The Information Gathering Strategies of Software Maintainers

Carolyn B. Seaman
UMBC, Baltimore, USA
and
Fraunhofer Center – Maryland

Abstract

In examining software maintenance processes for improvement opportunities, an obvious choice is information flow. Obtaining accurate, up-to-date, and useful information about a system being maintained is a major task. It is also a difficult task because the sources of this information are often limited, inaccessible, or unknown. Clearly this impacts maintenance productivity - simply because of the time it takes to find and use the appropriate information sources - as well as the quality of system changes, which depends on the quality of the system information available. This paper describes the results of a survey study that aims to discover the information gathering strategies that software maintainers employ. The survey was completed by 45 software professionals in two different organizations with varying degrees of experience in maintenance. Their responses, on the surface, simply show that maintainers overwhelmingly rely on source code, which is not surprising. However, a deeper analysis of the responses show that other sources of information, in particular human sources, some types of CASE support, and lessons learned recorded from previous projects are at least as valuable than source code under some conditions. The results of this ongoing survey study are meant to determine a set of hypotheses about information gathering strategies, which will then be empirically evaluated in future studies.

1. Keywords

Software maintenance, survey study, information flow, information sources

2. Introduction

Software maintenance offers a significant payback opportunity for process improvement. It is commonly believed that maintenance consumes a large proportion of total system cost (often up to 85% as reported by Lientz and Swanson [4] in 1978; no more recent studies have attempted to update this finding). For this reason alone, lowering maintenance cost is a profitable goal. Further, improving the quality of maintenance practices could lengthen the useful life of a software system, thus providing a greater return on investment with respect to the system's original development costs.

In examining software maintenance processes for improvement opportunities, an obvious choice is information flow. Curtis et al. [3] have shown that information flow is a key factor in the success of software development and maintenance efforts. In maintenance in particular, obtaining accurate, up-to-date, and useful information about the system being maintained is a major task. It is also a difficult task because these sources of information are often limited, inaccessible, or unknown. Clearly this impacts maintenance productivity - simply because of the time it takes to find and use the appropriate information sources - as well as the quality of system changes, which depends on the quality of the system information available. The situation is further complicated by the growing trend towards component-based (or COTS-based, or package-based) systems because maintainers need information about the components as well as the system being maintained.

Currently, software maintainers rely on various sources for information about the system they are trying to modify, adapt, or update. These sources range from artifacts of the development process (requirements documents, design documents, documentation within code, test plans and reports, etc.) to user documentation (user manuals and configuration information) to the system itself (both the running system and the source code) to direct access to the system's original developers and current users. Many of these sources, especially development artifacts, are produced at least in part to facilitate maintenance. A lot of effort is required to both produce and maintain the documentation, but little effort has been made to determine how useful it is for maintenance.

The objective, then, of the work described here is to discover the ways that maintainers gain information about the systems they are maintaining. The eventual goal of this line of research, however, is to use these discoveries to design, implement, and test process improvements that

facilitate the best information gathering strategies for a particular context. The larger research project includes plans for future empirical evaluation of hypotheses grounded in the data collected by the study described in this paper.

3. Related Work

Probably the most famous and most often referenced empirical study of software maintenance practice is Lientz and Swanson's [4] survey of 69 software maintenance organizations in 1978. This in-depth study produced the often-cited figure of 50-85% as the proportion of system costs devoted to maintenance. However, this study is quite dated at this point and we have no way of knowing if these results still hold in today's maintenance environments. A more recent study of maintenance practice is Singer's [5] study of 10 maintenance organizations. This study, unlike the earlier one, was a qualitative study and focused more on process than outcomes. Maintainers were asked (in interviews) about how they went about doing maintenance and what resources they used and needed. The eventual objective of this study was to inform the development of maintenance tools.

Another example of using qualitative methods in studying maintenance was Briand et al.'s study of a NASA maintenance organization [2]. In this study, interviews were used, along with an organizational modeling technique called Actor-Dependency modeling [9], to gain insight into the organizational issues and problems in the maintenance environment. The results uncovered problems such as organizational bottlenecks and a lack of participation from certain stakeholders in key process steps.

