Observing Open Source Programmers' Information Seeking

Khaironi Yatim Sharif

Department of Computer Science
University Of Limerick
khaironiyatim.sharif@u.lie

Jim Buckley

Department of Computer Science
University Of Limerick
jim.buckley@u.lie

Keywords: POP-II.B Maintenance, Program Comprehension, Problem Comprehension

Abstract

Several authors have proposed information seeking as an appropriate perspective for studying software maintenance, and have characterized information seeking empirically in commercial software evolution settings. This paper addresses the parallel issue of information seeking in Open Source software evolution. Open Source software evolution exacerbates information-seeking problems, as team members are typically delocalized from the other members of their team.

This paper employs an analysis schema from our previous study (Sharif et.al 2008), generated through open-coding, to characterize information seeking in Open-Source, programmers' mailing-lists, the medium they predominantly use for communication. A preliminary study using this schema had several interesting conclusions. Specifically, the analysis has shown that Open Source programmers rely somewhat on documentation, that many of their information seeking activities are process orientated and that their information seeking goals change over time.

1. Introduction

Software maintenance and evolution are considerable parts of the software development process. The amount of software lifecycle effort consumed during this phase has been estimated to range between 60% and 80% of the entire lifecycle effort (Lientz et.al 1978, Mayrhauser et.al 1993, Pressman 2000, Zayour et.al 2001).

Maintenance itself can be divided into two general stages: "Understanding the program and actually performing the change" (Prechelt et.al 1998). The time invested by the programmer in order to achieve an understanding before a successful modification can consume a considerable part of the maintenance phase, with typical estimates of the effort consumed in studying the code ranging from between 50% and 90% of the entire maintenance effort (De Lucia et. al 1996).

Information-seeking has been defined as the searching, recognition, retrieval and application of meaningful content (Kingrey 2002). It has been recognized as a core subtask within this phase of software maintenance (Curtis et.al 1988, Seaman 2002, Singer 1998, Sim 1998). Sim (1998), for example, refers to maintenance programmers as task-oriented information seekers, focusing specifically on getting the answers they need to complete a task using a variety of information sources.

Within this research area O'Brien (2005) and Vaclav (2005) have studied the information-seeking *processes* of programmers during the maintenance of commercial software systems. In complimentary research, Singer (1998) and Seaman (2002), have studied the information *sources* that programmers use when seeking information. However, there have also been several empirical studies that aim to inform on the *types* of information sought by programmers in the context of software comprehension (Singer et.al 1998, Ko et.al 2007, Letovsky 1986, Pennington 1987, Good 1999, Wiedenback et.al 1991, O'Shea 2006). These studies focus on the information that programmers' need and the information that they find difficult to obtain during software maintenance, thus potentially informing the design of software tools.

Several of these studies focus their efforts on small programs or on student programmers (Letovsky 1986, Pennington 1987, Good 1999). Others (Seaman 2002, O'Brien et.al 2005, Ko et.al 2007) report on commercial software development in collocated teams. The work reported on here extends this research by focusing on delocalized Open Source (OS) development, in the tradition of O'Shea (2006), where the developer mailing lists of OS projects are analyzed to inform on the programmer's comprehension efforts. However, in contrast to O'Shea(2006), this work does not focus on the information available to programmers in programs alone (Pennington 1987). Instead it places no restrictions on the information source, using open coding as the basis from which to characterize all the information programmers seek from other programmers via mailing lists. The resulting schema is presented in section 2[1].

Subsequent content-analysis of two OS mailing lists with this provisional schema has resulted in several surprising findings. Specifically, in some mailing lists there seems to be an emphasis on system documentation, a finding which seems to contradict previous studies in the area (Singer 1998, Seaman 2002, Sousa et.al 1998) where documentation was largely perceived as untrustworthy. Likewise the findings suggest a process orientation in developer's information seeking, again contradicting the source-code emphasis suggested in other works (Singer 1998, Sousa et.al 1998). In addition, several characteristics of developer's information seeking seem related to the time sampling of the mailing list. The results of applying the schema are presented in section 4 of this paper with the empirical study described in section 3.

