Work Practices as an Alternative Method to Assist Tool Design in Software Engineering

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

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

Norman Vinson
Institute for Information Technology
National Research Council
Ottawa, K1A 0R6, Canada
+1 613 993 2565
vinson@iit.nrc.ca

ABSTRACT

This paper reports our experiences studying the work practices of professional software engineers(SEs) with the goal of designing tools that enhance, rather than displace or replace, these work practices. The rationale being that the tools we build will actually be used because they have been created to mesh with existing behaviour. We provide our reasons for following this approach, and describe some details of our approach 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 talk about how these studies have influenced the design of a software engineering exploration environment.

Keywords

Work practices, tool design, software engineering tools, programming tools, work patterns, shadowing.

INTRODUCTION

Software engineers (SEs) are heavy users of software themselves, be that software CASE tools, configuration management systems, exploration environments, etc. Unfortunately, little effort has been expended in understanding how SEs work, what is entailed in their day-to-day activities, and how some of their difficulties might be alleviated. This in turn has led to a problem in academic and industrial efforts to help professional SEs: the problem of shelfware. SEs often lament that the tools produced to help them don't solve the problems they face and hence end up sitting on the shelf, rather than really being used.

This shelfware problem has led our research group to study the work practices of professional SEs with the goal of designing tools that enhance, rather than displace or replace, these work practices. Our rationale is that the tools we build will actually be used because they have been explicitly designed to permit incremental modification of existing behaviour.

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.

LACK OF EXISTING RESEARCH

We began our research by reviewing the existing literature. With respect to work practices, we found a burgeoning field of endeavor describing specific methodologies (e.g., [3, 6, 15]), research in diverse areas (e.g., [4, 8, 7, 14]), and the relationship of these studies to human-computer interaction and design (e.g. [11, 17]).

With respect to SE work practices though, we found little that could help us. Eleven years ago at the First Workshop on Empirical Studies of Programmers, Bill Curtis posed the question, "By the way, did anyone study any real programmers?" [5] We could add an addendum to this question, "And by the way, did anyone study any real programmers really working on real programs?" Unfortunately, with little exception, the answer to this question eleven years ago was "no", and remains "no" today.

We are interested in designing tools for individual SEs working in a maintenance environment. The few papers we did find dealt with the design of new software systems (e.g., [16, 19]) or collaboration among developers (e.g., [2, 9]). There were no papers describing the day to day activities of SEs working on real maintenance problems.

There are clear differences between maintenance and the design of new systems. Indeed, there are many skills necessary for success in one that are not necessary for success in the other. For instance, many software maintainers we have interviewed describe their work experiences by making an analogy to detective work, i.e., delving into someone else's code to find a problem. In contrast, we've never heard designers of new systems talk this way about their work. Due to this, research literature that talks only about design of new systems is, at best, only marginally relevant to our research.

With regard to collaboration, we are interested in designing tools to help individual SEs. SEs might use our tool to collaborate with others, but this is not our primary vision of its use. A tool for an individual SE cannot be designed by looking at how groups of SEs work together.

One specific area of research we thought would be helpful was the Empirical Studies of Programmers literature, but there are two problems with this research. First, the vast majority of it has been conducted with graduate and advanced undergraduates serving as expert programmers (but c.f., [18]). It is clear that these subjects do not accurately represent the population of industrial programmers. Consequently the results of studies involving students cannot be generalized to SEs in industry.

Second, even when professional SEs have served as subjects, researchers have used programs that are very small (both in terms of lines of code and logic) relative to industrial software. This is done to control as many variables as possible. However it poses another generalization problem: Approaches to working on small programs will not necessarily scale up to very large programs. For these two reasons, the ESP literature fell short of our expectations.

This lack of relevant literature motivated our decision to pursue work practice studies of professional SEs. The studies are described in more detail in the "Work Practice Studies of Software Engineers" section. Below, we present our reasons for following this approach.

WORK PRACTICES

Advantages of Work Practices

To design software maintenance tools, we could have employed a top-down approach as follows. First, analyze the structure of maintenance tasks, then apply that analysis to general models of cognition to generate a cognitive model of software maintenance. This model would provide specific hypotheses that could be tested under controlled conditions in a lab. However, as noted above, conclusions based on the results of lab experiments may not generalize to an industrial setting. Additionally, a tool design does not immediately follow from a specification of the user's mental model. Several designs can be generated and tested, requiring the designers to collect usability data. Even so, this approach does not guarantee that the resulting tool is useful [4]. Moreover, the top-down approach does not guarantee that SEs would use the tool. As noted earlier, getting SEs to use a new tool is quite difficult.

