Abstract

Information-seeking has been recognized as a core subtask in software maintenance. While much of the research in this area attempts to identify the information sought by programmers, the work to date has been typically based on pre-defined information types limiting the potential for new information types to arise. Hence, this paper presents a small empirical study on open-source programmer mailing lists, to iteratively refine a schema of the information that programmers seek based exclusively on an analysis of these programmers’ naturalistic communication. Subsequently it places this schema in the context of the existing literature in the area. The analysis was undertaken on information requests in 288 emails, taken from 2 open source project mailing lists; The Java Bean Scripting Framework (BSF)[1] and the Java Development Tool (JDT)[2]. This work produced a preliminary schema of 3 main categories divided into 18 sub categories.

1. Introduction.

Software maintenance and evolution are considerable parts of the 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 [3-6].

Maintenance itself can be divided into two general stages: “Understanding the program and actually performing the change” [7]. 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 [8].

Information-seeking has been defined as the searching, recognition, retrieval and application of meaningful content [9]. It has been recognized as a core subtask in software maintenance[10], [11] [12], [13], [14]. Sim [13], 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.

In this research area O’Brien [15] and Vaclav [16] concentrate on the information-seeking processes of programmers during maintenance. Other researchers, for example Singer [17] and Seaman [11], have studied the information sources that programmers use. However, there here have been several empirical studies that also aim to inform on the information types sought by programmers in the context of software comprehension [17-23]. Several of these studies [20-23] derive from a theoretical analysis of the information available in programs originally carried out by Pennington [20]. While this analysis is a valuable contribution, its predefined nature places limits on the discovery of new information types.

Ko [18] and Letovsky [19] are 2 exceptions to this limitation, as they used an open-coding approach to explore their data. However, in Letovsky’s research programmers were asked to study an artificially small system and both studies required programmers to think-aloud, thus affecting programmers’ normal maintenance protocol.
This work attempts to replicate and improve upon Letovsky’s [19] and Ko’s [18] work. It is performed in an open-source context where programmer mailing lists provide a near complete, explicit, and yet entirely natural, dataset of their communications. In addition, it provides insight on the type of information programmers who maintain realistically large systems require and, in particular, information needs in a geographically-distributed context. Hence, developing such a schema will be an important contribution in the area.

The paper will first discuss our empirical study. It will describe how the schema was developed and refined over the course of our analysis. Subsequently it will review the other literature in the area and place the schema in the context of this literature. Thus it will highlight new insights that have been gained from this naturalistic empirical approach to open-source distributed software development activity.

2. The empirical study

This study described in this paper is a pilot study that examined the information sought by open source programmers during software evolution of the Java Bean Scripting Framework (BSF) and Java Development Tool (JDT) project. It was designed in such a way that the participants were unaware of any monitoring, thus heightening the naturalistic nature of the data-set. The BSF programmers’ mailing list was captured for the period from January to August 2007 (only available archive in this mailing list during the analysis), while for the JDT, mails were captured from January to December 2003 (first year of this archive) and this served as the data for open-coding. The resultant data set consisted of 288 emails communicating with the programmers’ community and from this data-set all questions asked by the programmers were extracted.

2.1. Open-coding

Open-coding is a qualitative data analysis method. Bogdan and Biklen [24] define qualitative data analysis as “working with data, organizing it, breaking it into manageable units, synthesizing it, searching for patterns, discovering what is important and what is to be learned, and deciding what you will tell others”. We used open-coding to identify categories of questions in these open source mailing list. This is called as inductive data analysis in open-coding [25]. Data are compared and similar incidents are grouped together and given the same conceptual label if appropriately close. The process of grouping concepts at a higher, more abstract, level is termed as categorizing [26]. The goal is to create descriptive, multi-dimensional categories which form a preliminary framework for analysis as suggested by Hoepfl [25].

Using open-coding on open source mailing lists can provides a wealth of information on a naturalistic data-set. The medium of email list communication, was described by Mockus et al. [27], 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, 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.

2.2. Categorization

Initial investigations showed that most of the questions in programmers’ emails were asked without explicit indicators like question marks or signaling words such as ‘what, where…’. As a result, the questions in the mailing list had to be extracted manually. 288 emails were analysed in this fashion and 98 questions was extracted.

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 [21], [28]. The first author carried out a detailed analysis of this data, naming and categorizing each question asked by the programmers. This open-coding procedure (see section 2.1) and is carried out without the aid of a coding manual or schema, the coder effectively creating the categories from scratch. Accordingly, the first author 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 create categories based on the contents of portions of the transcripts being examined (as suggested by O’Brien [29]). This analysis was performed iteratively, each iteration marked by a discussion review with the second author.

