Understanding Software Maintenance Tools: Some Empirical Research[1]

Timothy C. Lethbridge
tcl@site.uottawa.ca
School of Information Technology and Engineering
University of Ottawa, K1N 6N5, Canada

Janice Singer
singer@iit.nrc.ca
Institute for Information Technology
National Research Council, Ottawa, K1A OR6, Canada

Abstract

In this position paper we discuss research into tools for software maintenance, where tools include everything from large scale integrated CASE products down to simple one-function commands or features. The objective of this research is to learn more about software engineers' work with and needs for tools, so that they can become more productive. We discuss several categories of questions to be asked in this research, and various methods for gathering data. We present some data from our research to date which indicates, among other things the need for efficient software exploration mechanisms. Finally we propose some directions for collaborative research.

Keywords: Software maintenance tools, work practices, studying software engineers

1. The software maintenance topic and problem we are addressing

A key part of our research is to study tools to help software engineers (SEs) perform maintenance. In this paper, we discuss the problem of understanding what tools and tool features are or would be useful. Our goal is to help software engineers become more productive.

In this position paper, we follow the format requested for this workshop; this also happens to correspond very well with the GQM (Goals-Questions-Metrics methodology) [1]. Our *goal* is defined in the previous paragraph. In section 2, we list *questions* that we want answered. In section 3, we list *metrics* and observations that would help answer the questions. In sections 4 and 5, we describe how to gather and interpret this data. We include some results from our work in section 6. Finally, in section 7, we propose a collaborative approach to help achieve our goal.

This work fits within the broader scope of our project, whose goal is to help software engineers to improve their productivity. Other discussions of our project can be found in [2, 3, 4, 5].

1.1 What is a tool?

When we use the word 'tool' we include anything functional which can help the software maintainer solve maintenance problems. In general, such tools are computer programs or parts of computer programs. We consider tools to be organized into hierarchies: Many programs provide multiple services or 'features'; each of these may be a tool in its own

[1] This work is supported by NSERC and sponsored by the Consortium for Software Engineering Research.

right. Many of these tools can be further subdivided into successive levels of finer-grained tools.

Not all ‘features’ of a higher-level tool will be considered tools. We only consider something a tool if a software engineer uses it when performing some clear software engineering task. For example, a feature of a program that merely listed the authors of that program would not be considered a tool.

To illustrate the tool hierarchy, consider a tool that many software engineers use: Emacs. Emacs is a high level tool; it also has many features that we consider lower-level tools in their own right, one of which is the editor. The editor, in turn, has a series of still lower level tools built into it (e.g. various kinds of search commands).

For the rest of this paper, we only use the general term ‘tool’ and do not distinguish between high-level and low-level tools or features.

2. What we expect to learn

As a result of this work, we expect to obtain answers to the following questions:

Q1. What tools do SEs use and for what tasks?
Q2. What difference do the tools make to the SEs’ work?
Q3. Why software engineers choose to use or not to use particular tools?
Q4. What new tools or improvements to tools could be made?
Q5. How can tools be introduced to software engineers?

3. What we can observe and measure

Table 1 shows some basic data we can gather from each software engineer; the columns of the table are the questions that the metrics can contribute to answering. Gathering these metrics would be in addition to obtaining simple subjective answers to the questions.

	Q1	Q2	Q3	Q4	Q5
M1. Which tools are used in a given period?	**	*		*	*
M2. Number of times each tool was used?	**	*		*	*
M3. Elapsed time spent using each tool?	**	*		*	*
M4. Goals and tasks for particular usages of a tool?	**		*	*	
M5. Lists of positive attributes for tools in general or of particular tools?		*	**	**	
M6. Lists of negative attributes of tools in general or of particular tools?		*	**	**	
M7. Time it takes to perform a given task with a given tool?		*	*	*	

Table 1: Some metrics that can contribute to answering the questions posed in section 2. Two asterisks indicates a strong contribution; one asterisk indicates a lesser contribution.

4. Collection of data

We have been studying a team of software engineers since mid-1996. Over that time we have been refining our techniques. The following are the data collection techniques that are currently part of our repertoire[2].

