

# Studying Work Practices to Assist Tool Design in Software Engineering\*

Janice Singer  
Institute for Information Technology  
National Research Council  
Ottawa, K1A 0R6, Canada  
+1 613 991 6346  
janice.singer@iit.nrc.ca

Timothy Lethbridge  
School of Information Technology and  
Engineering  
University of Ottawa  
Ottawa, K1N 6N5, Canada  
+1 613 562-5800 x6685  
tcl@site.uottawa.ca

## ABSTRACT

*This paper reports our experiences studying the work practices of professional software engineers (SEs). We provide our reasons for following this approach, and describe details such as the discovery of work patterns, and the use of synchronized shadowing. We outline several studies we are currently conducting in a large telecommunications company and explain how these studies influenced the design of a software engineering exploration environment.*

## 1. Introduction

Studying program comprehension is fraught with difficulties. One of the dilemmas researchers must face is whether to work in a lab setting or in the field: laboratory experiments are difficult to generalize to the field because variables that normally vary across field settings are fixed in experiments and thus it is hard to say how the results will apply in the 'real world.' In field studies however, it is often impossible to adequately control variables, hence it is not always clear what factors had a role in the results. Another dilemma is whether to use students or industrial software engineers (SEs): Students are easily available to participate in studies but do not necessarily perform the same type of work as those in industry; on the other hand SEs in industry often have no time to participate in studies.

Due to their respective limitations, then, a number of approaches must be followed in order to fully understand program comprehension,. A researcher pursuing one approach will be able to tell us certain things about program comprehension, while a researcher pursuing another approach will learn different things. Where different approaches converge around the same concepts, we will start to have a central corpus of knowledge about the processes used in program comprehension.

Since many laboratory studies already exist, we have chosen to study industrial practitioners in the field. Using a set of largely qualitative analysis techniques we are studying the *work practices* of software engineers. The study of work practices is a relatively new field [1, 2, 5] which integrates methodologies and theories from several different fields including cognitive and social psychology, human-computer interaction, business-process re-engineering, and anthropology. From this understanding, work practice researchers strive to design appropriate technologies for the workplace.

This paper describes our experiences with the work practice approach to tool design. First we explain why we chose the work practice approach. Second, we briefly review a series of studies we undertook to examine the work practices of a group of SEs at a large telecommunications company. This sets the stage for the description of a software engineering exploration tool based on our studies. Finally, we discuss how work practices in general can influence tool design.

---

\* This work is supported by NSERC and Mitel Corporation and sponsored by the Consortium for Software Engineering Research (CSER).

## 1.1 Work practices, work patterns and tool design

The Knowledge Based Reverse Engineering (KBRE) project's ultimate 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 this end, we decided that our first step must be to understand SEs' daily activities as they modify and come to comprehend the source. We began by searching the literature for a cataloguing of SE maintenance activities; i.e., a list of activities SEs engage in during their work - for instance, work with the maintenance control system, go to meetings, read documentation, search the source, etc. However, in the literature, we found no clear cataloguing as such of exactly how SEs go about solving problems and/or comprehending the source code. While many models of programmer comprehension exist, generally, they do not describe details of the comprehension process (but, c.f., [6]). Consequently, we decided that our first step would be to uncover both the low-level activities and high-level goals of SEs as they are solving a problem.

For us, a critical aspect of this endeavor was that it focus on the workplace and the work that occurs there. Because our ultimate goal had us placing a tool in a particular site, we wanted to make sure that a) the tool fit into the existing work practices of the site and b) it solved existing problems. This is because we felt that the best way to ensure tool acceptance was to solve the real problems SEs faced, while at the same time not requiring them to re-think all of the activities in which they currently engaged. This has been termed an evolutionary as opposed to a revolutionary approach to tool generation.

Thus, to understand SE work practices, we began by designing and implementing a series of studies (described below) from there proposing technical solutions to some of their problems. In the course of these studies, we began to see that specific series of work practices would be repeated, forming re-occurring patterns

Thus, we use the term *work pattern* to designate a re-occurring sequence of work practices meeting a goal. We incorporate work patterns into our tool by having the tool, rather than the user, accomplish many of the intervening actions. This saves the SE from issuing several commands. This approach can potentially be iterated to discover higher-level work patterns, that are themselves composed of work patterns, thus producing a hierarchy of work patterns from the bottom up.

