Communication in Open Source Software Development Mailing Lists

Anja Guzzi[1], Alberto Bacchelli[2], Michele Lanza[2], Martin Pinzger[3], Arie van Deursen[1]
1: Department of Software and Computer Technology - Delft University of Technology, The Netherlands
2: REVEAL @ Faculty of Informatics - University of Lugano, Switzerland
3: Institute for Informatics Systems - University of Klagenfurt, Austria

***Abstract*—Open source software (OSS) development teams use electronic means, such as emails, instant messaging, or forums, to conduct open and public discussions. Researchers investigated mailing lists considering them as a hub for project communication. Prior work focused on specific aspects of emails, for example the handling of patches, traceability concerns, or social networks. This led to insights pertaining to the investigated aspects, but not to a comprehensive view of what developers communicate about. Our objective is to increase the understanding of development mailing lists communication.**

We quantitatively and qualitatively analyzed a sample of 506 email threads from the development mailing list of a major OSS project, Lucene. Our investigation reveals that implementation details are discussed only in about 35% of the threads, and that a range of other topics is discussed. Moreover, core developers participate in less than 75% of the threads. We observed that the development mailing list is not the main player in OSS project communication, as it also includes other channels such as the issue repository.

I. Introduction

Open source software (OSS) development teams use electronic means, such as emails, instant messaging, or forums, to communicate. Conversations in OSS settings are typically conducted in an open public manner and are stored for later reference [8]. For this reason, OSS communication repositories offer a rich source of historical information, which can be used, for example, to observe software processes [28], to understand software developers communication dynamics [27], and to improve development practices [29].

Mailing lists have been considered—historically—the hub of project communication at the inception of the first OSS communities, such as Linux and Apache. For this reason, when studying OSS developers' communication, many researchers focused on development mailing lists: For example, to investigate the handling of patches [7], [24], traceability concerns [3], or developers' social networks [8].

These and other studies (*e.g.,* [1], [9], [21], [22], [25], [26]) are mostly based on the conventional wisdom that the role and usage of the development mailing lists (of the analyzed project) are similar to that of Linux [23] or Apache [19] in their first years. This leads to a number of assumptions, such as that development mailing list "*are primarily concerned with the software under development*" [22], and that "*communications by means of* [e]*mail is the only possible way for* [OSS developers] *to interact with each other.*" [5]

Nevertheless, there is no clear, updated, and well-rounded picture of the communication taking place in open source development mailing lists that supports these assumptions. In fact, at our disposal, we only have either abstract and outdated knowledge (*e.g.,* obtained as a side effect of the analysis of the Linux project), which does not consider the recent shift of interest to new social platforms (*e.g.,* GitHub and Jira), or a very specialized understanding (*e.g.,* regarding specific information, such as the process of code review [25]), which does not take into account all the information that can be distilled from development emails.

Our goal is to increase our understanding of development mailing lists communication: What do participants talk about? How much do they discuss each topic? What is the role of the development mailing lists for OSS project communication? Answering these questions can confirm or cast doubts on the previous assumptions, and it can provide insights for future research on mining developers' communication and for building tools to help project teams communicate effectively.

To answer these questions, we conducted an in-depth analysis of the communication taking place in the development mailing list of one major OSS software system, *i.e.,* the Apache Lucene project. We set up our study as an exploratory investigation. We started without hypotheses regarding the content of the development mailing list, with the aim of discovering the topics of communication, the prominency of implementation details, the position of developers, and the role of the development mailing list as communication channel. To that end, we manually inspected and classified 506 email threads comprising over 2,400 messages, we manually resolved the aliasing among more than 310 email addresses, and focused on gaining a holistic view on the information exchanged in the mailing list.

In this paper we make the following contributions:

- A coding system that is reusable for analysis of developer communication in general, and mailing lists in particular (Section IV).
- An assessment of relative frequency of topics in developer mailing lists (Section V).
- An assessment of relative participation of developers in developer mailing lists (Section VI).
- A qualitative evaluation of the role of development mailing list for project communication (Section VII).
- Two manually created benchmarks: one for email thread categorization and one for resolving aliases of participants.

978-1-4673-2936-1/13/$31.00 © 2013 IEEE

MSR 2013, San Francisco, CA, USA

Our results show that, although the declared intent of *development* mailing list communication is to discuss project internals and code changes/additions, only 35% of the email threads regard the implementation of code artifacts. Instead, development mailing list communication also covers a number of other topics, such as social norms and infrastructure. Also, project developers participate in less than 75% of the overall threads and they start only half of the discussions. Finally, the development mailing list is not the sole player in OSS project communication: It is complemented by other channels (*e.g.,* issue repository) from which it is disconnected.

Based on our findings, we analyze and discuss the implications for researchers and practitioners (Section X).

