An Empirically-based Characterization and Quantification of Information Seeking through Mailing Lists During Open Source Developers' Software Evolution

Khaironi Y. Sharif[1], Michael English[2], Nour Ali[3], Chris Exton[2] J.J. Collins[2] and Jim Buckley[2]

[1] Software Engineering Research Group (SERG), FSKTM, Universiti Putra Malaysia, Malaysia
[2] The Irish Software Engineering Research Centre (LERO), University of Limerick, Ireland
[3] University of Brighton, UK

Abstract

Context

Several authors have proposed information seeking as an appropriate perspective for studying software evolution. Empirical evidence in this area suggests that substantial time delays can accrue, due to the unavailability of required information, particularly when this information must travel across geographically distributed sites.

Objective

As a first step in addressing the time delays that can occur in information seeking for distributed Open Source (OS) programmers during software evolution, this research characterizes the information seeking of OS developers through their mailing lists.

Method

A longitudinal study that analyses 17 years of developer mailing list activity in total, over 6 different OS projects is performed, identifying the prevalent information types sought by developers, from a qualitative, grounded analysis of this data. Quantitative analysis of the number-of-responses and response time-lag is also performed.

Results

The analysis shows that Open Source developers are particularly implementation centric and team focused in their use of mailing lists, mirroring similar findings that have been reported in the literature. However novel findings include the suggestion that OS developers often require support regarding the technology they use during development, that they refer to documentation fairly frequently and that they seek implementation-oriented specifics based on system design principles that they anticipate in advance. In addition, response analysis suggests a large variability in the response rates for different types of questions, and particularly that participants have difficulty ascertaining information on other developer's activities.

Conclusion

The findings provide insights for those interested in supporting the information needs of OS developer communities: They suggest that the tools and techniques developed in support of co-located developers should be largely mirrored for these communities: that they should be implementation centric, and directed at illustrating "how" the system achieves its functional goals and states. Likewise they should be directed at determining the reason for system bugs: a type of question frequently posed by OS developers but less frequently responded to.

1 Introduction and Motivation

Software maintenance and evolution are large components of a successful 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 [1-4]. While the empirical basis for such statements is dated and suggestions have been made that it should be revisited [4], the increasing scale and complexity of newer software systems [3, 5] implies that the effort invested in maintenance of successful systems can only have increased. Thus, research is required towards the discovery and evolution of supportive approaches or tools, which could improve efficiency in this effort-intensive activity.

Software maintenance can be divided into 2 general stages: “Understanding the program and actually performing the change” [6, 7]. The time invested by a developer in order to achieve an understanding before (and during) a successful modification can be considerable, with typical estimates ranging from between 50% to 90% of the entire maintenance effort [8]. Information seeking has been recognized as a core subtask in software comprehension within software maintenance [9-14]. Sim [13] for example, explicitly makes the link between software maintenance and information seeking, referring 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.

Kingrey [15] defines information seeking as the searching for, recognition, retrieval and application of meaningful content. It is a difficult task in software evolution with developers often confused as to what to search for [16] and how to search for it [17]. In certain circumstances the information may not be readily available to them and this may stop their progress, an issue referred to as ‘information blocking’[18]. While Ko [19] states that information blocking is not 'inherently unproductive', empirical evidence suggests that, in practice, it is. Liu et al [20], for example, find that information blocking can consume up to 60% of a developers time. Likewise Mockus et al's [21] study shows that information blocking could delay tasks by up to 0.9 of a day when the information is available in the same geographical location and by up to 2.4 days when the information has to come from a different geographical location.

The nature of Open Source (OS) software development makes it as an important context in which to study the difficulty of information acquisition during software maintenance and evolution. OS software is becoming increasingly important in its own right: many OS software systems are considered critical components of today's software landscape [22, 23] and the OS community produce many long-lived systems that wide populations depend upon [24-26] over time.

Yet OS software development generally involves (or has the potential to involve) large, globally distributed communities of developers collaborating primarily through the internet [27, 28] under differing governance models. As stated above, this typically widely-distributed, and asynchronous development environment would seem to make information seeking more difficult [27, 29, 30]. Thus information blocking in OS contexts is likely exacerbated when compared to co-located proprietary development [19, 31].

This work focuses on the issue of information seeking for developers evolving Open Source projects. Specifically, it studies information seeking in 6 OS developer communities, as defined by developers' communication in their mailing lists, over 17 years, in total. It should be noted that mailing lists are not the only communication channels open to OS developers. In different OS governance models alternative channels may also be employed to facilitate communication, such as instant messaging applications, email, and/or face-to-face communication. These governance models and practices are discussed in more detail in Section 2.1, and the OS projects used in this study are characterized in terms of these governance models in Section 3.3.1 Analysis of alternative communication channels like email or instant messaging is outside of the scope of this paper. In addition this work does not consider solitary information seeking where the programmer refers to the code or documentation to seek the information they need. However there is an acceptance in the literature that mailing lists are the predominant communication channel of distributed OS development teams [30], [21], [32] and so can be informative in this regard. This is best illustrated with a quotation from the OpenOffice community [33].

“Mailing lists are the backbone of open source communications".

As a result, this research considers mailing lists as strongly representative of Open Source Developers' communication and argues that studying this communication channel can provide important insights on their Information Seeking needs. It identifies the information types they seek through this medium, focusing on their prevalently expressed information needs, post deployment. Subsequently it identifies the information types that seem difficult to acquire through this medium. It provides 3 specific contributions:

- A schema of information types sought by Open Source programmers through their mailing lists as they evolve software systems;
- Identification of the prevalent information needs across these OS communities, as expressed in their mailing lists;
- A determination of the response rates for different types of these information requests;

Identifying these prevalent question types allows researchers a framework in which to discuss response rate and quality. Analyzing the response rates with respect to these question types demonstrates the question types that the community are more (or less) willing to answer, thereby generating suggestions for information and tool support for these communities.

The paper is structured as follow: Section II discusses the related information-seeking work, contextualizing how the work reported here augments the existing body of work in the area. Section III describes the derivation of the schema used in the characterization of OS developers’ mailing list questions. Section IV reports on the resultant schema, the prevalent information request-types and the response rate of the community. Section V discusses these results and Section VI, concludes the paper.

It should be noted that this research is an extension of the work reported by the first and last authors at the Psychology of Programmers Interest Group (PPIG) [29, 34]. The work reported on at PPIG describes a more preliminary version of the schema, based on a much smaller data-set (364 questions). This data-set did not cover several core categories of Open Source Software Systems, as reported in taxonomies such as Feller and Fitzgerald [27] and Daniel et al. [35]. The preliminary schema derived was then used to analyse that smaller data-set for prevalent information types and their associated response rates.

Here a more refined schema is presented, based on a larger, more representative data-set (708 questions) and more iterations of data analysis. This refined schema was used in this research to analyse the complete data-set giving a stronger empirical basis for refined insights regarding developers' information needs. A more detailed literature review is also included in this paper for comparison against these new findings.

2 Related Work

2.1 OS Governance

Information seeking in OS projects is contextualized by the governance model and working practices employed in those projects. The work presented in this paper focuses specifically on one aspect of governance, that is the use of information and tools [36]. This occurs in the context of two other aspects, these being the software development processes, and community management that implicitly guides these processes.

The two main types of governance models are where a single individual governs a project (known as a benevolent dictatorship) or alternatively a model that is governed by a group where control and membership of the group is regulated and often determined by the contributions of individuals to the project (known as a meritocracy) [37]. In the former model "seniority prevails" and individual contributors can become "responsible for one or more parts of the software" [38].

This is likely to influence the amount of communication between project contributors: those working on one module may not need to communicate with those working on another, resulting in less communication overall. In addition, projects with a small number of contributors, and associated informality, may be more likely to ignore rules for communication and thus may utilize alternative communication mechanisms such as email or instant messaging. Likewise, when governance is driven by commercial entities and the systems evolution is largely performed by their developers, it is likely that these developers will know each other informally and may even be co-located, suggesting greater use of face-to-face communication, instant messaging and/or email.

Stol et al. [39] highlight some common OS development practices that have been suggested in the literature, including universal access to development artifacts, a fishbowl development environment where every contribution can be seen and reviewed, informal communication channels including mailing lists and instant messaging, self-selection of motivated contributors and around the clock development due to the globally distributed nature of development practices. The development practices adopted by a project are likely to impact the types and means of communication between contributors.

The working practices within an open source project may also influence communication mechanisms adopted. For example, an open source process that involves large communities of globally distributed developers, typically utilizes independent peer review and rapid release schedules [27] both of which involve significant communication between contributors. In such instances mailing lists are likely to be the main communication mechanism adopted.

In considering governance practices in open source projects, Midha and Bhattacherjee [40] focused on how to manage developers in open source projects under the categories of participation management and responsibility management. They found that tasks focused on enhancing the software were completed in lesser time when the responsibility was not assigned to anyone whereas corrective maintenance tasks were completed in lesser time when responsibility for task completion was delegated.

The governance model and working practices within an open source project may change when a project is forked [38]. Gamalielsson and Lundell [41] investigated the impact of forking by examining the OpenOffice project and the LibreOffice project which forked from OpenOffice. They suggested that "effective work practices are appreciated by community members and are fundamental for long-term sustainability".

2.2 Programmers' Information Seeking

There have been several empirical studies that aim to inform on the types of information sought by programmers in the context of software comprehension. These include Pennington [42], Good [43], Wiedenback et.al [44], O'Shea [32] and Buckley et al [45]. These studies focus on the types of information that programmers' might obtain during software comprehension of source code: operation-information, control-flow, data-flow, state, and function. Typical findings from this research suggest that programmers initially obtain more control flow and operational information, deriving delocalized data-flow and functional insights later. However, these studies are predicated on an existing 'information-types' schema developed by Pennington [42]. As this schema was developed through a theoretical review of the information available to individuals in small segments of code, it is entirely plausible that it ignores other artifacts produced by a development team and that it ignores some information types specific to larger code-bases [29]. This study adopts a more grounded approach [46, 47] to identifying developer's information seeking needs where the question types are identified using data derived from larger software systems in team-based, in-vivo contexts.

Using a data-driven approach, Letovsky [48] studied programmers as they tried to understand a small software system, by noting the questions programmers asked as they proceeded. He identified 5 different question types "What", "Why", "How", "Whether" and "Discrepancy", associating the first 2 question types to bottom-up comprehension and the third to top-down comprehension. He found that programmers were "opportunistic processors", changing their efforts as informational clues became available to them.

Other grounded studies have focused more on developers working with larger systems. Ko et al [19], for example, observed co-located developers, while they were working in-vivo with proprietary, commercial software development teams. The open-coding of observational data in that study suggested that developers wanted to maintain awareness of the team's activities and the resources they had changed, sought reassurances on the correctness of their changes, and often needed to determine the cause of program states or bugs. Likewise, Sillito et al [49, 50], used a grounded approach in their analysis of professional programmers' and paired student programmers. They identified 4 categories of code-specific information seeking queries: identification of focus points in the system, relating focus points to directly associated entities and concepts, relating multiple entities and concepts that have a unifying theme and relating across these unifying themes.

Other studies have probed Open Source communities. Most notable in this regard was Johnson and Erdem's [51] study of the questions asked about Tcl/TK on Usenet. Subsequently Erdem [52] did a more detailed literature review and developed a 3-tuple categorization of the questions developers asked in the original study. The derived tuple consisted of the entity-of-interest field, the aspect-of-interest field (essentially the 5WH framework, but without 'who' type questions, though they do acknowledge that 'who' type questions could be incorporated), and a relationship field. For example, the question, "What are the arguments to that procedure?" would have an entity-of-interest: procedure, an aspect-of-interest: what, and a relationship: argument. While this study provides a detailed, considered information seeking framework, it should be noted that the questions predominantly reflected users (and not developers) of an OS initiative, That is, it concentrated on programmers who were trying to use Tcl/TK for their own systems. Consequently, this and Johnson and Erdem's original question set, largely reflect end-user's questions on the functionality of the (Tcl/TK) OS system and not the evolution of that OS system. In contrast, Silvia et al [53] focus on information seeking in OS evolution, but they specifically focus on information seeking in a debugging context. For example, they studied question prevalence in terms of a bug's lifecycle. In contrast, this study aims to provide a more holistic characterization of OS developers' information acquisition beyond, but including, debugging, as they evolve their system.

