Journal of Computer Science 7 (7): 1060-1071, 2011
ISSN 1549-3636

Open Source Programmers' Information Seeking During Software Maintenance

[1]Khaironi Yatim Sharif, [2]Mohd Rosmadi Mokhtar and [3]Jim Buckley
[1]Software Engineering Research Group,
Fakulti S. Komputer and T. Maklumat,
University Putra Malaysia, Malaysia
[2]Centre of Computer Science FTSM, University Kebangsaan, Malaysia
[3]The Irish Software Engineering Research Centre (LERO),
University of Limerick, Ireland

Abstract: Problem statement: Several authors have proposed information seeking as an appropriate case study for studying software maintenance and evolution that have provided empirical classifications of information seeking in commercial software evolution settings. **Approach:** However, there is minimal research in the literature describing the information seeking behavior of Open Source programmers, even though Open Source contexts would seem to exacerbate the information seeking problems to a certain extend; where team members are typically delocalized from each other and they are often forced into asynchronous communication. **Results:** This study reports on an empirical study that classifies Open-Source programmers' information needs generated through open-coding of questions that appear on developers' mailing lists. Based on the generated Information Seeking Schema (ISS), details of the information sought by programmers on 6 different mailing lists over several years are analyzed and discussed. **Conclusion/Recommendations:** The result shows several interesting findings that describe the programmers' information needs across the mailing lists. Firstly, there are a similar pattern of information artifact and attribute across all projects. Secondly, majority of the programmers' information seeking concentrated on the systems' implementations. Thirdly, the OS programmers have also shown to be team-oriented and they tended to rely on documentation more than what have previously reported. These results suggest the applicability of the ISS in evaluating OS programmers information seeking.

Key words: Information seeking, program comprehension, open source, software maintenance, probed artifacts, theoretical review, theoretical harness, information seeking schema

INTRODUCTION

Software maintenance has been part and parcel of a software system's lifecycle ever since the first computer software was introduced more than half-century ago. Lientz *et al*. (1978), defines software maintenance as, "activities which keep systems operational and meet user needs" while Boehm's (2007) defines the process of software maintenance as "the process of modifying existing operational software". The software mantainance activities make the software systems change over time. In this context Belady and Lehman (1976), defines software evolution as "the dynamic behavior of programming systems as they are maintained and enhanced over their lifetimes."

Software maintenance and evolution are large components of a software system's lifecycle. 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 and Vans, 1993; Pressman, 2004; Zayour and Lethbridge, 2001). While the empirical basis for such statements are dated and suggestions that they should be revisited have been made (Kemerer and Slaughter, 1999), the increasing scale and complexity of newer software systems (Pressman, 2004; Sommerville, 2008; Stein *et al*., 2005) implies that the effort invested in maintenance of successful systems can only have increased. Thus research in this area is vital towards the discovery and evolution of supportive methods or tools, which could aid maintainers in their software maintenance efforts.

Software maintenance can be divided into 2 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

Corresponding Author: Khaironi Yatim Sharif, Software Engineering Research Group, FSKTM, UPM Malaysia, Malaysia

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).

Kingrey (2002) defines Information-Seeking as the searching, recognition, retrieval and application of meaningful content. Information seeking has been recognized as a core subtask in software comprehension within software maintenance (Curtis *et al*., 1988; Seaman, 2002; Singer, 1998; Singer and Lethbridge, 1998; Sim, 1998; O'Brien and Buckley, 2005). 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, Singer and Lethbridge (1998) in their case study of programmers' maintenance activities in the telecommunications domain, found that programmers perform more searching (i.e., grep-based navigation) than any other activity. A more recent study by Cleary (Cleary *et al*., 2008; Cleary and Exton, 2007) suggests that information seeking was a very credible model for describing the goal orientated, opportunistic software comprehension strategies employed by software engineers.

The nature of Open Source (OS) software development could make it as a very important context in which to study information seeking in software maintenance. The Open Source Software (OSS) development process generally involves (or has the potential to involve) large, globally distributed communities of developers collaborating primarily through the internet (Feller and Fitzgerald, 2001; Fitzgerald, 2004; Iskandarani, 2008). Internet is seen as successful collaborative environment (Sullabi and Shukur, 2008). However, in OS context, the typical widely distributed, asynchronous development would seem to make information seeking more difficult (Sharif and Buckley, 2009a; Gutwin *et al*., 2004). But open source programmers seem to manage to deal with large scale code with high-code complexity (Daniel *et al*., 2009). However, to date, there is little research to inform on information seeking among OS programmers.

In addressing this issue, this research aims to characterize information seeking in open source software projects in term of the information artifact probed by programmers and the information sought within the probed artifacts, henceforth referred to as the information attributes in this article. In this context, information artifact can be described as the entity that the programmer seeks information about: the focus of information or the object of the programmers' information seeking attention. On the other hand, the information attribute refers to the features of the information been sought. For example, when a programmer ask about the location of particular document, the document is the artifact, while the location (i.e., where) is the attribute of the information.

To contextualize this, we first discuss the related information-seeking work by illustrating how the work reported here differs from the existing body of work in the area. Then, the next section provides discussion on software maintenance in OSS, presenting the characteristic of OS that could makes different its software development and maintenance nature. In the following section, the process of generating the information-seeking categories employed in this study is presented and the fully documented classification schema is described. Then, the following section reports on the results of the empirical study carried out, before we finally conclude this paper in the last section.

