such as familiarity, organizational distance, and physical distance are related to how much time is spent on discussion and tasks.

Porter, et al. also used the participant-observer method [21]. One of the researchers, a doctoral student, joined the studied development team as a means of tracking an experiment's progress, capturing and validating data, and observing inspections. Here the field study technique was used in the service of more traditional experimental methods.

Advantages: Respondents are more likely to be comfortable with a team member and to act naturally during observation. Researchers also develop a deeper understanding of the cognitive aspects of a task after performing them.

Disadvantages: Joining a team is very time consuming. It takes a significant amount of time to establish true team membership. Also, an researcher who becomes too involved may lose perspective on the phenomenon being observed.

B. Second Degree Techniques: Indirect Contact

Second degree techniques require access to SEs' environment as they work, but do not require direct contact between the participant and researcher during data collection. As a result, they require very little time from the SE's and are appropriate for longitudinal studies.

1) Instrumenting Systems

This technique requires "instrumentation" to be built into the software tools used by the SE. This instrumentation is used to record information automatically about the usage of the tools. Instrumentation can be used to monitor how frequently a tool or feature is used, patterns of access to files and directories, and even the timing underlying different activities. This technique is also called system monitoring.

In some cases instrumentation merely records the commands issued by users. More advanced forms of instrumentation record both the input and output in great detail so that the researcher can effectively play back the session. Others have proposed building a new set of tools with embedded instruments to further constrain the work environment [3].

Kay and Thomas instrumented an existing UNIX editor, `sam`, to learn about long-term use of the tool [13]. They wanted to know how power users evolved so they recorded the commands and operators used from within the editor. They found that a small set of commands were used very frequently, another small set of commands were used occasionally, and a large set of commands were almost never used. Commands from the second group tended to phase in and out of favor. From this data, the authors were able to build documentation to assist in a computer-aided instruction system for the editor.

Advantages: System monitoring requires no time commitment from SE. People tend to be very poor judges of factors such as relative frequency and duration of tool use and this method can be used to provide this information accurately.

Disadvantages: It is difficult to analyze data from instrumented systems meaningfully; that is, it is difficult to determine the SE's thoughts and goals from a series of tool invocations. The instruments might tell you what the SE was doing, but it does not direct you to the logic behind that action. This problem is particularly relevant when the working environment is not well-understood or not constrained. For example, SE's often customize their environments by adding scripts and macros (e.g., in `emacs`). One way of dealing with this disadvantage is to play back the events to the SE and ask them to comment.

2) Fly on the Wall

"Fly on the Wall" is a hybrid technique, it allows the researcher to be an observer of an activity without being present by requesting the participants to record themselves. The recordings could be video or audio.

Berlin asked mentors and apprentices at a software organization to audiotape their meetings in order to study how expertise is passed on [2]. She later analyzed these recordings for patterns in their conversations. She found that their discussions were highly interactive in nature, using techniques such as confirmation and re-statement to verify messages. Mentors not only explain features of the system, they also provided design rationale. While mentoring is effective, it is also time-consuming, so Berlin makes some suggestions for documentation and short courses for apprentices.

Walz et al. had assistants videotape team meetings during the four-month design phase of a development project [36]. Researchers did not participate in the meetings and these tapes served as the primary data for the study. The goal of the study was to understand how the team work, goals, and design evolved over a period of four months. Initially the team focused on gathering knowledge about the application domain, then on the requirements for the application, and finally on design approaches. The researchers also found that the team did not record a lot of key information and as a result they re-visited issues that had been settled at earlier meetings

Advantages: This method requires very little time from the participants and is very unobtrusive. Although there may be some discomfort in the beginning, it fades quickly.

Disadvantages: The participants may forget to turn on the recording equipment at the appropriate time and as a result the record may be incomplete or missing.

C. Third Degree Techniques: Analysis of Work Artifacts

Third degree techniques attempt to uncover information about how SEs work by looking their output and by-products. Examples of their output are source code, documentation, and reports. By-products are created in the process doing work, for example work requests, logs and output from configuration management and build tools. These records, or archives, can serve as the primary information source. Sometimes researchers recruit SEs to assist in the interpretation or validation of the data. There are some advantages and disadvantages that are common to all third degree techniques.

Advantage: Third degree techniques require almost no time commitment from SEs.

Disadvantages: The data collected is somewhat removed from the actual development process. Also, it is sometimes difficult to interpret the data meaningfully. The information available may not be sufficient to perform some analyses relevant to research goals.

1) Work Request Analysis

In most large maintenance organizations, modifications to a program are accompanied by a paper trail. An SE is assigned a work request in the form of a problem report for a defect repair or a change request for other maintenance tasks. A single change request or a series of such requests can serve as the data source.

Change requests can be used as the basis for inquiries using first degree techniques. For example, a researcher can review a recently completed work request with the SE to document problems encountered or strategies used along the way.

These requests can be used in a number of ways. Pfleeger and Hatton analyzed fault reports for a system to evaluate the effect of adding formal methods to the development process of an air traffic control system [20]. Each module in the software system was designed using one of three formal methods or an informal method. Although the code designed using formal methods tended to have fewer faults, the results were not compelling even when combined with other data from a code audit and unit testing.