A study, agnostic to OS or proprietary software development, is Treude et al.s' [54] study of the web Q&A site: Stack Overflow, a site where any developer can pose programming questions. They identify a categorization based on 385 questions which they analysed from this website: how-to questions, discrepancy questions, questions on the programming environment, error-focused questions, decision help, conceptual questions, reviews, questions focused on non-functional attributes and noise (a bucket category for questions not related to programming). Interestingly, the structure of the website allowed them to report on the presence and quality of answers. Overall, they found that approximately 50% of the questions asked by developers were answered to a level that was acceptable to the person who originally posed the question. Again, this study is not reflective of software evolution in OS developer communities specifically: the developers who visit Stack Overflow may be developing proprietary or OS software, and may be involved in de-novo development or evolution of many distinct projects, discussing their issues with people who are not on their team. Likewise, and probably because of this last issue, the questions are predominantly constrained to the programming language topic.

Some work has moved beyond characterizing the information sought, to focus on the difficulties programmers face when trying to obtain their desired information. O'Brien [14], for example, showed that interruptions are a prevalent problem for programmers working on proprietary systems. Ko et al [19] found that such programmers working in a co-located environment struggled to obtain rationale information on the program's behaviour, bug causes and design. Other studies have used surveys [31, 55] and interviews [11, 12, 56] to study the questions developers reported as difficult: Similar to Ko et al [19], LaToza and Myers, found that proprietary commercial developers thought design-rationale, bug-cause errors and code behaviour questions were difficult to answer. However, because these studies are based on indirect empirical measurement their ability to rank the difficulty of these questions is limited: in-vivo observational measures would provide a more direct measure.

3 Deriving the Information seeking Schema

3.1 Research Objective

This research has two objectives. The first is to empirically derive a schema of information types sought by Open Source programmers through mailing lists, during post-deployment activities like maintenance and evolution. The second is to quantify the prevalent types of information sought through this medium and the response rates for those queries. This section discusses the empirical process used to derive the information type schema. For a fuller description of the derivation process and the evolution of the schema itself, interested readers are directed to [57] and to the associated interim papers [29, 34].

3.2 Research Design

Quantitative measures are usefully employed where the scientific area is mature, where controls can be imposed, and where there are specific hypotheses to evaluate [14]. That is, quantitative methods seem to emphasize ‘confirmation’ (and quantification of that confirmation) information [58]. However, the scarcity of work in the area of OS developers’ Information Seeking and the diversity of relevant viewpoints (as illustrated in Section II) suggests an immaturity in this field which in turn suggests that *confirmation* of hypotheses would be premature. In developing an information seeking schema, this work is more interested in *forming* hypotheses about the information types OS developers seek as they maintain and evolve software systems. In such situations qualitative methods are more suited to this task [59, 60].

Bogdan and Biklen [61] describe qualitative data analysis as “working with (non-numeric) data, organizing it, breaking it into manageable units, synthesizing it, searching for patterns, (and) discovering what is important and what is to be learned”. This is the approach that was adopted in this work: Specifically OS developer mailing lists were analysed, broken down into their component questions and responses, patterns of similarity were sought and consolidated into meaningful information categories. The qualitative approach employed here is largely based on Grounded Theory, where researchers “bring up” theory that resides in the data through an immersive, iterative data-analysis dance [62]. This iterative dance is continued until no new insights arise from the collection of new data, a situation called theoretical saturation [47].

Each round of data sampling is based deductively on findings inductively obtained from the previous round of data-analysis. The data-analysis activity primarily serves to build, evaluate, refine and augment the evolving theory. Strauss and Corbin [47] suggest that during analysis, researchers should follow a basic analysis workflow of: *Open Coding*, where repeated themes are identified and coded as concepts, *Axial Coding*, where relationships among categories are identified and *Selective Coding* which focuses on directed ‘umbrella’ categories, after the researcher establishes strong analytical direction from their previous coding (see Figure 3.3 in Harwood [63]).

3.3 Research Method

In Figure 1, the specific methods of data collection and analysis employed in this study are outlined. Developer mailing lists, from projects that represent coverage of the major categories of OS software [27, 35] and various stages of OS software’s evolution were collected over 4 different sampling iterations. This number of iterations was not predefined, but instead reflects the number of iterations after which refinements to the evolving schema, as suggested by new samples, seemed negligible.

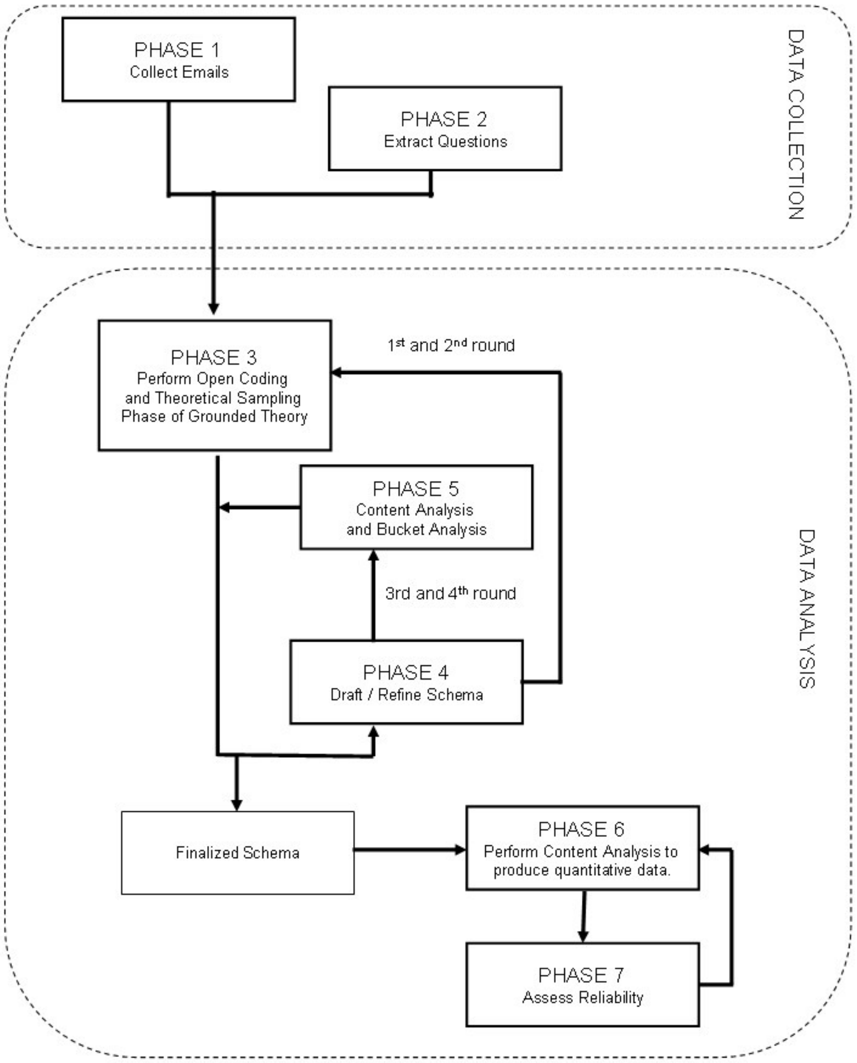

Fig. 1 The Data Collection and Analysis Model

In total, 708 questions were extracted from 2104 emails, representing 17 years of developer mailing list archives from six OS projects. See Table 1 for a breakdown of OS projects over the analysis iterations. The last two columns in this table categorize the OS projects in terms of two major classification schemas for Open Source software: Feller and Fitzgeralds's [27] functionality-based classification schema and Daniels et al.'s [35] classification schema which is based on the degree of success/adoption of the OS project. This classification was undertaken to ensure a wide and appropriate sampling of OS projects during the formation of the schema (see Section 3.3.1).

Table 1: Characterizing the OS projects used as a Dataset over the iterations of Schema Derivation

Iteration	Project Name	No. of Emails	No of Questions	Years of Evolution Phase	Duration of Data Set (Months)	Daniels et al. Categorization	Feller / Fitzgerald Categorization
1	BSF	128	44	6th	9	User Centered	Sys. Environment
	JDT	147	100	2nd	12	User Centered	Dev. (Development)
2	BSF	147	36	6th	3	User Centered	Sys. Environment
	JDT	81	42	1st	12	User Centered	Dev., Application
	JDT	100	69	3rd	12	User Centered	Dev., Application
3	BSF	391	102	2nd and 3rd	24	User Centered	Sys. Environment

	ECS	398	80	1st to 8th	96	Abandoned	Sys. Environment., Dev.
4	EBoard	182	45	1st	12	User Centered	U.I., Amusement
	SwingWT	302	108	1st	12	Controlled	Sys. Environment, Dev.
	Resciprocate	228	107	5th	12	Counter Cultural	System Environment
Totals		**2104**	**708**		**204**		

3.3.1 Sampling

The 4 data gathering iterations in this study were undertaken between 2008 and 2010, with periodic analysis up to, and during, 2012. In the first iteration, 2 OS projects were selected, for analysis with the primary selection criteria of having an active user base, and an active developer base, as evidenced by their mailing lists. These characteristics suggest that the projects would have an elongated post-deployment phase, where information seeking needs to occur, and where further elongated longitudinal studies might be performed. At the time of sampling, the Apache Jakarta [64] project was home to major open source Java projects, and the Bean Scripting Framework (BSF) was selected at random from the sub-projects within that project that adhered to these criteria (subsequently Apache Jarkarta was subsumed in late 2011 following the transfer of most of the sub-projects, including BSF, to Apache Commons).

At the time of sampling, BSF was an OS sub-project concerned with allowing Java applications to contain embedded languages, through an API to scripting engines. The vibrancy of the sub-project at that time is demonstrated by the user [65] and developer mailing list activity [66]. Likewise, JDT, an OS project with a vibrant user and developer community, both at the time of sampling and to this current day, was selected as the 2nd OS project for analysis in the first iteration: JDT is reported as being in the top 2% of OS projects regarding activity [67] and is concerned with enabling Eclipse for Java development. Developer emails were captured for BSF for the first 9 months of 2007 (the only months available for that year at the time of analysis) and for JDT for all of 2003: the 2nd year of that developer mailing list. The 2nd iteration concentrated on building up the longevity of this dataset, expanding the BSF data set to a year's duration and the JDT dataset to 3 years duration.

The third iteration again concentrated on enlarging the time sampling of the data set. The BSF mailing list emails were gathered for the 2nd and 3rd years after deployment. This provided a 3-year dataset, similar to the JDT dataset. In addition, another longitudinal dataset was gathered to see if any trends were apparent over a longer post-deployment phase, and to widen the sampling of OS projects: The Element Construction Set (ECS) was again an active OS sub-project at the time of sampling, taken at random from the Apache Jakarta project. It was aimed at allowing programmers to generate markup code in Java Objects.

In the last iteration, the emphasis was on a sampling that provided greater coverage over the major categories of OS software, as reported in the literature. Feller and Fitzgerald [27] developed a taxonomy of OS project types based on their end-user functionality whereas Daniels et al [35] constructed an orthogonal taxonomy of OS projects based on their degree of success. While ECS, JDT and BSF covered a number of Feller and Fitzgerald's categories, as illustrated in Table 1, they did not cover 'amusement of end-users' and 'support of UI development'. Hence these categories were identified for inclusion in the fourth iteration.

With respect to Daniels et al.'s taxonomy, it was decided at this stage that only projects with elevated success indicators should be considered, as these projects would be more likely to have elongated post-deployment phases, and thus the developers would have larger (over time) information needs. Consequently, 3 categories of Daniels et al.'s schema were considered appropriate for sampling coverage:

- User Centered projects are successful in terms of usage and development activities. They have high numbers of downloads, high numbers of bug report activities and a high number of releases. JDT and BSF are examples of such systems. EBoard, a user friendly chess interface for the Internet Chess Server, was also chosen as an example based on its high number of downloads. But it differed from the other projects in terms of having one owner and thus a low amount of developer activity. This system also provides coverage for the U.I. and Amusement categories of the Feller and Fitzgerald taxonomy.

- Controlled projects have high download rates over long periods of time and increasing size, but they have fluctuating rates of structural complexity as periodic efforts are made to simplify the code base. SwingWT was chosen as the exemplar for this category.