The study of work practices avoids many of the problems mentioned above. First, the work practice approach involves collecting field observations; that is, observing the daily activities of SEs in their work environment. This avoids the common problem of generalizing to industrial settings: since we collected our data in the field, our findings apply directly to SEs working on industrial code. However, the possibility that our findings do not generalize to other industrial settings must be noted. Nonetheless, it is likely that the similarity between two industrial settings is greater than the similarity between a lab experiment and an industrial setting.

Second, a tool designed using the study of work practices is less likely to end up as shelfware. The reason is that since data gathered in this manner reflects the existing behaviour of SEs, it enables us to design a tool that will mesh with that behaviour. Note that this includes behaviour that is not related to the tool (such as consulting colleagues, documentation, notes etc.) but forms part of the work context into which the tool has to fit. Since the tool fits into the work context and meshes with existing behaviour it should be easier for SEs to incorporate tool use into their daily activities.

Third, such a tool is likely to be useful because it is designed to increase the effectiveness of existing behaviour. In other words, the SEs will continue to do what they had done before, except that they will do it more effectively. For instance, they will still issue search commands, but they will issue them to a search engine that is better suited to their work.

Below we talk about the relation of work practices to tool design. In the section "Work Practice Studies of Software Engineers" we discuss our work-practices studies. We then explain how we designed a tool using participatory design and work practice studies as sources of information.

Work Practices, Work Patterns, And Tool Design

We have encountered two main arguments against moving directly from work practice data to tool design. In this section we present and deal with these arguments.

The first argument is that the type of data collected in work practice studies is not helpful in tool design. Instead of recording actions (e.g. the issuing of a grep command), we should instead focus on the goals of those actions (e.g. searching for a variable V in a series of files). Only when goals are known, the argument posits, can a tool be built to help SEs meet those goals. Moreover, the argument continues, one needs to develop a complex cognitive model of the task in order to understand goals properly.

We agree that it is essential to understand goals. We can contrast two approaches to extracting user goals and relating them to user behaviour: the top-down approach, which is used with cognitive models, and the bottom-up approach, which we have used. In the top-down approach, a pre-existing cognitive model of the task is used to categorize sequences of user behaviour, such as sequences of user actions or verbal "think-aloud" protocols,. It is in the categorization process that user goals are inferred [6]. Examples of this approach can be found in [12] and with SEs as users in [18].

In contrast, our bottom-up approach is data driven and does not require a cognitive model of the task. For example, consider the two following sequence of user actions. First, the user issues a search command (e.g. grep) over several files for a particular string S. Second, the user opens one of the flagged files in an editor, and searches (e.g. with

CNTRL-S) for string S in that file. The search commands, although syntactically different, are both categorized as search commands. In addition, the open file command is recognized as such. Finally, both search strings are encoded as being the same search item, even though one string may have been truncated. These categorizations do not require a model of software maintenance. They simply require a rudimentary understanding of the SE's interface and tools. As for goals, they can be inferred from the result of user actions. To continue our example, we can infer that the goal here was to examine the search string in the context of the program code. Alternatively, the goal can be expressed verbally by the user, as his work is observed. (E.g. "I'm now looking for this string."). These are the types of goals that interest us with respect to the tool. Determining this type of user goal does not require a cognitive model.

We use the term *work pattern* to designate a re-occurring sequence of user actions meeting a goal, such as the one illustrated above. 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.

Our search for the most significant work patterns also proceeded without reference to a model. Rather, we simply 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.

To conclude, it is true that some understanding of user goals is important to tool design, but this can be achieved in a bottom-up fashion. Thus, in designing our tool, we avoid the costs of constructing and verifying a complex cognitive model of software engineering or maintenance.

The other argument we have encountered against work practice studies is that by designing a tool that incorporates existing behaviour, we are entrenching the inefficient work practices of SEs. That is, rather than reinforcing their behaviour, we should be changing it.

The assumption that work practices of SEs are inefficient is unwarranted. Other than our data, there is no catalogue of the work practices of SEs engaged in software maintenance. Therefore, there is no empirical evaluation of the efficiency of these work practices. In fact, it is more likely that the work practices of experienced software maintainers are efficient and reflect expert performance. This is because experienced maintainers are the only people who have the domain knowledge necessary for expert software maintenance. Indeed, twenty years of research on expert behaviour has shown that a large body of domain knowledge is a pre-requisite for expertise [1]. By the same token, it is extremely unlikely that people with little experience in software maintenance could design tools that increased the effectiveness of maintainers without incorporating the work practices of experienced maintainers into the design of the tools. We have done this through participatory design and the study of work practices.