Another study, by Sousa and Moreira [6], was intended to contribute to maintenance process improvement in financial institutions in Portugal. This in-depth study, which involved interviews, observation, and questionnaires, collected and analyzed data on maintenance effort, defects, and processes, as well as the information sources used by maintainers. The latter subject is not treated in any depth in the Sousa paper, but does provide a basis for comparison for the results presented below.

A slightly more recent study by Tjortjis and Layzell [7] specifically attempted to describe the learning strategies of maintainers. This study presented purely qualitative results, and focused more on strategies than information sources. Although a variety of organizations were studied, there was no attempt to tie variations in findings to variations in the organizations (e.g. application area, experience, etc.).

There is a major body of literature in program comprehension that addresses how programmers study source code in order to understand a software system. Much of this work ([1] and [8] are two good examples) focuses on the comprehension processes of maintainers in particular. Specifically, it addresses only one source of information, the source code, and it looks at how maintainers process information once they have obtained it. In contrast, the study described here focuses on information gathering activities, i.e. how maintainers get information, from whatever source.

4. Study Design

Before describing the design of the survey study in detail, it would be helpful to explain its research context. The larger line of research, of which the survey study is one stage, has the goal of improving how maintainers gather the information they need to carry out maintenance tasks. This line of research constitutes three stages, with each stage depending on the findings of the earlier stages. The last stage is the proposal of improvements to maintenance practice and the empirical demonstration of their effectiveness through case studies. Before that can be done, however, we must learn more about how maintainers currently gain information about systems. In particular, we need to learn what information sources are efficient, effective, and available in different situations. This is the focus of the second stage of the proposed work, which includes semi-controlled hypothesis-testing experiments that compare different strategies. But there is another prerequisite even to this type of study. It is not currently clear what variables and factors need to be considered in the study of information sources in maintenance. What is the best measure of the "effectiveness" of an information source? What constitutes an "information source"? What other factors (programmer background, application domain, system size, etc.) might affect how a maintainer uses an information source? If we do not gain a better understanding of these questions than currently exists, then the second stage of this research may be seriously misguided. Thus, the first stage of this work is an exploratory stage that addresses these and similar questions. The survey study described in the remainder of this paper is part of this exploratory stage.

The starting point for this study, as it is exploratory, is not a set of hypotheses, but a set of research questions:

- What do maintainers consider "information sources"?
- How is an information source judged to be "good"?
- What factors affect whether or not a given information source is "good"?

There is some literature that provides suggestions for answers to these questions, and this literature informed the design of the survey. For example, the source code is one obvious information source that is the subject of much of the program comprehension research [8, 1], as mentioned earlier. Other information sources suggested

by [5] are other people and execution traces. How to measure "goodness" of an information source has less support in the literature, but some factors that seem reasonable are the amount of time required to consult the source (its convenience) and whether or not the source provided the information the maintainer sought (its usefulness). Some intervening factors that are suggested by the literature are program structure (e.g. unstructured vs. structured vs. object-oriented) and programmer background [1]. Other factors that seem obvious are the maintenance task being performed, the perceived quality of different sources, and the availability of different sources.

This paper reports on survey results from the first two organizations that responded to the maintenance survey, Computer Sciences Corporation (CSC) and the United States Internal Revenue Service (IRS). The CSC maintainers that responded to this survey were all associated with CSC's Space and Earth Technology Systems (SETS) program. Most of these respondents were CSC employees, but a few were from subcontractors to CSC. All of the respondents from the IRS were IRS employees, working on maintaining a variety of systems for enforcing US tax law and managing tax data. Although participation in the study was not mandatory, 38 responses were received from CSC, resulting in a response rate of 75% at that organization. At the IRS, however, the response rate was quite low. Although the survey was made available to over 50 maintainers, only 7 IRS responses were received. All respondents, both from CSC and the IRS were located in the suburban Maryland area.