2. A Preliminary Schema for OS Programmers' Information Seeking

The developed schema is presented in Figure 1. This schema was developed by the author through open coding (Krippendorff 2004) analysis of the questions contained in 2 mailing lists: specifically the Java Bean Scripting Framework (BSF) and the Java Development Tool (JDT). The BSF developers' mailing list used for this purpose was captured from January to August 2007 (as August was the last month of the mailing list at the time of analysis). The JDT mailing list was captured for all of 2003, the first year of that archive. These captures resulted in a data set of 288 email communications from which 98 questions were extracted.

The open coding was carried out without the aid of a coding manual, the coder effectively creating the categories through organizing the data, "breaking it into manageable units, synthesizing it, searching for patterns, (and) discovering what is important" (Singer 1998) . Similar occurrences were grouped together and given the same conceptual label if appropriately close (Krippendorff 2004). This analysis was performed iteratively, each iteration marked by a discussion review with an independent researcher in the field. (Readers interested in the detail of forming this schema are invited to review Sharif (2008)).

Subsequently, a schema was distilled where every question identified in programmers' emails had one attribute from each of the following categories ; i) *Question Strategy*, ii) *Information Focus* and iii) (pre-existing) *Knowledge Strength*.

[1] It is intended that this schema will be iterative refined in the tradition of grounded theory (Pandit 1996), until it becomes saturated (Howit 2008). This is in line with Basili's assertion that knowledge should be evolved through 'modeling, experimenting, learning and remodeling' (Basili 1996)

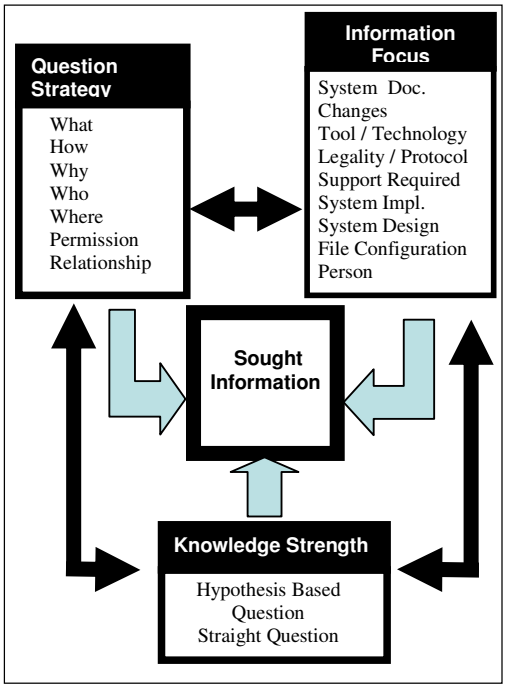

Figure 1 . Preliminary Schema for OS Programmer's Information-seeking (Sharif et.al 2008)

2.1. Information Focus

Information focus refers to the external representation that the programmer refers to in their search for information. There were 9 information foci identified. Table 1 contains a definition for each of these. Please note that all of examples presented in Table 1 are taken from the data-set captured.

Information Focus	**Definition and Example**
System Documentation	Questions referring to the documentation: Example: *"Is there any Apache official guidelines on this?"*
Changes	Questions that refer to changes that programmer has made. . Example: "Here is a patch for the changes I had to do…. Please look into it, I may have broken many exception handling policies here".
Tool / Technology	Questions that refer to technology or tools. Example: *"Can we use JIRA for bug reporting for this issue instead…."*
Legality / Protocol	Questions about the protocol to follow. Example : *" Did you got the approval to contribute your work to BSF? "*
Support Required	Questions that ask another programmer to take on responsibility or tasks. Example: *"There are 2 non-filed open issues….. Are there any taker? "*
System Implementation	Questions that aim to understand the code. Example *: "(Given a situation..)I have no idea why this is happening. Please help me solve this problem"*
System Design	Question referring to the system's design. Example : *"Is jdt.core.jdom built on top of jdt.core.dom? Can you get to the underlying jdom model?"*
File Configuration	Question about configuration management. Example *: " What is the distribution directory in the src zip/tgz? "*
Person	Question about the person in-charge for some task. Example : *"Who is the team / person in charge for documentation?"*

