

# ULRR

## Further observation of open source programmers information seeking

Item Type	Meetings and Proceedings
Authors	Sharif, Khaironi Yatim;Buckley, Jim
Citation	The 21st Annual Psychology of Programmers Interest Group Conference (PPIG);05/2009
Download date	2026-05-16 23:58:52
Item License	<a href="https://creativecommons.org/licenses/by-nc-sa/1.0/">https://creativecommons.org/licenses/by-nc-sa/1.0/</a>
Link to Item	<a href="https://hdl.handle.net/10344/1807">https://hdl.handle.net/10344/1807</a>

# Further Observation of Open Source Programmers' Information Seeking

Khaironi Yatim Sharif

*Department of Computer Science  
University Of Limerick  
khaironiyatim.sharif@ul.ie*

Jim Buckley

*Department of Computer Science / Lero  
University Of Limerick  
jim.buckley@lero.ie*

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 evolution, and have characterized information seeking empirically in commercial software evolution settings. However, there is little research in the literature describing the information seeking behaviour of Open Source programmers, even though Open Source contexts would seem to exacerbate information seeking problems. That is, team members are typically delocalized from each other and they are often forced into asynchronous communication.

This work reports on an empirical study that classifies Open-Source programmers' information needs, as generated through open-coding of the questions that appear on their developer mailing lists. The study details the information sought by Open Source programmers on 3 different mailing lists over several years and characterizes the responses they obtained. In doing so, several interesting observations are made about the information these programmers seek, the likelihood that they will receive responses and the number of responses they are likely to get.

## 1. Introduction

Software maintenance and evolution are considerable components of a software system's lifecycle. The amount of effort consumed by these activities has been estimated to range between 60% and 80% of the entire lifecycle effort (Lientz et al 1999, 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 (and during) a successful modification can consume a considerable portion of the maintenance activity, with typical estimates of this effort ranging from between 50% and 90% of the entire maintenance effort (De Lucia et al 1996).

Information-seeking, defined as the searching, recognition, retrieval and application of meaningful content (Kingrey 2002), has been recognized as a core subtask in software comprehension within software maintenance (Curtis et al 1998, Seaman 2002, Singer 1998, Sim 1998 and O'Brien et al 2006). 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. Likewise, in their case study of programmers' maintenance activities in the telecommunications domain, Singer et al (1997) found that programmers perform more searching (i.e. grep-based navigation) than any other activity.

There are several previous works (Good 1999, Buckley et al 2004, O'Shea et al 2006 and O'Shea et al 2004) that inform on the types of information that programmers seek, but most of these studies are derived from an existing 'information-types' schema developed by Pennington (1987). As this schema was developed through a theoretical review of the information available in segments of code, it is

possible that it ignores other artefacts produced by the development team and that it ignores some information seeking requirements specific to larger code-bases. An illustrative example is the ‘*location*’ information type identified by O’Shea et al (2006), where programmers sought the location of a specific piece of code within the software system.

In contrast, Ko et al (2007), observed programmers while they were working in-vivo and he identified the information that they sought through his observations, in an open-coding fashion. The work reported here mirrors this approach in that it relies on a schema derived from observations of the information types that programmers seek in-vivo. However, in this instance, it is Open Source programmers being observed, through the medium of the questions they ask on their developer mailing lists. A fuller description of this schema’s derivation is provided in (Sharif et al 2008a and Sharif et al 2008b).

It is important to do such a study in an Open Source (OS) context. The typical widely distributed, asynchronous nature of OS development teams would seem to make their information seeking more difficult. However, to date, there is little research to inform on information seeking among OS programmers.

In addressing this issue, this paper will first discuss related information-seeking work (section 2) illustrating how the work reported on here differs from the existing body of empirical work in the area. In section 3 the process of generating the information-seeking categories we employed in our study is described and the resultant classification schema fully documented. Section 4 reports on the empirical study carried out and the data we obtained. Section 5 discusses these results and interprets them. Finally section 6 concludes the paper.