The survey underwent a significant review and tailoring process before being distributed to maintainers. The first drafts of the survey were based on the literature described above, and were then reviewed by several experts in survey design. Then managers and process engineers at both CSC and the IRS scrutinized the modified survey, made some terminology changes, and tailored it to support each organization's process improvement goals. Finally, one or two maintainers in each organization pretested the latest version and further debugged it for terminology and clarity. We also timed the maintainers during the pretest, determining that it took about 30 minutes to complete the survey. The final version of the survey was then sent via email (in the form of an MS Word document) to approximately 50 maintainers in each organization by their managers.

There were four separately managed projects at CSC and two at the IRS participating in the study. At CSC, maintainers were given a week to complete the survey, print it out, and return it to a designated person in each project group. These designated persons then packaged together the paper copies and mailed them to the author for analysis. After the surveys were received, interviews were conducted with four of the respondents (about 10%) in order to provide further clarification and elaboration on the survey questions, and also to ask a few specific questions that had arisen from the initial analysis of the survey data. These interviews were each about an hour in length and were audiotaped. At the IRS, the surveys were distributed at two different times to two different groups, and maintainers were given two weeks to complete the surveys. The completed surveys were sent via email directly to this author's research assistant for analysis. No follow-up interviews have been conducted to date at the IRS.

The survey itself begins with a few brief questions about the respondent's experience in software development in general and in maintenance in particular. Then there is a section asking some characterization questions about a representative maintenance project (platform, language, age of the system, etc.). The next part of the survey presents a list of information sources that could potentially be used by software maintainers (e.g. source code, original developers, CASE tools, etc.). The respondent is asked to rate each of the sources in terms of usefulness, convenience, availability, and frequency of use on a scale of 1 to 5. The respondent is also asked to add any information sources not on the list. Finally, a series of questions asks the respondent which information sources are most frequently used, least frequently used, and why, as well as which currently unavailable sources they would most like to have (their "wish list"). A version of the survey can be seen at http://research.umbc.edu/~cseaman/maintenance.htm, as well as more general information on this project.

The analysis of the survey responses and interview notes included both quantitative and qualitative techniques. The ratings of the information sources on the survey were grouped in various ways according to different attributes (system age and type, for example) and averaged. This simple quantitative analysis helped answer questions about which sources were rated higher on which criteria (usefulness, convenience, etc.) under what conditions. The qualitative analysis utilized the software tool NVivo™ (developed by QSR International Pty Ltd, Australia), which is used to code textual data and find trends and insights.

5. Results

The results of the survey study are presented below for each organization. Then interesting differences and similarities between the two organizations are highlighted. In the following section, the findings are discussed in more detail.

5.1. CSC Results

The CSC respondents to the survey were highly experienced; all had at least 2 years experience in software development and maintenance but the vast majority had more than 10 years. About a third of the respondents had been involved in the original development of the system they were maintaining. The systems under maintenance varied in age but most were more than 5 years old. About half of the systems described were Unix/C/C++ systems, about a quarter were mainframe (usually Fortran) systems, and the rest were some mixture of platforms and languages.

Not surprisingly, source code was overwhelmingly the highest rated information source in terms of usefulness, convenience, and frequency of use. Respondents used terms like "does not lie", "the only truth", and "100% accurate" to explain why they rely so heavily on source code. On a related note, respondents varied a good bit on their view of comments in the source code. Some felt that they were very helpful while others felt they were largely inaccurate or irrelevant.

Related to source code, execution traces were also used frequently by some respondents, although the responses on this source were mixed. About a quarter of the respondents said that this source was not available to them. They were used most frequently on older systems (more than 5 years old), and they were most convenient to use on mainframe-based systems. Those who used execution traces frequently said that they were the most accurate and efficient way to understand a problem, particularly in combination with looking at the source code. They are especially useful for isolating complex bugs. However, those who did not use execution traces said that there were no tools to support them, thus they were too hard to configure and too time-consuming.

Many of the information sources that the respondents were asked to comment on were human (e.g. original developers of the system, writers of the original system requirements, current users and operators, other maintainers and experts, etc.). When rated on usefulness, all the human information sources rated higher than most other sources (except, of course, source code). Respondents said that these human information sources were "handy" and "accurate", and were "unique" sources of otherwise undocumented information. Information that could only be found in human sources included specific project and system experience, design rationale, and previously solved problems.