Table 1 . Information Focus (Sharif et.al 2008)

2.2 Question Strategy

Question strategy refers to the type of information sought by the programmers. 7 question strategies were derived by open coding of OS programmers' email communication. The strategies are presented in Table 2.

Question Strategy	Definition and Example
What	Questions which ask what source code or software tool elements do. When referring to source code, these questions represent the bottom-up program comprehension strategy employed by programmers (Letovsky 1986) Example: *"What is the .rep file?"*
How	Questions which attempt to identify how some goal of the system is achieved, how some software tool feature is employed or how to proceed. Example: "*Does anyone know how I can fix this?*
Why	Asking about the purpose / explanation of a system behaviour or design. This also represents bottom-up program comprehension by programmers (Letovsky 1986). Example:"*I am getting an exception being thrown when trying to create new java class and I was wondering if anyone could shed any light on why?"*
Who	Asking for the relevant persons for their task. Example: "Are *there any takers?"*
Where	Asking about the location for software artefacts, tool etc. Example:*"Where I can find the sources for plug in so I can create a patch?"*
Permission	Permission to do something. This strategy is normally related with Legality / Protocol. It seeks permission to do something. Example:*"BTW, can we use JIRA for bug reporting for this project instead ..."*
Relationship	Relationship between 2 or more things. It differs from other questions in that it directs itself at relationships between entities rather than at entities themselves. Example:"*What is the dependence between PackageFragementRoot and PackageFragment?"*

Table 2 . Question Strategy (Sharif 2008)

2.3 Knowledge Strength

Knowledge strength refers to the pre-existing knowledge implied by the programmer in phrasing their question. There are 2 such types of question phrasing, presented in Table 3.

Knowledge Strength	Definition and Example
Hypothesis Based Question	Questions that are asked with an idea as to their answer already in mind. That is, the question comes with a suggestion for the answer. This type of question is asked to validate, to confirm or to correct the provisional answer. Example: *"I see there's a JIRA issue now, and my changes would've been needed anyway, so I hope you're ok?"*
Straight Question	A straight question is a question that is asked without a proposed answer in mind. The person who asks this type of question knows little about the information that he/she asked for and is looking for a related information source. Example : *"Is there any news on access to the JSR-223 TCK? Or any idea how long it might take to get access?"*

Table 3 . Knowledge Strength (Sharif 2008)

3. The Empirical Study

This study described in this paper is a study based on the schema presented in section 2, which examined the information sought by OS programmers during software evolution of the Java Development Tool (JDT) project and the Java Bean Scripting Framework (BSF). The JDT is an OS project concerned with enabling Eclipse for Java development. The JDT programmers' mailing list (http://dev.eclipse.org/mhonarc/lists/jdt-dev/maillist.html) was captured for the period of 3 years from January 2002 to December 2004. The BSF is an OS project concerned with allowing Java applications to contain embedded languages, through an API to scripting engines. The BSF programmers' mailing list (http://jakarta.apache.org/site/mail2.html) were captured from January to August 2007. The resultant data set consisted of 469 emails, and from this data-set 237 questions asked by the programmers were manually extracted. Content analysis was then applied to this dataset.

3.1 Content Analysis

The content analysis method has been widely employed in many disciplines including anthropology, ethnography, history, linguistics, literature, political science and psychology. Krippendorff (2004) provided perhaps the most widely accepted definition of content analysis:

> Content analysis is a research technique for making replicable and valid inferences from texts (or other meaningful matter) to the contexts of their use (Krippendorff 2004)

In this study, content analysis was performed on questions contained in programmers' mailing lists, using the derived schema, in order to assess the information seeking of programmers as the OS software systems evolved.