Related works: Within this research area O'Brien and Buckley (2005) has 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, also in commercial scenarios. The work reported in this thesis extends this research by focusing on delocalized OSS 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.

There have been several empirical studies that aim to inform on the types of information sought by programmers in the context of software comprehension, such as Singer (1998); Ko *et al*. (2007); Letovsky (1987); Pennington (1987); Good (1999); Wiedenback and Corritore (1991); O'Shea (2006) and Buckley *et al*. (2004). These studies focus on the information that programmers' might obtain and the information that they find difficult to obtain during software maintenance, thus potentially informing the design of software tools.

However, 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 to individuals in small segments of code, it is possible that it ignores other artifacts produced by a development team and that it ignores some information seeking requirements specific to larger code-bases (Sharif and Buckley, 2009a). An illustrative example is the 'location' information type identified. O'Shea empirically established that programmers sought the location of a specific piece of code within the software system, in her Ph.D. research. While this finding was in line with feature and concept location work, O'Shea attributed the lateness of this finding to her adoption of Pennington's schema. This 'theoretical harness' thus potentially constrained O'Shea's work and has the same potentially constraining possibilities for this entire body of research.

In contrast, Ko *et al*. (2007), observed programmers while they were working in-vivo with proprietary or commercial software development teams 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. This frees it from any potentially constraining theoretical harnesses. Instead it places no restrictions on the information source to derive a holistic information seeking schema. However, as stated above, this study will focus on OS programmers.

Software mantenance for OSS development: The public availability of source code for OSS makes a difference in its software development and maintenance nature. In an open source setting, for example, anyone can amend the code and contribute change to the software. With the source code available to all OS developers, they tend to work in parallel (Feller and Fitzgerald, 2001), with different individuals or groups working on the system simultaneously. Several other characteristics of generic OSS development process have been suggested (Feller and Fitzgerald, 2001; Gacek and Arief, 2004) that might impact on the software maintenance process, such as the involvement of large global communities, parallel development, independent review, prompt feedback, motivated developers and users, as well as rapid release schedule.

These characteristics can be seen as factors that will give impact on OS programmers' information seeking activities. For example the large and global communities could impact the information seeking activities among OS programmers. The extremely delocalized OS programmers might cause them to actively looking for information to organize their task. Likewise, availability of source code to all members in the communities is possibly make them inquiry about code's version, questions about code comprehension. This also could lead them to ask question about design decision that has made for particular sets of codes. At the same time, the huge number of community's members might contribute in active response for information request from the community. 'Parallel development' possibly makes OS programmers seek more information to coordinate their task-such as other programmers' job status, software activities and related source code process.

MATERIALS AND METHODS

Schema developement: Early investigations showed that considerable number (estimated at 20%) of questions in programmers' emails were asked without explicit indicators like question marks or signaling words such as "what" and "where" (Sharif and Buckley, 2008a; 2009a). As a result, all the datasets used in this study-the questions in the mailing list had to be extracted manually. Later, all of the questions were individually isolated in a spreadsheet, ready for analysis. This is a prerequisite for data preparation when analyzing textual data in this fashion (Good, J., 1999; Sharif and Buckley, 2009b).

The researcher carried out a detailed analysis of this data, naming and categorizing each question asked by the programmers. This open-coding procedure is carried out without the aid of a coding manual or schema, the coder effectively creates the categories from scratch. Accordingly, the researcher immersed himself in the transcript data, seeking to gain as many insights as possible into the information-seeking behavior of the programmers, and began to produce categories based on the contents of portions of the transcripts being examined as suggested by O'Brien *et al*. (2001) and Pandit (1996).

The open-coding procedure was done iteratively, each iteration marked by a discussion review with another researcher where a random sample, was categorized by both the first author and this other researcher. The results were compared and ver time, a number of provisional categories began to emerge by consensus. Those categories were then applied to other question datasets and refined by means of reflection, dual review, discussion, merging and renaming. Finally, a set of categories seemed increasingly resistant to change and these became the final schema.

In refining the schema the following datasets were employed:

- A random dataset (a comparatively small dataset as initial dataset during pilot study)
- Datasets from different stages of software evolution
- A larger, time-scaled dataset
- A dataset that reflected successful OSS projects as per the characterization presented by Daniel *et al*. (2009)

As a result, the schema employed in this study was developed through open coding (Krippendorff, 2004) and content analysis of the questions contained in dataset that consisting of 17 (yearly) archives taken from 6 OSS projects. This dataset resulted in 2104 email communications from which 708 questions were extracted. Table 1 describes the 6 OSS projects and the different dataset used in this study.

Initially, the archive from BSF 2007 and JDT 2003 were used in modeling the preliminary Information Seeking Schema (ISS). Then all the archives were used in refining this schema with respect to modeling information seeking in maintenance over time and further analysis on these initial findings. The process of schema creation and refinement were discussed earlier in our previous work (Sharif and Buckley, 2008a; 2008b; 2009a).