Over time, a number of provisional categories began to emerge with respect to these sections. Those categories were then applied to other question data-sets and refined by means of merging and renaming categories. Finally, a set of categories seemed increasingly resistant to change and a final set of 3 main categories, divided into 18 sub categories was established.

During early stages of the analysis, 10 provisional types of information were identified. These were (the number in bracket reflect the number of occurrences of these question type for BSF mailing list) Request for Documentation (6), Validation for Changes / Product Sufficiency (5), Request for Bug Fix Support (4), Legality / Protocol (4), Tool/Technology Changes (4), Request for Awareness (3), Validation for Correctness of Changes (2), Design (2) Correction Procedure (2) and, File Configuration (1).

After a few more cycles of open-coding, several common features shared by 2 or more categories were found. So we decided to divide the category into 2 levels, one general and one specific. This is shown in Table 1.

Table 1. Refinement of information type

<table>
<thead>
<tr>
<th>Level I</th>
<th>Level II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td><strong>Correctness of Changes</strong></td>
</tr>
<tr>
<td>Questions to validate something.</td>
<td>This category is for questions that are asking others to verify the correctness of changes the programmer made.</td>
</tr>
<tr>
<td><strong>Sufficiency of Changes</strong></td>
<td>Sometimes programmers are not sure that the changes they are doing are sufficient or still required to solve certain problems.</td>
</tr>
<tr>
<td><strong>Procedure</strong></td>
<td><strong>Tool / Technology</strong></td>
</tr>
<tr>
<td>Questions to ask information on the correct procedure.</td>
<td>This category of question is asking how new tools / software are used</td>
</tr>
<tr>
<td><strong>Legality / Protocol</strong></td>
<td>This category is asking about the protocol for doing something. Open source programmers need to follow certain protocols to, for example, make sure a programmer is qualified to contribute.</td>
</tr>
<tr>
<td><strong>Awareness</strong></td>
<td><strong>Enhancement</strong></td>
</tr>
<tr>
<td>Pushing</td>
<td>Sometimes programmers want to update / enhance their knowledge / information. This type of question is use to ask for the latest information</td>
</tr>
<tr>
<td><strong>Request for Support Required</strong></td>
<td>This category of question is to ask another programmer to take the responsibility for a task.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information focus</th>
<th>System Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical question</td>
<td>These are questions that refer to the code base</td>
</tr>
<tr>
<td><strong>System Documentation</strong></td>
<td>Open source programmers often need to confirm something using documents such as official guidelines and emails.</td>
</tr>
<tr>
<td><strong>System Design</strong></td>
<td>Question to check the design of the software.</td>
</tr>
<tr>
<td><strong>File Configuration</strong></td>
<td>Question about the system’s configuration.</td>
</tr>
<tr>
<td><strong>Bucket Category</strong></td>
<td>Questions that do not fit with any of above category.</td>
</tr>
</tbody>
</table>

Later a more general framework was uncovered where we identified the target of the request, the type of information sought on that target and the certainty with which the question was phrased. Hence the final schema was developed based on these main categories; i) Information focus; ii) Question Strategies; iii) Knowledge Strength.

3.1. The schema

The resultant schema is presented in Figure 1. Every question identified in the mailing list has one attribute from each of these categories.
3.2. Information focus

There are 9 information focuses identified. Table 2 contains a definition for each of these. Please note that all of examples presented in Table 2 are taken from the open-source dataset of our BSF study:

<table>
<thead>
<tr>
<th>Information Focus</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Doc. Changes</td>
<td>Questions referring to the documentation: Example: “Is there any Apache official guidelines on this?”</td>
</tr>
<tr>
<td>Tool / Technology</td>
<td>Questions that refer to technology or tools. Example: “Can we use JIRA for bug reporting for this issue instead...”</td>
</tr>
<tr>
<td>Legality / Protocol</td>
<td>Questions about the protocol to follow. Example: “Did you got the approval to contribute your work to BSF?”</td>
</tr>
<tr>
<td>Support Required</td>
<td>Questions that ask another programmer to take on responsibility or tasks. Example: “There are 2 non-filed open issues..... Are there any taker?”</td>
</tr>
<tr>
<td>System Implementation</td>
<td>Question to understand the code. Example: “(Given a situation...)I have no idea why this is happening. Please help me solve this problem”</td>
</tr>
<tr>
<td>System Design</td>
<td>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?”</td>
</tr>
<tr>
<td>File Configuration</td>
<td>Question about configuration management. Example: “What is the distribution directory in the src zip/tgz?”</td>
</tr>
<tr>
<td>Person</td>
<td>Question about the person in-charge for some task. Example: “Who is the team / person in charge for documentation?”</td>
</tr>
</tbody>
</table>