Good examples of highly useful, but largely unavailable, human information sources were the developers of the original release and the writers of the original system requirements. About half the respondents said that the requirements writers were not available to them (availability was a little better for the original developers). Respondents commented that these people were often busy on other projects, or could only be bothered when the situation was urgent. In a few cases, there were problems even finding out who the original developers or requirements writers were. People who had been maintaining a system for a long time, or who were involved in the original development of the system, tended to have better access to original developers. Not surprisingly, there were more problems with the availability of original developers and requirements writers on older systems, as compared to younger systems (more or less than 5 years old, respectively). Several cases were cited where allowance was made (funding, management permission, etc.) for people involved in the original release of the system being maintained to join the maintenance effort. Maintainers found this very useful but said that it happens rarely.

Those who did have access to the requirements writers and original developers rated them above average on usefulness. They were also the most often listed sources on respondents' "wish lists". Respondents said that the writers of the original requirements would be a useful resource because they have such an in-depth knowledge of the requirements, and thus the original intent of all aspects of the system. Original developers, on the other hand, were the best source for interpreting the code, especially in the absence of reliable comments, and for discovering design rationale.

Many respondents also cited current users and operators of the system under maintenance as crucial information sources, even if they were not consulted very frequently. System users and operators were useful when a maintainer was trying to isolate a defect, designing a fix or enhancement, or trying to understand how the system is used (which is sometimes different than it was envisioned). Respondents were generally favorable about the convenience and availability of users and operators. On many projects, the maintenance team was in very close proximity to the current users and operators, and this was cited as a major benefit. However, a few respondents noted that users and operators lacked the necessary knowledge of the system design.

CASE tools in general were not rated particularly high in terms of availability, usefulness, convenience, or frequency of use, although the results were more favorable among those maintaining younger systems and those who have been maintaining a system for less than 4 years. Integrated development environments, however, were described much more favorably. About a third of respondents said that such an environment was not available to them, but the others rated the environments as above average on all counts. These results are difficult to interpret, as the meaning of "integrated development environment" was different in different projects. Some projects had available to them a development environment consisting of a set of simulators that mimicked the operational environment of the software

being maintained, while others had a set of Java development tools (compilers, debuggers, editors) that they considered their "IDE". Others were referring to a set of tools used to draw, analyze, and generate code from data flow diagrams. Configuration management systems were generally rated low on usefulness, convenience, and frequency of use, although they were generally available.

Lessons learned reports were viewed rather differently by different groups of respondents, depending on whether or not they had such reports available to them. Several of the respondents who did not have lessons learned available to them (4 out of 12) wished that they did because they felt such reports would help them avoid previously-committed errors. It was also noted that lessons learned, if they were done well, could be used to capture the project and system experience that currently is only available directly from people (in particular original developers and requirements writers). However, half the respondents who did have lessons learned reports available to them indicated that they were among the least frequently used information sources because they contained no useful information, were not up-to-date, were not written well, and were not easily accessible. It should be noted, however, that one respondent listed lessons learned as one of their most frequently used information sources because they describe "unpredictable" situations. Another observation by one respondent was that properly recording and archiving lessons learned was a budgetary issue; i.e. there were not enough resources to do it well. In general, lessons learned were a bit more available and more frequently used on mainframe systems and less available on younger systems.

5.2. IRS Results

The IRS respondents to the survey were highly experienced; all had at least 10 years experience in software development and/or maintenance, with only two respondents having less than 5 years experience specifically in maintenance. Very few of the respondents had been involved in the original development of the system they were maintaining. The systems under maintenance varied in age but most were less than 5 years old. With one exception, all of the systems described were mainframe-based systems, largely written in COBOL.

As with CSC, source code was rated highest on usefulness, convenience, and frequency of use, and it appeared more than any other information source on the list of the three most frequently used sources. Accuracy and availability were cited as reasons. The only respondent who did not have source code available placed it on their "wish list".

