Workshop on Empirical Studies of Software Maintenance, Monterey, Nov 1996
Fraunhofer ISBN 3-00-001337-7 p105-109

Methods for Studying Maintenance Activities

Janice Singer[1]
singer@iit.nrc.ca
Institute for Information Technology
National Research Council, Canada

Timothy C. Lethbridge
tcl@csi.uottawa.ca
School of Information Technology and Engineering
University of Ottawa

1. Introduction

The Knowledge Based Reverse Engineering Project's goal is to provide a group of software engineers (SEs) in an industrial telecommunications setting with a toolset to help them maintain the system more effectively. To achieve this goal, we have adopted a user-centered design approach to tool development. Thus, the first (and current) phase of our project involves an intense study of the SEs[1, c.f., 2, 3]. This paper describes the methods that we have considered and highlights those that we have used.

First, though, we have found that in studying maintainer's work, there are several special considerations. To begin, maintainers follow a development cycle. This means that they are doing different things at different times. Therefore, you must sample the SEs' behavior at several different time points. Finding out what they do during an enhancement cycle does not necessarily reflect what they do during a bug-fixing cycle. Therefore, you must look at various time points to get an overall view of their work.

Another focal consideration for us has been the SEs' lack of time. We need to find, to the greatest extent possible, unobtrusive measures that allow the SEs to continue their work. Unfortunately, it is not always possible to gather the type of information we are interested in by being a 'fly on the wall.' When a time commitment is required from the SEs, we need to make sure that we get the largest possible return for that time.

We are also concerned about obtaining domain knowledge for the telecommunications system. It would be impossible for us to become experts; however, we need to have knowledge of the base domain (telephony), the maintained system, the specialized tools, the base language and operating system, the documentation requirements, and the development cycle. Sometimes, this knowledge is not available. Other times it is hard to locate, and sometimes, not worth locating because it is erroneous. This puts us in catch-22 situation. We need to understand the SEs' work before we study them, but the only way to understand their work is to study them.

One final issue has to do with privacy. In this type of study, we are gathering sensitive data that could be used by other companies and/or by management. It is very important that we assure and maintain the privacy of all of our participants.

---

[1] We would like to thank the members of the Mitel SX2000 team for their participation. We would also like to thank Jelbar Sayyad and Pierre Fauvel for comments. This project has been sponsored by CSER (Consortium for Research in Software Engineering). Support has also been provided by NSERC.

The considerations of development cycles, time, knowledge, and privacy are important to address before undertaking empirical studies. We have not found solutions to all our problems; however, we have found it important to know that they are there.

Below is a description with pros and cons of some of the methods that we have considered adopting for our study. The ones we have used are highlighted.

2. Inquisitive techniques

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.

2.1 Brainstorming

In a brainstorming session, several people get together and focus on a particular issue. This technique usually works better with a moderator.

We have used brainstorming in our project and found that it works well. We obtained a good overview of where to focus our maintenance research.

Pros: Generates ideas that lead to testable hypotheses. Allows the participants to guide the experimental process. Best when you are starting a project and looking for ideas.

Cons: Even with a moderator, can be too unfocused. People can be shy in a group and not say what they really think. Hard to schedule a session with the busy schedules of the group members.

2.2 Interviews and questionnaires

Interviews and questionnaires are used to get specific information from the group members. Interviews allow for personal interaction, so that an interesting lead can be followed. Questionnaires allow the SEs more privacy and more time to reflect.

For these techniques, it is important to design appropriate questions. If time permits, it makes sense to design pilot questionnaires and then redesign them as you see which questions really attack the pertinent issues.

Pros: Easy way to get specifics about the project and the engineers, e.g., amount of experience.

Cons: When time is limited, SEs may not find the time. This means that you have to find some way of encouraging them to participate. It is not always clear how to interpret the data.

3. Observation techniques

Observation techniques allow for an accurate portrayal of the types of activities maintainers do. However, it is often hard to analyze the data, either because it is dense, or because it requires an enormous amount of knowledge to interpret correctly. Observation techniques can be used at randomly chosen times, as we have implemented them, or when the SE is engaging in a specific activity (such as using the debugger).

3.1 Think-aloud protocols

In think-aloud protocol analysis [4], the experimenter asks the SE to think out loud while performing a task The experimenter may occasionally remind the SE to continue thinking out loud. We are planning on implementing a variation of think-alouds where the SE thinks-aloud for a short time, and then reviews the tape with the experimenter to explain their problem-solving process.

Pros: Shows problem solving process and gives access to mental model. Established.

Cons: Original method was developed for an experimenter who could map out the entire problem space. It's not clear how it translates to this domain where it is impossible to know a priori what the problem space is. Very hard to analyze. Extremely time intensive.

3.2 Shadowing

In shadowing, the experimenter follows the SE around and records what activities they engage in. We are currently using this technique, and have found it to be useful in giving us a better idea of the SEs' work and how they spend their time

Pros: Easy to implement, fast results, requires no special equipment.