Table 1: Description for OSS projects used in this study

OSS project	Description	Dataset used year	# of emails	# of questions
The Java Bean	BSF is an OS project concerned with allowing	2003	284	73
Scripting	Java applications to contain embedded	2004	107	18
Framework (BSF)	languages, through an API to scripting engines.	2007	275	85
Java Development	JDT is an OS project concerned with	2002	81	43
Tool (JDT)	enabling Eclipse for Java development.	2003	147	90
		2004	100	61
The Element	ECS is an OS project to create Java APIs for	2001	162	37
Construction Set (ECS)	generating elements for various markup	2002	39	17
	languages that allows user to use Java Objects	2003	131	11
	to generate markup code.	2004	21	2
		2005	17	5
		2006	6	4
		2007	2	1
		2008	20	2
Eboard	User-friendly chess interface for Internet Chess Servers (ICS)	2001	182	45
SwingWT	Implementation of the Java Swing and AWT APIs	2004	302	107
Resiprocate	Dedicated to maintaining a complete, correct and commercially usable implementation of Session Initiation Protocol (SIP)	2009	228	107
Total		17	2104	708

Table 2: Description for information artifact categories

Information artifact	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….”
Protocols adhered to	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-Enhancement	Questions that aim to understand the code in order to make change. Example : “…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 “
System Implementation-Debug	Questions that aim to understand the code in order to trace a bug. 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?”
File configuration	Question about configuration management. Example: “What is the distribution directory in the src zip/tgz?”
Owner	Question about the relevant person for some task. Example: “Who is the team / person in charge for documentation?”
Task-Testing	Question related to testing. Example: “Can I invoke all junit test cases in one or more source folders in one movement without testsuites”
Task-Implementation	Question about tasks that are related to Implementation. Note that this is not about comprehending the code b ut more directed at the task to be undertaken. Example :”Maybe you need to post more code, or maybe you need to update ecs-1.4.1?
Stage/Completion	Question about completion of a certain task or stage. Example: “Has jakarta-ecs seen substantial dev work in that time? Ie is ecs2 still effectively the latest work?”

Resultant schema: Through the series of iterative refinements mentioned above, where 2 independent coders applied the developing classification schema to samples of these datasets, a coding schema was distilled where every question identified in programmers’ emails was categorized with respect to information artifact and information attribute.

Information artifact: Information artifact refers to the external representation that the information search refers to. There were 13 individual foci identified. Table 2 contains a definition for each of these and examples taken from the dataset 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 artifact 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.

Table 3: Description for information attribute categories

Information attributes	Definition and example
What	Questions which ask what the does (the source code or software tools). 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 an information artifact achieves its goal, how some information artifact is employed or how to proceed. When applied to source code, it often refers to a top-down comprehension strategy (O'Brien *et al*.,2004) Example: "Does anyone know how I can fix this?
Why	Asking for a purpose/explanation of the information artifact. 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?"
Who	Asking for the relevant persons. Example: "Are there any takers?"
Where	Asking about the location of something within the information artifact or about the location of an information artifact. For 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 the Protocol information artifact. Example:"BTW, can we use JIRA for bug reporting for this project instead ..."
Confirmation	Questions that confirm certain information/actions/tasks. Example : "… will it be incorporated into the latest version of BSF?"
Instruction	Question that are asking a community member to do something Example: "Would you consider donating your patch to Apache?"

Table 4: Relationship between information artifact categories and information attribute categories

Info. Focus and Quest. Strategy	What	How	Why	Who	Where	Permission	Confirmation	Instruction
Changes	2	4	1	0	1	4	3	4
File Config.	2	7	2	1	5	1	1	7
Legality and Protocol	1	4	2	0	1	1	1	13
SI-Debug.	15	16	6	33	4	0	0	20
SI-Enhance.	25	12	3	10	3	5	0	27
Stage/Completion	4	11	2	1	0	3	1	15
Support required	2	1	29	1	0	2	0	3
System design	4	13	2	7	1	0	0	38
System document	0	11	5	0	18	3	0	11
Task-Impl.	28	16	23	2	1	2	5	34
Task-Testing	3	4	2	1	0	0	0	5
Tool/Tech.	46	15	10	9	4	5	1	49
Owner	0	0	9	0	0	1	0	0

Information attribute: Information attribute refers specifically to the aspect of information sought by the programmer based on the information artifact. 10 information attributes were derived by open coding of the OS programmers' email communication. These attributes are presented in Table 3. Note that the examples shown in Table 2 and 3 are the actual questions found in the dataset.

RESULTS

The empirical studies: The empirical results described in this section are based on the schema presented above. The schema was used to examine the entire dataset that was used in creating the schema. That is, when the schema was finalized the entire data set was revisited for analysis. When all 708 questions were extracted, they were individually isolated in spreadsheet cells to facilitate categorization with respect to the schema. We then applied content analysis to this dataset, categorizing each question asked by the programmers with the aid of the current schema. We then separated the results of the analysis into different tables. The relationship between information artifact categories and information attribute categories is presented in Table 4. Discussions on these results are presented as follow.

DISCUSSION

Information artifact: The graph in Fig. 1 visualizes result that we gathered for information artifact across all mailing lists across all years of the dataset. Based on this, it shows a similar trend of information artifact across all projects. The essential pattern seems to hold with small emphases in different system. This suggest a high reliability of the schema in characterizing OS programmers' information seeking, The different results for different data sets most probably because of different characteristic among the projects that impact on the result. However the similar trend over all project suggest the reliability of the schema.