In general, the data shows that the IRS respondents do not feel a need to consult more than they do with other people related to the maintenance effort. Recall that respondents were asked about people involved in the original development of the system they were maintaining (developers and requirements writers), people currently involved with the system (users, operators, customers), and colleagues (other maintainers and experts). Although the respondents said that a number of these human sources of information were not available to them, none of them listed these types of sources on their "wish list". It should be noted, however, that in general less experienced maintainers found human sources of information more useful and convenient (and used them more frequently) than more experienced maintainers.

Developers and requirements writers of the original system release were said to be available in most cases, somewhat useful, and in many cases convenient, but they were not consulted very frequently. One reason cited for this was that the expertise of these people was out of date due to constant changes in the systems. It was also mentioned that often the code was not documented with the names of the original developers, making it extremely inconvenient to find them. Respondents not involved in the original release of the system under maintenance were less likely to have access to the original developers and requirements writers, not surprisingly. Along the same lines, people who were involved in the original release found the original requirements writers a more useful source of information.

More useful, convenient, and frequently used human sources of information were the people currently involved with the system being maintained. This includes users, customers, and operators. Although, on average, original developers and requirements writers were rated a little bit higher on usefulness, the respondents clearly preferred to consult with people with a more current understanding of the system. All respondents indicated that users, customers, and operators were available to them. Current system users were described as the "best source to verify system functionality and the need for enhancements." Current operators were described as knowledgeable about all aspects of the system, especially the database structure. Customers were found to be more convenient and were more frequently used by less experienced maintainers.

The survey respondents were also asked about their use of colleagues as information sources, in particular "other maintainers" and "other local experts". Many of the respondents indicated that they did not have access to such people, but those who did rated them high on usefulness, convenience, and frequency of use. In fact, other maintainers and local experts were the highest rated human information source, on average, with respect to usefulness. However, none of the respondents who did not have other maintainers and experts available put them on their "wish list". One could conclude, then, that maintainers who have other maintainers and experts readily available appreciate them as information sources,

but those who don't do not see a need for them. Some respondents noted that they consulted other maintainers frequently to brainstorm solutions, and that other maintainers usually provide accurate information. Experienced maintainers were more likely to rely on their fellow maintainers.

Respondents were asked to comment on three different classes of tools: configuration management (CM), CASE tools, and integrated development environments (IDEs). Most respondents said that they had CM support and found it fairly convenient to use. CM, on average, was rated moderately useful and some respondents identified it as their most frequently used information source. However, one respondent, whose maintenance work was quite different from the rest (configuring a PC-based tool), found CM unhelpful because it was "always an afterthought and usually murky". Only one of the two respondents without CM support said that they wanted it. Those who used CM frequently said they did so because its use was required, or because it contained the most accurate system documentation.

More than half the respondents, all of them more experienced maintainers, said that they did not have CASE tools available for their maintenance work. Half of these respondents, furthermore, put CASE tools on their wish lists. Design tools and analyzers in particular were mentioned on some respondents' wish lists. Those who did have CASE tools available, though, said that they did not use them very frequently and did not find them particularly useful or convenient.

System documentation was universally available and among the most frequently used sources of information for maintainers, along with system files. In one case, system documentation took the place of source code when the source code was not available. Those respondents who were not involved in the original release of the systems they were maintaining found system documentation especially convenient. However, one respondent indicated that he/she did not rely on system documentation because the examples used were not realistic.

Lessons learned reports or documents are clearly something that many of the respondents would find useful if they were available. Among respondents who said that they did have some sort of lessons learned available to them, they rated them the second most useful information source (after source code). However, lessons learned were also reported to be one of the most unavailable, least convenient, and least frequently used information sources. Almost all the respondents who did not have access to lessons learned said that they wished they did, mainly to avoid "repeating mistakes", "duplicating effort and multiplying errors". The only respondent to say that he/she used lessons learned frequently also was the only respondent that found the lessons learned convenient to use.