Advantages: A large amount of data is often readily available. The data is stable and non-reactive to researchers.

Disadvantages: Little control over the quantity and quality of information on the work request forms. It is also difficult to gather additional information about requests, especially if they are very old or the SE who worked on it is no longer available.

2) Documentation Analysis

This technique focuses on the documentation generated by SEs. Data from this category can come from many different sources. Documentation includes comments and headers in the program code, as well as any written materials describing the software system. Data collected from these sources can also be used in re-engineering efforts, such as subsystem identification. Other sources include local newsgroups, group e-mail lists, memos, and process documents.

Kemerer and Slaughter wanted to test a model of relationships between types of repairs performed during maintenance and attributes of the modules being changed [14]. Their analysis was based on the change history of the modules which contained information about the module's creation date and author, the function of the module, the SE making the change, and a description of the change. They found support for their regression models and posit that software maintenance activity follows predictable patterns. For example, modules that are more complex, relatively large and old need to be repaired more often, whereas, modules with important functionality tend to be enhanced more often.

Advantages: Documents written about the system often contain conceptual information and present a glimpse of at least one person's understanding of the software system. They can also serve as an introduction to the software and the team. Comments in the program code tends to provide low-level information on algorithms and data. Using the source code as the source of data allows for an up-to-date portrayal of the software system.

Disadvantages: Studying the documentation can be time consuming and it requires some knowledge of source. Written material and source comments may be inaccurate.

3) Analysis of Tool Logs

Some of the information gathered using other techniques can be gleaned from software tools with built-in logs, such as configuration management tools, work assignment tools, and tool servers. This technique can be viewed as a quick-and-dirty version of system instrumentation.

In the Wolf and Rosenblum study mentioned in the section on Shadowing, the authors analyzed the log files generated by the build tools [37]. They developed tools to automatically extract information from relevant events from these files. This data was input into a relational database along with the information gathered from other sources.

Advantages: The data is already in electronic form, making it easier to code and analyze. The behaviour being logged is part of SEs normal work routine.

Disadvantage: Companies tend to use different tools in different ways, so it is difficult to gather data consistently when using this technique with multiple organizations.

III. Applying the Methods

In the previous section, we describe a number of diverse techniques for gathering information in a field study. The utility of data collection techniques becomes apparent when they can help us to understand a particular phenomenon. In this section, we explain how these methods can be used in an empirical study of software engineering. Some of the issues we deal with are: how to choose a data collection method, how to record the data, and how to analyze the data.

A. Designing the Study

The first step in designing any study is to establish a set of well-understood goals, because many of the subsequent design decisions depend on them. The goals should describe the phenomenon being studied and the purpose of the study, i.e. how will the results be used. Field studies in software engineering have had a variety of goals. Some of them try to develop tool requirements by studying SEs [27][29][32]. Others want to make the management of software maintenance easier by characterizing maintenance activities [12][14]. Other studies attempt to understand existing development and maintenance processes, so they can prescribe new ones [19][24][28][37].

The goals of the research drive the formulation of the research questions, which in turn drive the research design, which in turn dictate the choice of data collection technique(s). This relationship between goals and selection of data collection technique is depicted in Figure 1. When the goals are clear, many of the later decisions are simplified. It should be noted that goals are distinct from hypotheses, the latter are formulated at the same time as the research design. As mentioned earlier, a hypothesis is not necessary to perform a field study.

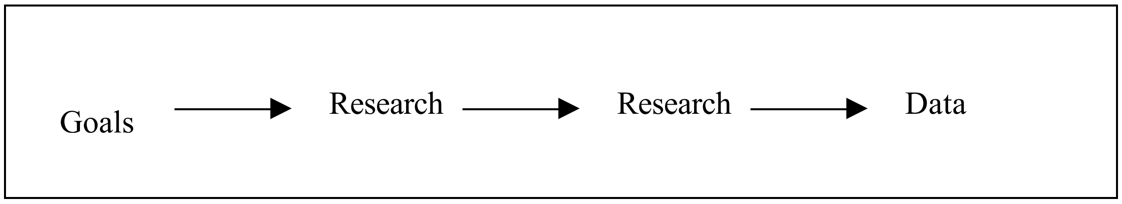

Figure 1: Design Decision Sequence

The goals should be fairly specific, so that a small number of research questions can be derived from them. The questions should be short, so they can be answered easily. There should not be too many questions and the scope of each question should be narrow, lest the study become unmanageable. A study can be built around a single well-defined question.

The research question combined with background knowledge generates a small number of research designs. There are three basic types of studies that can be performed: case study [38], surveys [8], and controlled experiments . Any of these designs can be carried out in the field. If there is little background knowledge and the questions are broad, a case study could be used. On the other end of the continuum, if there is a great deal of background knowledge and the questions are specific, a controlled experiment can be performed. A survey is suited towards quantifying a phenomenon, such as how many programmers there are and what languages they know. Many of the field studies in software engineering tend to be exploratory in nature, because we are still gathering basic knowledge about the people factors surrounding software development and maintenance. As a result, a case study design is commonly used and the study results in a theory or model that can be tested later. As our knowledge base grows, we can employ designs that test these theories or models.