Figure 1 suggests that OS programmers' information seeking is very implementation centric where Tools and technology, System Implementation-Enhancement, System Implementation-Debug and Task Implementation gained high requests across all projects in the dataset.

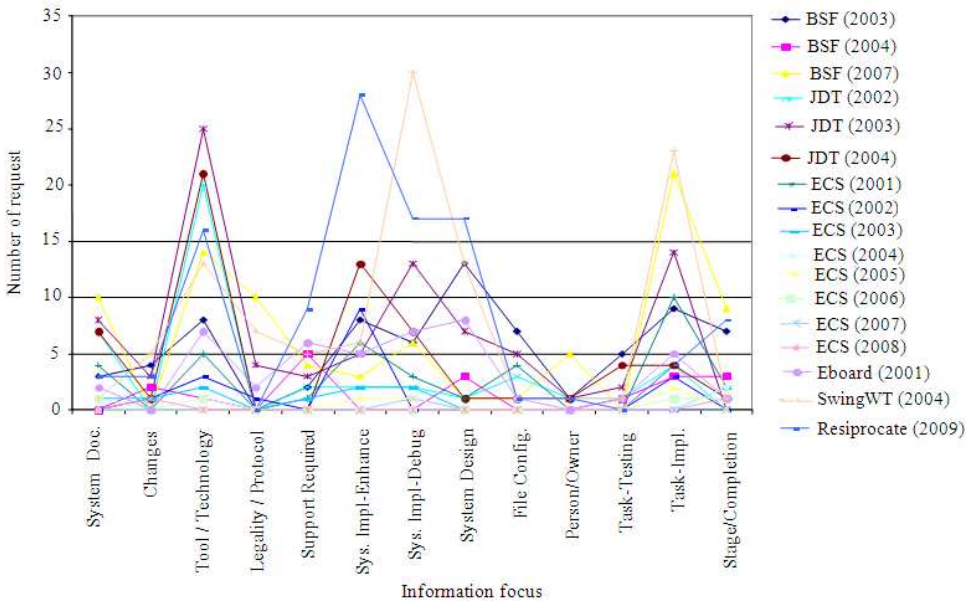

Fig. 1: Patern for information artifact request

Table 5: Ranking for information artifact categories

#	Information artifact	%
1	Tool/Technology	19.21
2	Task-implementation	14.55
3	System implementation-debug	13.42
4	System implementation-enhancement	12.43
5	System design	9.18
6	System documentation	6.64
7	Support required	5.37
8	Stage/Completion	5.37
9	Legality/Protocol	3.39
10	File configuration	3.39
11	Changes	2.97
12	Task-testing	2.12
13	Owner	1.55

This graph also suggests a lesser emphasis on Documentation, System Design and Stage Completion and Protocol across all projects in the dataset. This also suggest that behavior of OS programmers' information seeking seems to less related to information seeking rather than physical artifact seeking such request for helps or seek a person to do a job.

Likewise, when we refer to Table 5 that presents the ranking of these information artifact categories, the result shows emphasis on Tool/Technology (19.21%), Task-Implementation (14.55%), System Implementation-Debug (13.42%) and System Implementation-Enhancement (12.43%). Hence, as suggested in our previous work 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 40.4% of all questions were directed at the code base.

In addition, closer examination of the 'Tool/Technology' focus showed that 89% 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 artifact, this makes in total, 57.7% of request in the dataset was focused on the code based. Hence, this suggests a strong code focus for the all the OSS projects that we have studied.

Such high request for information in 'Tool/Technology' might reflect the rapid changes in tools that used by OS programmers. For example, a version of Java Development Kit namely Java SE 6, had 6 updates released within 11 months in 2010 (Wikipedia, 2010). This rapid change is likely to give impact on the programmers' works such as coding and debugging. This is suggested by examples such as: "Do you remember what version of RELOAD was current, the time you dealt with it?"

Another possible rationale for the high request for 'Tool/Technology' is that many tools available for OS projects. OS programmers might be asking a lot of questions to choose a tool that suitable for them: "Can we use JIRA for bug reporting for this issue instead…." The high request for this type of question also might be related with request of software document. For example, there is question in the dataset asking about user manual for specific tool in use: "Is there any Apache official guidelines on this?"