## 2. Related Work

Within the area of information seeking, O’Brien et al (2005) and Vaclav et al (2005) have focused on the information-seeking processes programmers employ when maintaining commercial software systems. In complimentary work, Singer (1998), Seaman (2002) and Sousa et al (1998), have studied the information sources that programmers use when seeking information. In summary, they found that, while code is the most valuable information source for programmers, execution traces and trusted colleagues are also valued sources of information. In general, they found that documentation was less trusted and thus less valued. However, they did find that, the more abstract the documentation, the higher its perceived trust.

There have also been several empirical studies that characterize the types of information sought by programmers in the context of software comprehension (Good 1999, O’Shea 2006, Corritore et al 1991, Pennington 1987, Ko et al 2007 and Letovsky 1986) 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 visualization tools.

Several of these studies focused their efforts on small programs or on student programmers (Letovsky 1986, Pennington 1987 and Good 1999). A notable exception is Ko et al (2007) who report on commercial software development in collocated teams. They used an open coding protocol to characterize programmers’ information needs as they maintained commercial software and identified 21 such information needs (in the context of 7 maintenance tasks). The most prevalent were:

- Information on the ripple effects of their changes;
- Information on the causes of specific program states and bugs and;
- Information on the changes performed by colleagues.

They also went on to identify the information types that programmers had difficulty obtaining during maintenance and, by means of a survey, the information types that the programmers thought were important. They found that programmers thought that it was important to know the causes of a specific program state or failure, the program’s goals, the implications of a change and what a

specified code failure looks like. Programmers found it particularly difficult to obtain information on the causes of specific program states and the ripple effects of changes they had made.

The work reported on here extends this research by focusing on delocalized OS development, in the tradition of O'Shea (2006) where the developer mailing lists of OS projects are analyzed to inform on the programmers' information seeking efforts. In her work, O'Shea used Good's (1999) enhanced version of Pennington's (1987) information-type schema as the basis for her analysis. This schema was derived from a theoretical analysis of the information available in small computer programs.

The work reported on in this paper can best be described as an amalgamation of the protocol of Ko and the target domain of O'Shea. It studies OS programmers' information needs via their developer mailing lists but, in contrast to O'Shea's work, this research is not based on the information types of Good's schema. Instead it employs a schema derived from open-coding of the questions contained in OS programmers' developer mailing lists: A schema that places no preconceived restrictions on the information types that programmers might seek (Sharif et al 2008b). This was deemed important for 2 reasons:

- The original schema was developed for procedural, small-scale programs only and may be of lesser relevance to large-scale OO software applications;
- The findings of Singer (1998), Seaman (2002) and Sousa et al (1998) suggest that programmers in development teams rely on more than the information available in static source code when performing software maintenance.

This schema was subsequently employed to report on the information sought by OS programmers in 3 OS projects' developer mailings lists. Consistent with previous work in the field, it assessed the availability of this information and did this through the proxy measures of '*Number of responses*', and '*Time taken to obtain responses*'.

### 3. The Information-seeking Schema

This current schema was developed by the first author through open coding (Krippendorff 2004) analysis of the questions contained in 3 mailing lists: specifically the Java Bean Scripting Framework (BSF), the Java Development Tool (JDT) and Element Construction Set (ECS). The JDT is an OS project concerned with enabling Eclipse for Java development and its developer mailing list resides at Eclipse.Org (2008). The BSF is an OS project concerned with allowing Java applications to contain embedded languages, through an API to scripting engines. Its developers' mailing list resides at Jakarta's Apache.Org (2007). The ECS (Apache.Org 2009) is an OS project to develop the Java API for generating elements for various mark-up languages. It directly supports HTML 4.0 and XML, but can easily be extended to create tags for any mark-up language. These 3 projects were picked at random, from a sub-set of OS projects that had strongly active developer mailing lists in their first year post-release.

The open-coding procedure was done by the first author iteratively, all iterations marked by a discussion with the second author, where the second author reviewed the data, categorized it independently and the results were compared. Detailed information on the coding process can be found in Sharif et al (2008a and 2008b).