Eisenstadt did a study to characterize particularly difficult bugs [6]. He collected data by posting a request for anecdotes about such bugs on several electronic forums. The research questions were: Why were the bugs difficult to find? How were the bugs found? and What were the root causes of the bugs? Ninety-five anecdotes received in two batches were used to answer these questions. About half the bugs were difficult to find because there was a large gap between where the error occurred and where the error appears, or the bug rendered debugging tools inapplicable. Data gathering, such as using print statements, and hand simulation accounted for 80% of the reported strategies for finding a bug. The two main root cases were memory overwrites and faults in vendor-supplied hardware and software.

In this study, Eistenstadt essentially used a questionnaire to collect data. The first batch of responses was analyzed using bottom-up techniques, such as generating distributions, summary statistics, and grounded analysis. From these results, he generated a set of categories and criteria that he used to analyze the second batch of responses in a topdown manner. This research design permitted easy collection of data from a large number of sources about a phenomenon that was relatively uncommon—very difficult bugs. One drawback of the design is that the respondents may not have reported strategies that are difficult to describe.

The goal of the study was simply to characterize difficult bugs. The research question reflected this goal because they are exploratory in nature. Such questions can be answered using either a case study design or a survey design. Although a questionnaire was used, there are other ways of answering the same research questions using field study techniques. Indeed, these alternate data sources could serve to validate the results found by Eisenstadt. Some other approaches are presented below.

Characteristics of difficult bugs can be studied using work diaries, work requests, or work management tool logs. Use of the latter two sources depends on the information available from the archives. With a modified work diary technique, SEs complete a form after repairing a bug to describe the process along key dimensions. The researcher then interviews SEs about bugs that took a long time to resolve or were rated as very difficult. One advantage of this approach is that it provides information about the relatively frequency of different types of bugs. A disadvantage is the researcher has to deal with uninteresting bugs, such as those that are not difficult.

Another approach it to observe SEs, with or without think-aloud, when they are repairing bugs. These bugs could be chosen by the SE on the basis that they appear to be difficult (based on the work request) or the SE has spent some time working on the bug already and is having little success. Advantages of this approach are that researchers get to see strategies as they are being used and they may also see bugs that are difficult, but not memorable. Disadvantages of this approach is it takes more time to watch SEs repair bugs than to read reports of them, and it may be difficult to co-ordinate observations sessions.

Any of these approaches could have been used to characterize difficult bugs. Depending on the goals of the research and the field site, one option usually is more attractive than the others. When there are a small number of equally good options, all of them can be used to create a more complete picture of the phenomenon. The method chosen by Eisenstadt is appropriate for his goal of acquiring a basic understanding of the bugs. However, if the goal of the study to develop debugging tools, then the observation and think-aloud protocols may be more appropriate. If the goal of the study is to better manage maintenance processes with respect to debugging, then the strategy using problem reports/work diaries coupled with interviews may be more appropriate. Hence, it is important to have well defined goals as the basis of designing a study because they guide important research decisions.

B. Record-Keeping Options

First degree contact generally involves one of the following three data capture methods: videotape, audiotape, or manual record keeping. These methods can be categorized as belonging to several related continua. First, they can be distinguished with respect to the completeness of the data record captured. Videotape captures the most complete record, while manual record keeping captures the least complete record. Second, they can be categorized according to the degree of interference they invoke in the work environment. Videotaping invokes the greatest amount of interference, while manual recording keeping invokes the least amount of interference. Finally, these methods can be distinguished with respect to the time involved in using the captured data. Again, videotape is the most time-intensive data to use and interpret, while manual record keeping is the least time-intensive data to use and interpret.

The advantage of videotape is that it captures details that would otherwise be lost, such as gestures, gaze direction, etc [5]. However, with respect to video recording, it is important to consider the video camera's frame of reference. Videotape can record only where a video camera is aimed. For instance, consider videotaping a software engineer to follow his eye movements. To accomplish this, it is necessary to have coordinated videotaping:

[5] It is often felt that videotaping will influence the participants actions. However, while videotaping appears to initially influence behavior, the novelty wears off quite quickly [11].

one capturing the software engineer's back and computer screen[6]; the other capturing his eye movements as he looks at the screen. Moving the video camera a bit to the right or a bit to the left may cause a difference in the recorded output and subsequently in the interpretation of the data. Another difficulty with videotape is that video formats are generally of far poorer resolution than that of computer screens – thus it is hard to capture enough of what happens on the screen,

Audiotape also allows for a fairly complete record, however details of the physical environment and interaction with it will be lost. Audiotape does allow, however, for the capture of tone. If a subject is excited while talking about a new tool, this will be captured on the audio record.

Manual record keeping is the most data sparse method and hence captures the least complete data record, however manual record keeping is also the quickest, easiest, and least expensive method to implement. Manual record keeping works best when a well-trained experimenter identifies certain behaviors, thoughts, or concepts during the data collection process. Using synchronized shadowing as in our laboratory, in order to attain the benefits of manual record keeping, while at the same time keeping its disadvantages at

[6] System logging of the computer screen may provide an alternative in this situation, but it is still necessary to consider the video frame from the perspective of what data is required.

bay, we used two record keepers for the same session with each record keeper trained to capture different data. This method has worked particularly well for capturing high-level goals, while at the same time capturing low-level actions.