5.3. Similarities and Differences

Clearly, there is a heavy dependence on source code as an information source in both organizations surveyed. However, there are numerous differences in terms of which other sources are viewed as most useful and convenient, and which are used most frequently. One big difference is in how maintainers in the two organizations viewed other people as sources of information. CSC maintainers rated all human sources of information more useful than all other information sources, other than source code. All types of people (those involved in the original development and current use of the system, as well as colleagues) were consulted frequently when they were available (although there were some major availability problems). At the IRS, on the other hand, people were not so highly regarded as sources of information. The IRS maintainers seemed to have fewer human sources of information available to them, but did not express a need for more access to people.

In particular, people involved in the original development of the system under maintenance were largely unavailable, but highly sought after, at CSC. At the IRS, original developers and requirements writers were mostly available, but were not consulted frequently, at least in part because their knowledge and expertise were seen as out of date. One might conclude from this that the systems being maintained at the IRS were older than those at CSC were, but this is not the case. Most of the systems described by IRS maintainers were less than 5 years old, while the CSC systems were of all ages (but more than half were more than 5 years old). Rate of system change may be the more relevant factor here, but that was not an issue addressed by the survey, unfortunately.

CSC and IRS maintainers did agree on their use of current system operators, users, and customers as valuable sources of information. In both organizations, these people were generally available to maintainers, and were cited as useful for such things as understanding system functionality, interpreting change requests (including the rationale for the requests), and designing enhancements. One interesting difference, though, was the frequency with which operators, users, and customers are consulted in the two organizations. CSC maintainers said that these people were very useful when consulted, but there was not a need to consult with them all that often. On the other hand, IRS maintainers consulted with them quite frequently. System operators, users, and customers for the IRS systems were generally part of the same organization as the maintainers, while the CSC system operators, users, and customers were generally in a

different organization (although sometimes they were co-located with the maintainers).

The other category of human information sources addressed in the survey was the maintainers' colleagues ("other maintainers" and "other local experts"). At the IRS, maintainers who had such colleagues readily available appreciated them as information sources, but those who didn't did not see a need for them, as evidenced by the fact that colleagues did not appear on any "wish lists". The CSC maintainers rated their colleagues just as useful as did the IRS maintainers, but at CSC these colleagues were not consulted nearly as frequently. The availability of colleagues was better at CSC, but CSC maintainers who did not have colleagues available felt they would be useful more often than IRS maintainers without available colleagues.

The difference between the two organizations in attitudes towards CASE tools is also interesting. Neither organization had a high availability of CASE tools, but at the IRS there was an expressed desire for such tools, especially design tools and code analyzers, that was not expressed among CSC maintainers. However, in both organizations, maintainers with CASE tools available to them did not rate them very high on usefulness or convenience, and did not use them often. The IRS maintainers are more positive about their CM infrastructure than CSC maintainers, according to the survey results. This may be related to the fact that system documentation is heavily relied upon at the IRS, and the CM system seems to be the best source for that documentation. At CSC, use of system documentation was somewhat mixed, and it did not seem to be linked to the CM system.

It's interesting to compare the findings on lessons learned in the two organizations. Lessons learned were not available to about a third of the respondents in both organizations, and both sets of maintainers rated them very low on convenience and frequency of use. However, the CSC maintainers rated lessons learned as far less useful than did the IRS maintainers. In fact the IRS maintainers who had lessons learned available found them highly useful, and those that didn't have them available generally wished that they did. At CSC, however, most maintainers without access to lessons learned did not say that they wanted them. So the major barrier to the use of lessons learned at the IRS seemed to be availability and convenience, while at CSC there was also a feeling that the lessons, even if one could find them, would not be very helpful.

6. Discussion

Given the severe inequity in the numbers of survey responses from the two organizations (38 from CSC, 7 from the IRS), one useful way to synthesize them is to use the IRS data to indicate which of the findings from the CSC data might be generalizable and which are probably not. That is the perspective we will take in this section. However, it is worth saying that, even considering the responses from both organizations, the findings of this study are limited by the scope and the types of data collected. The findings presented here are based strictly on the survey responses and interviews in two organizations. An attempt has been made to collect information that will help contextualize the findings (e.g. system age and type, maintainer experience, etc.), so the scope of applicability of the findings will be evident.