The medium of mailing list communication, was described by Mockus et al.(2002) as the primary means of communication for open source 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 high ecological validity (O'Shea 2006). Also, the mailing list medium can be viewed as containing a substantial proportion of the information passed between programmers on such globally distributed projects, making mailing lists a rich source of data.

The BSF developers' mailing list used for this purpose was captured from November 2002 to December 2003 (the first year of that archive) and the JDT mailing list was captured for 3 years; from 2002 to 2004 (the first 3 years of that archive). These time-frames were chosen to provide a realistic

time-frame for stress-testing the schema. The ECS mailing list was captured for 8 years from 2001 to 2008 (from the beginning of that archive to the last archive at the time of analysis), in order to assess trends in information-seeking over time.

This data set resulted in 1117 email communications from which 364 questions were extracted manually. Manual extraction was necessary because initial investigations (Sharif et al 2008a) showed that many of the questions in programmers' emails were asked without an explicit indicator like a question mark or explicit signaling words such as 'what, where...'.

<b>Information Focus</b>	<b>Definition and Example</b>
<i>System Documentation</i>	Questions referring to the documentation. Example: <i>"Is there any Apache official guidelines on this?"</i>
<i>Coding Conventions</i>	Questions referring to coding conventions. Example: <i>"Is there a preferred coding standard"?</i>
<i>Changes</i>	Questions that refer to changes that the programmer made. Example: <i>"Here is a patch for the changes I had to do.... Please look into it, I may have broken many exception handling policies here".</i>
<i>Tool / Technology</i>	Questions that refer to technology or tools. Example: <i>"Can we use JIRA for bug reporting for this issue instead...."</i>
<i>Protocols Adhered to</i>	Questions about the protocol to follow. Example: <i>" Did you got the approval to contribute your work to BSF? "</i>
<i>Support Required</i>	Questions that ask another programmer to take on responsibility or tasks. Example: <i>"There are 2 non-filed open issues..... Are there any taker? "</i>
<i>System Implementation - Enhancement</i>	Questions that aim to understand the code in order to make change. Example: <i>"...but I need to understand the refactoring currently in Eclipse now. Can anyone suggest me where about in the code is a good starting point in understanding how the component works "</i>
<i>System Implementation – Debug</i>	Questions that aim to understand the code in order to trace a bug. Example: <i>"(Given a situation..)I have no idea why this is happening. Please help me solve this problem"</i>
<i>System Design</i>	Questions referring to the system's design. Example: <i>"Is jdt.core.jdom built on top of jdt.core.dom?"</i>
<i>File Configuration</i>	Questions about configuration management. Example: <i>" What is the distribution directory in the src zip/tgz? "</i>
<i>Owner</i>	Questions about the relevant person for some task. Example: <i>"Who is the team / person in charge for documentation?"</i>
<i>Task-Testing</i>	Questions related to testing. Example: <i>" Can I invoke all junit test cases in one or more source folders in one movement without testsuites"</i>
<i>Task-Implementation</i>	Questions about tasks that are related to Implementation. Note that this is not about comprehending the code but more directed at the task to be undertaken. Example: <i>"Maybe you need to post more code, or maybe you need to update ecs-1.4.1?"</i>
<i>Stage/Completion</i>	Questions about completion of a certain task or stage. Example: <i>"Has jakarta-ecs seen substantial dev work in that time? Is ecs2 still effectively the latest work?"</i>

Table 1 . Information Focus ( Sharif et al 2008a and Sharif et al 2008b)

Through a series of iterative refinements, where 2 independent coders applied the developing classification schema to samples of these data-sets, a coding schema was distilled where every question identified in programmers' emails was categorized with respect to *Information Focus* and *Question Strategy*. These categories are described in sections 3.1 and 3.2, but a more detailed description of the schema's formation is given in Sharif et al (2008a and 2008b).

### 3.1 Information Focus

*Information Focus* refers to the external representation that the information search refers to. There were 14 individual foci identified. Table 1 contains a definition for each of these and examples taken from the data-set captured. Please note that while these seem to bear similarity to the 'information source' research carried out by Singer (1998), Seaman (2002) and Sousa et al (1998), they differ, as the focus in this research is *the artefact the programmer is looking for information about*, not *the source through which they choose to acquire the information*. In this research the source through which they choose to acquire the information is always the mailing list.