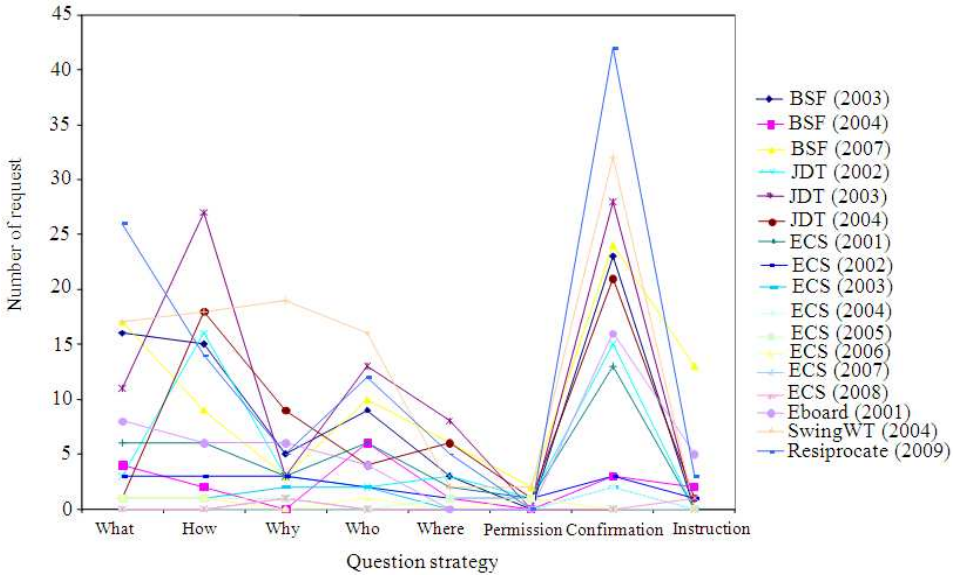

Fig. 2: Patern for information attribute request

The higher figures for Tool/Technology, Task-Implementation, System Implementation-Debug and System Implementation-Enhancement with respect to System Design might reflect the OSS Development Life Cycle. According to Feller and Fitzgerald (2001), in OSS development, planning, analysis and design stages are concatenated and performed typically by a single developer or small core group. Design decisions are generally made in advance, before the larger pool developers starts to contribute. Hence, most of OS programmer's contribution is directed at the systems' implementations. However, since the design decisions were made in advance, it also possible that OS programmers looking for information about system design as they were not involved in making the design decision. This could be the reason for the considerable proportion for System Design question (9%) in the maintenance phases of these projects. This also may also be reflected in the high proportion of implementation-based queries, although this theme is consistent of studies of commercial programmers (O'Shea, 2006; Ko *et al*., 2007).

We have also previously reported an unanticipated finding (Sharif and Buckley, 2008a; 2008b; 2009a; 2009b) with regards to programmers' 'System Documentation' requests. Specifically we noted that documentation seemed to play an important part in OS programmers' information requests. This was unusual because other 'information source' literature suggested that programmers distrusted documentation (Singer 1998; Seaman, 2002; Sousa *et al*., 1998).

The data shown in Table 5 reinforces this findings but with smaller emphasize, System Documentation was ranked the 6th most requested artifacts across all projects in the dataset. Over all years of all projects, almost 7% of the questions were 'System Documentation' questions. Given the large number of total questions (708 questions from 2104 emails) in the dataset, it suggests that documentation is regarded as an important issue for programmers (55 questions). The delocalized nature of OS programmers might be the reason for a higher than expected reliance on this documentation.

As OS programmers 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 possible that 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 perhaps trust documentation more than in the traditional case.

Information attribute: As with the information artifact, the result that we obtained in Fig. 2, presents a similar pattern of information attribute across all projects with the essential pattern seems to hold with small emphases in different system. Such pattern indicates the high reliability of the schema. This also means that the overall trend of OS programmers' information seeking behavior is team oriented in nature. Such trends imply that OS programmers are often asking for confirmation for their sought information and want to know who is relevant for that particular information. Both are expected in a delocalized environment. This graph also align closely with the findings by Letovskys (1987); where lots of what and how questions and a lesser number of why questions.

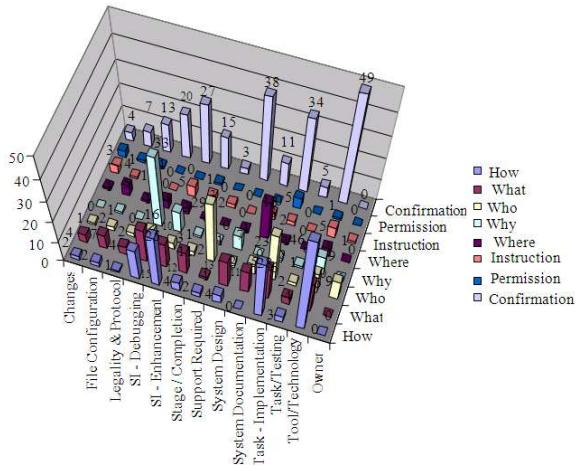

Fig. 3: Relationship between information artifact and information attribute

Table 6: Ranking for information attribute categories

Rank	Information attribute	%
1	Confirmation	31.64
2	How	19.35
3	What	16.24
4	Who	12.29
5	Why	9.04
6	Where	5.37
7	Instruction	3.67
8	Permission	1.41

Table 6 present the ranks of information attribute employed in the information request in the dataset. The 5 most frequent information attribute traced in the dataset were Confirmation (32%), How (19%), what (16%), Who (12%) and Why (9%). In addition, the 6th ranking strategy of 'Where' questions with 5% request rate were also considered as significant. This is in accordance with our previous studies that suggest that there is strong team-orientation among programmers, and the existence of Location type queries among programmers.