- Counter Cultural projects are relatively successful in attracting developer activity, but not in attracting user interest. Resiprocate, a system dedicated to maintaining a complete and correct implementation of SIP, was chosen as an example here because it attracts a relatively normal amount of developers, undergoes an active development phase but does not attract a high number of downloads.

The specific projects were chosen via personal correspondence with the authors of the taxonomy, as specific example systems were not always noted in their paper.

These projects ranged from true meritocracy initiatives (Resiprocate) in foundations like the Apache Software Foundation (BSF, ECS) and the Eclipse Foundation (JDT), through projects which did have committees but which seemed more governed by "Benevolent Dictators" (SwingWT), to EBoard, a project that seemed very much under the "Benevolent Dictatorship" model. The mailing list data suggests a strong tendency towards globally distributed software evolution happening in parallel but focused on different parts of the system. While most projects were similar in the degree to which they were (non) commercially driven, the JDT is an outlier in this regard, as demonstrated by its IBM origins and through IBM retaining "technical leadership of key projects" [68] within JDT.

3.3.2 Analysis

The questions were manually extracted from these mailing lists, as preliminary analysis showed that often lexicons that could explicitly signal a question ('what', 'why', '?') were omitted from the questions in the mails. For example, *"Is anyone interested in that"* represents a 'who' question, but (automated) lexical analysis is insufficient to determine this without the explicit 'who' keyword or the question mark.

More than 1 question was often found per mail in this data-set. In such cases the questions were examined to see if they were re-phrasings of the same question in slightly different forms or genuinely different questions. In all 22 occurrences they were found to be re-phrasings of the original question. The resultant questions were then placed in a Microsoft Excel spreadsheet with their corresponding location address and unique identifiers for any follow-on emails. In the first 2 iterations (see Figure 1), a Grounded-Theory based approach was used to create an initial schema that guided further data collection. That is: open coding was employed to identify categories of questions and axial coding was performed to identify an initial clustering of these categories. But, with the focus on identifying all categories of developer questions, rather than building an encompassing theory, selective coding was omitted.

The first author did this analysis by immersing himself in the transcribed data, seeking to gain as many insights as possible into the questions the programmers asked, and began to create categories based on the contents of portions of the question-set being examined, as suggested by Pandit [69] and O'Brien et al. [70].

During both iterations there were fortnightly discussion meetings with the last author, where the questions and preliminary categorizations were discussed and debated. Specific topics discussed included:

- Ambiguity of question sentences that made it difficult for the main researcher to identify categories.
- Defining new and refined categories emergent from the data-set;
- Defining the relationships among concepts. This is related to Axial Coding.

After the first month, the first and last author had monthly review meetings where the primary author randomly chose 30 questions from the sample and both he and the final author applied the evolving schema independently. The meetings then compared results to assess the schema's reliability and identify points of issue. Towards the end of each iteration, discussions were held with the other authors, again evaluating the evolving categorization schema.

With the schema partially established, after the 2nd iteration, the analysis concentrated more on refinement: Instead of adopting a Grounded approach, the preliminary schema was used as the basis for further evolution. That is, the questions identified in the data-set were either categorized into the existing schema or, if they could not be categorized in this fashion, they were placed in a bucket category. All the questions in the bucket category were then analysed to determine new categories they suggested. Retrospectively, all the other questions in the data-set were then revisited to assess their categorization with respect to the new and existing categories. Hence, the four iterations were based on a mixture of Grounded analysis and a 'Content Analysis' approach where initial Grounded analysis made way to classification into a-priori categories [71] that identified anomalies through the questions that were characterized as 'bucket' questions. Finally when no new 'bucket' category questions arose, during the 4th iteration, it was taken as an indicator of theoretical saturation and the derivation process was terminated.

4 The Empirical Study Results

4.1 The Information Types schema

The information schema derived from the mailing lists is presented in Figures 2 and 3. These Venn Diagrams present a hierarchical representation of the schema, each diagram representing 1 of the 2 top-level categories revealed by axial coding: Information Focus and Information Aspect. Focus refers to the target-entity that information is sought about, and Aspect refers to the type of information sought about that target-entity, so each question has a focus and an aspect. For example, in the case where the developer is asking about the location of some documentation on the system or an associated standard, the focus is 'Documentation' and the aspect is 'Location'. It should be noted that the word 'Focus' here does not refer to the source of the information; here the source of the information is always the mailing list/other developers. Instead, it is the target: the external representation that information is sought about.

Within the focus categorization, there are a number of sub-categories: questions targeted at the System itself, questions targeted at the Task, and questions targeted at the Contextual Technology. Questions targeted at the System refer to aspects of the code-base, directly, attributes that shape the code-base (*design*, its *operating environment*) and the associated *documentation*. Task questions focus around the task at hand, requesting *support* in undertaking the task from the community, describing its *implementation*, or querying its *stage* of progression. Contextual Technology questions refer to the working environment of the programmers: typically their *IDEs* or their means of *communicating* with the rest of the development community. Each of the individual question types within these sub-categories are described in more detail in Table 2.

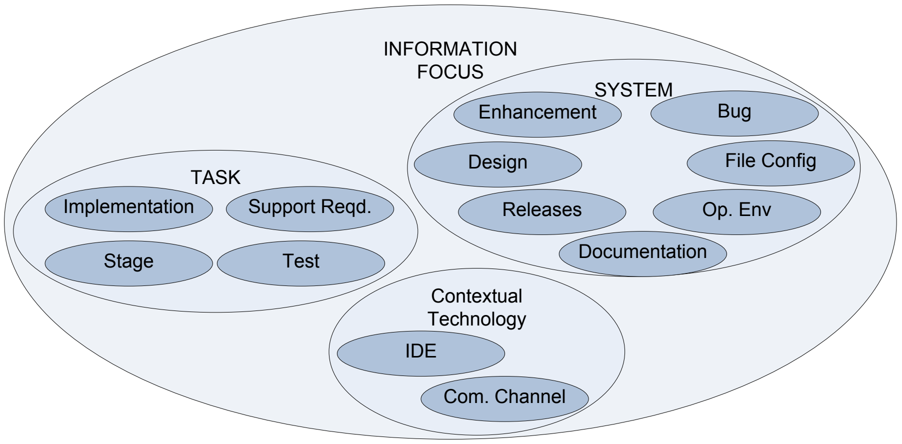

Fig. 2 Information Focus

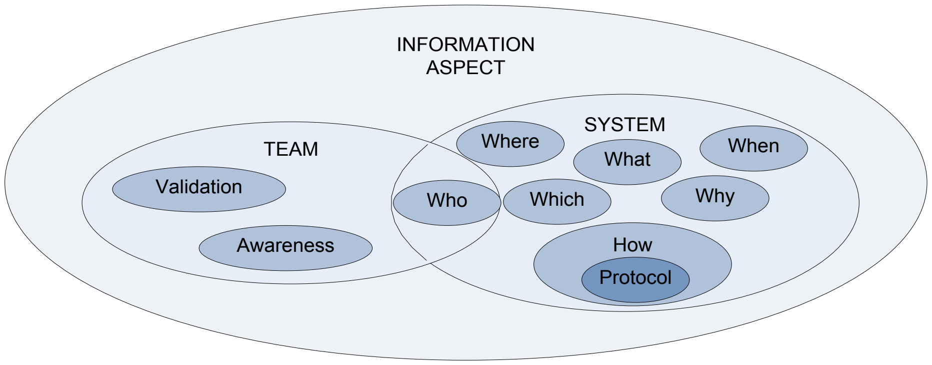

Fig. 3 Information Aspect

Within the Aspect categorization, there are 2 main sub-categories. The first can be considered a 6WH category: This is an expansion of the 5WH (*Who, What, Where, When, Why and How*) categorization, with an additional "*Which*" category. This category reflects questions where the developer outlines or implies 2 alternatives and seeks guidance on the best way forward. The second sub-category reflects the Team based nature of open source development and overlaps with the 6WH categorization through *Who*-type questions, which often probe the owner of a part of the system or associated artifact. Another Team-based question type seeks *Validation* for changes made or proposed changes that might be made in the future. Finally, *Awareness* type questions seek knowledge of others' changes or often alert others to changes that they have been made. Again, the individual question types are described in more detail in Table 3.

Table 2: Information Focus.

Info. Focus	Definition and Example
System Enhancement	Definition: Questions that aim to understand the code before making an evolutionary change. Example : *"...but I need to understand the refactoring currently in Eclipse now. Can anyone suggest me ... a good starting point in understanding how the component works* "
System Bug	Definition: Questions that aim to understand the code in order to trace a bug. Example : *"(Given an error situation..)I have no idea why this is happening. Please help me solve this problem*"
System Design	Definition: Question referring to the system's design. Example: *"Is there any plan to add pattern functionality to the JDT UI, i.e.. (given an example)"*
File Configuration	Definition: Question about configuration management. Example *: " What is the distribution directory in the src zip/tgz? "*
Distribution / Release	Definition: Question about distribution or release of source code / software product to public. Example : *"What do people think about cutting a release of BSF V3?"*
Operation Environment	Definition: Questions asking about any related surrounding technical context that is involved in the application running such as operating system, and plug-ins. These questions tend largely to request guidance ("how can I?"). Examples: *"Can I invoke update manager using bash scripts to install our plugins and features?"*.
Documentation	Definition: Questions referring to the documentation. Example: *"Is there any Apache official guidelines on this?"*
Task-Implementation	Definition: Questions about tasks related to implementation. Note that this is not about comprehending code but focused on the task. Example: "*…Maybe you need to update ecs-1.4.1?"*
Support Required	Definition: Questions that ask another community member to take on responsibility or tasks. Example: *"There are 2 non-filed open issues..... Are there any taker? "*
Stage	Definition: Questions asking about completion or stage of certain activities or a project. Example: *"I've not seen any noise on this list since I joined. Is there any life in the BSF sub-project??"*.
Task-Test	Definition: Questions related to testing. Example : "Is renameParticipants working? I am doing a spike test but cannot get it work so far."
Integrated Development Environment (IDE)	Definition: Questions asking about the IDE used in the project. These questions largely request guidance ("how can I?"), or ask what features the IDE has. Example: *"How to use Eclipse (for java) with CVS"*
Communication Channel	Definition: Question that refer to the communication channel such as mailing lists. Again, these tend to reflect requests for guidance ("How do I?"): Example : *Is there something that needs to be done to get the SVN commits i do have notifications sent to the BSF dev list?*

Table 3: Information Aspect

Info. Aspect	Definition and Example
Where	Definition: Asking about the location of software artefacts, tools, etc. Example:*"Where I can find the sources for plug in so I can create a patch?"*
What	Definition: Questions which ask what source code, system design or software tool elements do or what state they are in. Example: *"What is the value of X at this line"*, *"What is the features of GTK that can be used with Motif?"*
When	Definition: Questions asking about timeline or time of occurrence. Example : "*When is the next BSF release expected?"*
Which	Definition: Question that reflects a choice between one and another subject (information focus). Example: *"can we use JIRA for bug reporting for this project instead .. Thoughts ?"*

Why	Definition: Questions asking for the purpose / explanation of a system behaviour, bugs or rationale of design. *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?"*
How	Definition: Questions which attempt to identify how some goal of the system is achieved or how some software tool feature is employed. Example: *"How X can be 5?","How do I compile my Java source in Eclipse?"*
Legality / Protocol	Definition: Questions about the protocol to follow within the project. Example: *"Did you [get] the approval to contribute your work to BSF?"*
Who	Definition: Questions asking for the relevant persons to seek information from or to perform a task. Example: *"Can someone please point me to the information development team that wrote the used documentation?*
Validation	Definition: Questions asking permission to do something or to check if something done is OK. This strategy is normally related with Legality / Protocol. It seeks permission to do something. Example: *"I think once the docs are done we can release. Markus?"*
Awareness	Definition: Question to make sure that the asker or the audience has up to date information about current changes in the software Example : *"Do we need to tell anyone in Apache we're doing this?"*

4.2 Prevalence of Information Seeking Types

The next phase of the analysis targeted the prevalence and satisfaction of OS developers information needs (again as expressed through the questions on their mailing lists), based on the frequency of request and the communities' response rates. Consequently, the schema discussed in Section 4.1 was applied holistically to the entire data set, to identify patterns of prevalence in OS developer's mailing list information seeking. That is, the primary author used content-analysis to re-code all the questions and thus to determine the prevalence of each question-type over the data-set. At the start and end of this exercise, the final author independently categorized a sample of these questions (2*30 questions) and comparisons were carried out to evaluate reliability and evaluate the prospect of interpreter drift [72]. The Kappa [73] obtained at the start was 0.752 with a p value of 0.081, where the strength of agreement was considered to be good. The Kappa obtained from the sample at the end was 0.683 with a p value of 0.087 again suggesting a 'good' agreement between the coders. This implies that the schema was reliably applied by an independent coder and that interpreter drift over the course of this analysis was insufficient to affect that level of reliability.