Question Strategy	Definition and Example
<i>What</i>	Questions which ask what the information focus does (the source code or software tools). When referring to source code, these questions represent bottom-up program comprehension (Letovsky 1986). Example: "What is the .rep file?"
<i>How</i>	Questions which attempt to identify how an information focus achieves its goal, how some information focus is employed or how to proceed. Example: "Does anyone know how I can fix this?"
<i>Why</i>	Asking for a purpose / explanation of the information focus. When directed at code, 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?"
<i>Who</i>	Asking for the relevant persons. Example: "Are there any takers?"
<i>Where</i>	Asking about the location of something within the information focus or about the location of an information focus. For example: "Where can I find the sources for plug in so I can create a patch?"
<i>When</i>	Questions about the time of occurrence. Example: "When is the next BSF release expected?"
<i>Permission</i>	Permission to do something. This strategy is normally related with the Protocol information focus. Example: "BTW, can we use JIRA for bug reporting for this project instead ..."
<i>Confirmation</i>	Questions that confirm certain information/actions/tasks. Example: "... will it be incorporated into the latest version of BSF?"
<i>Relationship</i>	Questions that probe the 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?"
<i>Instruction</i>	Questions that are asking a community member to do something : Example: "Would you consider donating your patch to Apache? "

Table 2 . Question Strategy ( Sharif et al 2008a and Sharif et al 2008b)

### 3.2 Question Strategy

*Question Strategy* refers specifically to the aspect of information sought by the programmer, from the information focus. 10 question strategies were derived by open coding of the OS programmers' email communication. These strategies are presented in Table 2.

## 4. The Empirical Study

The study described in this paper is based on the schema presented in section 3. The schema was used to examine the entire data-set as described in section 3: The JDT developer mailing list from 2002 to 2004, the BSF developer's mailing list from November 2002 to December 2003 and the ECS-Java API developer mailing list from 2001 to 2008.

### 4.1. Results and Data Analysis

When all 364 questions were extracted, they were individually isolated in spreadsheet cells to facilitate categorization with respect to the schema. The first author then applied content analysis (Krippendorf 2004) to this dataset, categorizing each question asked by the programmers with the aid of the current schema. The results of the study are presented in Tables 4 and Table 5 in the Appendix. (The number in brackets in the heading reports on the number of question identified in a particular project for each year). As the BSF mailing list started in 2001, the BSF column in each category reports on an early stage in this product's evolution. Likewise, the JDT columns report on the early stages of its evolution but the 'ECS-Java API' columns report from the very beginning through to its current evolution. Several interesting findings from this data-set are presented in following sections.

### 4.2 Information Focus

Overall, the three biggest information foci in the BSF project were '*System Design*', '*Tools/Technology*' and '*Task-Implementation*'. Likewise the JDT showed the same emphasis on '*Tools/Technology*' and '*Task-Implementation*', but with '*System Documentation*' and '*System Implementation-Debug*' also important. The most prevalent information focus in the ECS project was '*System Implementation – Enhancement*' followed by '*Task – Implementation*' and '*Tools/Technology*'.

Overall, this resulted in a large emphasis on *Tools/Technology* (24.18% of total), '*Task Implementation*' (13.46% of total) and '*System Implementation – Enhancement*' (11.81% of total). Other information sought frequently was '*System implementation – Debug*' (9.34% of total) and '*System Documentation*' (8.79% of total).

Hence, as suggested in our previous work (Sharif et al 2008a, Sharif et al 2008b), and in line with other research (Sousa et al 1998, Singer et al 1998), much of the programmers' information seeking was directed at the systems' implementations. Taking '*System Implementation – Enhancement*', '*System Implementation – Debug*' and '*Task-Implementation*' as reflecting a focus on the code base, 30% of all BSF questions were directed at the code base. Likewise 31% of all JDT queries were directly code based and 50% of ECS queries were directly code based. In addition, closer examination of the '*Tool/Technology*' focus showed that 92% of the questions aimed at this focus related to working with the code (editing code, submitting changes, debugging and settings). As '*Tool/Technology*' was the biggest information focus this suggests a strong code focus for the JDT, ECS and BSF projects.