DC1. Questionnaires: We use a web-based questionnaire to obtain basic answers to many of the above questions from a team of software engineers.

DC2. Interviews with software engineers

a) General structured interviews: 60-90-minute sessions where we ask SEs general questions about their work, using a 10-page protocol. We are able to be more open-ended than with questionnaires, although interviews take much more time, especially for analysis. So far we have interviewed 12 software engineers using our most recent protocol, and a similar number using an earlier protocol. This is the only technique where we have worked with people outside the core team we are studying.

b) Regular debriefings: 30-60 minute sessions where we ask SEs to describe what they have been doing since the last debriefing. We work with the same SE every few weeks in order to obtain a long-term perspective on his work. We have been primarily following two software engineers in this manner for over 7 months. This is our most labor intensive technique – particularly with regard to analysis of the data. We have started to use the MacShapa tool to analyze our data.

c) Tool reviews: 30-60 minute sessions where we ask SEs a series of questions about a specific tool, and all its subtools. This allows us to systematically evaluate all aspects of a tool - even features that cannot readily be logged because they are not 'commands'. We have only just started using this technique.

DC3. Observation sessions

a) Observations of real work: 30 minute sessions where we observe the Ses regular work (assignmed by their managers) and note each activity. The Ses work should be as unaffected as possible by our watching: We ask them to use the same tools and techniques as if we weren't there. We have done this every time we held a debriefing (DC2b)

b**) Observations of 'artificial' tasks:** Similar to usability evaluations, where we try to determine weaknesses in tools by observing difficulties software engineers have performing a prescribed task. We have only just started using this technique in conjunction with tool reviews (DC2c).

DC4. Automated logging of tool use: We record detailed usage of tools used by an organization (tools developed by us, as well as basic operating system tools and third-party products). We have been using this technique for four months, although the first month's data may be unreliable due to technical problems.

Table 2 indicates which data collection techniques we believe are useful to obtain which data

[2] It is important to note that while using these techniques, we also gather data about other aspects of software engineering (e.g. software engineering process and training requirements). In this paper, we only consider tool use, which is the most important thrust of our research.

5. Interpretation of data

In all but technique DC2a, we have been working with a single small group of software engineers. We therefore cannot make very broad claims. In particular, the following variables may restrict the applicability of the results:

- The team we have worked with have a well defined process that has developed as part of the organization's internal culture. It may well be the case that data from other organizations may be different.
- The team is working on complex real-time software.

Although the sample size and variables prevent us from drawing statistically significant conclusions about the broader population of software engineers, we can nevertheless apply what we learn to this team. It is our hope that if we make a significant improvement to this team, then we will be able to take the ideas and test them in the broader context.

	Q1 What tools are used?	Q2 What difference do they make?	Q3 Why are they chosen?	Q4 What improve-ments needed?	Q5 How to introduce them?
DC1 Questionnaires	+	+	+	++	
DC2a General interviews	+	++	++	++	+
DC2b Regular debriefings	++	++	++	+	
DC2c Tool reviews	+	+	+	++	+
DC3a Work observation	++	+		+	+
DC3b Structured observation	+	+		++	
DC4 Automated logs	++	+			

Table 2: Data gathering techniques that can provide information to help answer questions of interest in this research

6. Some interesting data and observations.

Space permits us to only highlight some key findings:

There was strong evidence that software engineers in our study desire tools to help them explore software. They use such tools heavily already and want improvements. There was evidence for this from all of the data collection techniques.

- After editors, the most heavily used tools as measured using automated logging were grep-like searching tools.
- As indicated in appendix 2, users who were interviewed or answered questionnaires indicated various kinds of analysis tools on their wish-lists.
- When observing users, we found them to be exploring software as much as editing.

When we asked about positive features of tools, the most important aspects of tools were ease of use, useful features and rapid response time. Tool reviews and observation sessions add to our confidence that these factors are of importance. A summary is in Appendix 1.

When we asked about negative features of tools, the main complaints were lack of integration with other tools, and the wrong mix of features. Data is in Appendix 2.