To find work patterns, we analyzed our work practices data to find the most frequent, time consuming, and important activities. Further studies focused on these activities to extract work patterns.

The next section describes our work practice studies. In it, we give some general information about the workplace, followed by a description of the various methodologies we pursued, with some related results.

## 2. Work practice studies of software engineers

We began our study of work practices by finding out what it is in general that SEs do when they do their work. Our approach was fourfold; we conducted a web questionnaire, performed intensive shadowing of an experienced SE who was a newcomer to this project, performed various studies of a whole group, and collected company-wide tool usage statistics. Section 2.1 provides some general information about the workplace we were studying. Following this, sections 2.2 - 2.5 outline the methods and results of each of these studies (for greater detail see [3,4]). After using the data to understand individual activities we then analyzed it further to discover work patterns: Our preliminary analyses of these is in section 2.6 and 2.7. A discussion of the tool that we developed using this information is in section 3.

### 2.1 Workplace characteristics

The group we are studying maintains one of the key products of the company: a large telecommunications system. The management of the group is fairly informal, with group members often able to select the problems on which they work.

Group members work in close proximity and often walk over to each other's desks with questions. The group also makes use of a laboratory in which the target hardware is installed.

The system includes a real-time operating system and interacts with a large number of different hardware devices. The system contains several million lines of code with over 16000 routines in over 8000 files. It is also divided into numerous layers and subsystems written in a proprietary high-level language.

The system was first fielded in the early 1980s and has since been continually updated. Its importance to the company and its evolution are expected to continue for many years to come.

Approximately 13 people actively work on various aspects of the system at the current time. Over 100 people have made changes to the source code during the life of the system.

The group follows a well-defined process for creating new system features. They also keep detailed records of problem reports and the consequent changes to the system.

Other important documents include the ‘practices’ that are followed by those who install and run the system in the field.

Careful attention is paid to quality control in the form of design reviews, informal code inspections, and an independent test team.

Development work is done on the Sun platform, although the SEs must spend a considerable amount of time installing and running the software on various configurations of the target hardware.

## 2.2 Questionnaire Study

We began our research by administering a web-based questionnaire (6 of the 13 SEs completed the questionnaire). The questionnaire covered many different aspects of the SEs’ work. Here we report their answers to two questions about what they spent their time doing.

The first question was open-ended, meaning that SEs had to identify activities for themselves, rather than choosing activities from a list. The activity listed by the

most SEs was reading documentation; many also reported spending time looking at source code, writing documentation, attending meetings, and writing source code. Other activities included consulting, both answering and asking questions, working with the hardware, testing, designing, and fixing bugs.

The second question asked the SEs how they divided their time: On average, 57% of their time was spent fixing bugs, and 35% of their time making enhancements to the system.

Due to the questionable validity of self-reports, we only used the questionnaire to obtain a rough initial indication of what the SEs’ work involved. The next two subsections of the paper describe studies that allowed us to improve our knowledge by obtaining direct observations.

## 2.3 Individual study

We have been following one SE longitudinally from the time he first joined the company (November, 1996). For the first six months, we spent about 1-1/2 hours per week with B. However, as B has become more knowledgeable about the system, we meet only once every 3 weeks. This is because new things happen less frequently: B has fewer experiences with new tools and at the same time is working on larger problems that require long periods performing tasks such as reading documentation or reproducing the problem. B is an experienced SE (he was previously a team-leader), thus while he is new to the company, he is certainly not new to either maintenance or telecommunications software.

Our sessions with B consist of 3 distinct components. First we talk about what has transpired since the last time we met. This could be anything from code reviews, to learning about a new tool, to reading documentation, etc. Second, we ask B to look at a diagram of the system that he previously constructed and ask him to modify it if it does not reflect his current understanding of the system. Finally, we ‘shadow’ B as he works for half an hour. In this paper, we report the data from the shadowing.

We classified B’s shadowed events into the 14 distinct categories described in Table 1. The data reported in this paper reflects 14 shadowing sessions with B.