The medium of email list communication, was described by Mockus et al.(2002), as the primary means of communication for OS projects 'where programmers work in arbitrary locations, rarely or never meet face to face, and coordinate their activity almost exclusively by means of email and bulletin boards'. Hence this is an entirely naturalistic communication medium for these programmers and thus has highly ecologically validity. Also, the mailing list medium can be viewed as containing a substantial proportion of the information passed between programmers of globally distributed projects, making mailing lists a rich source of data.

3.2 The Study

Initial investigations (Sharif 2008) showed that many of the questions in programmers' emails were asked without explicit indicators like question marks or explicit signalling words such as 'what, where…' As a result, the questions in the mailing list had to be extracted manually. 469 emails were analyzed in this fashion and 237 questions were extracted. Later, all of the questions were individually isolated in a spreadsheet, ready for analysis. The first author carried out a detailed analysis of this data, categorizing each question asked by the programmers with the aid of the current schema (Sharif 2008).

4. Result and Data Analysis

The result of the study is presented in Table 4. (the number in brackets in the heading shows the number of question identified in a particular project for each year)

Question Strategy	JDT 2002 (39)	JDT 2003 (99)	JDT 2004 (66)	BSF 2007 (33)
What	8 (20%)	22 (22%)	22 (33%)	15 (45%)
How	18 (46%)	41 (41%)	22 (33%)	7 (21%)
Why	2 (5%)	5 (5%)	5 (8%)	3 (9%)
Who	3 (8%)	7 (7%)	2 (3%)	3 (9%)
Where	7 (18%)	17 (17%)	14 (21%)	1 (3%)
Permission	1 (2%)	5 (5%)	1 (2%)	4 (12%)
Relationship	0 (0%)	2 (2%)	0 (0%)	0 (0%)

Table 4 . Content Analysis Result, Question Strategy Category.

Information Focus	JDT 2002 (39)	JDT 2003 (99)	JDT 2004 (66)	BSF 2007 (33)
System Doc.	5 (13%)	11 (11%)	7 (11%)	6 (18%)
Changes	1 (3%)	3 (3%)	3 (4 %)	5 (15%)
Tool / Technology	20 (51%)	34 (34%)	11 (17%)	5 (15%)
Legality / Protocol	1 (3%)	7 (7%)	3 (4 %)	5 (15%)
Support Required	1 (3%)	1 (1%)	0 (0%)	4 (12%)
System Impl.	6 (15%)	31 (31%)	31 (47%)	4 (12%)
System Design	1 (3%)	3 (3%)	9 (14%)	2 (6%)
File Configuration	4 (10%)	8 (8%)	2 (3 %)	1 (3%)
Person	0 (0%)	1 (1%)	0 (0%)	1 (3%)

Table 5 . Content Analysis Result, Information Focus Category.

Knowledge Strength	JDT 2002(39)	JDT 2003 (99)	JDT 2004 (66)	BSF 2007 (33)
Hypothesis Based Question	5 (13%)	29 (29%)	11 (17%)	12 (36%)
Straight Question	34 (87%)	70 (71%)	55 (83%)	21 (64%)

Table 6 . Content Analysis Result, Knowledge Strength Category.

Table 4, 5 and 6 show the result of this content analysis on the JDT mailing list for 3 years, and BSF for 9 months of one year. The BSF mailing list started in 2001, so column 4 reports on a mature stage in this product's evolution. The novel findings from this study are presented in following sections.

4.1 Information Focus

In line with other research (Singer 1998, Sousa et.al 1998), much of the programmers' information seeking was directed at the implementation of the system. For 2 of the 3 years reported on for JDT, this was the 2nd biggest information focus and, in the 3rd year it was the biggest. More surprising, was the programmers' focus on the tools and technology they used. In this instance however, interest in the tools and technology fell over the 3 years, suggesting that programmers were familiarizing themselves with their environment and becoming more comfortable with it as time went on.