Team orientation question: Upon closer analysis, the data presented in Table 6 is in line with Sullabi and Shukur (2008) and in keeping with another of our preliminary finding (Sharif and Buckley, 2008a; 2008b; 2009a; 2009b), as there is a strong emphasis on, the 'Confirmation' questions and the 'Who' questions. 'Confirmation' questions accounted for approximately 31.64% of all questions, and were the most frequently asked type of question. Likewise 'Who' type of questions was also popular, accounting for 12.29% of all questions. This emphasis on confirmation and who questions may reflect the effort to maintain awareness among delocalized programmers reported by Gutwin *et al*. (2004). The Confirmation questions also reflect the Pre-Commit Testing stage (Jorgensen, 2001) in OSS life cycle for changes. Pre-Commit Testing test is done before the new code is integrated with the other codes in project's repository or released to other developers to prevent the new code from breaking the other tested code. In this context, it is understandable if OS programmers need to confirm their changes with the community members.

With regard to the high percentage for who question (12.29%) in the mailing list, given Ko *et al*. (2007) findings; this is not an entirely unexpected result. This is because, 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.

Location type queries: On the other hand, the data in Table 6 have also shown to correspond to our original findings that exhibit the presence of 'Location' type queries in the questions that we obtained. This finding is in accordance with. Based on this, we manage to identified 38 questions which were location oriented (i.e., the 'Where' questions). This type of question represents 5.37% of all questions asked, suggesting that this is a significant information seeking type for OS programmers maintaining large systems. Theses finding add empirical credence to Marcus *et al*. (2005) 'Concept Location' work.

Relationship between information artifact and information attribute: In order to further analyze these results, we present a 2-dimensional relationship between information artifact categories and information attributes categories by number of request for both dimensions in Fig. 3.

The top 5 highest request in the dataset was Confirmation on Tool/Technology (49 requests), How questions on Tool/Technology (46 requests), Confirmation on System Design (38 requests), Confirmation on Task-Implementation (34 requests) and Why questions on System Implementation-Debug (33 requests). This figure reinforces our finding that OS programmers' information seeking is very implementation centric and they have strong team-oriented nature.

Task-Implementation and System Implementation-Debug are reflecting focus on implementation. System design wills not normally reflecting system implementation. However, in closer view, the Confirmation on System Design information is likely asked to confirm certain design issues with intention to do implementation task. Given the fact that in OS settings system design is normally done by "small-

core" group and the OS development is located primarily at the implementation phase (Feller and Fitzgerald, 2001), request for System Design is seems to have the agenda of code implementation. The high request for how (Tool/Technology) and why (System Implementation-debug) aspect of information is also reflecting the implementation focus in the questions.

Likewise, most of the questions for Tool/Technology (as shown in Fig. 4) in the information artifact is around the How questions, which suggests a strong implementation centric in the questions. The other information attribute with highest request in Tool/Technology category is the Confirmation questions.

Another interesting finding shown earlier in Fig. 3 is the request on System Documentation that is higher than previously reported for non OS programmers. Upon closer analysis, Fig. 4 shows that majority of request for System Documentation is targeted on Where aspect. This indicates the needs for a tool or webpage that can point them to required document. Another high request for this category was on What (11 requests) and Confirmation (11 request). This suggesting OS programmers tendencies to refer to document to get confirmation on certain information and to know about newly found (what) subject.

Besides that, Fig. 3 also shows that OS programmers often asked the Why questions for System Implementation-Debug (33 requests), the How question for System Implementation-Enhancement (25 questions) and the How question for Task-Implementation (28 request). This is intuitively understood. According to a popular definition, debugging is a methodical process of finding and reducing the number of bugs. Hence, it is understood when programmers asking why to find cause of errors. Likewise, how questions is likely asked to get guide or suggestion to enhance particular piece of code or guide in doing the enhancement (task).

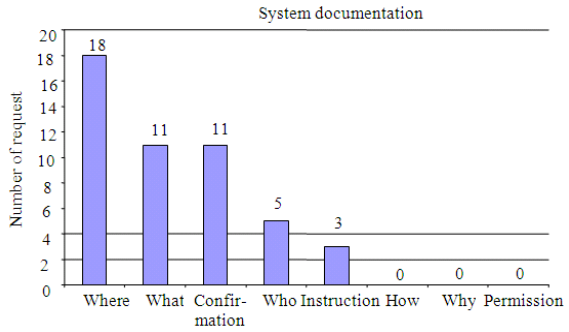

Fig. 4: Information attributes for system documentation

CONCLUSION

This study observed OS programmers information seeking through questions found in OS projects mailing list. All the extracted questions were analyzed based on Information Seeking Schema employed from previous studies. In doing this, we found results that reinforce previous findings. Specifically, we found:

- A similar pattern of information artifact and attribute across all projects
- Supporting evidence that OS programmers were highly implementation centric when much of the programmers' information seeking was directed at the systems' implementations
- OS programmers have also shown to repeatedly require location information and that they are quite team-oriented
- OS programmers tended to rely on documentation more than what have been previously reported for non OS programmers

These findings from different insights demonstrate the applicability of the ISS of OS programmers. This schema is an open schema allowing further evaluation and refinement and can be replicated for future research in this area (This schema is available from the first author on request). By determining the information that the programmers frequently seek, this research defines the requirements for visualization tools that truly support programmers in their maintenance of 'information-seeking' endeavors.

ACKNOWLEDGMENT

This study was supported, in part, by Science Foundation Ireland grant 03/CE/I303_1 to Lero-The Irish Software Engineering Research Centre (www.lero.ie).