Searching and interacting with the hardware were the most likely events to occur on a daily basis, each occurring on 8 of the 14 days. B studied the source code using simple editors on 6 of the days. The reason that B searched on more days than he studied the source code is that searching occurred when interacting with the hardware and debugging. B only looked at documentation on 2 of the 14 days. This is surprising because, at the time, B was still a relative novice to the software system and it is

<i>Activity</i>	<i>Description</i>
<b>Call trace</b>	Looking at an execution trace of the program
<b>Consult</b>	Either being consulted or consulting someone else
<b>Compile</b>	Linking or compiling a program
<b>Configur- ation management</b>	Entering and using the in-house configuration management system (sometimes for updating, and sometimes to search for past updates)
<b>Debug</b>	Using either the high-level or low-level debugger
<b>Document- ation</b>	Looking at documentation
<b>Edit</b>	Changing the source code
<b>Management</b>	General software activities, such as meetings, code reviews, etc.
<b>In-house tools</b>	Using one of the in-house tools, primarily static software analysis tools
<b>Notes</b>	Taking notes, or reading past notes
<b>Search</b>	Using grep, in-house search tools, or searching in an editor
<b>Source</b>	Studying source code using editors or code viewers
<b>Hardware</b>	Interacting with the hardware, e.g., loading software, running software, configuring the hardware, etc.
<b>UNIX</b>	Issuing a general Unix command, e.g. ls

**Table 1: Categories of activities performed by the SE we shadowed.**

commonly assumed that novices will spend much of their time reading the documentation to get a handle on what they are doing. The data show that this was not the strategy B pursued. However, because B was a novice, it was not surprising to find that editing code, compiling, and management were each done on only 1 of the 14 days.

If instead of daily activities, we look at the overall frequency of activities, we see that B searched more often than he did anything else (37 times over the 14 days). He also frequently studied the source code (33 times over the 14 days). While B was likely on any particular day to work with the hardware, he did so on only 22 distinct occasions.

Thus overall, both in terms of daily activities and frequency of different activities, search for information about the system, whether through `grep`, in-house search tools, or within a particular editor or debugger, figured most prominently in B's attempts to comprehend the system. A significant amount of B's effort was also expended interacting with the hardware and studying the source code.

## 2.4 Group study

In the last section, we discussed intensive studies of one individual. To generalize our findings, we conducted several studies that focused on various aspects of the work of an entire group of SEs.

We collected four types of data from the group. First, we asked the SEs to draw a diagram or picture of their current understanding of the system, a conceptual map, if you will. Second, we conducted intensive interviews with the SEs. Some of these asked about their work in general, while others focused on how they solved a real problem with the software. The latter generally involved several 1-hour interviews over the course of several days. Finally, we spent one hour shadowing each SE as they went about their work. This report focuses on this third type of data; the shadowing data.

Eight group members participated in the shadowing study. Their experience ranged from the most expert member of the group (8 years) to the least experienced (6 months, a recent college graduate). All but one of the shadowed subjects worked on the main controller of the hardware. One of the subjects worked primarily on the database component.

The subjects were expert in a wide variety of platforms and languages, and had experience in both development and maintenance environments.

Like B's data, the shadowed events were classified according to the 14 distinct categories described in Table 1. 356 distinct events were recorded.

All 8 SEs looked at the source, conducted a search, and changed the source code at least once during the hour. Most of the SEs also engaged at least once in several other activities, with 5 of the 8 SEs interacting with the hardware, debugger, or the in-house tools. On the other hand, only 3 SEs looked at a call trace, while only one SE performed a management activity.

Of the total of 356 events (counted over the 8 SEs), issuing a Unix command was the most frequent activity, occurring 54 times. A close second was studying the source which was done 52 times. Interacting with the hardware (36 times) or the debugger (32 times), searching (31 times), and changing the source code (30 times) were the next most frequent activities. Configuration management, consulting, compiling, and working with in-house tools were each done about 20 times.

Surprisingly enough, reading the documentation, although performed by 6 of the 8 SEs, accounted for only 12 separate events. Clearly, the act of looking at the documentation is more salient in the SEs' minds (as evidenced by the questionnaire data) than its actual occurrence would warrant.

SEs only occasionally wrote notes, looked at the call trace or did management activities. This is not to say that these events are not important, but merely that they did not occur as frequently as other events.