All three data capture methods have advantages or disadvantages. The decision of which to use depends on many variables, including privacy at work, the subject's degree of comfort with any of the three measures, the amount of time accorded for data collection and interpretation, the type of question asked and how well it can be formalized, etc. It is important to note that data capture methods will affect the information gained and the information that it is possible to gain. But again, these methods are not mutually exclusive. They can be used in conjunction with each other.

C. *Coding and Analyzing the Data*

Field study techniques produce enormous amounts of data—a problem referred to as an "attractive nuisance" [16]. The purpose of this data is to provide insight into the phenomenon being studied. To meet this goal, the body of data must be reduced to a comprehensible format. Traditionally, this is done through a process of coding. That is, using the goals of the experiment as a guide, a scheme is developed to categorize the data. These schemes can be quite high level. For instance, a researcher may be interested in noting all goals stated by a software engineer during debugging. On the other hand the schemes can be quite specific. A researcher may be interested in noting how many times `grep` was called in a half-hour programming session. Often, two raters will categorize some overlapping portion of the data to ensure to the greatest degree possible that objective categorization is occurring. Interrater reliability measures quantify the degree to which raters agree on categorizations, and some acceptable level is determined beforehand. Generally disagreements about the coding scheme are resolved before coding continues.

Audio and videotape records are usually transcribed before categorization, although transcription is not necessary. Transcription requires significant cost and effort, and may not be justified for small, informal studies. Having made the decision to transcribe, getting an accurate transcription is challenging. A trained transcriber can take up to 6 hours to transcribe a single hour of tape (even longer when gestures, etc. must be incorporated into the transcription). An untrained transcriber (especially in technical domains) can do such a poor job that it takes researchers just as long to correct the transcript. While transcribing has its problems, online coding of audio or videotape can also be quite time inefficient as it can take several passes to produce an accurate categorization. Additionally, if a question surfaces later, it will be necessary to listen to the tapes again, requiring more time. The issue of finding transcribers is further discussed below under “Staffing” in Section VI.

Once the data has been categorized, it can be subjected to a quantitative or qualitative analysis. Quantitative analyses can be used to provide summary information about the data, such as, on average, how often `grep` is used in debugging sessions. Quantitative analyses can also determine whether particular hypotheses are correct, such as whether high-level goals are stated more frequently in development than in maintenance.

When choosing a statistical analysis method, it is important to know whether your data is consistent with assumptions made by the method. Traditional, inferential statistical analyses allow for tests of statistical significance, which can include characterization of population parameters with error estimates. However, these tests are only applicable in well-constrained situations. The type of data collected in field studies often require nonparametric statistics. Nonparametric statistics are often called "distribution-free" in that they do not have the same requirements regarding the modeled distribution as parametric statistics. Additionally, there are many nonparametric tests based on simple rankings, as opposed to strict numerical values. Finally, many nonparametric tests can be used with small samples. For more information about nonparametric statistics, Siegel and Castellan [25] provide a good overview.

Qualitative analyses do not rely on quantitative measures to describe the data. Rather, they provide a general characterization based on the researchers' coding schemes. For example, after interviewing SEs at 12 organizations, Singer [31] found characteristics common to many of the organizations, such as their reliance on maintenance control systems to keep historical data. Again, the different types of qualitative analysis are too complex to detail in this paper. See [17] for a very good overview.

In summary, the way the data is coded will affect its interpretation and the possible courses for its evaluation. Therefore it is important to ensure that coding schemes reflect the research goals. They should tie in to particular research questions. Additionally, coding schemes should be devised with the analysis techniques in mind. Again, different schemes will lend themselves to different evaluative mechanisms. However, one way to overcome the limitations of any one technique is to look at the data using several different techniques (such as combining a qualitative and quantitative analyses). A triangulation approach will allow for a more accurate picture of the studied phenomena.

As a final note, with any type of analysis technique, it is generally useful to go back to the original subject population to discuss the findings. Subjects can tell researchers whether they believe an accurate portrayal of their situation has been achieved. This, in turn, can let researchers know whether they used an appropriate coding scheme and analysis techniques

IV. Logistical Issues in Studies of Software Anthropology

Since studies of software anthropology are undertaken in the field, many logistical issues have to be resolved. Some logistical issues arise in all research involving human subjects (e.g., scheduling sessions, obtaining ethics approval, etc.). Other logistical issues are unique to field studies (e.g. making contact with organizations, negotiating access to informants, obtaining permission to publish results, etc.). In this section, we describe some of the practical concerns associated with software anthropology. Drawing upon our experience performing such studies, we also give suggestions for dealing with these issues.

A. Establishing Field Research Co-operation with Organizations

The biggest practical problem in studying work practices is obtaining participants. Although it is possible to conduct a study using participants who are solicited individually, it is usually necessary to work with teams within an organization**.** Hence, participation needs to be obtained from organizations.