Table 4: Information Focus by OS Project

Info Focus	**BSF**	**JDT**	**ECS**	**Eboard**	**SwingWT**	**Resiprocate**	**TOTAL**
System Enhancement	18	28	11	9	9	31	**106 (14.3%)**
Task-Implementation	32	19	16	5	27	5	**104 (14.1%)**
System Bug	13	25	7	6	30	18	**99 (13.4%)**
System Design	17	21	2	6	9	16	**71 (9.6%)**
IDE	10	39	8	1	10	1	**69 (9.3%)**
Support Required	28	8	8	4	8	11	**67 (9.1%)**
Documentation	18	21	12	2	2	3	**58 (7.8%)**
Communication Channel	17	8	1	3	1	10	**40 (5.6%)**
Operation Environment	4	24	1	4	3	4	**40 (5.6%)**
Distribution / Release	16	1	5	2	4	1	**29 (4.1%)**
File Configuration	7	11	4	1	2	1	**26 (3.6%)**
Task-Test	3	6	2	2	3	1	**17 (2.4%)**
Stage	4	1	3	0	0	5	**13 (1.8%)**

Table 5: Information Aspect by OS Project

Information Aspect	BSF	JDT	ECS	Eboard	SwingWT	Resiprocate	Total
How	45	90	12	13	27	40	**227 (30.7%)**
What	44	41	13	10	17	31	**156 (21.1%)**
Why	15	23	13	10	28	9	**98 (13.3%)**
Who	29	7	8	4	14	14	**76 (10.3%)**
Awareness	15	6	20	4	6	6	**57 (7.7%)**
Where	12	28	4	0	5	6	**55 (7.4%)**
Legality / Protocol	11	8	2	2	7	0	**30 (4.2%)**
Validation	8	7	7	1	2	0	**25 (3.5%)**
When	6	0	0	1	1	0	**8 (1.1%)**
Which	2	2	1	0	1	1	**7 (1.0%)**

Tables 4 and 5 present the results of the analysis for each of the projects studied. They are arranged in order of decreasing prevalence over all OS projects. Table 4 identifies the prevalence of questions for each Information-Focus over the six OS projects showing that *System Enhancement, Task Implementation* and *System Bug* type questions dominate information seeking, providing 310 of the 708 questions. Table 5 presents the prevalence of questions for each Information Aspect, again over all the projects. It shows that *How* questions, *What* questions and *Why* questions dominate, in this case providing 481 out of the 708 questions. In Table 6 the Information Focus and Information Aspect are combined in a matrix to provide a table where the prevalence of questions with a specific focus and a specific aspect can be identified. In this table seven cells are highlighted to emphasize prevalent question combinations. While prevalence is less emphasized in this table, it can be observed that questions focusing on "*How* a *System Enhancement* might be achieved", "*Who* can provide *Support* for tasks" and "*Why* a *System Bug* has occurred" are more prevalent in the dataset.

Table 6: Information Focus, Quantified by Information Aspect

Focus vs Aspect	How	What	Why	Who	Aware ness	Where	Legality Protocol	Valid ation	When	Which
System Enhancement	**71**	11	10	1	7	3	0	2	0	1
Support Required	2	1	1	**59**	3	0	0	1	0	0
System Bug	20	13	**55**	3	1	4	0	0	0	3
System Design	20	**37**	9	0	1	2	0	1	0	1
Task-Implementation	**36**	18	2	5	19	4	6	13	1	0
IDE	**34**	21	6	2	0	4	0	0	0	2
Documentation	3	8	0	1	11	**30**	3	2	0	0
Communication Channel	11	5	6	3	2	2	9	2	0	0
Operation Environment	19	17	2	0	2	0	0	0	0	0
Distribution / Release	0	4	1	1	5	0	11	2	5	0
File Configuration	8	6	1	0	2	6	1	2	0	0
Task-Test	2	6	5	1	3	0	0	0	0	0
Stage	0	8	0	0	3	0	0	0	2	0

4.3 Response Analysis

For each category of question, the community's response rate was also analyzed, using the response location markers noted during the original transcription. Table 7 represents analysis of the responses received for the most popular query types by focus and aspect posted by the OS developers on the mailing lists. The first column reports on the information sought (its focus or aspect). 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 percentage 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. Column six reports on the average number of days which passed before the query posted obtained an initial response, again for those queries that received at least one response. Finally, column seven reports on the average duration of responses where duration refers to the amount of time that passed between a query being posted and its final response. The table shows that more than half the questions received at least one response in all prevalent categories and that it was normal to receive 2-3 responses when the questions were answered at all. However, in line with the work of Mockus et al [21], it does suggest that the developers also had to wait for their responses.

Table 7: Response Rate by Information Focus and Information Aspect

Information Focus	**Total No of Question**	**% of Total Request**	**% Answered**	**Average No of Response**	**Average Delay before 1st response (Days)**	**Average Timespan of Response (Days)**
System Enhancement	106	14.3	64.2	2.4	0.99	1.5
Task - Implementation	104	14.1	63.5	2.8	1.68	2.9
System Bug	99	13.4	55.6	2.4	0.91	3.0
System Design	71	9.61	69.0	2.6	1.59	2.8
IDE	69	9.34	62.3	2.4	1.0	1.6
Support Required	67	9.1	55.2	2.8	1.11	3.6
Documentation	58	7.8	58.6	2.6	2.18	3.4
Information Aspect						
How	227	30.7	58.6	2.3	1.5	3.3
What	156	21.1	59.6	2.7	0.81	1.6
Why	98	13.3	67.3	2.2	1.26	1.9
Who	76	10.3	60.5	2.8	0.98	3.3
Awareness	57	7.7	57.9	2.9	0.76	1.9
Where	55	7.4	60.0	2.8	2.06	4.1
Average			61.3	2.5	1.29	2.5

5 Discussion

5.1 The Schema

The schema derived from the mailing lists can be aligned partially with existing schemas, most notably with Erdem's [52]. Erdem proposed that all information seeking events could be classified into Topic, Question type and Relation type. The Topic was the entity referenced in the question, and the Question Type consisted of *Why, What, Where, When, How* and *Verification* type questions. Erdem identified 9 different Relation types: Topic, Behaviour, Structure, Function, Use, Goal, Preconditions, Post-conditions, and Context. Of the 3 classifications, there is most overlap between Erdem's Question Type dimension and the Aspect schema reported on here, as shown in Figure 4. Both contain *Where, What, When, Why* and *How* questions and Erdem's Verification questions closely align with the *Validation* questions in this schema. In addition Erdem acknowledges that *Who*-type questions might be legitimately included, again in agreement with our Aspect dimension. However, the empirical data reported on here suggested 3 additional categories: *Protocol* questions probing the allowed or desired community standards that developers should follow, *Awareness* questions that seek to raise awareness across the team's activities and *Which* questions that propose 2 (or more) alternatives and request community members to select the more optimal choice. Given that Erdem studied users of Tcl/TK that were not otherwise in a common development community, it is unsurprising that he did not find evidence of the first 2 question types. However, he did note Letovsky [74] as an influence and thus it is somewhat surprising that he did not include Letovsky's Whether-type questions: questions that are analogous to the *Which* questions reported on here. In contrast LaToza and Myers schema did propose a policy information need, analogous to the *Protocol* information need identified here (again shown in Figure 4) in their 2010 paper [55] but like Erdem et al. [52], did not note *Awareness* or *Which* information needs.

When we assess Erdem.'s two other dimensions their schema and the one presented here diverge more. Specifically Erdem's study looked at users of Tcl/TK and, as such, the entities of interest (Topics) in the original study differ substantially [51]. They discuss topics like environmental set-up variables and troubles when installing Tcl/TK. In the later reporting of this work [52] efforts are made to extend this categorization to entities of more general interest to software developers, based on a literature review. But the entities of interest are never explicitly defined in the article. Instead a representative set of examples are given in the text and in Table 1 of that paper. The work reported on here empirically formalizes this Topic dimension for OS developers involved in software maintenance.

The Relation Type dimension in Erdem's classification schema is explicitly defined and the subcategories (as declared in the first paragraph of this section) seem relevant to software developers. Indeed, they seem to offer increased granularity over the schema proposed here. This is because these relation types largely focus on specific attributes of source code like preconditions, behaviour, and structure and, as such, this dimension works at the level beneath the schema proposed here. This can be seen in Figure 5. This figure shows that several of the types identified by Erdem et al. map to the *System Enhancement* and *System Bug* categories, although some, like Context and Structure, do provide a more one-to-one mapping between the schemas. The same 'increased-granularity' comment can be made with respect to Pennington's schema [42] and LaToza and Myers [55] work, again as illustrated in Figure 5. These latter authors propose a schema that has code specific categories like Control Flow, Data flow and more generalized (code) Dependencies that map many-to-1 to the *System Enhancement* and *System Bug* categories of the schema presented here.

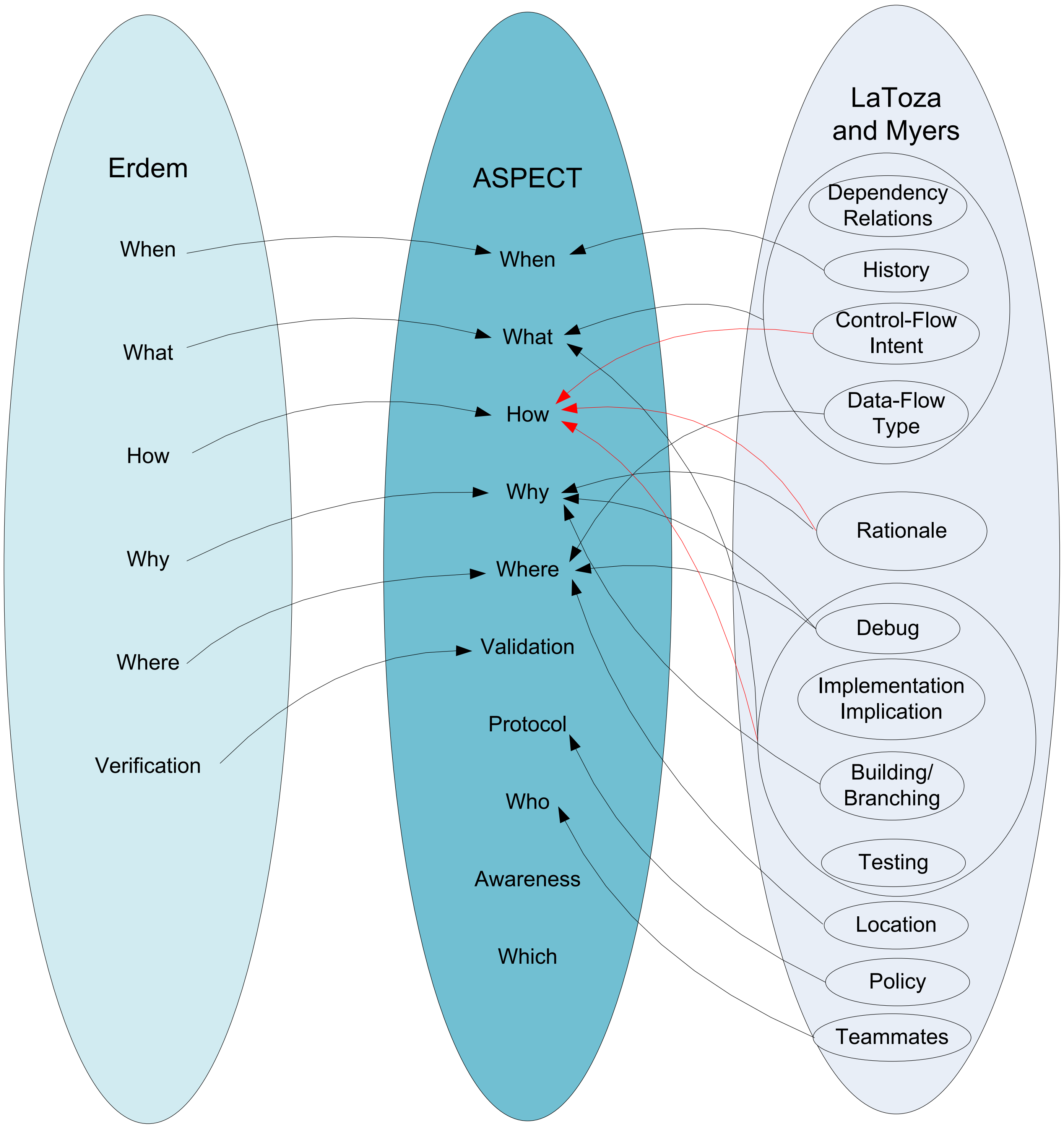

Fig. 4 Relating the Aspect Dimension to Erdem's and LaToza and Myers' Schemas

It can be concluded that the schema presented here is at a broader level than the schemas identified in the literature but it would be interesting to merge the quite diverse, code-level schemas of Pennington, Erdem and LaToza and Myers to assess if they could provide a third, comprehensive, detailed dimension for code questions in the schema presented here. Additionally, this effort could be adapted to *Task* focused questions, *Contextual Technology* focused questions and other subtypes of System focused questions (*Design, Release, Documentation, File Configuration*) to provide more detail across the entire schema.