As B did, members of the group frequently examined the source code. Every SE in the group made at least one search during their shadowing session, but search was less prominent than in B's activities. Search ranked as the most frequent event type for B, while it was the 4th most frequent for the group.

Code editing and compiling were more prominent activities in the group data than in B's data. This is probably because B was still learning the system at the time we shadowed him, so he was not yet in a position to make many changes. This may also explain the higher incidence of working with the call trace in his data: doing the latter may be effective in gaining an initial understanding of a system.

Interestingly, in-house tools and documentation were both relatively infrequent activities for both the group and B.

The group data converge with B's data to suggest that looking and searching through the source are prominent activities for SEs in attempting to comprehend a system. Editing and compiling are also important. This concurs with what we would expect in that their work revolves around the source code.

## 2.5 Company Study

The final study we report concerns company-wide tool usage statistics. These data were obtained from the company's tool group. This group is responsible for acquiring, updating, and maintaining the company's tools. Collecting usage statistics is part of their mission.

The data presented here represent one week of Sun tool usage by 367 users in late May 1997. Note that this week occurred before 'vacation season,' so is fairly representative of peak tool usage. There were 79,295 separate tool calls logged from the Sun operating system.

Invocations of compilers occurred 32,422 times (41% of all events recorded) due to regular automatic load-builds; therefore we excluded this data from our studies.

When we factored compiling out, the overwhelming finding from the company data is that search is done far more often than any other activity. In fact, search accounts for 21,146 events over the course of the week, or an average of about 58 searches per individual user. Compression and un-compression tools are also used often (We never actually observed anyone using these tools so we assume that they are also mostly used by automated scripts).

The configuration management system was activated 2819 times, accounting for approximately 4% of all events. At this company, the configuration management system is central to the work process, both for retrieving files, filing changes, and searching through past changes (along with associated documentation).

Editors and viewers account for approximately 3190 events, or 4% of the total number of events. This low frequency could be due to counting particularities that apply only to editors. In the company tool data, an editor command is counted only when the editor is opened. Once an editor is open, it generally stays open, regardless of how many changes are made, or how many files are viewed. In contrast, in the shadowing data, an edit was recorded each time the source was changed, and a source event was counted each time the source was examined, whether the editor was already open or not. Consequently, it comes as no surprise that in the shadowing data, edit and source frequency is higher than it is in the company-wide data.

Again, the in-house tools are not used very frequently, but that belies their importance. These tools are important because they perform necessary functions that cannot be performed by other tools.

Search is the most frequently used tool at the company wide level. Grep and its variants are the most frequently used search tools, accounting for 21,117 separate invocations. Clearly, search is an important aspect of SEs work practices.

## 2.6 Synchronized shadowing to discover work patterns

The above studies of SE work practices highlighted two primary activities: search and navigation. In continuing our research, we are focusing on these areas. In particular, we are interested in the recurring sequences of actions that SEs follow to execute search, i.e., search work patterns (this term is discussed in section 1.1).

To find work patterns, we implemented a methodology we call *synchronized shadowing*. Here, two observers shadow an individual SE at the same time. Each observer records observations on their own laptop computer. The clocks on the two computers are synchronized, so that the two data sets can later be matched. One researcher records the low-level work practices of the SE, such as 'execute grep', 'open an editor', 'look at the source', etc.<sup>1</sup> The other researcher has the SE "think-aloud" while working, and records the SE's immediate goals and whether and when they are achieved. For example, the first researcher may record that the SE executed grep, while the second researcher would have recorded that the SE was looking for a variable, looking for a constant, looking for a routine name, etc. The high-level goals recorded are only those directly mentioned by the SE; the very low-level goals are partly interpreted or inferred from the sequences of actions and from the higher level goals (see section 1.1).

To find the work patterns, then, the two data sets are merged so that the goals can be matched up with the specific actions that were taken to achieve them. Over time and after studying many more SEs, we expect that certain goals will always be matched with the same or very similar actions to form work patterns.

We implemented the synchronized shadowing method because we found that no other technique would work effectively. A single researcher could not record both types of information; videotaping was too time-consuming, and automated recording missed important data.