II. Related Work

By analyzing OSS development mailing lists, researchers provided insight in social aspects of software development. For example, researchers exploited email *metadata* (*e.g.,* author, date, and time) to conduct quantitative social analyses: Bird *et al.* proposed techniques to mine email social networks [8], and investigated social interactions in OSS projects [9]; Ogawa *et al.* visualized social interaction among participants in OSS projects [21]; and Shihab *et al.* showed that mailing list activity is related to source code activity [29]. Researchers also quantitatively analyzed the *text* of emails: Pattison *et al.* studied the frequency with which terms of software entities are mentioned in emails, and correlating it with the number of system changes [22]; Baysal and Malton searched for a correlation between discussions and software releases [5]; and Bacchelli *et al.* analyzed the correlation between email discussions and software defects [1].

Most of the aforementioned work is quantitative and based on the premise that development mailing list communication mostly regards the implementation of source code artifacts. This assumption derives from the knowledge about OSS systems provided by seminal literature such as "The Cathedral and The Bazaar" [23]. Few studies analyzed the *content* of OSS mailing list communications and mostly focused on specific traits of the communication. Gutwin *et al.* read mailing list archives to study group awareness in distributed development [14]. Rigby *et al.* analyzed mailing lists to study the OSS code reviewing process (*e.g.,* [25]). Mockus *et al.* studied the Apache Server development process finding that the mailing list play a central role for communication, coordination, and awareness [19]. We want to obtain a comprehensive knowledge of communication in development mailing lists of OSS projects.

Our work is also related to data quality: By knowing what data is available in mailing list repositories, we can devise better techniques for extracting relevant, unbiased, and comprehensible information. In this vein, researchers have studied bug repositories [31] and code repositories [16] to understand what information is more relevant. They also analyzed the impact of data quality on mining approaches and analyses (*e.g.,* [20]). In the context of mailing list data, Bettenburg *et al.* showed the risks of using email data without a proper cleaning pre-processing phase [6].

III. Methodology

To explore and understand the communication taking place in development mailing lists, we performed an in-depth analysis of the development mailing list of Apache Lucene, an OSS information retrieval framework and API.

We chose Lucene for the following reasons: (1) Lucene is a mature project with a large user base and an established community of developers. (2) It was started in 1999 by a single developer, who initially guided it as a "benevolent dictator". In 2001, Lucene joined the Apache Software Foundation and became a *foundation*, with a well-organized, hierarchical governance structure and formalized policies[1]. (3) The previous work describing the communication occurring in the development mailing list of OSS projects (*e.g.,* [17], [23]) dates back to the early 2000s, it is high-level and focuses on Linux, which is more of an exception than the rule in OSS projects [23]. Lucene's organizational structure sets it apart from the benevolent dictatorship of Linux; by choosing Lucene we aim at having an updated knowledge of contemporary developers' communication in the development mailing list in a more common OSS setting. (4) Lucene has a *publicly* archived development mailing list with a *declared* intent: The developer discussion `dev@lucene` list is " *where participating developers of the Java Lucene project meet and discuss issues concerning Lucene* [...] *internals, code changes/additions, etc.*"[2]

A. Research Questions

Our investigation revolves around four research questions:

RQ1: What topics are development mailing list participants talking about?

RQ2: How often do participants talk about each topic? How prominent are implementation details?

RQ3: Is the development mailing list just for developers? What do developers focus on?

RQ4: What is the role of the development mailing lists for the communication in the OSS at large?

B. Research Method

We followed the approach depicted in Figure 1: (**1**) we modeled all the mailing list emails, (**2**) we reconstructed threads of discussion (removing auto-generated ones), and (**3**) we randomly extracted 1,000 threads. Using open card sort [4] (see Section III-D), we manually analyzed the threads and extracted categories of discussion (**4**). To ensure the integrity of the extracted categories, we sorted threads several times and iteratively refined the catalogue (**5**). During the cart sort, we took notes about the mailing list, its role, and the communication occurring in it (**6**). We validated the resulting catalogue of categories using closed card sort (**7**). We complemented the automatically collected email data by resolving aliases and by adding information about which participants were developers (**8**). Finally, we conducted a statistical analysis on the obtained categorized threads (**9**).