However, the most surprising finding was in regard to programmers' *System Documentation* requests. This was the third most frequently sought information focus in the first 2 years and the 4th in the final year of the analysis for the JDT. In the 2002 archive 12% of the requests were for documentation. The

trend was maintained in the following year when 11% of the requests were for documentation and often these requests seemed important: for example: "*Could anyone please tell me if Eclipse Platform is J2EE compliant, where could I get some more documentation on it. This piece of information is really critical for me"*.

This is at odds with previous studies that suggest software document is not the preferred reference material for programmers (Singer 1998, Seaman 2002, Sousa et.al 1998). It is possible that OS programmers rely much more on documentation than these other programmers based on their delocalization. As they cannot rely on informal communication with their team, they are more likely to need reference material in hand while doing their job. In addition, it is also possible that due to delocalization, OS programmers may be motivated to produce better documentation and therefore they perhaps can trust on documentation more than in the traditional case. Further study will be done to investigate if this is a widespread phenomenon.

4.2 Process Oriented

Our previous findings (Sharif 2008) suggest a largely process-orientation nature to open-source programmers' information-seeking (albeit based on a much smaller data-set). The data showed in Table 4 however portrays a slightly different picture. In the 'Information Strategy' dimension, 'who' questions and 'permission' questions directly reflect a process-oriented nature, while many of the 'how' questions also reflect this aspect of software development. Even disregarding the proportion of relevant 'how' questions, nearly 8% of questions are explicitly processes based. Likewise, looking at the relevant 'Information Focus' questions (person, protocol), nearly 10% were process oriented.

While these results do not imply the same degree of process-oriented information seeking as was suggested in our original findings, this category of information seeking is still significant for OS programmers and deserving of further study, given that most studies to date have not concentrated on this area.

4.3 Development Size (location and team-based)

Several of the previous empirical studies that also aim to inform on the information types sought by programmers in the context of software comprehension (Pennington 1987, Good 1999,Wiedenback et.al 1991, O'Shea 2006) derive from a theoretical analysis of the information available in programs, originally carried out by Pennington (1987). In one typical example of this work, O'Shea (2006, 2004) did content analysis on one OS mailing list based on Pennington's schema and her resultant schema was heavily reflective of the original. Indeed, only late in O'Shea's work did she identify new information types independent of Pennington's. For example, she identified a '*location*' information type where programmers discussed the locations of fixes and functionalities in the code (O'Shea 2007).This category wasn't present in Pennington's initial analysis, probably because Pennington only considered individual programmers studying small code pieces (Penington 1987). In such a scenario, location wasn't an issue. This suggests that Pennington's schema should be expanded to consider context specific factors like larger systems and team-based development. Indeed, our findings mirrored O'Shea (2007), in that we identified 39 questions which were location oriented ('*where*' questions). This represented approximately 14% of all the questions asked suggesting that this is a significant information seeking issue for OS programmers maintaining large systems.

The *'who'* questions identified in the dataset above refer to the team-based nature of the development, requesting information on the member(s) of the team who (for example) implemented a specific part of the system. Likewise the *'Permission'* questions asked others (inside or outside the team) for

permission to take some course of action. While these questions were present, they accounted for approximately only 7% of the questions asked.

Our study shows a big number of *Where* questions asked in the JDT mailing list (refer to Table 4). . This is in line with the works in the concept-location area (Vaclav et.al 2005). However these were not as obvious in BSF mailing list. This might be because the BSF mailing list represented a mature stage of development where the team knew the location of the resources they required.

4.4 Changes over time

There are several trends that can be identified over time for the JDT (from Table 4, 5 and 6):

- Various information seeking issues decrease over time.

Tool/Technology questions decreased over time. A possible reason for this is that both OS programmers become more familiar with the tools they use over the lifetime of their projects and so have fewer information requests in this area. Similar reasoning can be applied to the decrease in the request for documentation over the lifetime of the project.