REFERENCES

Belady, L.A. and M.M. Lehman, 1976. A model of large program development. IBM Syst. J., 15: 225-252. DOI: 10.1147/sj.153.0225

Boehm's, B.W., 2007. Software Engineering Economics, in Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research. 1st Edn., Wiley-IEEE Computer Society Pr., ISBN-10: 047014873X, pp: 832.

Buckley, J., C. Exton and J. Good, 2004. Characterizing programmers' information-seeking during software evolution. Proceedings of the in International Workshop on Software Technology and Engineering Practice, Sept. 17-19, Chicago, IL., pp: 7-29. DOI: 10.1109/STEP.2004.7

Cleary, B. and C. Exton, 2007. Assisting concept location in software comprehension. Proceedings of the 19th Annual Psychology of Programming Interest Group Conference, Aug. 16, Joensu, Finland. http://www.enterpriseresearch.ie/?p=251

Cleary, B., C. Exton, J. Buckley and M. English, 2008. An empirical analysis of information retrieval based concept location techniques in software comprehension. Empirical Software Eng., 14: 93-130. DOI: 10.1007/s10664-008-9095-3

Curtis, B., H. Krasner and N. Iscoe, 1988. A field study of the software design process for large systems. Commun. ACM, 31: 1268-1287. DOI: 10.1145/50087.50089

Daniel, S., K. Stewart and D. Darcy, 2009. Patterns of evolution in open source projects: A Categorization Schema and Implications. http://misrc.umn.edu/workshops/2009/spring/Stewart.pdf

De Lucia, A., A.R. Fasolino and M. Munro, 1996. Understanding function behaviors through program slicing. Proceedings of the 4th International Workshop on Program Comprehension, Mar. 29-31, IEEE, Berlin , Germany, pp: 9-18. DOI: 10.1109/WPC.1996.501116

Feller, J. and B. Fitzgerald, 2001. Understanding Open Source Software Development. 1st Edn., Addison-Wesley, Pearson Education Limited, ISBN-10: 0201734966, pp: 224.

Fitzgerald, B., 2004. A critical look at open source. Computer, 37: 92-94. DOI: 10.1109/MC.2004.38

Gacek, C. and B. Arief, 2004. The many meanings of open source. Software, IEEE, 21: 34-40. DOI: 10.1109/MS.2004.1259206

Good, J., 1999. Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension. The University of Edinburgh: Edinburgh, UK. http://www.dart-europe.eu/full.php?id=299441

Gutwin, C., R. Penner and K. Schneider, 2004. Group awareness in distributed software development. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, Nov. 06-10, ACM, Chicago, Illinois, USA., pp: 72-81. DOI: 10.1145/1031607.1031621

Iskandarani, M.Z., 2008. Effect of Information and Communication Technologies (ICT) on non-industrial countries-digital divide model. J. Comput. Sci., 4: 315-319. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.8042&rep=rep1&type=pdf

Jorgensen, N., 2001. Putting it all in the trunk: incremental software development in the FreeBSD open source project. Inform. Syst. J., 11: 321-336. DOI: 10.1046/j.1365-2575.2001.00113.x

Kemerer, C.F. and S. Slaughter, 1999. An empirical approach to studying software evolution. Software Eng., IEEE Trans., 25: 493-509. DOI: 10.1109/32.799945

Kingrey, K.P., 2002. Concepts of information seeking and their presence in the practical library literature. Library Philosophy Practice, 4: 1-8. http://www.webpages.uidaho.edu/~mbolin/saunders.PDF

Ko, A.J., R. DeLine and G. Venolia, 2007. Information needs in collocated software development teams. Proceedings of the 29th International Conference on Software Engineering, May 20-26, IEEE Computer Society Washington, DC, USA., pp: 344-353. DOI: 10.1109/ICSE.2007.45

Krippendorff, K.H., 2004. Content Analysis: An Introduction to its Methodology. 2nd Edn., Sage Publications, , Inc., ISBN-10: 9780761915454, pp: 440.

Letovsky, S., 1987. Cognitive processes in program comprehension. J. Syst. Software, 7: 325-339. DOI: 10.1016/0164-1212(87)90032-X

Lientz, B.P., E.B. Swanson, and G.E. Tompkins, 1978. Characteristics of application software maintenance. Commun. ACM, 21: 466-471. DOI: 10.1145/359511.359522

Marcus, A., V. Rajlich, J. Buchta, M. Petrenko and A. Sergeyev, 2005. Static techniques for concept location in object-oriented code. Proceedings of the 13th International Workshop on Program Comprehension, May 15-16, IEEE Computer Society Washington, DC, USA., pp: 33-42. DOI: 10.1109/WPC.2005.33

Mayrhauser, A.V. and A.M. Vans, 1993. From code understanding needs to reverse engineering tool capabilities. Proceedings of the 6th International Conference on Computer-Aided Software Engineering, Jul. 19-23, IEEE, pp: 230-239. DOI: 10.1109/CASE.1993.634824

O'Brien, M.P. and J. Buckley, 2005. Evolving a model of the information-seeking behavior of industrial programmers-an empirical approach. Proceedings of the 13th International Workshop Program Comprehension, May 15-16, IEEE, pp: 125-134. DOI: 10.1109/WPC.2005.24

O'Brien, M.P., J. Buckley and T.M. Shaft, 2004. Expectation-based, inference-based, and bottom-up software comprehension. J. Software Maintenance Evolut.: Res. Practice, 16: 363-447. DOI: 10.1002/smr.v16:6

O'Brien, M.P., T.M. Shaft and J. Buckley, 2001. An open-source analysis schema for identifying software comprehension processes. Proceedings of the 13th Workshop of the Psychology of Programming Interest Group (WPPIG'01), Bournemouth UK, pp: 129-146. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.5397&rep=rep1&type=pdf

O'Shea, P.A., 2006. An Investigation of Views and Abstractions Employed by Software Engineers during Software Maintenance. 1st Edn., University of Limerick, Limerick.