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 have pursued, with some related results.

WORK PRACTICE STUDIES OF SOFTWARE ENGINEERS

Workplace Characteristics

We are studying a group that maintains one of the key products of their 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.

Studies

When we began our strides, we found there was no clear 'cataloging' as such of exactly how SEs go about solving problems. Consequently, 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. The methods and results of each of these studies is briefly presented in the next four subsections (see [13, 10]). After using the data to understand individual activities we then analyzed it further to discover work patterns: Our preliminary analyses of these is in the section entitled "Synchronized Shadowing to Discover Work Patterns" and a discussion of the tool that we developed using this information follows.

Questionnaire Study

We began our research by administering a web-based questionnaire. The questionnaire covered many different aspects of the SEs' work. Here we report their answers to two questions about what they spend their time doing.

In response to the first question, the 6 (of 13) SEs who responded had to decide how to describe their work. The question was open ended, meaning they 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 include consulting, both answering and asking questions, working with the hardware, testing, designing, and fixing bugs.

In another question, we asked the SEs how they divided their time: They reported an average of 57% of their time was spent fixing bugs, and 35% of their time making enhancements to the system.

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

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

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.

Individual study

We have been following one SE longitudinally from the time he 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 expert, we have found that it makes more sense to meet 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 much 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 new to neither maintenance nor 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 categorized the shadowing events into 14 distinct categories which are described in Table 1. Each of B's events was then classified as belonging to one of these categories. This data reflects 14 distinct 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 occasions. 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 because searching was an activity that also 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 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 supposing to find that editing code, compiling, and management were each only done on only 1 of the 14 days.

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

Thus overall, in terms of both 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, figures most prominently. A significant amount of effort was also expended interacting with the hardware and studying the source code.

Group study

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

We have 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 one of the most expert members 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 categorized into the 14 distinct categories that are described in Table 1. Each of the events was then classified as belonging to one of these event categories. 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 357 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 or the debugger, searching, and changing the source code were done on 36, 32, 31, and 30 occasions respectively. 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 prominence 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. Editing and compiling also seem important. This concurs with what we would expect in that their work revolves around the source code.

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 had to exclude 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 individual 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.

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 have focused 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 the section “Work Practices, Work Patterns, And Tool Design”).

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.[1] 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 second researcher would record 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 “Work Practices, Work Patterns, And Tool Design”).

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.

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.

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.

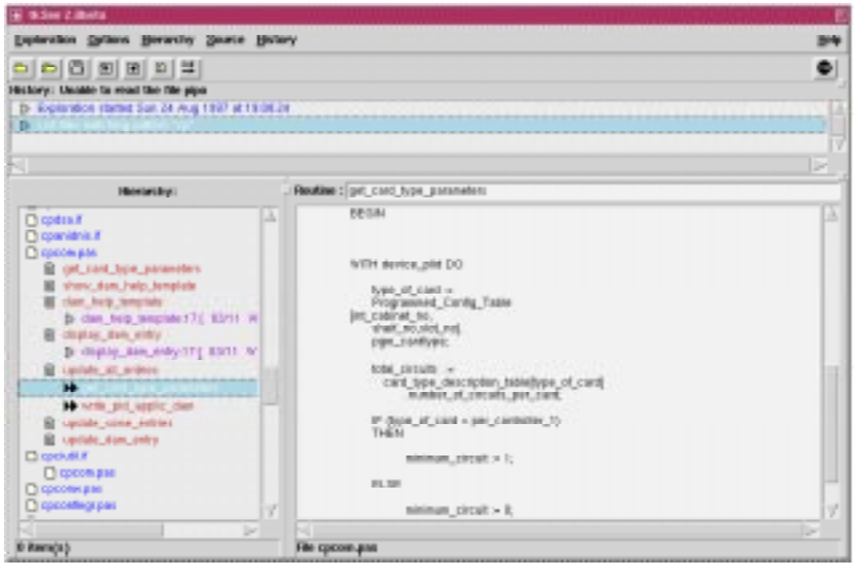
The next section describes how we have developed a tool that helps SEs more effectively achieve the goals implicit in the above patterns.

DEVELOPING A TOOL USING THE RESULTS OF WORK PRACTICE STUDIES

In this section, we discuss how we have 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. We performed two

[1] Unfortunately, this level of observation cannot be done by automatic logging of keystrokes and mouse movements.