Another information seeking issue that decreased over time is '*how*' questions. '*How'* questions reflect reasoning about how a goal is achieved, or how to proceed. The decrease in '*how to proceed'* questions is to be expected. However, it would be surprising if reasoning about how a goal is achieved in the system decreases over time, as this reflects top-down comprehension (Letovsky 1986), a strategy associated with programmers who are increasingly familiar with their application domain (O'Brien et.al 2004). Further work will sub-categorize these 'how' questions to see if a masking effect is in place.

- *What'* questions have grown over time.

This is a genuinely surprising finding, especially given that '*what*' questions, as described in the above schema, represent bottom-up comprehension of the system and ignorance of the facilities in software tools. Given that other information issues seem to become less prevalent as programmers become more familiar with the system, it is surprising that this type of question increases. Further qualitative analysis will be carried out to address this phenomenon.

5. Limitation

There is validity threat issue on the evidence strength discussed in this paper. This paper looks at questions posted on two open source programmers' mailing lists consisting of over 400 emails containing 237 questions. Even though this is a large number, the data is split by year leading to four groups (with 33-99 questions in each) and each group is analyzed in three dimensions with 2-9 categories. Thus, the number of data points in each category varies from 0 to 70, with 80% of all categories having less than 10 questions. As a consequence, the discussion is based on considerably weak evidence. Further investigation with larger data set to provide more evidence for each category will be done.

6. Conclusion

This study applied an analysis schema for open-source programmers' information seeking to questions taken from OS projects to investigate preliminary findings in (Sharif 2008). Specifically, the preliminary findings are the high request rate for documentation, the process-oriented nature of the requests and further issues that related to large system and team-based development. Evidence in this study showed that OS programmers do rely on documentation that they are process oriented (albeit to a lesser degree than was originally reported) and that location and team based issues are an important part of their information seeking. In addition, a number of interesting findings over time became apparent.

Our future work concerns refining the schema, and carrying out further analysis with this refined schema. Specifically, we intend to probe our novel findings with respect to larger datasets, and to identify the difficult information seeking issues for OS programmers.

5. Acknowledgements

We would like to thank the Malaysian Government and Lero (supported by Science Foundation Ireland grant number 03/CE2/I303_1).

5. References

AJ Ko, R. D., G Venolia (2007). *Information Needs in Collocated Software Development Teams*. Paper presented at the 29th International Conference on Software Engineering (ICSE'07).

A. Mockus, R. T. F., and J. D. Herbsleb. . (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Basili, (1996)*"The Role of Experimentation in Software Engineering : Past, Present and Future,"* in *Keynote Address : International Conference on Software Engineering*.

Curtis, B., Herb Krasner, and Neil Iscoe. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Cynthia L. Corritore , S. W. (1991). What Do Novices Learn During Program Comprehension? *International Journal of Human-Computer Interaction, 3*.

De Lucia, A., Fasolino, A. R. & Munro, M. (1996). *Understanding function behaviors through program slicing*. Paper presented at the International Workshop on Program Comprehension (IWPC'96).

Dennis Howit, D.C (2008), *Introduction to Research Methods in Psychology*, 2 ed. Essex, England: Pearson Education Limited

Good, J. (1999). *Programming Paradigms, Information Types and Graphical Representations : Empirical Investigations of Novice Pogram Comprehension.* Unpublished PhD Thesis, The University of Edinburgh, Edinburgh , UK.

Khaironi Yatim Sharif , Jim Buckley (2008). *Developing Schema for Open Source Programmers' Information-Seeking*. Paper presented at the International Symposium on Information Technology 2008 (ITSIM '08).

Kingrey, K. P. (2002). Concepts of Information Seeking and Their Presence in the Practical Library Literature. *Library Philosophy & Practice, 4*(2).

Krippendorff, K. (2004). *Content analysis: An introduction to its methodology*: Sage Publications.

Letovsky, S. (1986). *Cognitive Process in Program Comprehension*. Paper presented at the First Workshop on Empirical Studies of Programmers.