Finding suitable organizations is the first hurdle. While many researchers or their institutions may have a few companies that are their perennial 'contacts' in industry, software anthropology researchers should give thought to involving companies of several different types to avoid introducing bias. The companies most likely to be willing to participate are those already involved in research – particularly medium to large companies whose primary business is software or computer products. Much harder to penetrate are companies in other industries that develop specialized software or in-house software, for example, banking and health care. In the past, we have experienced considerable frustration finding suitable managers to contact. Our only advice is that unbiased research often requires considerable effort of this type.

Two levels of management must be convinced to participate: Higher management must agree to the involvement of the company as a whole, while first-level managers must agree to the involvement of their teams. In both cases, obtaining commitment can be hard. Management will naturally be concerned about the costs of the research, particularly in terms of time. Researchers have to effectively, but realistically, show that there are benefits to the company which can balance the costs. Some of these benefits are presented below:

1) The results of the research can sometimes help the company to improve its product or process. Work-practices studies can show, for example, the benefits of certain techniques, or highlight training requirements of the staff. Since there is considerable *a priori* uncertainty about the success of such research, this assistance should not be presented as the only benefit. The benefits below may prove more convincing.

2) Students will be exposed to the company and thus may be more inclined to work there when their research is finished. Furthermore, they will have a head-start on the learning curve faced by new employees. In today’s market, this can be very attractive to companies.

3) Cooperation between companies and faculty can result in technology transfer in both directions. Companies absorb knowledge from faculty through presentations or informal discussions. By the same token, faculty absorb knowledge about the corporate environment that they will take back to academia.

4) The research can improve the company’s profile through publicity in the press and in the academic literature.

It is easier to make a case for the above benefits when establishing a long term relationship with a company. We have found companies are more open to anthropological studies when other members of the research team are tackling the company’s engineering problems.

Once a company has established its willingness to participate, it is important to reach agreement on a number of issues. The formality of the agreements varies with the size and duration of the research. A study with a low level of involvement over a short period of time only need casual discussions. A very large project requires more detailed negotiations, particularly if financial support is involved. Sometimes a company will be interested in the project, but reluctant to make certain commitments because it is unfamiliar with empirical software engineering or the proposed methods. In such cases, the researcher should treat educating the organization as part of the negotiation process, so they can proceed as partners in the endeavor. The following are areas where agreements should be established to help ensure the project’s success.

- **Commitment to project**. The first point of mutual agreement should be the level of commitment to the project. What is the expected duration of the project? How much support (e.g. space, time, equipment) is expected from the company? What kind of results or deliverables are expected from the researchers? Agreement on these issues often forms the basis for agreement on other issues below.

- **Access to informants.** Both sides need to agree on how many employees will participate in the study and how much time is required from each employee. Sometimes an organization will find it difficult to provide the personnel required by the ideal research design and some compromise may be necessary.

- **Selection of informants.** A company may want only certain employees to participate, e.g. ones not involved in critical tasks or ones who present a particular point of view. A researcher needs to ensure that the sample participating in the study is representative. However, a researcher does not need to retain complete control over the selection process to obtain an appropriate sample.

- **Confidentiality of data.** Some data need to be kept confidential for corporate reasons; for example a company may not allow highly sensitive information to be taken off-site, for example aggregate source code or defect logs. Other data need to be kept confidential for reasons of experimental ethics: A researcher needs to keep raw data confidential, in order to protect the informants involved. Data that are not confidential for either of the above reasons can serve as the basis for discussions of the next point, publication of results.

- **Publication of results.** It is difficult to predict which results will be sufficiently interesting to publish, particularly before data collection has begun. Understandably, companies are reluctant to give blanket approval to disclosure of information. One solution is to set some ground rules at the beginning, and deal with publications on a case-by-case basis. Although this approach adds a step to the process of writing a paper, it has the benefit of providing researcher swith an opportunity to verify their observations and conclusions. Often, a paper is reviewed for publication by companies at the same time that peer review occurs[7]. We have never had a case where a company rejects a paper outright.

  Another decision to be made is whether or not to identify the organization in the publication. A company may want its contributions acknowledged, or it may not want to be associated with "negative" findings. Also, it may not be possible to publish the identity of the company without compromising the anonymity of the participants. This question can be dealt with in using the same approach described above for results.

[7] If corporate lawyers review a research agreement, they may insist on a lengthy pre-submission approval period (e.g. 90 days) to protect the company. Researchers should be careful not to unwittingly accept such a clause.

A final comment regarding the co-operation of companies: One should keep in mind the possibility of a long term relationship with the company. After going through the effort of establishing a relationship it will likely be useful to extend it either by performing a series of different studies, each building on the previous, or by performing longitudinal studies where software engineers are followed over many years.

B. Establishing Research Co-operation with Individual Participants

After establishing a research relationship with the company, the next step is to establish relationships with individual participants. Whether potential respondents are willing to participate depends on several factors:

- **The type of research**. Being watched is of more concern to most people than, for example filling out a survey. Also, long-term or time-consuming research will likely attract fewer participants.

- **Whether the participants perceive management to be supportive**: We have found it essential that management appear enthusiastic about the work, so that employees feel they are not at risk of being penalized for not getting their ‘regular’ job done while taking time out for the research.

- **Whether the participant perceives some benefit to participation**: Some participants will enjoy taking time away from their daily work; others may be interested in the research for its own sake or because they feel they may gain something from the results.