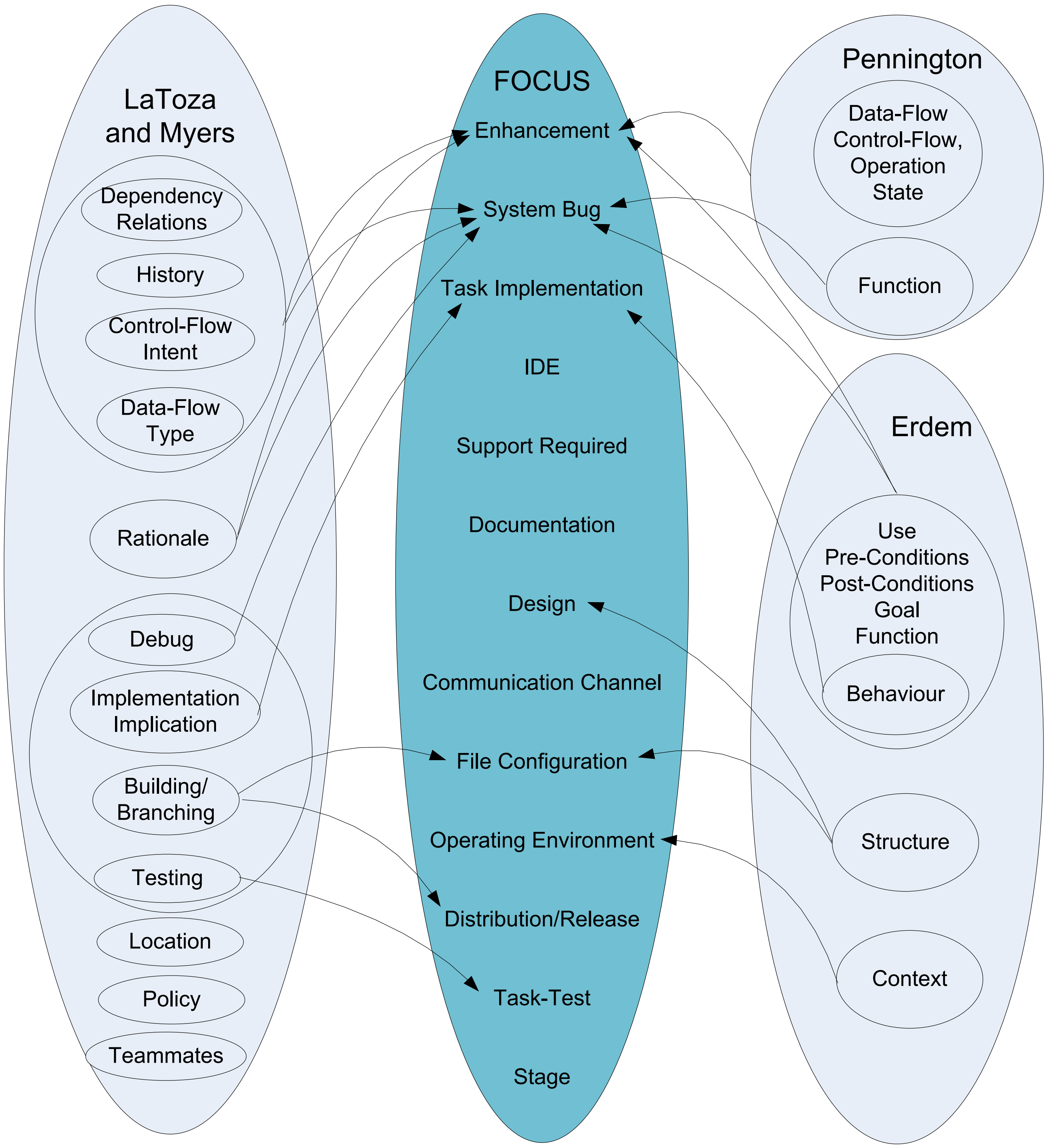

Fig. 5 Relating the Focus Dimension to Erdem, LaToza and Myers and Pennington's Schemas

Another difference between the work reported here and LaToza's schema is their inclusion of questions relating to architectural concerns (Architecture, Contracts, Concurrency and Performance) and questions concerning the implications of their changes. Consequently the questions in our schema were revisited with these categories in mind and re-analysed. However, no questions were found regarding the architecture of the application explicitly (baring those that might be considered design-rationale questions) and only 3 questions were found concerning the Implications of Change. This is an interesting finding, suggesting that OS developers on these projects, at this stage of the projects' evolution, were less concerned with architectural concerns like performance and contracts and less worried about the implications of their change. One possible reason is that the data-set is largely taken from the initial stages of evolution, after the major architectural decisions have been made and before they need to be revisited. Likewise, it is possible that, during these initial stages of evolution, developers are less aware of the potential ramifications of their changes, as the code is still relatively integral and does not need to be refactored or reengineered. However, this is only one possible rationale: it could alternatively reflect a different mindset between commercial and OS developers, the difference between a data-set gathered by surveying programmers about hard-to-answer questions and an observation-based data-set of mailing lists or, it may just reflect questions that programmers ask themselves rather than ones they reflect to other developers. Regardless, these questions were reported as hard-to-answer, and so, are legitimate for inclusion in a schema of information seeking that attempts to provide a basis for future evolution-supporting software-tool developers. Further research should be directed at ascertaining the rationale for such differences.

It is interesting to note that*,* *Documentation* and *Contextual Technology* sub-categories noted in our schema were not reflected in the findings of Erdem or LaToza and Myers. Again, this may be an artifact of their protocols where they were basing their findings on a literature review/surveying hard-to-answer questions. Alternatively there may be some semantic mapping of these differently named categories. For example, our *Documentation* location questions might really be targeted at other attributes like LaToza and Myers' Rationale and Policies. Otherwise the schemas are quite consistent, although it should be noted that the dimensional organization present in the schema proposed here is missing from their work, making it harder to disentangle the entity of interest and the aspect of interest.

Similarly in Truede et al.'s discussion of their preliminary work in this area on Stack Overflow [54], this organization is missing, but there are high overlaps with their observations and ours. Specifically, they identified *How* questions, *Explain* (*Bug*) *Behaviour* questions, *Environment* questions (that map to our *IDE* questions, *Communication Channel* questions and *Operating Environment* questions), *Conceptual* (*Why*) questions, Error *(Bug)* questions and requests for review (or *Validation* questions). Their *Decision Help* questions also seem highly similar to our '*Which*' questions. In contrast though, and similar to LaToza and Meyer, they had a category of questions targeted at non-functional requirements like performance or memory issues, further strengthening the case for inclusion of this category in the schema.

Silvia et al. [53] provided a classification of questions specifically related to debugging, but still there is plenty of overlap with the Schema presented here. Their Correcting the Bug questions and Debugging questions map onto our *How* to undertake *Implementation* questions, and their Clarification (or understanding) questions map directly onto our *Enhancement* category, especially those *Enhancement* questions that ask *How* something is achieved (see Table 6). Their *Status* and *Resolution* questions map onto our *Stage* and *Awareness* type questions, and their process questions map directly onto our *Protocol* questions. They did, however, find 2 additional categories of questions that were not present in our data set: questions concerning the Triaging of bugs and questions concerning Missing Information that is required to recreate a bug.

Ko et al. [19] in their classification of co-located programmers' information needs also suggested information needs around Triaging and Reproducing a Failure, breaking the former down into assessing the legitimacy of the problem, assessing its complexity and assessing the value of a fix. In terms of Reproducing a Failure they found that developers asked the conditions under which a bug occurred, mapping quite closely to questions in our schema that probed the *How* and *Why* of *System Bug*s. In terms of writing code Ko found questions that focused in on the data structures and functions that should be used. While our schema did not discuss this level of detail, an analogous category from our schema was Task Implementation questions, particularly those that probed *How* to make the change. Retrospective analysis of the questions in our dataset found very few questions that probed specific data structures or functions. Ko's *Submitting a Change* question-type closely aligns with our *Validation, Protocol* and *Test*-based questions and his Understanding Behaviour category aligns with our *System Enhancement* questions. Finally, both schemas have an *Awareness* category and a *Design* category. With respect to the former, the 2 categories look very similar but, from Table 6, we can see that the questions observed in this study focused more on *Awareness* of change, while in Ko et al.s' work they also focused on 'other information relevant to the task'. The *Design* category is also subtly different. In Ko's work the design questions probed the rationale for specific pieces of the implementation. However in our case the observed questions reflected a desire to map from known design rationale to implementation (see Section 5.2, third paragraph).

In conclusion the schema presented here is largely consistent with other schemas, providing support for the position that the schema is reliable. Where it does not reflect the categories in other schemas (for example with respect to information needs related to Architecture, Concurrency, and non-functional Attributes in general), the data-set has been retrospectively analysed for expression of these information needs. This analysis suggests that these information needs were not expressed in this OS-developer, mailing-list data-set. Finally the schema suggests additional information needs; information needs not noted in other schemas: *Documentation* and *Contextual Technology*. The prevalence of these and the other information needs will now be discussed.

5.2 Prevalence-Based Findings

One of the most obvious findings from the tables is the implementation-centric nature of the questions posed. Over 42% of the entire question set is targeted at low-level implementation: understanding the system code (*System Enhancement*), understanding the change task (Task Implementation) and understanding bugs (see Table 4). This is in line with much of the other research in the field [11, 75] , which states that programmers' information seeking is largely directed at systems' implementations.

From Table 6, over 55% of the system-bug focused questions are *Why* questions, reflecting the developers wish to understand the cause of the bug. Examples include: *"Dialog pops up and vanishes before I can read it - any ideas?"* or *"(Given an error situation)...I have no idea why this is happening. Please help me solve this problem".* This is in line with the findings of Ko et al. [19] and LaToza et al. [17] who cited the cause of system failures/bugs as a major information need for co-located commercial developers working on proprietary software.

What is more interesting here though is that, contrary to Ko's findings, the developers asked very few *Why* questions, regarding the design. *Why* type questions have been noted as a bottom-up type comprehension activity [76]) where, for a specific implementation, programmers ask why it is done that way [74]. Possible reasons for OS developers not asking design questions could be because they already know the design rationale, that they don't notice design features while working at the implementation detail (and hence don't wonder why they are incorporated) or for the related reason that design information is irrelevant to their task.