We had significant difficulty introducing new tools. One technique that seems to hold promise is to train a single individual (in our case somebody new to the organization) and have him or her act as a consultant for others. The consultant can recommend use of the tool when others in the team run into difficulties that the tool could help them solve. Without the consultant, software engineers will be either unlikely know about the tool or not willing to try to learn it. In our experience, a major new tool went unused for many months until word started spreading from *within* the team that the tool was truly useful. Our own earlier efforts to advertise the tool were not effective.

7. Parallel or subsequent studies that could be performed

The above techniques are helping us discover useful information about software engineering tools. Unfortunately there are a number of difficulties in reaching strong scientific conclusions. The first is that studying tool use is time-consuming and requires trained people. This means that individual researchers will find it difficult to gather enough data. The second problem is that there are significant variables about which to be concerned.

The solution is collaborative studies with several groups of researchers each working with different groups of software engineers.

1. Researchers with contacts in a variety of companies could distribute questionnaires and arrange for logging tool use in these companies.

2. Researchers could observe different groups of software engineers using the same tools or performing similar tasks.

3. Researchers could use the same structured interview format with a variety of groups of engineers.

To make such collaborative research function effectively, the following would be necessary:

• The collaborating researchers would have to agree on the questionnaires, interview protocols etc. This could be somewhat difficult, because there are very many possible questions to ask, yet the number of questions must be limited.

• For interviews and observation sessions, researchers would have to physically work together on at least a few initial studies to ensure that their methods are similar. Effectively, each researcher would train the others regarding their methods.

References

[1] Basili V.R. and Rombach H.D. (1988), “The TAME Project: Towards Improvement-Oriented Software Environments,” *IEEE Trans. Software Eng.* **14**, 758-773.

[2] Singer, J. and Lethbridge, T. (1996), “Methods for Studying Maintenance Activities”, *Proc. 1st Workshop on Empirical Studies of Software Maintenance*, Monterey, November.

[3] Lethbridge, T., Singer, J. (1996), “Strategies for Studying Maintenance”, *Proc. 1st Workshop on Empirical Studies of Software Maintenance*, Monterey. November.

[4] Sayyad-Shirabad, J., Lethbridge, T. and Lyon, S. (1997), “A Little Knowledge Can Go a Long Way Towards Program Understanding”, *Proc. International Workshop on Program Comprehension*, Dearborn, May.

[5] Singer, J., Lethbridge, T. and Vinson N. (1997), “An Examination of Software Engineering Work Practices”, *Proc CACON ‘97*, IBM Center For Advanced Studies, Toronto, November.

Appendix 1. Aspects of tools that software engineers found positive.

The following is combined data from 19 software engineers, using data collection techniques DC1 (questionnaires) and DC2a (structured interviews).

Usability issues	**19**
Short learning curve	1
Easy to use	5
Familiar	1
Flexible	1
Useful or necessary features	4
Integrated	1
Multi-user	1
Improves productivity	1
Responsive performance	3
Reliable	1
Specific tools or features liked very much	**2**
Modeling and manipulating	1
Artistic design tools	1
Virtual environment	1

Appendix 2. Aspects of tools that software engineers found negative.

The following is combined data from 19 software engineers, using data collection techniques DC1 (questionnaires) and DC2a (structured interviews).

Usability issues	**26**
Hard to learn	2
Hard to use	2
Interface inconsistencies	2
Inflexible	1
Hard to install	1
Too many features / too big	3
Too much GUI	1
Not powerful enough / features missing	4
Poorly integrated / incompatibilities	6
Restricted to one platform	1
Functionality is hidden	1
Bugs / unreliable	2
Specific tools desired	**11**
Better debugging	1
Finding memory leaks	1
Better exploration tools	3
Tools to analyze coding or naming conventions	1
Easy general web front end package	1
Automated testing tools	2
Tools to ease installation	1
Tool that doesn't require you to know what you are looking for	1
Miscellaneous	**4**
Too expensive	1
Physical environment	1
Tools are too obscure	1
Tools not part of culture	1