- **The personality and beliefs of the participants**. We have found some employees are more willing to participate than others. In fact, we have had situations where participants actively dissuade us by saying that the work they are doing would not be interesting enough for us to study. It is important but difficult to guard against these types of biases introduced into the research .

Some difficulties can be overcome if management demonstrates their commitment to the project. One way is to have a group meeting where they introduce the researchers, and explain the nature of the research and its benefits to the company. However for ethical reasons, managers should make it clear that they are not ordering people to participate. In the case of long-term research relationships, management and researchers should report on the project's progress at a similar meeting at least once a year.

An important step before collecting data from participants is obtaining written, informed consent. To do this, we use two forms. The first form is signed by the participant's manager and given to each respondent when their participation is solicited. The document serves as evidence of management support for their participation. The second form is signed by the participant and ensures that respondents understand the nature of the research and their rights.

C. Planning Software Anthropology Field Studies

There are almost as many potential delays in a study of a software engineering project, as there are for a software engineering project itself. The researcher planning the study must take these factors into consideration.

1) Ethics Approval

As a standard part of their work, social scientists submit research plans for approval by their institutions' Human Subjects Research Committee (HSRC), or equivalent. Software engineering researchers should do the same when they are studying human beings.

The HSRC normally wants quite detailed plans, and evaluates them to ensure that there is little risk of violating ethical norms. Research should not start until the HSRC gives approval.

2) Staffing

Work practice studies involve a lot of work that is not traditional in computing departments. Normally, most of the legwork in academic research is put in by graduate students. However, software engineering graduate students often lack the skills to design studies, perform interviews, and analyze the data. They also lack the interest to perform rudimentary tasks such as videotaping, transcribing recordings, and coding the data.

Software engineering graduate students enter the field expecting to work with technology, not people. They may be willing to do endless programming or statistical analysis, but will likely complain about transcribing or coding videotapes. This is partly because they are not likely to enjoy the latter work, and partly because they feel with justification that it will not help their careers. For the students who are interested in learning to perform good field studies, a small number of courses in research methodology from related disciplines can provide the necessary research skills.

We see no immediate remedy for the lack of interested students and qualified staff–researchers should seriously consider this to be the single biggest obstacle to effective work practices research. Some possible solutions are as follows:

- **Use secretarial support**. We have unsuccessfully tried this on several occasions. Having a secretary transcribe a tape filled with project-specific technical jargon and incomplete sentences, tends to give a result that takes as much time to edit as if the researchers had transcribed the text themselves. Even an engineer unfamiliar with the context was unable to produce an accurate transcript. And coding the transcript so that data can be statistically analyzed takes very considerable skill. It might be possible to engage a technically knowledgeable secretary to work full time on the project (i.e. to be present at interviews and observation sessions so he understands the context), but such people are hard to find and the process is expensive.

- **Involve social science experts**. University researchers might be able to establish collaborative research projects with faculty from psychology, sociology, or anthropology departments. The work of course is not lessened, but graduate students in these disciplines might be more motivated to do what is necessary. The drawback is that students and researchers are not always able to understand the technical issues and terminology from outside their own disciplines.

Given the above difficulties, the work practices researcher might feel inclined to fall back on data collection methods that require far less work to analyze because they are inherently quantitative: Surveys with closed-ended questions, and logs. Unfortunately such methods only give a partial picture.

3) Pilot Studies and Training

It is essential for the entire research team to practice and refine the research methodology before taking it on the road, otherwise many mistakes will be made and data will be lost. Researchers unfamiliar with the techniques discussed in this paper will be surprised about how many difficulties can arise. For example the wording of questions must be thoroughly tested to remove ambiguities. Also the process of setting up cameras, recording, transcribing, and coding should be well rehearsed.

In addition to understanding field study techniques, researchers should spend considerable time in learning about the field site. “Prior ethnology” needs to be established so researchers can effectively interact with the participants and correctly interpret data. The researchers needs a basic understanding of key aspects of the respondents’ work, such as the problem domain, the business context for the application, and the tools and process they are using. Some of this knowledge can be gained during the study itself, but we have found it more effective to have a learning phase in advance of the study.

4) Scheduling Access

Field research can be mentally intense for researcher and informant. In order to get the most out of the work, the pace should not be rushed. Plenty of flexibility should be built into the day’s schedule and no more than two sessions should be held in any day.

It is also important to understand that software engineers follow a development cycle. This means that they are doing different things at different times. Finding what SEs do during design and coding does not necessarily reflect what they do during bug-fixing or requirements gathering. Therefore, data collection has to focus on one aspect of the development cycle, or must extend over several time points to get an overall view of SE work.

Another consideration is SE's time constraints. Researchers need to find, to the greatest extent possible, data collection methods that do not affect SE's productivity. Unfortunately, it is not always possible to gather key information researchers unobtrusively. When a time commitment is required from SEs, researchers need to make sure that they get the largest possible return for that time.

In this section we discussed many of the issues we have faced in doing software anthropology. We also gave suggestions for resolving these issues. Our goal in presenting this information is to provide guidance for others undertaking similar studies for the first time. The issues discussed, such as establishing contact with organizations and respondents, staffing, and scheduling, can be accommodated through planning.