In previous works ( Sharif et al 2008a and Sharif et al 2008b) we reported a surprising finding in regards to programmers' '*System Documentation*' requests. Specifically we noted that documentation seemed to play an important part in OS programmers' information requests. This was surprising because other 'information source' literature suggested that programmers distrusted documentation (Singer 1998, Seaman 2002 and Sousa et al 1998).

The data shown in table 5 reinforces our original findings, but the trend is not as emphasized here. Specifically, Documentation was sought frequently in the JDT project but was ranked 5th in the ECS project and only 11th in the BSF project. However, over all years of all projects, almost 10% of the questions were ‘*System Documentation*’ questions.

It is possible that this is due to the delocalized context of programmers in this study. OS programmers may be motivated to produce better documentation because of this delocalization, and therefore trust documentation more than in the traditional case. Alternatively, it is also possible that, because of delocalization, OS programmers cannot rely on informal communication within their team and so must resort to the documentation. This phenomenon will be explored in future work.

### 4.3 Team-Oriented Questions

In our initial work (Sharif et al 2008a), in line with Ko et al (2007), we suggested that there is a strong team-orientation to the questioning (albeit based on a much smaller data-set). The data showed here in Table 4 (with a much larger data set) is in line with our initial finding, as there is a strong emphasis on, ‘*Who*’ questions and ‘*Confirmation*’ question. ‘*Confirmation*’ questions accounted for approximately 31% of all questions, and this was by far the most frequent question strategy. Likewise ‘*Who*’ questions were also popular, accounting for 12.09% of all questions. While this latter category may reflect the increased effort in allocating and breaking down work in a delocalized context, the underlying information need is still founded upon team awareness and team dynamics. Given Ko et al.’s findings, this is an unsurprising result: If co-located programmers need to ascertain their team-mates, and their roles, then it is likely that delocalized programmers will also have increased information needs in this regard.

### 4.4 Development Size

The data in Table 4 is also inline with our original findings (Sharif et al 2008a and Sharif et al 2008b) with regard to the presence of ‘*Location*’ type queries, as suggested by O’Shea (2007). This category of information wasn’t present in previous research that aimed to inform on the information types sought by programmers in the context of software comprehension (Good 1999, O’Shea 2006, Corritore et al 1991 and Ko et al 2007). However, its empirical recognition by O’Shea is echoed in this work. We identified 25 questions which were location oriented (‘*Where*’ questions). This represents approximately 7% of all questions asked, suggesting that this is a significant information seeking type for OS programmers maintaining large systems. These finding add empirical credence to Rajlich’s body of ‘*Concept Location*’ work (Vaclav 2005).

In our previous work (Sharif et al 2008b) we suggest that programmers’ higher familiarity with code and resource location would probably lead to a reduction of ‘*Location*’ questions over time. Our current findings are in line with this hypothesis. Table 4 shows a high number of ‘*Location*’ requests for the BSF project and JDT project (given that the mailing lists captured for these projects represent early stages of the evolution of both products). In the ECS project, ‘*Location*’ questions are only obvious on the first and second year. However, this and similar trends in ‘*Tools and Technology*’ and ‘*How*’ questions, could also be attributed to significantly decreasing activity in this mailing list over time.

## 5. Responses to Queries

Table 5 presents an analysis of the responses received for the most popular query types posted by the OS programmers on the mailing lists. The first column reports on the information sought (its *focus* and its *strategy*) and the top five ranking query types in each dimension are presented. Column two reports on the number of queries identified for each information-type and column three presents this as a percentage of the whole. Column four shows the % of these queries that received a response and column five reports on the average number of responses received, for queries that received at least one response. Finally column six reports on the average number of days which passed between the query being posted and the final response, again for those queries that received at least one response.