Pandit, N.R., 1996. The creation of theory: A recent application of the grounded theory method. Qualitative Report, 2: 1-20. https://ueaeprints.uea.ac.uk/28591/

Pennington, N., 1987. Comprehension strategies in programming. Proceedings of the 2nd Workshop in Empirical Studies of Programmers (WESP'87), Ablex Publishing Corp, NJ, USA. ISBN: 0-89391-461-4

Prechelt, L., B. Unger, M. Philippsen and W. Tichy, 1998. Re-evaluating inheritance depth on the maintainability of object-oriented software. Int. J. Empirical Software Eng., 1-16. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.7652&rep=rep1&type=pdf

Pressman, R., 2004. Software Engineering: A Practitioner's Approach. 6th Edn., McGraw-Hill Science/Engineering/Math, England, ISBN-10: 007301933X, pp: 880.

Seaman, C.B., 2002. The information gathering strategies of software maintainers. Proceedings of the 8th International Conference on Software Maintenance, Oct. 3-6, Montreal, Quebec, Canada, pp: 0141. http://doi.ieeecomputersociety.org/10.1109/ICSM.2002.1167761

Sharif, K.Y. and J. Buckley, 2008a. Developing schema for open source programmers' information-seeking. Proceedings of the International Symposium on Information Technology, Aug. 26-28, IEEE, Kuala Lumpur, pp: 1-9. DOI: 10.1109/ITSIM.2008.4631611

Sharif, K.Y. and J. Buckley, 2008b. Observing open source programmers' information seeking. Proceedings of the 20th Annual Psychology of Programming Interest Group Conference (APIGC'08), Lancaster University, Lancaster, United Kingdom, pp: 1-10. http://www.ppig.org/papers/20th-sharif.pdf

Sharif, K.Y. and J. Buckley, 2009a. Further Observation of Open Source Programmers' Information Seeking. Psychology of Programming Interest Group, University of Limerick, Ireland, pp: 1-12. http://www.ppig.org/papers/21st-sharif.pdf

Sharif, K.Y. and J. Buckley, 2009b. Observation of open source programmers' information seeking. Proceedings of the IEEE 17th International Conference Program Comprehension, May 17-19, IEEE Computer Society, Vancouver, British Columbia, Canada, pp: 307-308. DOI: 10.1109/ICPC.2009.5090071

Sim, S.E., 1998. Supporting Multiple Program Comprehension Strategies during Software Maintenance. A thesis submitted in conformity with the requirements for the degree of Master of Science Graduate, Department of Computer Science, University of Toronto. http://www.ics.uci.edu/~ses/msc/thesis.pdf

Singer, J. and T. Lethbridge, 1998. Studying work practices to assist tool design in software engineering. Proceedings of the 6th International Workshop on Program Comprehension, Jun. 24-26, IEEE, Ischia, Italy, pp: 173-179. DOI: 10.1109/WPC.1998.693348

Singer, J., 1998. Work practices of software maintenance engineers. Proceedings of the 4th International Conference on Software Maintenance Mar. 16-19, Bethesda, Maryland, pp: 139. http://www.computer.org/portal/web/csdl/doi/10.1109/ICSM.1998.738502

Sommerville, I., 2008. Software Engineering. 7th Edn., Pearson Education India, ISBN: 8131724611, pp: 864.

Sousa, M.J. Castro and H.M. Moreira, 1998. A survey on the software maintenance process. Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Washington, DC, USA., pp: 265-274. ISBN: 0-8186-8779-7

Stein, C., G. Cox and L. Etzkorn, 2005. Exploring the relationship between cohesion and complexity. J. Comput. Sci., 1: 137-144. http://www.doaj.org/doaj?func=abstract&id=116220

Sullabi, M.A. and Z. Shukur, 2008. CSCW for preparing formal software specifications: issues and implementation. J. Comput. Sci., 4: 333-340. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.8548&rep=rep1&type=pdf

Wiedenback, S. and C.L. Corritore, 1991. What do novices learn during program comprehension? Int. J. Hum.-Comput. Interact., 3: 199-222. DOI: 10.1080/10447319109526004

Wikipedia, 2010. Java version history. http://en.wikipedia.org/wiki/Java_version_history

Zayour, I. and T.C. Lethbridge, 2001. Adoption of reverse engineering tools: A cognitive perspective and methodology. Proceedings of the 9th International Workshop on Program Comprehension, May 12-13, IEEE, Toronto, Ont., Canada, pp: 245-255. DOI: 10.1109/WPC.2001.921735