## 2.7 Work patterns of particular importance

During the course of our studies we began to notice several important work patterns. We are still in the process of extracting these patterns, but our preliminary attention became focused on patterns common to several SEs and having mechanical, time-consuming and/or inefficient elements that could perhaps be automated.

---

<sup>1</sup> Unfortunately, this level of observation cannot be done by automatic logging of keystrokes and mouse movements.

The following are four of the most important such patterns:

1. Searching for some target string using `grep`, successively opening each file that had `grep` 'hits', searching for the same target in the file, and then studying the code around the hits. In most cases the `grep` target was a very simple string and the search involved a very large number of files; the SE was often forced to wait for many seconds for the result, and some SEs developed the habit of starting searches in the background, and then performing some other task while they waited for the search to complete.
2. Saving the results of `grep` searches to act as checklists for future work (either lists of places to study, or lists of places where changes are needed), and then working through the checklists. This pattern was fraught with errors, however: On several occasions we observed SEs repeating searches because they could not find previous results (e.g. they had scrolled too far off the top of the screen).
3. Suspending an investigation of a checklist item to perform some other search or study, then resuming work at a later time. This task switching involved considerable overhead, and it was hard for the SEs to keep their work organized.
4. Jumping back and forth between tools, primarily Unix command line (performing `grep`) to editor and back. This jumping involved the use of cut and paste to transfer data and was frequently awkward.

The next section describes how we developed a tool that helps SEs more effectively achieve the goals implicit in the above patterns.

### 3. Developing a tool using the results of work practice studies

In this section, we discuss how we used work-practices studies to inform the design of a software engineering tool.

In late 1995 we started our research project whose goal was to discover techniques whereby SEs could more effectively maintain large legacy systems.

#### 3.1 The first release

For the first release, we brainstormed a group of SEs for their needs, and then designed, with their continued involvement, a tool called SEE (Software Exploration Environment); its main features were:

a) Hypertext-like abilities to select any word in the code, and build a list of relevant information that describes

that word (e.g. a variable, a routine or even a word in comments).

b) Abilities to build, in a hierarchical manner, a list of items related to the file, routine, identifier etc. on the screen.

Both of these facilities were ranked high in the brainstorming sessions. They proved useful to the SEs (as evidenced by ongoing use) and remain, in improved form, in the current version.

#### 3.2 Work studies based design: The second release

Our work practice studies proceeded in parallel with the above, and have so far been underway for over a year. These studies clearly could not inform tool design for the first release since we had to amass data. We therefore used them to develop the second release.

We used the work patterns discussed in section 2.2.6 to guide our tool design, and thus implemented the following features in the second version of SEE. We call this `tksee`, and a screen snapshot is shown as figure 1:

- Persistent hierarchical history<sup>2</sup>. This facility automatically records the entire state of each exploration and presents it to the user in a compact, but graphical manner. It allows the user to jump among states or return to earlier states, and thus facilitates work patterns 2 and 3. Information recorded in each state includes the object the SE was studying (right pane in figure 1), as well as the exploration hierarchy – i.e. the path that led the SE to this object (left pane). Each time the SE starts a new search, a new history record is created (top pane). These history records are themselves hierarchical. Any given level of the hierarchy represents search tasks that the user considers to be peers; if the user selects one of these search records and performs new search work, then a lower level in the hierarchy will be started.
- Visual `grep`. Although the user can perform useful queries with a combination of hypertext and relationship-expanding that were available in the original tool, users persisted in using `grep`, jumping from our first release to the command line and back. In order to help users better perform work patterns 1 and 4, therefore, we integrated `grep` into `tksee`. There are three ways to access this functionality: 1) Requesting an 'ordinary' `grep` whereby the search hits are displayed as a fresh search in the history hierarchy. 2) Selecting some items in the bottom left pane and requesting a `grep` in each of these; the hits being displayed indented below

---

<sup>2</sup> Although a rudimentary version of history was available in the original tool, it was little use since it was not persistent nor automatic enough