Lientz, B. P., Swanson, E. B. & Tompkins, G. E. . (1978). *Characteristics of application software maintenance*. Communications of the ACM, 21(6), 466-471.

Michael P. O'Brien,Jim Buckley, Norah Power. (2006). *Empirically Refining a Model of Programmers' Information Seeking Behaviour During Software Maintenance.* Paper presented at the 18th Annual Psychology of Programming Interest Group (PPIG) Workshop,, Brighton, UK.

Michael P. O'Brien, J. B. (2005). *Modelling the Information-Seeking Behaviour of Programmers - An Empirical Approach*. Paper presented at the 13th International Workshop on Program Comprehension (IWPC'05).

Michael P. O'Brien , J. B., Teresa M. Shaft (2004). Expectation-based, inference-based, and bottom-up software comprehension. *Journal of Software Maintenance and Evolution: Research and Practice, 16*(6), 20.

O'Shea,P.A. , C. Exton. (2004). *The Application of Content Analysis to Programmer Mailing Lists as a Requirements Method for a Software Visualisation Tool.* Paper presented at the International Workshop on Software Technology and Engineering Practice (STEP'04).

O'Shea, P.A ,*Location' Information Type*, Personal Correspondence, 2007.

O'Shea, P. A. (2006). *An Investigation of Views and Abstractions Employed by Software Engineers during Software Maintenance - An Empirically Founded set of Guidelines for Visualisation Tools Supporting Comprehension.* Unpublished PhD Thesis, Limerick Ireland.

Pandit, N.R (1996) ,*The Creation of Theory:A Recent Application of the Grounded Theory Method,* in *The Qualitative Report* vol. 2

Pennington, N. (1987). *Comprehension strategies in Programming.* Paper presented at the Empirical studies of programmers: second workshop.

Prechelt, L., Unger, B., Philippsen, M. & Tichy, W. (1998). *Re-evaluating inheritance depth on the maintainability of object-oriented software.* International Journal of Empirical Software Engineering, 1–16.

Pressman, R. S. (2000). *Software Engineering: A Practitioner's Approach* (5 ed.). Shoppenhangers Road, Maidenhead, Berkshire SL6 2QL, England.: McGraw-Hill Publishing Company.

Seaman, C. B. (2002). *The Information Gathering Strategies of Software Maintainers.* Paper presented at the International Conference on Software Maintenance (ICSM02).

Sim, S. E. (1998). *Supporting Multiple Program Comprehension Strategies During Software Maintenance.* Unpublished Masters Thesis, University of Toronto.

Singer, J. (1998). *Work Practices of Software Maintenance Engineers.* Paper presented at the International Conference on Software Maintenance (ICSM '98), Washington, Federal District of Columbia, USA.

Singer, J. L., T. (1998). *Studying work practices to assist tool design in software engineering.* Paper presented at the 6th International Workshop on Program Comprehension (IWPC'98).

Singer, J. (1998, November 1998). *Practices of Software Maintenance.* Paper presented at the International Conference on SoftwareMaintenance, Bethesda, MD.

Sousa, M. J. C., and Helena Mendes Moreira. (1998, November 1998). *A Survey on the Software Maintenance Process.* Paper presented at the International Conference on Software Maintenance, Bethesda, MD.

Vaclav Rajlich, Andrian Marcus, Joseph Buchta, Maksym Petrenko, Andrey Sergeyev. (2005). *Static Techniques for Concept Location in Object-Oriented Code.* Paper presented at the 13th International Workshop on Program Comprehension (IWPC'05).

Von Mayrhauser, A. & Vans, A. M. (1993). *From code understanding needs to reverse engineering tool capabilities*. Paper presented at the Sixth International Conference on Computer-Aided Software Engineering (CASE'93). 230-239.

Zayour, I. & Lethbridge, T. C. (2001). *Adoption of reverse engineering tools: a cognitive perspective and methodology*. Paper presented at the 9th International Workshop on Program Comprehension (IWPC'01).