Explanation
Options
Hierarchy
Source
History
Help
History: Unable to read the file pipe
Exploration started Sun 24 Aug 1997 at 19:06:24
Hierarchy:
Routine : get_card_type_parameters
File cpcom.pas

```
BEGIN


WITH device_pist DO

   type_of_card :=
   Programmed_Config_Table
[int_cabinet_no,
   shelf_no,slot_no]
   pgm_conftype;

   total_circuits :=
     card_type_description_table[type_of_card]
        number_of_circuits_per_card;

   IF (type_of_card = pgm_controller_1)
   THEN

        minimum_circuit := 1;

   ELSE

        minimum_circuit := 8;
```

Figure 1: An example of the Tksee main window.

approaches in parallel: participatory design and work practices.

Participatory Design: The First Release

The participatory design approach rapidly yielded the first version of our tool: 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 version developed after we incorporated the results of our work-practice studies.

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 earlier 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[2]. 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 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.

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

[2] Although a rudimentary version of history was available in the original tool, it was little use since it was not persistant nor automatic enough

incorporate some of the facilities we have 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 have 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.

Our work also allows us to compare participatory design with work-practices-based design. We feel that both are important and should be used together, however in our case no amount of brainstorming and discussion with users gave rise to ideas about history or the details of how to integrate grep. It was only the work-practices studies that allowed us to make the extra step – and obtain considerably higher rates of tool adoption as a result.

CONCLUSION

This paper has described experiences with several techniques that can help SEs and user interface specialists to develop systems that are not only usable, but are also used.

We have demonstrated that by studying work practices, it is possible to develop tools that meet the needs of users, and are adopted by those users. Furthermore, this can be done without the need to perform cognitive modeling.

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 have 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. Bédard, J., & Chi, M.T.H. Expertise. *Current Directions in Psychological Science, 1* (1992*)*, 135-139.
2. Berlin, L. Beyond program understanding: A look at programming expertise in industry, in *Empirical Studies of Programmers, Fifth Workshop*, (New Brunswick, NJ, 1991.)
3. Beyer, H., & Holtzblatt, K., Apprenticing with the customer. *Communications of the ACM 38* (1995*),* 45-52.
4. Blomberg, J., Suchman, L., & Trigg, R., Reflections on a Work-oriented Design Project. *Human Computer Interaction 11*, (1996), 237-265
5. Curtis, B. By the way, Did anyone study any real programmers. In *Empirical Studies of Programmers, '86*, 256-261.
6. Ericsson, K.A., & Simon, H.A. *Protocol Analysis: Verbal Reports As Data*. MIT Press, Boston MA, 1984.
7. Harper, R., Sellen, A.. Collaborative tools and the practicalities of professional work at the International Monetary Fund, in *Proceedings of CHI '95* (Denver CO, May 1995), ACM Press.
8. Hutchins, E.. Cognition in the Wild. MIT Press: Boston, MA, 1996.
9. Kraut, R. & Streeter, L. Coordination in large scale software development. *Communications of the ACM 38, 3,* (1995) 69-81.
10. 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), to appear.
11. Nardi, B. Studying context: A comparison of Activity Theory, Situated Action Models, and Distributed Cognition In B. Nardi (ed.), *Context and Consciousness : Activity Theory and Human-Computer Interaction*. MIT Press, Boston, MA, 1996
12. Newell, A., & Simon, H. Human Problem Solving. Prentice Hall:Englewood Cliffs, NJ, 1972.
13. Singer, J., Lethbridge, T., Vinson, N. and Anquetil N. An Examination of Software Engineering Work Practices, in *Proceedings of CASCON '97* (Toronto, November 1997), to appear.
14. Sellen, A., Harper. R. Paper as an analytic resource for the design of new technologies. In *CHI '97.*
15. Snelling, L, Bruce-Smith, D. The work mapping technique. *Interactions*, *4, 4*, (1997) 25-31.
16. Sonnetag, S., Brodbeck. F., Heinbokel, T., & Stolte., W. Stressor-burnout relationship in software development teams. *J. Occupational and Organization Psychology*, *67*, (1994), 327-341.
17. Vicente, K and Pejtersen, A. *Cognitive Work Analysis*, in press.
18. von Mayrhauser, A and & Vans, A., Program Comprehension During Software Maintenance and Evolution, *Computer* Aug. 1995, 44-55.
19. Walz, D., Elam, J., Curtis, B. (1993). Inside a software design team: Knowledge Acquisition, sharing, and integration. *CACM 36, 10*, (1993), 63- 76.