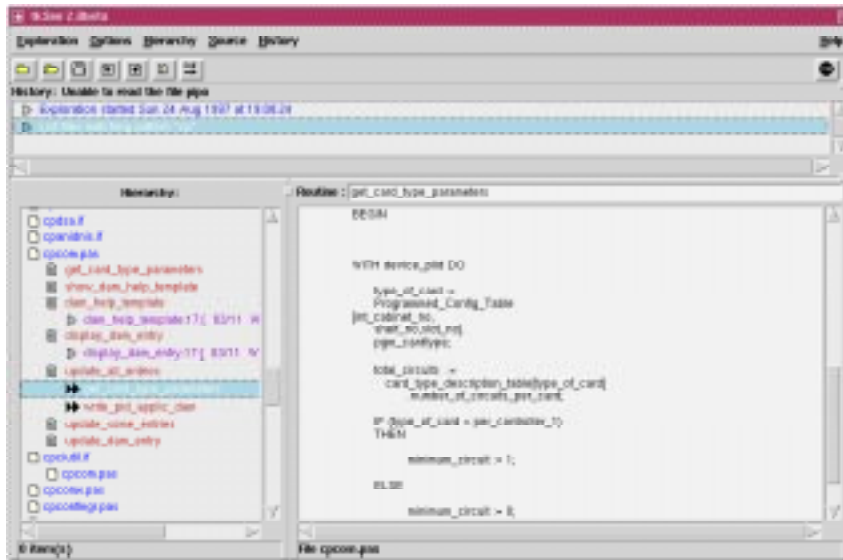


Figure 1: An example of the Tksee main window.

the places where they were found in the bottom-left pane. 3) Selecting (in the bottom-left pane) an item that is the destination of a relationship, and requesting that the places in that item that establish the relationship be highlighted as grep hits. In all three cases, the user can select a grep hit and immediately see the context of the hit in the right pane.

### 3.3 Conclusions from tool development

The second release of the tool has been eagerly adopted by a variety of SEs. This is an achievement, since it is hard to encourage these people to adopt new techniques – many of them have not even adopted emacs, and prefer to use more primitive editors they know better.

We attribute our success to the following: a) we focused on tasks that they do most frequently (i.e. search); b) we developed tools that specifically helped with work patterns that appeared cumbersome previously; c) we allowed them to continue their existing work practices (e.g. use of grep), rather than forcing them to adopt a radical new paradigm.

One criticism we have received about our research is that there are already commercial and freeware tools that incorporate some of the facilities we developed. Well known examples include Sniff+ and emacs. Our counter-argument to this is to ask, why are those tools not being used by our SEs? We believe that our work-practices studies allowed us to develop a tool that fits more precisely with the SEs' needs. The other tools either do not integrate all the facilities needed (especially the persistent hierarchical history) or are overly complex.

## 4. Conclusions and future work

This paper has described experiences with several techniques that can help us to develop systems that are not only usable, but are also used.

In our work practice studies, we first discovered what our users did in broad terms using interviews, shadowing, questionnaires, etc. Then we focused on the most frequently performed tasks to discover what we call work patterns.

We have also shown that by using the technique we call synchronized shadowing, it is possible to gather information about both activities and goals fairly efficiently. The analysis of this data leads to the discovery of work patterns that are most amenable to automation.

We will now continue our work-practices research and perform tool development iteratively. We will study the extent and manner with which the SEs use the facilities we developed as a result of this research. We also plan to study the work patterns of SEs in more depth as we amass more observations.

## Acknowledgments

This research is supported by NSERC and sponsored by the Consortium for Software Engineering Research (CSER). We would like to thank the SEs who participated in our studies.

## References

1. Beyer, H., & Holtzblatt, K., Apprenticing with the customer. *Communications of the ACM* 38 (1995), 45-52.
2. Blomberg, J., Suchman, L., & Trigg, R., Reflections on a Work-oriented Design Project. *Human Computer Interaction* 11, (1996), 237-265
3. Lethbridge, T., and Singer, J. (1997). Understanding software maintenance tools: Some empirical research, in *Workshop on Empirical Studies of Software Maintenance*, (Bari, Italy, October 1997), pp. 157-162..
4. Singer, J., Lethbridge, T., Vinson, N. and Anquetil N. An Examination of Software Engineering Work Practices, in *Proceedings of CASCON '97* (Toronto, November 1997).
5. Vicente, K and Pejtersen, A. *Cognitive Work Analysis*, in press.
6. von Mayrhauser, A and Vans, A., Program Comprehension During Software Maintenance and Evolution, *Computer* Aug. 1995, 44-55.