However, there are many events that cannot be anticipated. For example, participants may leave the team or company, change their work, or withdraw from the study. Similarly, companies may reorganize or lose interest in the research. A contingency plan for such situations is to work with two or three different field sites. But in cases where this option was not possible or feasible, valuable lessons can still be learned from the data collected. The data can reported as preliminary results, they can serve as a point of departure for a new study, or it can be combined in a later study to triangulate a phenomenon. We have been able to apply information collected from an aborted study to an entirely different user study.

Finally, these unexpected events serve to illustrate why field studies are necessary. If software engineering sites were completely understood and predictable, we would no longer need to study them.

V. Conclusions

In this paper we have discussed issues that software engineering researchers need to consider when studying practitioners in the field – an activity we call software anthropology. Software anthropology is one of several complementary approaches to software engineering research and is based on a recognition that software engineering is fundamentally a human activity: Software Anthropology is particularly useful when one is gathering basic information to develop theories or understand practices.

The material presented in this paper will be useful to both the producer and consumer of software engineering research. Our goal is give researchers a perspective on how they might effectively perform a field study – we believe that more such studies are needed. The material presented here will help others evaluate published field studies: For example, readers of a field study may ask whether appropriate data gathering or analysis techniques were used.

In this paper, we divided the set of field study techniques into three main categories. First-degree techniques such as interviewing, brainstorming, and shadowing place the researcher in direct contact with subjects. Second-degree techniques allow researchers to observe work without needing to communicate directly with subjects. Third-degree techniques involve retrospective study of work artifacts such as source code, problem logs, or documentation. Each technique has advantages and disadvantages that we described in Section II.

To perform good field studies a researcher must first create effective plans. The plans should describe the study techniques and also how various practical issues are to be handled. To choose study techniques, we espouse a modification of the GQM methodology, originally developed to choose metrics, but described here to choose data collection techniques. The researcher must have firm goals in mind, choose study questions that will help achieve the goals, and then choose one or more techniques that are best suited to answer the questions

In addition to deciding which techniques to use, the researcher must also determine the level of detail of the data to be gathered. For the first degree techniques a typical choice, in increasing order of information volume and hence difficulty of analysis, is manual notes, audio-taping and videotaping. In all three cases, a key difficulty is encoding the data so that it can be analyzed.

Regardless of the approach to gathering and analyzing data, software anthropology also raises many logistical concerns that should be dealt with in the initial plan. For example: How does one approach and establish relationships with companies and employees in order to obtain a suitable sample of subjects? Will the research be considered ethical, considering that it involves human subjects? And finally, will it be possible to find research staff who are competent and interested, given that most of the techniques described in this paper are labor intensive but not yet part of mainstream software engineering research?

In conclusion, software anthropology provides empirical studies researchers with a unique perspective on software engineering. As such, we hope that others will pursue this approach. The techniques described in this paper are well worth considering to better understand how software engineering occurs, thereby aiding in the development of methods for improving software production.

VI. References

[1] V. Bellotti and S. Bly. "Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team" *Conference on Computer Supported Cooperative Work (CSCW96)*, pages 209-219, Cambridge, MA.

[2] L.M. Berlin. “Beyond Program Understanding: A Look at Programming Expertise in Industry”, *Empirical Studies of Programmers, Fifth Workshop.*, pages 6-25, Palo Alto, USA, 1993.

[3] J. Buckley and T. Cahill. “Measuring Comprehension Behaviour Through System Monitoring”, *International Workshop on Empirical Studies of Software Maintenance (WESS’97)*, pages 109-113, Bari, Italy.

[4] B. Curtis, H. Krasner, and N. Iscoe. “A Field Study of the Software Design Process for Large Systems”, *Communications of the ACM*, pages 1268-1287, Volume 31, Number 11, November, 1988.

[5] D.A. DeVaus. *Surveys in Social Research, Fourth Edition.* UCL Press: London, 1996.

[6] M. Eisenstadt. “My Hairiest Bug War Stories”, *Communications of the ACM*, Volume 40, Number 4, April, 1997.

[7] K.Ericcson, and H. Simon. *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: The MIT Press, 1984

[8] W. Foddy. *Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research.* Cambridge University Press, 1994.

[9] J. Iivari. “Why are CASE Tools Not Used?”, *Communications of the ACM*, Volume 39, Number 10, October, 1996.

[10] B. Jordan. “Ethnographic Workplace Studies and CSCW”, In D. Shapiro, M. Tauber, and R. Traunmuller, editors, *The Design of Computer Supported Cooperative Work and Groupware Systems,* Elsevier. 1996.

[11] B. Jordan and A. Henderson. “Interaction Analysis: Foundations and Practice”, *The Journal of the Learning Sciences*, Volume 4, Number 1, pages 39-103, 1995.

[12] M. Jørgensen. “An Empirical Study of Software Maintenance Tasks”, *Software Maintenance: Research and Practice*, Volume 7, pages 27-48, 1995.

[13] J. Kay and R.C. Thomas. “Studying Long-Term System Use”, *Communications of the ACM*, Volume 38, Number 7, July, 1995.

[14] F. Kensing. “Prompted Reflections : A Technique for Understanding Complex Work.” *interactions*, pages 7-15, January/February, 1998.