Perhaps the most surprising finding is the low response rate overall. On average, a query had only a 55% chance of being responded to. Indeed, when the programmers were interested in seeking information on *'Tools and Technologies'* they had less than an even chance of getting a response (43%). As *'Tools/Technology'* was the most frequent information focus, this is problematic for the community. Likewise *'Task-Implementation'* questions were frequently posed, but had only a 52% chance of being responded to. Information strategies with a low success rate include *'How'* questions and *'Who'* questions, which together make up over 33% of all the questions posed. By far the highest response rate was recorded for *'System Documentation'* queries. Two thirds of these queries were responded to by the community.

It is difficult to hypothesize on the reasons for these response rates without in-depth qualitative analysis. However, it is possible that documentation requests, which would seem to be associated with brief answers, are appealing to the community. In contrast *'How'* questions may require a more detailed explanation, leading to a verbose, time-consuming answer. Indeed the answer itself may take some time to formulate, given that these question types often reflect reasoning about systems' achieving their goals. Another consideration is that only a small pool of the community may know the answer to such queries.

The low response rate to *'Who'* questions may be associated with an obscurity of roles within OS projects. That is, only a few of the community know the identities of the owners of specific artefacts or roles. Hence only a few of the community can respond.

It is difficult to explain away the low response rate to the *'Tools/Technology'* in such a fashion. Such questions should not be that difficult for the technical community and probably require a short answer. It may even be possible that there is a *'technical-snobbery'* phenomenon occurring here, where OS programmers are unforgiving of those who are not experts in the tools of their trade and thus would not countenance helping them. However, these are speculations only and would need to be probed by more in-depth, qualitative analysis of the community.

<b>Info. Strategy</b>	<b>Total no. of questions</b>	<b>% of total Requests</b>	<b>% Answered</b>	<b>Avg. no. of Resp.</b>	<b>Avg. Time-span of Resp. (Days)</b>
Confirmation	111	30.49	54	2.60	3.05
How	84	23.08	52	2.27	4.36
What	47	12.91	60	3.29	2.36
Why	32	8.79	53	1.65	2.29
Who	44	12.09	50	1.91	2.00
Info. Focus					
Tools/Tech	88	24.18	43	2.13	2.08
Sys-Doc	32	8.79	66	2.57	4.19
Task Impl	49	13.46	51	2.64	6.36
SI-Debug	34	9.34	59	1.80	2.35
SI-Enhance	43	11.81	60	2.54	1.27
Average			55	2.32	3.05

*Table 5. Analysis of the pattern of Response*

Regarding the questions that were answered, they did seem to provoke some discussion, resulting in an average of 2.32 responses for each (responded-to) query. This suggests a discursive community,

particularly with regard to ‘*What*’ questions (average response = 3.29). This is surprising as discussion implies a degree of animation in the community and ‘*What (does X do)*’ questions would not seem, on first impression, to have the contentiousness to stoke up such animation. Again, further qualitative analysis is required to probe this finding.

The least amount of discussion happened round ‘*Who*’ questions and ‘*System Implementation-Debug*’ type questions. Presumably, in the case of ‘*Who*’ questions, the relevant community member answered for themselves quite frequently, and thus prompted little discussion. With regard to ‘*System Implementation- Debug*’ type questions, understanding this lack of discussion is more difficult, but may be associated with the general apathy programmers feel for debugging existing code.

The largest distribution in the results set is in the response time-span, which spreads from 6.36 to 1.27 days. Interesting in this regard is the relationship between this data and the data in column 4. For example, even though ‘*What*’ questions had, on average, the largest number of respondents to each query, they had a below average time-span, suggesting that the responses came quickly after the original questions were posted. In contrast, ‘*Task Implementation*’ type queries had an above average number of responses, but over a much longer time-span, suggesting a drip-effect in information retrieval. This could be because ‘*Task Implementation*’ type queries require greater reflection than ‘*What*’ type queries but again, this preliminary hypothesis would need to be tested by further empirical work.

## 6. Conclusion

This research probed the information seeking of OS programmers as they maintained and enhanced OS software systems. It extracted questions from 3 OS development mailing lists and found both expected and surprising results. Specifically, it found, in line with other studies, that programmers were implementation centric, that they often required location information and that they were quite team-oriented. Two surprising findings were that they tended to rely more on documentation than previous reports would suggest, and that they asked a lot of questions focused on the ‘*Tools/Technology*’ employed in the project.