3.3. Question strategy

We found 7 question strategies employed by open source programmers in their email communication. The strategies are presented in Table 3:

<table>
<thead>
<tr>
<th>Question Strategy</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>What</td>
<td>Questions which ask what a file, programming object does. Representing the bottom-up program comprehension strategy employed by programmers. Example: “What is the .rep file?”</td>
</tr>
<tr>
<td>How</td>
<td>Questions which ask the way some goal of the program / job is accomplished. Example: “Does...”</td>
</tr>
<tr>
<td>Question Strategy</td>
<td>Information Focus</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>How</td>
<td>System Doc 6</td>
</tr>
<tr>
<td>Where</td>
<td>Changes 5</td>
</tr>
<tr>
<td>Who</td>
<td>Tool / Technology 5</td>
</tr>
<tr>
<td>What</td>
<td>Legality / Protocol 5</td>
</tr>
<tr>
<td>Why</td>
<td>Support Required 4</td>
</tr>
<tr>
<td>Relationship</td>
<td>System Impl. 4</td>
</tr>
<tr>
<td>Permission</td>
<td>System Design 2</td>
</tr>
</tbody>
</table>

### 3.5. Schema application result.

Figure 2 below summarizes the results when the schema was applied to the BSF project mailing list.

**Table 4. Knowledge strength**

<table>
<thead>
<tr>
<th>Hypothesis Based Question</th>
<th>Knowledge Strength</th>
</tr>
</thead>
</table>
| Questions that are asked with an idea as to their answer already in mind. That is, the question comes with suggestion for the answer. This type of question is asked to validate, to confirm or to correct the suggested answer. Normally the person who asked this question has found a related information source, and has some confidence in the information that he/she has. They ask the question to reconfirm the status of information that he got. Example: "I see there's a JIRA issue now, and my changes would've been needed anyway, so I hope you're ok?"

<table>
<thead>
<tr>
<th>Straight Question</th>
<th>Question in BSF Mailing List</th>
</tr>
</thead>
</table>
| 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. So the difficulty level for the information sought is higher in this instance and the apparent availability of a source is low. Example: "Is there any news on access to the JSR-223 TCK? Or any idea how long it might take to get access?"

**Request for Documentation** was the most sought of information focus with 6 occurrences out of 33. ‘What’ was the most used question strategy.
As regard Knowledge Strength, the majority of the questions were Straight Questions.

In an informal attempt to assess the reliability of this schema, the first and second author randomly chose 10 questions from the sample and applied the schema independently. On comparing results there was a high correlation. We also applied the schema to another mailing list for the JDT [2] project, which consist of email archives for 2 years.[23] and again with similar reliability results.

4. Preliminary findings

While the categories we have identified may not be complete, the data has already produced several surprising findings with the current literature. These include the high request rate for documentation and the process-oriented nature of the requests.

4.1. Request for documentation

Request for Documentation was one of the most frequently sought information targets. 6 out of 33 questions were requests for documentation.

This is at odds with previous studies that suggest software document is not the preferred reference for programmers (see section 5).

Hence, it is possible that open source programmers rely much more on documentation than these other programmers. Open source programmers tend to work in different locations, and are extremely separated. Hence they cannot rely on informal communication with their team and are more likely to need some reference material in hand while doing their job. These provisional findings suggest new research questions on how working environment differences affecting the programmers’ tendencies to use documentation as a reference in software maintenance.

4.2. Process Oriented

Our findings suggest a largely process-orientation nature in open-source programmers’ information-seeking. In the Information Focus subcategories for example, only 15 of the 33 questions refer to process issues.

Again, this could be based on the geographically distributed nature of the development. Specifically, programmers who have never met, may not be able to use informal communication or observed cues to understand the protocol of software development within the development team. Consequently, they explicitly need to ask for this information. Again, these results are provisional and need re-enforcement through additional or enlarged studies.