[15] C.F. Kemerer and S.A. Slaughter. “Determinants of Software Maintenance Profiles: An Empirical Investigation”, *Software Maintenance: Research and Practice*, Volume 9, pages 235-251, 1997.

[16] M.B. Miles. "Qualitative Data as an Attractive Nuisance: The Problem of Analysis." *Administrative Science Quarterly*, Volume 24, Number 4, pages 590-601, 1979.

[17] M.B. Miles and A.M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook. 2nd Edition.* Thousand Oaks, CA: Sage Publications, 1994.

[18] J. Nielsen, “Evaluating the Thinking-Aloud Technique for Use by Computer Scientists”,. in H. Hartson and D. Hix, editors, *Advances in Human-Computer Interaction*, Vol. 3, Norwood, NJ: Ablex Publishing.

[19] D.E. Perry, N. Staudenmayer, and L. Votta. People, Organizations, and Process improvement. *IEEE Software*, July, 94, 37-45.

[20] S.L. Pfleeger and L. Hatton. “Investigating the Influence of Formal Methods”, *Computer*, pages 33-43, February, 1997.

[21] A.A. Porter, H.P. Siy, C.A. Toman and L.G. Votta, “An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development, *IEEE Trans. Software Engineering*, **23,** 6 pp. 329-346

[22] W.J. Ray, *Methods Toward a Science of Behavior and Experience*, Pacific Grove, CA: Brooks/Cole Publishing, 1993

[23] S.P. Robbins. *Essentials of Organizational Behavior*: Fourth edition. 1994. Englewood Cliffs, NJ: Prentice Hall.

[24] C.B. Seaman and V.R. Basili. “Communication and Organization: An Empirical Study of Discussion in Inspection Meetings”, *IEEE Trans.s on Software Engineering*, **24**, 6, June 1998.

[25] S. Seigel and N.J. Castellan, *Nonparametric Statistics for the Behavioral Sciences*, Second Edition, McGraw Hill, 1988

[26] A. Sen. The role of opportunism in the software design reuse process. *IEEE Transactions on Software Engineering*, 23, 7, pp. 418 – 436.

[27] S.E. Sim, C.L.A. Clarke, and R.C. Holt. “Archetypal Source Code Searches: A Survey of Software Developers and Maintainers”, *International Workshop on Program Comprehension (IWPC’98)*, pages 180-187, Ischia, Italy.

[28] S.E. Sim and R.C. Holt. “The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize”. In *20th International Conference on Software Engineering*, Kyoto, Japan, April, 1998.

[29] J. Singer, T.C. Lethbridge., and N. Vinson, (1998), "Work Practices as an Alternative Method to Assist Tool Design in Software Engineering", *International Workshop on Program Comprehension (IWPC'98)*, pages 173-179, Ischia, Italy.., 1998.

[30] J. Singer., T. Lethbridge., N. Vinson. and N. Anquetil. "An Examination of Software Engineering Work Practices", CASCON 1997, Toronto, October, pp. 209-223.

[31] J. Singer. "Practices of Software Maintainence," to appear in *International Conference on Software Maintainence*, Washington, DC, November, 1998.

[32] L. Snelling and D. Bruce-Smith. "The Work Mapping Tehnique". *interactions*, pages 25-31, July/August, 1997.

[33] A. VonMayrhauser, and A.M. Vans. "From Program Comprehension To Tool Requirements for an Industrial Environment". In Proceedings of the 2nd Workshop on Program Comprehension, pages 78-86, Capri, Italy, July 1993.

[34] A. VonMayrhauser, and A.M. Vans. "Program Understanding: Models and Experiments". In M.C. Yovita and M.V. Zelkowitz, editors, *Advances in Computers*, Vol. 40, pages 1-38, Academic Press, 1995.

[35] L.G. Votta, A. Porter and D. Perry. "Experimental Software Engineering: A Report on the State of the Art", *Proc. 17th Int. Conf. on Software Engineering*, April 1995, pp. 277-279.

[36] D.B. Walz, J.J. Elam, and B. Curtis. “Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration”, *Communications of the ACM*, Volume 36, Number 10, October, 1993.

[37] A. Wolf and D. Rosenblum. A study in software process data capture and analysis. In *Proceedings of the 2nd International Conference on Software Process*. February, 1993, pp. 115-124.

[38] R.K. Yin. *Case Study Research: Design and Methods, Second Edition.* Sage Publications: Thousand Oaks, 1994.

[39] M.V. Zelkowitz and D.R. Wallace. “Experimental Models for Validating Technology”. *Computer*, May, 1998.

Affiliation of the Authors

Timothy C. Lethbridge

University of Ottawa

School of Information Technology and Engineering

Ottawa, Ontario K1N 6N5

CANADA

tcl@site.uottawa.ca

Susan Elliott Sim

University of Toronto

Department of Computer Science

10 Kings College Circle, Toronto, Ontario M5S 3G4

CANADA

simsuz@cs.utoronto.ca

Janice Singer

National Research Council Canada

Institute for Information Technology

Montreal Rd, Building M-50

Ottawa, Ontario K1A 0R6

CANADA

janice.singer@iit.nrc.ca