Surprisingly, there was a low overall response rate to their queries from the community, particularly with regard to this ‘*Tools/Technology*’ focus, a cause for concern, given its prevalence. Greatest discussion focused round ‘*What*’ type questions, and this discussion typically happened quickly. Least discussed were ‘*Who*’ questions and ‘*System Implementation-Debug*’ and further empirical studies need to probe the rationales behind these phenomenon.

These findings suggest a number of directions for IDEs and visualization tools that support programmers involved in OS developments. Specifically it suggests that organizational charts be made available to inform new developers of the roles of other developers in the community. Likewise tutorials should be made available on the tools and technology the community uses as this seems like a significant information need for developers. Finally, there is a suggestion that tools and documentation should focus more of identifying where concepts are located in the software.

## 7. Acknowledgement

This work was supported, in part, by Science Foundation Ireland grant 03/CE/I303\_1 to Lero – The Irish Software Engineering Research Centre ([www.lero.ie](http://www.lero.ie))

## 8. Appendix

Question Strategy	BSF 391 emails 2002 (91)	JDT 81 emails 2002 (43)	JDT 147 emails 2003 (90)	JDT 100 emails 2004 (61)	ECS 162 emails 2001 (37)	ECS 39 emails 2002 (17)	ECS 131 emails 2003 (11)	ECS 21 emails 2004 (2)	ECS 17 emails 2005 (5)	ECS 6 emails 2006 (4)	ECS 2 emails 2007 (1)	ECS 20 emails 2008 (2)
What	20	3	11	1	6	3	1	0	1	1	0	0
How	17	16	27	18	6	3	1	0	1	1	0	0
Why	5	3	3	9	3	3	2	0	1	0	1	1
Who	14	2	13	4	6	2	2	0	0	1	0	0
Where	5	3	8	6	2	1	0	0	1	0	0	0
When	2	0	0	0	0	0	0	0	0	0	0	0
Permission	0	1	0	1	1	1	0	0	1	1	0	0
Confirmation	26	15	28	21	13	3	2	2	0	0	0	0
Relationship	0	0	0	0	0	0	3	0	0	0	0	0
Instruction	0	0	0	1	0	1	0	0	0	0	0	1

Table 4. Content Analysis Result, Question Strategy Category

Information Focus	BSF 391 emails 2002 (91)	JDT 81 emails 2002 (43)	JDT 147 emails 2003 (90)	JDT 100 emails 2004 (61)	ECS 162 emails 2001 (37)	ECS 39 emails 2002 (17)	ECS 131 emails 2003 (11)	ECS 21 emails 2004 (2)	ECS 17 emails 2005 (5)	ECS 6 emails 2006 (4)	ECS 2 emails 2007 (1)	ECS 20 emails 2008 (2)
System Doc.	3	7	8	7	4	0	1	0	1	0	0	0
Coding Standard	0	0	0	0	1	0	0	0	0	0	0	0
Changes	6	0	3	1	0	1	1	0	0	0	0	1
Tool / Technology	8	20	25	21	5	3	2	0	1	0	0	0
Legality / Protocol	0	0	4	0	0	1	0	0	0	0	0	0
Support Required	7	2	3	0	1	0	1	0	0	0	0	0
Sys. Impl-Enhance	9	2	5	13	6	9	2	0	0	1	0	0
Sys. Impl-Debug	5	2	13	7	3	0	2	0	0	1	1	0
System Design	16	1	7	1	1	0	0	0	0	0	0	0
File Configuration	6	3	5	1	4	0	0	0	0	0	0	0
Person	1	1	1	1	0	0	0	0	0	0	0	0
Task-Testing	6	1	2	4	0	0	0	0	0	0	0	0
Task-Impl.	14	4	14	4	10	3	0	0	1	2	0	0
Completion/Stage	10	0	0	1	2	0	2	2	1	0	0	1

Table 5. Content Analysis Result, Information Focus Category.