5. Findings in relation to existing literature

Our literature review shows that most of the research in this area suggests that information seeking is very code oriented, in agreement with Sousa’s [30] and Singer’s [12, 17, 31] finding that programmers rely predominantly on source code. On this basis, Pennington identified five information types available from the source code in her landmark studies [32, 33]. Subsequently, many studies in the area have been heavily influenced by her work, specifically taking her predefined schema as the basis of their studies [21], [34], [23, 35] and [22]. In one typical example of this work, O’Shea [23, 35] did content analysis on one open source 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 [36]. This category wasn’t present in Pennington’s initial analysis, possibly because Pennington only considered individual programmers studying small code pieces. In this scenario, location wasn’t an issue. Examples like this suggest that Pennington’s schema should be expanded to consider larger systems and team-based development. In addition, it also suggests that the schema might blinker researchers from seeing other information types associated with programmers’ needs.

In contrast, our inductive schema is quite process oriented. As can be seen in figure 2 and discussed in section 5.2, many of the subcategories on the mailing lists refer to the context of the development with only several sub categories referring to the code itself. This is re-enforced when the number of questions in each sub-category are summed. In the ‘Information Focus’ categories for example only 11 of the 33 questions refer to implementation (system implementation and changes). This shows that, based on open-coding from the data, the schema’s broad scope is appropriate in the context of distributed open-source development.

Additionally, while previous studies of the information sources used by programmers [12], [11] and [30] suggest that programmers rely heavily on the code, other (trusted) programmers, the customers
execution traces and to a lesser degree system
documentation. However, our preliminary findings
suggest that documentation is the most commonly
sought information focus in this context, as reported
on in section 3.5. This suggests that, at least in an
open-source distributed development context,
documentation plays a larger role.

In contrast to Pennington-based work in this
area, Ko’s experiment [18] was a relevant example
of an open-coding approach to identifying
programmer’s information-seeking needs. His study
looked at programmers information needs (and the
difficulty programmers have in meeting those
information needs) as they maintained commercial
software. The maintenance team were co-located and
this forced Ko to use think-aloud data, imposing a
slightly less realistic context on the study.

On the other hand, our study was done in open
source environment where the programmers are
located at different places. Thus, communication
happened naturalistically through the mailing list. In
addition, several of the information sources noted by
Singer[12] and Seaman[11] were unavailable and/or
difficult to find. This might be the reason why
documentation, legality/protocol and, “who”
question were more apparent in out findings. This
current study suggests that open source programmers
who are located in separate locations could require
documentation more often. It also suggest that open
source programmer have less information about
Legality and Protocol, again because of separate
location. The current study, for example, shows
Legality and Protocol were among the highest
information foci (5 out of 33) sought by
programmers. Open source programmers also tend to
ask about ‘who’ which is not as apparent among
collocated development team. Previous studies of
programmers studying unfamiliar code showed no
evidence of ‘who’ questions, probably because, in
that context (not a team context), ‘who’ questions
were irrelevant [20-22].

Ironically, this work reflects most closely one of
the earliest core works in the area: that of Letovsky
[19]. The category of question strategy extends the
‘why’, ‘what’ and ‘how’ question types in his
empirical study. Also the category of Knowledge
Strength map to the ‘whether’ and ‘discrepency’ in
Letovsky’s study. The ‘whether’ questions map quite
nicely onto Hypothesis Based Question and
discrepancy questions map to refuted hypothesis.
Again though, this work can be seen as extending his
work in the context of the team-based development of
a large software system. Specifically, the ‘who’
questions refer to the team-based nature of the
development, requesting information on the member
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. Finally,
the ‘Where’ questions referred, as in O’Shea’s [36]
study, to requests for the location of specific
functionality or fixes in the large code-base.

6. Conclusion

Our literature review suggests that limited
research has been carried out to develop a holistic
information-seeking schema based on an analysis of
programmers’ naturalistic communications
Essentially then, this paper examined the information
sought by open source programmers when
maintaining large open source project. Based on the
finding we proposed a schema for assessing questions
or requests for information in open source dialog.
While the categories we have identified may not be
complete, the data has already produced several
surprising findings: the high request rate for
documentation and the process-oriented nature of the
requests. These provisional findings suggest new
research questions for further investigations. For
future work, this schema will be applied on bigger
sample to refine it and the information needs of open-
source programmers will be elucidated.

7. Reference

(BSF),
Source:http://jakarta.apache.org/site/mail2.html,”,
November 2007

http://dev.eclipse.org/mhonarc/lists/jdt-
dev/maillist.html, January 2008

“Characteristics of application software maintenance,”
466-471.

understanding needs to reverse engineering tool
capabilities,” Proc. Sixth International Conference on
Computer-Aided Software Engineering (CASE’93),
1993, pp. 230-239.

Practitioner’s Approach, McGraw-Hill Publishing