From Table 4, we note that there are a large number of *System Design* questions, so design does seem to be of interest to OS developers. This is not surprising given Feller and Fitzgerald's [27] observation that in OS design is typically performed in advance by a single developer or small core group and thus, the larger development group carrying out evolution may not have all the design insights. From Table 6 it becomes apparent than many of these *Design* questions posed are *What* questions. Closer qualitative inspection of these questions suggests that the majority probe the basis for the design ("*What criteria are used for deciding where a particular BSF-engine should be a part of bsf or not?*"). It seems that the OS developers are not as inclined to move bottom-up from the implementation detail to derive design insights through questions. Instead they seem to question more from a position of design knowledge, where they know the generic design (possibly from other similar systems) and want to refine their understanding of its specifics in the case of the system/implementation they are dealing with. This hypothesis is reinforced by qualitative analysis of the *How, System Design* questions: questions that come next in frequency to the *What System Design* questions. Here programmers asked how a design criterion they expected, is implemented in the system. *How* questions have been explicitly noted by Letovsky [74]as a top-down comprehension activity, moving from abstracted goal (in this case a design-abstracted goal) to the implementation.

Table 6 also illustrates how most of the *System Enhancement* questions and a large proportion of the *Task-Implementation* questions are *How* questions. Qualitative analysis of these questions re-affirms that here the developers are asking how to do the *Task* at hand (*"How can I further optimize the below chunk of code...")* or *How* the system achieves a goal or state (*"How X can be 5?").* Again, these *How* questions reflect a top-down process where the programmers move from their knowledge of the goal to the implementation detail. While Ko et al. [19] noted that co-located commercial developers of proprietary software did pose questions related to the task ('How difficult will this problem be to fix?') this is the first reporting of developers asking how to do a task.

More generally, Table 5 shows that there is a predominance of *How* questions across the dataset. While a large proportion of these questions are focused at *System Enhancement* and *Task Implementation* as noted above, another associated focus is the *IDE*s employed. Qualitative analysis of these *How* questions show that developers want to know how to achieve certain goals with the *IDE* ("*How can I just compile a subset of source folder within the project-one working set"*).

In total, over 14% of all questions were focused at *Contextual Technology*: the *IDE* and the *Communication Channels* used by the developer community. This novel finding suggests that OS developers struggle with the technology around the project as well as the system itself. This finding could be related to new programmers coming on board or just the increased complexity and variety of *IDE*s and *Communication Channels* generally. However, this emphasis could also be biased by the inclusion of the JDT project, a project which is a plug-in for Eclipse. It is logical that a lot of *IDE* questions would be asked in that developer mailing list. *IDE* questions on the JDT project account for over 56% of all *IDE* questions, but over 50% of these JDT-IDE questions do probe *How* the *IDE* allows a (developer) user to achieve a functionality, suggesting this is still a fairly prevalent information need.

Like Ko, we found a large presence of team-oriented questions in the dataset. This is apparent in Table 5 where the *Who* questions (*"Can someone please point me to the information development team that wrote the used documentation?"*), Team *Awareness* questions (*"Robert are you still working on the texen stuff for ecs2 to generate the html and rtf classes?"*) and the *Validation* questions (*"I think once the docs are done we can release. Markus?"*) all suggest a team focus that was lacking in many early works in this area [42, 74, 77, 78] Table 6, shows that the vast majority of the *Who* questions are looking for community support in the form of undertaking a fix or enhancement (*"There are 2 non-filed open issues….. Are there any taker?"*). Qualitative analysis of the *Validation* questions suggest that developers are looking for validation of some task just undertaken by the developer and likewise, many of the *Awareness* questions are focused on the task done, to be done or finished ("*I was wondering if anyone was working on this code….*").

It is interesting to note that a relatively large number of the questions were directed at the *Documentation* and specifically where it is located (*"alternatively if you can point me to instructions…")*. Indeed, documentation searches accounted for over 51% of all *Documentation* questions and 55% of all *Where* questions. This is interesting because other reports from studies of co-located developers, suggest [11, 75] that documentation is not as trusted as source code or other programmers. Here, however, documentation was often sought and seemed important ("*Could anyone please tell me if Eclipse Platform is J2EE compliant, where could I get some more documentation on it. This piece of information is really critical for me"*). There are a number of possible reasons for this finding: It is possible that, due to the delocalized context of developers in this study, OS programmers may be motivated to produce better documentation, and therefore it is possible that the community trust documentation more than in the traditional case. Alternatively, the delocalized nature of OS developers, resulting in a perceived lack-of-availability of other developers, might be the reason for a higher-than-expected number of *Documentation* requests. Qualitative analysis of these *Documentation*-location questions finds that the vast majority of them refer to standard documentation outside of the OS project itself and this, allied with the increased request frequency, suggests that developers may trust this standard documentation more. But further research would need to be carried out to confirm this hypothesis.

Finally, a small analysis was performed on the 2 projects that exhibited different governance models, to determine if they differed from the others in any way. EBoard, governed by a Benevolent Dictator, seemed consistent with the other projects in terms of the relative prevalence of different question foci and aspects. In line with Viseur's (Viseur 2012) assertion that different developers may take responsibility for different parts of the system in this governance model, and that overall communication may thus decrease, the EBoard mailing list was much less active than the other projects. However, EBoard is not being developed anymore and the low activity rate observed on the mailing list is probably more to do with this and its small pool of developers, rather than its governance type.

JDT, the only commercially driven model did show unusual prevalence of certain question types. For example the JDT mailing list had elevated *IDE* and *Operating Environment* questions. Many of the IDE questions were questions that probed how the IDE allowed users/programmers to do a certain task, in line with the intent of the categories in the schema. Admittedly though, this could have been for the purposes of locating or testing certain functionalities of the IDE.

How and *Where* questions were also elevated in the JDT mailing list, but no obvious reason for this increased prevalence could be determined. It is possible though that the large scale of the project prompted more *Where* type questions.

In summary, the question analysis suggests that, similar to the information-seeking findings published on co-located developers, OS developers, as observed through their mailing list queries:

- Are implementation-centric;
- Concentrate on how the system achieves its functionality/goals/states, a top-down approach.
- Find it difficult to find the cause of bugs;
- Seek awareness of colleagues' activities and their team context in general;

However, several novel insights were obtained regarding OS developers through this analysis:

- They ask their colleagues how they would do a task;
- They rely more on standard documentation outside of the OS project itself but seem to find it difficult to locate that documentation;
- They ask their colleagues how to use the technology (*IDEs, Communication Channels*) employed in the project;
- They don't seem to work bottom-up towards design insights, tending instead to work from generic design knowledge to specific-instance design knowledge;

5.3 Response-Based Findings

The literature suggests two opposing perspectives that can be related to OS programmers' behavior in responding to questions. One perspective is that OS programmers are highly proactive and motivated contributors [79]. However, software maintenance is often portrayed as an undesirable task, and this suggests that OS programmers might tend to shy away from it [5].

In the context of this research, response analysis was carried out to evaluate the two opposing perspectives mentioned above and to see if the pro-activity associated with OS development overcomes the reticence of programmers with respect to maintenance, at least in their support of other programmers in the community. Specifically, this work investigated the likelihood that each OS programmer would receive responses to their mailing list queries and assessed the amount of responses they are likely to get. For this purpose, response analysis was performed on all the questions found in the dataset.

Table 7 shows the results from this analysis. Overall, the response rate was approximately 61%, over all the mailing lists analysed. But this average obscures disparities between different information foci (*System Design* 69%, *Support Required* 55.2%, *System Bug* 55.2%) and different information aspects (*Why* 67%, *Awareness* 57.9%). Given that most of the *Support Required* questions were *Who* questions (searching for someone to take on a task), the low response rate with respect to this category is perhaps unsurprising and is in line with other findings [53]. Qualitative analysis of this dataset showed that just more than half of these requests sought developers to perform 'fixes' on the code-base and, based on developers reported reluctance to embrace maintenance changes, a low response rate could be expected. This finding mirrors Midha and Bhattacherjee's [40] assertion that corrective maintenance tasks are completed in lesser time when responsibility for task completion was delegated by OS management. Interestingly, for the *Support Required* questions that were answered, they generated a higher-than-average number of communications (average 2.8), over a longer-than-average time-span (3.4 days), possibly implying that they were subject to more intense clarification, discussion or negotiation.

Slightly more difficult to understand is their reluctance to respond to *System Bug* questions. This finding is in contrast to another study of OS developers which found a higher (66%) response rate for this type of question [53]. Our study found that these questions are among the most frequently sought information focus and, hence, a low response rate could be problematic for the community. Most of the *System Bug* questions are *Why* questions (see Table 6) and Ko et al [76] note that *Why* questions are considered difficult to answer. This suggests that the developer communities might shy away from such questions. However, our findings with respect to *Why* questions in Table 7 contradict this assertion: *Why* questions were the most responded-to questions of all information aspects. A more detailed analysis reveals that *Why* questions directed at other information foci (not *System Bug*) were very frequently responded to, but *Why* questions directed at *System Bugs* were not.

Possible reasons for the low response rate for *Why* questions targeted at *System Bug*s' might be their tight association with the negatively perceived activity of maintenance, or because questions as to the causes of *System Bugs* are directed at very specific parts of the code base and directed at unanticipated behaviors of those parts of the code base. This latter rationale suggests that other programmers are unlikely to know the reasons behind a specific bug's behavior. If, in contrast, we look at the other information Foci where *Why* questions are asked (Table 6) we notice that these differ in terms of the number of developers who might be interested or knowledgeable. For example, there will probably be general expertise with respect to the *IDE* and *Communication Channel*. Likewise there may be a pool of expertise regarding the *System Design*. Only *System Enhancement* would seem to be as specific as questions directed at *System Bugs*. However, even these questions are not as specific, in that it is more likely that a developer in the community will know why the code-base is as it is (the original developer or someone who has worked on it in the past) than knowing the cause of a specific bug. This information would of course be unanticipated by the original developer.

Where questions took the longest number of days to respond to on average. From Table 6, most *Where* questions were directed at *Documentation*. Qualitative analysis of the response data for these questions suggests that the time-span can be explained by the nature of the communication. Specifically, while the developer who posed the question implicitly assumed (for example) that documentation existed, often this was not the case. In such instances, a response came back to state that it didn't exist, but that the respondent was willing to talk about the issue on which documentation was sought. Given that the majority of *Documentation* sought was external to the specific OS project, it is possible that respondents waited to see if the community did know if the documentation existed before offering this help.

Finally, *Awareness* questions have a low response rate. Qualitative analysis of the associated questions suggests a possible reason for this: *Awareness* questions are sometimes rhetorical, acting more as an announcement than a direct request for information. Nevertheless this is not true for the majority of such questions and the low response rate for genuine *Awareness* questions (in conjunction with developers announcing their actions through the mailing lists) suggests that OS communities should have improved facilities for informing other developers of on-going work. This is a conclusion reached independently by Ko et al. [19] in their study of co-located proprietary software developers.

In summary, the response rate for their mailing list queries suggests that OS developers are:

- Reluctant to respond to requests that ask community members to take on maintenance or system evolution *Tasks*;
- Respond frequently to Why questions in general, apart from questions that probe the cause of System Bugs. Responses to these questions are less frequent;
- Reluctant to answer questions that seek Awareness of the activities/changes of others on the team;
- Frequently responded to System Design questions.

5.4 Empirical Validity

The study reported is of high ecological validity, as it reports on the actual communications of OS developers in a completely naturalistic environment. However, it could be argued that it suffers from several potential limitations

- Reliability of the derived schema: The schema derivation was generated largely from the observations of the primary and final authors. However, the derivation protocol was performed in line with Grounded Theory and Content Analysis, two largely accepted protocols for working with qualitative data. In addition, moderation techniques were employed where categories were discussed extensively between the two primary authors in the early stages, discussed with the other authors periodically and the resultant categories were checked for inter-author reliability later over the course of the subsequent, holistic analysis. Finally, the resultant schema had significant congruence with elements of information seeking categories from the previous literature in this area. Specifically, in the Information Focus dimension, it has significant similarities with the research of Sousa et al [75], Singer [11], Seaman [10] and Ko et al [19]. In both Information Focus and Information Aspect dimensions there is a significant overlap with Erdem's [52] work in this area. These overlaps, along with retrospective analysis of areas where it was not congruent with these existing schemas, serve to raise confidence in the findings that do not align with previous work in this area.