## 5. References

- Buckley, J., Exton, C. and Good, J.(2004) ,Characterizing Programmers' Information-Seeking during Software Evolution, Proc. International Workshop on Software Technology and Engineering Practice (STEP'04)
- Corritore.C.L. Wiedenback, S.( 1991),*What Do Novices Learn During Program Comprehension?*, International Journal of Human-Computer Interaction, vol. 3.
- 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.
- 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).
- Eclipse Org. (2008), *Mailing List for JDT Developer (JDT)*, Internet Source : <http://dev.eclipse.org/mhonarc/lists/jdt-dev/maillist.html>
- 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.
- Jakarta Apache Org.(2007) *Mailing List for Java Bean Scripting Framework (BSF)*. Internet Source : <http://jakarta.apache.org/site/mail2.html>
- Jakarta Apache Org.(2009) *Mailing List for Element Construction Set (ECS)*. Internet Source : [http://mail-archives.apache.org/mod\\_mbox/jakarta-ecs-dev/](http://mail-archives.apache.org/mod_mbox/jakarta-ecs-dev/)
- Kingrey, K. P. (2002). *Concepts of Information Seeking and Their Presence in the Practical Library Literature*. Library Philosophy & Practice, 4(2).
- Ko, A.J. DeLine, R. Venolia,G. (2007). *Information Needs in Collocated Software Development Teams*. Paper presented at the 29th International Conference on Software Engineering (ICSE'07).
- Krippendorff, K. (2004). *Content analysis: An introduction to its methodology*: Sage Publications, ISBN 0761915451
- 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.
- Mockus, A. Fielding, R.T. and Herbsleb, J.D.(2002). *Two case studies of open source software development: Apache and Mozilla*. ACM Transactions on Software Engineering and Methodology, 11(3), 309-346.
- O'Brien, M.P., Buckley, J. (2005). *Modelling the Information-Seeking Behaviour of Programmers - An Empirical Approach*. Paper presented at the 13th International Workshop on Program Comprehension (IWPC'05).
- O'Brien, M.P., Buckley, J. and Power, N. (2006). *Empirically Refining a Model of Programmers' Information Seeking Behaviour During Software Maintenance*. Paper presented at the 18<sup>th</sup> Annual Psychology of Programming Interest Group (PPIG) Workshop,, Brighton, UK.
- O'Brien, M.P., Buckley, J. and Shaft, T.M. (2004). *Expectation-based, inference-based, and bottomup software comprehension*. Journal of Software Maintenance and Evolution: Research and Practice, 16(6), 20.
- 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*. PhD Thesis, Limerick Ireland.

- O'Shea, P.A (2007) ,*Location' Information Type*, Personal Correspondence.
- O'Shea,P.A. , Exton, C. (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).
- Pennington, N. (1987) *Stimulus structures and mental representations in expert comprehension of computer programs*. *Cognitive Psychology*, 19, 295-341
- 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).
- Sharif, K.Y., Buckley, J.(2008b), *Observing Open Source Programmers' Information Seeking Paper* presented at the 20th Annual Psychology of Programming Interest Group (PPIG) Workshop,Lancaster, UK (PPIG'08).
- Sharif. K.Y., Buckley, J.(2008a), *Developing Schema for Open Source Programmers' Information-Seeking*, Proc. International Symposium on Information Technology 2008 (ITSIM '08), IEEE, 2008.
- 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. Lethbridge, 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. Lethbridge, T. Vinson, N. Anquetil , N. (1997).*An Examination of Software Engineering Work Practices*, Proc. Centre for Advanced Studies on Collaborative research (CASCON'97).
- Sousa, M. J. C., and Moreira, H.M. (1998). *A Survey on the Software Maintenance Process*. Paper presented at the International Conference on Software Maintenance, Bethesda, MD, USA
- Vaclav Rajlich, Andrian Marcus, Joseph Buchta, Maksym Petrenko, Andrey Sergeyev. (2005). *Static Techniques for Concept Location in Object-Oriented Code*. Paper presented at the 13<sup>th</sup> International Workshop on Program Comprehension (IWPC'05).
- Von Mayrhauser, A. and 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. and 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).