One CSC respondent very neatly summarized the major findings of this study: "[What I need is] somebody to talk to and something to help me understand the code". Better access to human sources of information and better (and truly useful) tools for navigating and interpreting code seem to be the major needs of software maintainers at CSC. Looking at this in light of the IRS data, however, adds some complexity to this finding.

The use of source code as the primary source of information in both organizations is not surprising (this was also a finding of Singer's study of software maintainers [5] as well as Sousa and Moreira's study of Portuguese financial software maintainers [6]), but there are indications that the examination of source code could benefit from better tool support. No great enthusiasm for CASE tools is evident in the survey responses from either the IRS or CSC, but many CSC maintainers evidently do rely on execution traces to get a better understanding of the code and some mentioned that better tool support would be helpful in this area. In fact, many of the CSC respondents who chose not to use execution traces said that the reason is that they are too difficult to configure, too time-consuming, and that there is no tool support for using them. Further, there was a positive attitude towards integrated development environments among the CSC respondents, which indicates that there is not a general hostility towards automated tools of all kinds. Although the nature of IDEs varied greatly among the CSC respondents, most were either highly domain-dependent, or were focused on providing very basic services (editing, debugging, compiling, etc.). The IRS findings support the conclusion that the real need for tools lies in supporting the more basic functions – configuring and managing execution traces (debuggers, source code analyzers), supporting administrative tasks like finding source code and other relevant documents (CM or document managers), and basic programming tasks (compilers, editors) - as opposed to more sophisticated and higher-functioned CASE tools. Clearly, maintainers in both organizations are not satisfied with the tools they currently have available to them. Tjortjis and Layzell [7] also found a reluctance to rely on existing CASE tools in their study of software maintainers.

The CSC survey responses indicate a need for better access to human sources of information, particularly the

writers of the original system requirements and the original developers of the system, but the IRS findings do not support this. It may be that this is highly dependent on how much the system under maintenance changes over time. With a highly volatile system, the knowledge and expertise of those involved in its original development becomes out of date very quickly. However, on more stable systems, these people can be invaluable. In both organizations, current users, operators, and customers of the maintained systems are highly useful sources of information. Also in both organizations, maintainers viewed their colleagues as useful information sources, but varied a great deal in how frequently they relied on them. This is consistent with, but not as strongly indicated as, Tjortjis' [7] finding that maintainers sought, but were unable to find, a good substitute for the experience of original developers and others familiar with the system being maintained. Neither documentation nor tools sufficed.

The potential exists for at least a partial substitute for human experience in the capturing and management of knowledge, in particular lessons learned. The survey responses are mixed in the case of lessons learned documents, but they indicate that there is unrealized potential in this area. Even at CSC, some maintainers have found them useful, and some who do not have access to them feel that they would be helpful. However, from the responses of most who have tried to use them, it's clear that the way that lessons learned are currently documented and disseminated at CSC could be improved. The IRS results reinforce the idea that lessons learned can be done well and can be highly useful. In both organizations, there are problems with convenience and availability, however.

7. Conclusions

"Somebody to talk to and something to help me understand the code."

This seems to be what maintainers most need and, in some cases, what they find most lacking in their maintenance environments. The CSC survey respondents repeatedly expressed a desire to have better access to people who had been involved in the development of the systems they were maintaining, particularly the original developers and requirements writers. Of course, such access is often impossible for a variety of reasons. Thus, while efforts can be made to increase access to development personnel, other avenues must be explored for capturing the experience of these people for the use of maintainers. Research into experience repositories, lessons learned, and even knowledge management, could be brought to bear on this problem of capturing original design rationale and system intent in a way that is useful to future maintainers and thus reduces their reliance on development personnel. This study implies that current technology in this area does not suffice. It also indicates, however, that this need (for knowledge and expertise from those involved in the original development of the system) is not universal. At the IRS, this type of knowledge was not seen as valuable. What was valuable was interaction with current users of the systems being maintained and, as these people were generally available, there does not seem to be a need to capture their expertise in a persistent way. The lesson here is that the types of experience and expertise that need to be made available to maintainers may vary considerably between organizations. Thus the challenge in providing infrastructure and support for experience sharing lies not only in the form of the automated tools, but also in choosing the content to match the needs of maintainers. In some cases, experiences from the original development of the system are most valuable, while in others, more current experience (e.g. from other maintainers or users) is more useful. Attitudes towards documentation seem to vary considerably, so this needs to be studied more locally.