- Construct Validity: The mailing lists were chosen as the observation medium in this study. But, there are also other communication channels that could be used by OS programmers. This is particularly true of those involved in OS projects with different governance models. For example in commercially driven OS projects like JDT, it is entirely plausible that the majority of development and evolution originates from one commercial partner. In such cases, programmers may be familiar with one another and may use email or instant messaging applications to communicate. Indeed, this may even happen over time in projects with other governance models. Additionally the commercially driven projects may even have co-located developers who discuss issues face-to-face. Likewise, discussion forums, and generic programming query sites like Stack Overflow are alternative and potentially rich sources of communication that have been omitted from our sampling. However, Gutwin [30] , and Mockus and Herbsleb [21] state that mailing lists are one of the primary communication channels for OS developers. Likewise O'Shea [32] notes that a "substantial portion of the information" passed between OS developers is through the medium of mailing lists and so we feel confident that this data-set is a reasonable proxy for delocalized OS developers' Information Seeking when it involves communication with other developers.

  It should also be noted though that OS developers may not communicate with others when seeking information: instead they can study the source code, the documentation or even previous mailing list queries (and responses) in a solitary fashion. Indeed, it is likely that a significant proportion of their Information Seeking is of this nature. This raises questions as to the subset of Information Seeking observable through mailing lists. One possibility is that mailing list communication reflects hard-to-answer queries that force the developers to seek help. This position is supported by the meritocracy ideals behind many OS projects' governance models. Developers who wish to illustrate their prowess, and thus rise in the meritocracy, might be reluctant to expose their inability to address their information needs by themselves. But an equally valid hypothesis is that some developers just prefer to be part of, and communicate with, their OS community [80]. Another, in line with information foraging principles [81], assumes that that mailing list queries provide an easier information foraging trail and that information hunters will take this path of least resistance. The truth is probably a combination of several motivations and rationales for each individual programmer but regardless, does suggest that the sampling of mailing lists exclusively has potential bias. To address these issues, future studies should aim at capturing a more holistic data-set of OS developers' communication and complimentary empirical studies should be undertaken that assess their non-communication-based Information Seeking.

- Sample: While efforts have been made to achieve saturation, it is still possible that the dataset is insufficient for this purpose. The dataset used in this study was taken from six mailing lists that, combined, culminated in 17 years archive communication. However, Table 6 illustrates that, when taken at the granularity of Information Focus and Information Aspect combined, the number of questions in each category is small. Hence, there were several issues that were hypothesized in the research that require buttressing through the qualitative analysis of larger data-sets. In mitigation, to the authors' knowledge, this is one of the largest manual analyses of its type, which allows the richness of interpretation that comes with qualitative analysis.

  But there are also other issues not addressed within our dataset. For example, while different governance models exist within the dataset, only one of the OS projects is representative of commercially-driven OS development. Similarly, there is only one OS project that is exclusively "benevolent dictator" in governance. Likewise, we saw no strong evidence of forking in any of the projects we studied. Forking is an important consideration because it typically reflects tensions amongst contributors that signals changes in governance/work practices which may impact on developers usage of OS infrastructure (like the mailing lists). Future work should address these scoping issues but studying a greater range of OS projects under different governance models and specifically studying projects which have been forked.

- Success status of projects: Another related issue is that this study focuses on successful projects, as categorized by Daniel et al 's [35] OS project categorization. Mailing lists from successful projects were chosen as they suggest longer maintenance phases and thus their developers have more (longer duration and more immersive) information needs. Unsuccessful projects were thought to be less relevant as their maintenance phase is shorter and less active, suggesting lesser information needs. An orthogonal perspective should also be considered though: perhaps unsuccessful projects are stifled by the lack of relevant information and so developers became disillusioned. Analysis of these projects may have provided the opportunity to identify and rectify these developer issues, thus changing the projects from unsuccessful to successful.

- Programming language used: All the OS projects involved in this study used Java. Studies that focus on different programming languages might illustrate different cognitive processes that developers use while working with these languages, hence affecting their information seeking behavior [82].

- Changes over time: It is plausible that the types of information sought by OS developers through their mailing lists might change over time, as projects get more mature or as technologies change. However a chronological analysis of the dataset suggests no such trends. There were no apparent trends in the total number of questions posed over years or in the types of questions that were posted on the mailing lists. Periodically, there were spikes and troughs in specific question types associated with specific OS projects, but these probably reflected changes in the individual projects and were not systematic. For example in the BSF project in 2007, there was a spike in the number of *Communication Channel* questions posted but this can be explained by an increase in the number of communication channels used in BSF that year over previous years (a wiki, the developer mailing list, Bugzilla and JIRA)

6 Conclusion

This paper reports on the derivation of an Information Seeking schema for OS developers through a grounded analysis of 6 OS developer mailing lists, spanning 17 years of mail activity. The resultant schema is largely congruent with the findings of [11, 19, 74, 75] and closely echoes the schema proposed by Erdem [52]. However, several of the categories differ, particularly with respect to *Contextual Technology*, and *Documentation*.

The resultant schema was then applied to the dataset to quantify the different types of questions posed by OS developers, and to quantify their associated response rate. Question analysis suggests that, like their co-located commercial counterparts, OS developers are largely implementation-centric. They concentrate on how the system achieves particular functionality and states, how to make specified changes and why specific bugs arise. Additionally, like their co-located commercial counterparts, *Awareness* of team-member activities seems important for them (both declaring their own activities to the team and ascertaining other team-members activities). Novel findings include their focus on how to use the *IDE* and *Communication Channels* employed in the project, their top-down seeking of design information with respect to the specific system they are working on, and their seeking of more documentation, although often this documentation doesn't exist and responders provide the required information instead.

Regarding the response data, it does seem that the OS developers reported on here are reluctant to take on software maintenance tasks. While this reluctance has been suggested for software developers in general [5], evidence for the assertion with respect to OS developers is novel. Additionally they seem reluctant to answer questions on the causes of system bugs. However, this latter reluctance may be attributed to the difficulty of these questions: not only are these questions typically directed at very specific pieces of the code base, they are also directed at the unanticipated behavior of those specific pieces of the code-base. Finally, the preliminary rationale for the long response time for *Where Documentation* questions should also be evaluated on a larger dataset.

Interestingly, OS developers frequently responded to *Why* questions in general, and questions that probed the *Design* specifics of the systems they were working on. This may reflect a high level of (rationale and design) knowledge embedded in OS communities, but a larger and richer dataset would be needed to evaluate this hypothesis conclusively. Finally, the response data showed the need for a medium that allowed OS developers remain aware of the team's activities, an observation in line with Ko et al.'s [19] findings with respect to co-located commercial developers of proprietary software.

The findings provide insights for those interested in supporting the information needs of OS developer communities: They suggest that the tools and techniques developed in support of co-located developers should be largely mirrored for these communities: they should be implementation centric, and directed at illustrating *How* the system achieves its functional goals and states. Likewise they should be directed at determining the reason for system Bugs: a type of question frequently posed by OS developers but less frequently responded to.

Mapping functionality to implementation is a longstanding research-community endeavor [83-85] with varying levels of success, but the relevance of this agenda across schemas and comprehension theories suggests that the research effort should be continued. Efforts in the determination of Bug rationale are reflected in de-bugging environments, often included in IDEs such as MS Visual Studio and Eclipse, but sometimes as stand-alone prototype tools and approaches [86, 87]. More basic research into software *Bugs* has directed itself at classifying software errors [87, 88]and predicting where buggy code is [89-92]. This empirical study, allied with other empirical studies in the area, suggest that effective bug-rationale determination should continue to be an important goal for researchers and tool providers. Unfortunately, it is a difficult point to address as, as discussed above, bug causes are very specific. In addition, the causes can be as diverse as design, memory access and code logic so automated detection, at a meaningful level for industrial scale developers, is unlikely in the near future.

While the above information needs have been proposed before for co-located commercial software developers, this work suggests that OS developers also need support in terms of using the *IDE* and communication technology employed by the community. Additionally their need for design information seems to be top-down, where they know the design principle in advance and want to map these to the implementation details of design decisions in the specific system. Ultimately this means that approaches where design rationales are inserted in comments are less suited to OS development: These comments assume that the developer is looking at the code and trying to move up, from the code, to the design. This study suggests that OS developers seem to move from the design to the code and so traceability links from the design documentation to the code would seem more appropriate. Likewise it seems important to provide a, possibly generic, forum where information can be obtained regarding IDEs and communication channels used in OS projects. Venues like Stack Overflow could address this need.

Future work should concentrate on replicating and extending the study using (and possibly refining) additional communication channels (mailing-lists, IM, and email, for example), individual OS programmers working alone and the existing schema. Ideally it should contain examples of OS projects written in other programming languages, of different OS governance types, should include examples of unsuccessful OS projects and should incorporate other communication channels that OS developers might use: a data-set incorporating several communication channels could even probe potential differences in the different media used.

Additionally, it would be interesting to find some mechanism that allowed researchers to determine the efficacy of the responses obtained to questions. Very seldom in our data-set (<1% of queries) was there an explicit acknowledgement that the response had correctly addressed a developer's question. Most of the time the developer who posed the question did not respond at all, and so there was no way to ascertain if they were satisfied, if the response was incorrect, or if they even checked for the response (i.e. if they had found the information elsewhere). A measure of efficacy, in the spirit of [54] would allow more accurate determination of these developers' information needs than our "response-presence" measure.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

Reference

1. Lientz, B.P., E.B. Swanson, and G.E. Tompkins, *Characteristics of application software maintenance.* Commun. ACM, 1978. **21**(6): p. 466-471.
2. Mayrhauser, A.V. and A.M. Vans. *From code understanding needs to reverse engineering tool capabilities*. in *Sixth International Conference on Computer-Aided Software Engineering (CASE'93)*. 1993.
3. Pressman, R.S., *Software Engineering: A Practitioner's Approach*. 5 ed. 2000, Shoppenhangers Road, Maidenhead, Berkshire SL6 2QL, England.: McGraw-Hill Publishing Company.
4. Zayour, I. and T.C. Lethbridge. *Adoption of reverse engineering tools: a cognitive perspective and methodology*. in *9th International Workshop on Program Comprehension (IWPC'01)*. 2001. IEEE.
5. Sommerville, I., *Software Engineering*. 7 ed. 2004: Addison-Wesley.
6. Prechelt, L., et al., *Re-evaluating inheritance depth on the maintainability of object-oriented software.* International Journal of Empirical Software Engineering, 1998: p. 1–16.
7. Roehm, T., et al. *How do Professional Developers Comprehend Software?* . in *33rd ICSE 2011*. 2011.