Cons: Hard to see what SE is doing, especially when they are using macros and working quickly. However, for the general picture, e.g., they are now debugging, it works well. You need to have a fairly good understanding of the environment to interpret the SEs' behavior.

3.3 Starship

In the Starship method, the experimenter acts as a mediator between the SE and the computer; i.e., the SE tells the experimenter what to type.

Pros: Forces SE to think out loud.

Cons: Could change problem solving process. Not clear what happens when the SE needs to consult other team members. Not clear how it works for graphical commands such as scrolling in a window.

3.4 Fly on the wall

In the Fly on the wall method, the experimenter records the activities as two (or more) SEs work together on a problem.

Pros: Allows access to SEs' thought processes because they must talk to each other.

Cons: Could be a different problem solving process than working alone. Difficult and time consuming to analyze data. Having experimenter there might change the interaction.

4. Monitoring techniques

Monitoring techniques focus on collecting data over a long time scale, either automatically or manually. They do not require the experimenter to be present.

4.1 Instrumenting systems

Here a program automatically records information from the system (e.g., each time the SE sends email to a group member or changes the code).

Pros: Requires no time commitment from SE. Gives accurate information.

Cons: Hard to analyze, especially when SEs have individual macros (e.g., in Emacs) defined. Also, it might tell you what the SE was doing, but it does not direct you to the logic behind that action. Possibly, you could record the keystrokes, and then play it back to the SE (like a movie) and ask them to comment.

4.2 Logbooks

Using Logbooks, the SE records information each time they do some task, or at some specified time each day, or on the computer at random times.

Pros: Daily or otherwise time sample of activities. Allows the SE to accurately report daily frustrations.

Cons: Might interfere with the problem solving process. For instance, if SEs have to record each time they go and consult a colleague, they may do so less often. SEs might not record accurate information. They may also forget to record all the time. They may not record fine details that are important.

5. Historical analysis techniques

Using historical analysis, the researchers, either in conjunction with the SEs or not, analyze historical data on the system.

5.1 Problem report (PR) analysis

In most large maintenance organizations, there is a paper trail that accompanies a bug fix, a PR. Here, you review a recently completed PR with the SE to document the problems they encountered along the way.

Pros: Work is completed, so you know where it's going. Gives you a useful basis to talk to the SE.

Cons: Retrospective reports are notorious for being inaccurate. However, even the inaccuracies could point you in the direction that the SEs find the most problematic.

5.2 Comment /Document analysis

In comment/document analysis, you analyze the comments and/or documentation generated by the SEs.

Pros: Using the source code as the source of data allows for an up-to-date portrayal of the system. Does not require SEs' time. Leads to important variables, hotspots, efforts at impact analysis, etc.

Cons: Time consuming. Requires some knowledge of source. Source comments/documentation may be inaccurate.

6. System and user modeling

Modeling allows the experimenters to find out what the SEs know about the system and how the knowledge of various SEs differs. The mental models that different SEs use might have a significant impact on the kinds of tools they will find useful.

6.1 System illustration

Here, we ask the SEs to draw or describe the system on which they are working. We have collected system illustrations from the SEs in the group we are studying.

Pros: Good conceptual basis for mental model.

Cons: Hard to interpret if you do not have greater knowledge of the system than the SEs. For instance, the manager knew right away that the illustrations reflected the parts of the system that the SEs were *currently* working on. We could have never known this so quickly, if at all. Not clear how to analyze.

6.2 Building a knowledge base

Here, various techniques from the domain of knowledge acquisition are used to build a knowledge base. The experimenter, acting as knowledge engineer, builds an inheritance hierarchy describing the main concepts in the system and their properties.

Pros: Clearly shows what SEs think is important. Highlights differences in mental models.

Cons: Time consuming. Only worthwhile if there is a secondary benefit to having a knowledge base (e.g., to be used by expert-system tools)

7. Conclusion

We have presented in this paper a number of methods for studying the work of software maintenance engineers. Our goal is to understand the work environment and the problems SEs face so that we can build effective tools that will get used. Some of the described methods we have used, and some we plan to use. Each has provided us with a different type of information. The most important lesson that we have learned from this endeavor is that each method has its advantages and disadvantages. The only way to get a truly accurate picture of maintainer's work is use several methods at various points in the development cycle.

[1] Lethbridge, T., Singer, J., Strategies for studying maintenance, *Proc. Workshop on Empirical Studies of Software Maintenance*, Monterey, and *Empirical Software Engineering: An International Journal* (this issue).

[2] von Mayrhauser, A., & Vans, A. M. Program Comprehension during software maintenance and evolution. *Computer*, *28(9)*, 44-55.

[3] Boldyreff, C., Burd, E., Hather, R., Mortimer, R., Munro, M., Younger, E. (1995). The AMES approach to application understanding: A case study. In Proceedings of International Conference on Software Maintenance, Nice, France.

[4] Ericsson, K., & Simon, H. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, MA: MIT Press.