A related issue is this study's implications for the design of lessons learned systems. At both organizations, but at the IRS in particular, there is some indication that lessons learned can be highly useful if done well. However, there are serious barriers in terms of convenience and availability that effectively hinders their use. The potential payoff, though, justifies pursuing this issue in future work.

Software tools for maintenance, at least in the environments studied here, are found lacking by maintainers. There seems to be a need for support for the most basic of tasks, not for highly sophisticated analyses. Simple debugging tools, to help navigate the code and trace the execution of different scenarios, would go a long way towards facilitating the maintainer's job. Also, support for basic administrative tasks, such as that provided by CM tools and some IDEs, seems to be appreciated in both environments studied.

The dominant strategy employed by software maintainers, simply put, is to find a knowledgeable person (where "knowledgeable" means different things in different environments) and to explore the source code, not necessarily in that order. These two major sources of information are intertwined and, when possible, maintainers prefer to rely on both rather than either in isolation.

Acknowledgments

This study would not have been possible, of course, without the help of the 38 CSC maintainers and 7 IRS maintainers, who provided information via the maintenance survey, and their managers. In addition, many thanks go to several CSC process engineers and IRS managers who facilitated the logistics of this survey,

and obtained authorization to distribute it. Thanks also go to Weimin Hou, Dr. Seaman's research assistant who was invaluable in organizing and doing initial analysis of the data. Thanks also go to the reviewers of this paper who provided very valuable comments. Finally, this study was funded by the National Science Foundation, grant CCR-9984047.

References

[1] Boehm-Davis, Deborah A., Robert W. Holt and Alan C. Schultz. "The role of program structure in software maintenance." *International Journal of Man-Machine Studies*, 36:21-63, 1992.

[2] Briand, Lionel, Yong-Mi Kim, Walcelio Melo, Carolyn Seaman, and Victor Basili. "Q-MOPP: Qualitative evaluation of Maintenance Organizations, Processes, and Products." *Software Maintenance: Research and Practice*, 10:249-278, 1998.

[3] Curtis, Bill, Herb Krasner, and Neil Iscoe. "A field study of the software design process for large systems." *Communications of the ACM*, 31(11):1268-1287, November 1988.

[4] Lientz, B.P., E.B. Swanson, and G.E. Tompkins. "Characteristics of Application Software Maintenance." *Communications of the ACM*, 21(6):466-471, June 1978.

[5] Singer, Janice. "Practices of Software Maintenance." *Proceedings of the International Conference on Software Maintenance*, Bethesda, MD, November 1998, pp. 139-145.

[6] Sousa, Maria João Castro, and Helena Mendes Moreira. "A Survey on the Software Maintenance Process." *Proceedings of the International Conference on Software Maintenance*, Bethesda, MD, November 1998, pp. 265-274.

[7] Tjortjis, C; Layzell, P. "Expert Maintainers' Strategies and Needs when Understanding Software: A Case Study Approach." *Proceedings of the IEEE 8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, IEEE Computer Society Press, 2001, pp. 281-287

[8] von Mayrhauser, A. and A.M. Vans. "Identification of Dynamic Comprehension Processes During Large Scale Maintenance." *IEEE Transactions on Software Engineering*, 22(6): 424-437, June 1996.

[9] Yu, Eric S. K. and John Mylopoulos, "Understanding 'Why'' in Software Process Modelling, Analysis, and Design." *Proceedings of 16th International Conference on Software Engineering*, May 16-21, 1994, Sorrento, Italy, pp. 159-168.