8. De Lucia, A., Fasolino, A. R. & Munro, M. *Understanding function behaviors through program slicing*. in *International Workshop on Program Comprehension (IWPC'96)*. 1996. IEEE.
9. Curtis, B., Herb Krasner, and Neil Iscoe., *A field study of the software design process for large systems.* Communications of the ACM, 1988. **31**(11): p. 1268-1287.
10. Seaman, C.B. *The Information Gathering Strategies of Software Maintainers*. in *International Conference on Software Maintenance (ICSM02)*. 2002. IEEE.
11. Singer, J. *Work Practices of Software Maintenance Engineers*. in *International Conference on Software Maintenance (ICSM '98)*. 1998. Washington, Federal District of Columbia, USA.
12. Singer, J. and T. Lethbridge. *Studying work practices to assist tool design in software engineering*. in *6th International Workshop on Program Comprehension (IWPC'98)*. 1998. IEEE.
13. Sim, S.E., *Supporting Multiple Program Comprehension Strategies During Software Maintenance*, in *Department of Computer Science*1998, University of Toronto.
14. O'Brien, M.P., *Evolving a Model of the Information-Seeking Behaviour of Industrial Programmers*, 2007, University of Limerick.
15. Kingrey, K.P., *Concepts of Information Seeking and Their Presence in the Practical Library Literature.* Library Philosophy & Practice, 2002. **4**(2).
16. Starke, J., C. Luce, and J. Sillito. *Searching and skimming: An exploratory study*. in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. 2009. IEEE.
17. LaToza, T.D. and B.A. Myers. *Developers ask reachability questions*. in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. 2010. IEEE.
18. Bradac, M.G., D.E. Perry, and L.G. Votta, *Prototyping a process monitoring experiment.* Software Engineering, IEEE Transactions on, 1994. **20**(10): p. 774-784.
19. Ko, A.J., R. DeLine, and G. Venolia. *Information Needs in Collocated Software Development Teams*. in *29th International Conference on Software Engineering (ICSE'07)*. 2007.
20. Liu, W., et al. *A design for evidence-based software architecture research*. in *Workshop on REBSE'2005*. 2005.
21. Mockus, A., R.T. Fielding, and J.D. Herbsleb., *Two case studies of open source software development: Apache and Mozilla.* ACM Transactions on Software Engineering and Methodology, 2002. **11**(3): p. 309-346.
22. Wilson, C., *Network Centric Operations: Background and Oversight Issues for Congress," Congressional Research Service Report for Congress*, in *CRS Report for Congress*2007, United States Department of Defense.
23. Gasperson, T. *To Iraq and back: Soldier uses Linux in war*. 2006 30 September 2009]; Available from: http://www.linux.com/archive/articles/56216.
24. Raymond, E.S., *The Cathedral and the Bazaar.*, in *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*
2001, O'Reilly Associates, Inc. p. 241.
25. Torvalds, L. and D. Diamond, *Just for fun : The Story of An Accidental Revolutionary*. 2001, New York: HarperCollins.
26. Koponen, T. and V. Hotti, *Open source software maintenance process framework*, in *Proceedings of the fifth workshop on Open source software engineering*2005, ACM: St. Louis, Missouri.
27. Feller, J. and B. Fitzgerald, *Understanding Open Source Software Development*. 2002: Addison- Wesley , Pearson Education Limited.
28. Fitzgerald, B., *A critical look at open source.* Computer, 2004. **37**(7): p. 92-94.
29. Sharif, K.Y. and J. Buckley. *Further Observation of Open Source Programmers' Information Seeking*. in *Psychology of Programming Interest Group*. 2009. Limerick, Ireland: PPIG.
30. Gutwin, C., R. Penner, and K. Schneider, *Group awareness in distributed software development*, in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*2004, ACM: Chicago, Illinois, USA. p. 72-81.
31. LaToza, T.D., et al., *Program comprehension as fact finding*, in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*2007, ACM: Dubrovnik, Croatia. p. 361-370.
32. O'Shea, P.A., *An Investigation of Views and Abstractions Employed by Software Engineers during Software Maintenance - An Empirically Founded set of Guidelines for Visualisation Tools Supporting Comprehension*, 2006: Limerick Ireland.
33. OpenOffice. *Key Open Source "Best Practices" supported on this site*. [cited 2014 7 June]; Available from: http://www.openoffice.org/docs/bestpractices.html.en.
34. Sharif, K.Y. and J. Buckley. *Observing Open Source Programmers' Information Seeking*. in *The 20th Annual Psychology of Programming Interest Group Conference*. 2008. Lancaster University, Lancaster, United Kingdom.
35. Daniel, S., K. Stewart, and D. Darcy, *Patterns of Evolution in Open Source Projects: A Categorization Schema and Implications*, in *Patterns of Evolution in Open Source Projects: A Categorization Schema and Implications*2009, Management Information Systems Research Center, Carlson School of Management, University of Minnesota:

Minnesota.

36. Markus, M.L., *The governance of free/open source software projects: monolithic, multidimensional, or configurational?* Journal of Management & Governance, 2007. **11**(2): p. 151-163.

37. Raymond, E.S., *Homesteading the noosphere.* First Monday, 1998. **3**(10).

38. Viseur, R., *Forks impacts and motivations in free and open source projects.* International Journal of Advanced Computer Science and Applications (IJACSA), 2012. **3**(2): p. 117-122.

39. Stol, K.-J., et al., *Key factors for adopting inner source.* ACM Transactions on Software Engineering and Methodology (TOSEM), 2014. **23**(2): p. 18.

40. Midha, V. and A. Bhattacherjee, *Governance practices and software maintenance: A study of open source projects.* Decision Support Systems, 2012. **54**(1): p. 23-32.

41. Gamalielsson, J. and B. Lundell, *Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved?* Journal of Systems and Software, 2014. **89**: p. 128-145.

42. Pennington, N. *Comprehension strategies in Programming*. in *Empirical studies of programmers: second workshop*. 1987. Ablex Publishing Corp.

43. Good, J., *Programming Paradigms, Information Types and Graphical Representations : Empirical Investigations of Novice Program Comprehension*, 1999, The University of Edinburgh: Edinburgh , UK.

44. Wiedenback, S. and C.L. Corritore, *What Do Novices Learn During Program Comprehension?* International Journal of Human-Computer Interaction, 1991. **3**.

45. Buckley, J., C. Exton, and J. Good. *Characterizing Programmers' Information-Seeking during Software Evolution*. in *International Workshop on Software Technology and Engineering Practice (STEP'04)*. 2004.

46. Glaser, B. and A. Strauss, *The discovery of grounded theory: strategies for qualitative research*. 1967, London: Aldine de Gruyter.

47. Strauss, A. and J. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory (2nd ed.)*. 1998: Thousand Oaks, CA, US: Sage Publications, Inc. xiii, 312.

48. Letovsky, S. *Cognitive Process in Program Comprehension*. in *First Workshop on Empirical Studies of Programmers*. 1986. Washington, DC: Ablex Publishing Corporation.

49. Sillito, J., G.C. Murphy, and K. De Volder. *Questions programmers ask during software evolution tasks*. in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. 2006. ACM.

50. Sillito, J., G.C. Murphy, and K. De Volder, *Asking and answering questions during a programming change task.* Software Engineering, IEEE Transactions on, 2008. **34**(4): p. 434-451.

51. Johnson, W.L. and A. Erdem, *Interactive explanation of software systems.* Automated Software Engineering, 1997. **4**(1): p. 53-75.

52. Erdem, A., W.L. Johnson, and S. Marsella. *Task Oriented Software Understanding.* in *The 13th International Conference on Automated Software Engineering*. 1998.

53. Silvia, B., et al., *Information needs in bug reports: improving cooperation between developers and users*, in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*2010, ACM: Savannah, Georgia, USA.

54. Treude, C., B. 0, and S. M.A.. *How do Programmers Ask and Answer Questions on the Web?* , in *ICSE (NIER Track)* 2011. p. 804-807.

55. LaToza, T.D. and B.A. Myers. *Hard-to-answer questions about code*. in *Evaluation and Usability of Programming Languages and Tools*. 2010. ACM.

56. Fritz, T. and G.C. Murphy. *Using Information Fragments to Answer the Questions Developers Ask*. in *32nd ICSE 2010*. 2010.

57. Sharif, K.Y., *Open Source Programmers' Information Seeking.*, in *Department of Computer Science and Information System*2012, University of Limerick.

58. Reid, A. and S. Gough, *Guidelines for Reporting & Evaluating Qualitative Research: What are the Alternatives?* Environmental Educational Research, 2000. **6**(1): p. 66-91.

59. Westbrook, L., *Qualitative research methods: A review of major stages, data analysis techniques, and quality controls.* Library & Information Science Research, 1994. **16**(3): p. 241-254.

60. Neill, J. *Qualitative versus Quantitative Research: Key Points in a Classic Debate*. 2007 [cited 2010 19 March]; Available from: http://wilderdom.com/research/QualitativeVersusQuantitativeResearch.html.

61. Bogdan, R.C. and S.K. Biklen, *Qualitative Research in Education. An Introduction to Theory and Methods.* 1998, 160 Gould St., Needham Heights, MA 02194: Allyn & Bacon, A Viacom Company.

62. Riley, R.W., *Revealing socially constructed knowledge through quasi-structured interviews and grounded theory analysis.* Journal of Travel & Tourism Marketing, 1996. **5**(1-2): p. 21-40.

63. Harwood, I.A., *Developing scenarios for post-merger and acquisition integration: a grounded theory of'risk bartering'*, 2001, University of Southampton.

64. Jakarta. *The Apache Jakarta Project*. 2007; Available from: http://jakarta.apache.org/.

65. Apache. *user@commons.apache.org Archives*. [cited 2014 9 June]; Available from: http://mail-archives.apache.org/mod_mbox/commons-user/.
66. Apache. *dev@commons.apache.org*. [cited 2014 9 June]; Available from: http://mail-archives.apache.org/mod_mbox/commons-dev/.
67. Black Duck Software Inc. *Eclipse Java Development Tools (JDT): Project Summary:Factoids - Ohloh*. 2014 [cited 2014 9 June]; Available from: http://www.ohloh.net/p/eclipse-jdt/factoids#FactoidActivityIncreasing.
68. Chesbrough, H., W. Vanhaverbeke, and J. West, *Open innovation: Researching a new paradigm*. 2006: Oxford university press.
69. Pandit, N.R. *The Creation of Theory:A Recent Application of the Grounded Theory Method*. The Qualitative Report, 1996. **2**.
70. O'Brien, M.P., T.M. Shaft, and J. Buckley. *An Open-Source Analysis Schema for Identifying Software Comprehension Processes*. in *13th Workshop of the Psychology of Programming Interest Group*. 2001. Bournemouth UK.
71. Krippendorff, K., *Content analysis: An introduction to its methodology*. 2004: Sage Publications.
72. Oates, B.J., *Researching information systems and computing*. 2005: SAGE Publications Limited.
73. Hartmann, D., *Considerations in the choice of inter-observer reliability estimates.* Journal of Applied Behaviour Analysis 1977: p. 103-116.
74. Letovsky, S., *Cognitive Processes In Program Comprehension.* Journal of Systems and Software, 1987. **7**(4): p. 325-339.
75. Sousa, M.J. Castro, and H.M. Moreira. *A Survey on the Software Maintenance Process*. in *International Conference on Software Maintenance*. 1998. Bethesda, MD.
76. Ko, A.J. and B.A. Myers, *Extracting and answering why and why not questions about Java program output.* ACM Trans. Softw. Eng. Methodol., 2010. **20**(2): p. 1-36.
77. Von Mayrhauser, A. and A.M. Vans, *Program comprehension during software maintenance and evolution.* Computer, 1995. **28**(8): p. 44-55.
78. Soloway, E. and K. Ehrlich, *Empirical Studies of Programming Knowledge.* Software Engineering, IEEE Transactions on, 1984. **SE-10**(5): p. 595-609.
79. Bonaccorsi, A. and C. Rossi, *Why Open Source software can succeed.* Research Policy, 2003. **32**(7): p. 1243-1258.
80. Ghosh, R.A., *Understanding free software developers: Findings from the FLOSS study.* Perspectives on free and open source software, 2005: p. 23-46.
81. Pirolli, P. and S. Card, *Information foraging.* Psychological review, 1999. **106**(4): p. 643.
82. Thomas, R.G.G., *Instructions and descriptions: some cognitive aspects of programming and similar activities*, in *Proceedings of the working conference on Advanced visual interfaces*2000, ACM: Palermo, Italy.
83. Dit, B., et al., *Feature location in source code: a taxonomy and survey.* Journal of Software: Evolution and Process, 2013. **25**(1): p. 53-95.
84. Quilici, A., S. Woods, and Y. Zhang, *Program plan matching: experiments with a constraint-based approach.* Science of Computer Programming, 2000. **36**(2): p. 285-302.
85. Rich, C. and R.C. Waters, *The programmer's apprentice*. 1990: ACM press Addison-Wesley.
86. Arcuri, A. *On the automation of fixing software bugs*. in *Companion of the 30th international conference on Software engineering*. 2008. ACM.
87. Hangal, S. and M.S. Lam. *Automatic dimension inference and checking for object-oriented programs*. in *Proceedings of the 31st International Conference on Software Engineering*. 2009. IEEE Computer Society.
88. Ko, A.J. and B.A. Myers, *A framework and methodology for studying the causes of software errors in programming systems.* Journal of Visual Languages & Computing, 2005. **16**(1): p. 41-84.
89. Catal, C. and B. Diri, *A systematic review of software fault prediction studies.* Expert Systems with Applications, 2009. **36**(4): p. 7346-7354.
90. English, M., et al. *Fault detection and prediction in an open-source software project*. in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. 2009. ACM.
91. Nagappan, N., T. Ball, and A. Zeller. *Mining metrics to predict component failures*. in *Proceedings of the 28th international conference on Software engineering*. 2006. ACM.
92. Kim, S., et al. *Predicting faults from cached history*. in *Proceedings of the 29th international conference on Software Engineering*. 2007. IEEE Computer Society.