[1] http://www.apache.org/foundation/

[2] http://lucene.apache.org/core/discussion.html

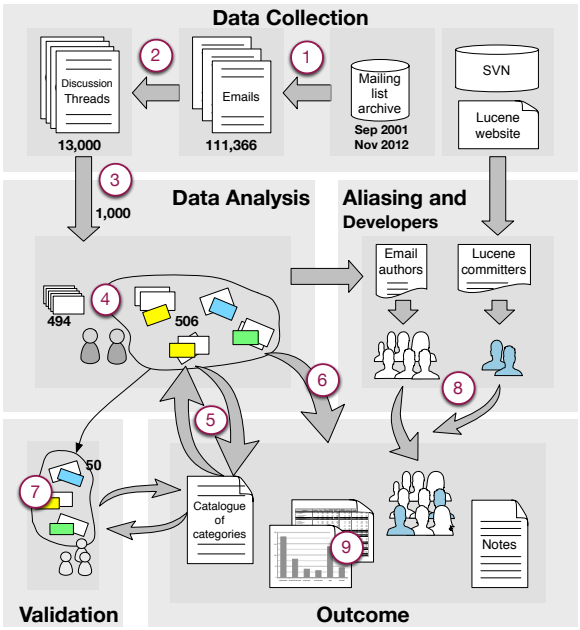

Fig. 1. The mixed approach research method applied.

C. Data Collection

In our previous work, we presented MILER, a toolset to explore email data [2]. It crawls the website of MARKMAIL[3] an online service for searching mailing lists. MARKMAIL has two drawbacks: It obscures email addresses for privacy reasons, and it does not always reconstruct email threads. To correctly recognize participants, their roles, and the discussions threads, we extended MILER to collect, extract, and model data from a more complete source than MARKMAIL: MBOX files. This implies challenges also mentioned by Bettenburg *et al.* [6].

Extracting messages: We wrote an MBOX importer tool in PYTHON to download and model emails. Although the PYTHON library `mailbox`[4] gives reliable support for loading the different messages from MBOX files, we had to write an algorithm to automatically correct wrong date formats.

Reconstructing threads: An email *discussion thread* is a set of messages that are logically related, *i.e.,* replies in the same chain of emails. To reconstruct discussion threads, we use two complementary heuristics: (1) Whenever possible, we consider the 'message-ID' (a globally unique identifier for emails) and 'in-reply-to' (used to specify the 'message-ID' of the email that it is replying to) fields to reconstruct threads. (2) Otherwise, we consider email subjects. By manually inspecting the LUCENE mailing list, we found that participants are conservative in keeping the subject consistent with the discussion: When a thread changes topic, participants accordingly modify the subject of the subsequent emails. Thus, if two emails have the same subject, or two slight variations of it (*e.g.,* they are prefixed by *Re:*), we place them on the same thread, using the timestamp for sorting.

[3]http://markmail.org
[4]http://docs.python.org/2/library/mailbox.html

Removing automatically generated emails: Many OSS projects forward a number of special automatically generated emails to development mailing list, for example, from the versioning or the issue tracking systems. For the purpose of our research, aimed at understanding what *participants* talk about in a mailing list, we filter out these automatically generated emails, unless they are answered by a person. Although this filter has to be customized to the mailing lists under analysis, we used an approach that can be adapted to other lists. It focuses on the quantity and the thread subject. In fact, automatically generated emails often outnumber those manually generated and have a well defined subject pattern. We aggregated threads with a subject starting with the same 10 characters and manually analyzed their distribution. This approach found almost all generated emails.

D. Card Sort

To group the email threads we used *card sort*, a technique used in information architecture to create mental models and derive taxonomies from input data [4]. We used it to organize the threads into groups to abstract and describe mailing list communication. A card sort has 3 steps: (1) *preparation* (select card sort participants and create the cards); (2) *execution* (sort cards into meaningful groups); and (3) *analysis* (form abstract hierarchies to deduce general categories).

Preparation: We created all cards from the sample resulted from the data collection. Each card (exemplified in Figure 2) represents a thread and includes: (1) number of emails, (2) subject, (3) duration, with timestamp of the first and last emails, (4) the first 15 lines (removing white lines) in the body of the initial email, (5) email addresses of the participants involved, and (6) an univocal `id` for later reference.

Execution phase: The first two authors analyzed the cards applying *open* (*i.e.,* without predefined groups, as they emerge and evolve during the sorting process) card sort, adapting the guidelines for the analysis of qualitative data with grounded theory [13]: They avoided information related to LUCENE (*e.g.,* its website) and the literature closely related to mailing list communication, as this could have sensitized them to look for concepts related to existing theory, thus hindering innovation in organizing the threads. They often interrupted the card sorting to memo an idea or concept potentially useful for later analysis (see Section VII). When necessary they consulted the entire thread online. Since the rigor of the card sorting method is in its analysis [18], instead of working separately on different cards, and checking the consistency of the sorting and merging the cards in a later phase, they used *pair-sorting*. This requires significantly more time, but it brings more value to the analysis as they discussed discrepancies in their thoughts for each card during the card sorting itself.

Analysis phase: To ensure the integrity of the emerging categories, the first two authors did a second pass on all the analyzed cards, starting from small groups that could not be included in any larger group, and re-categorizing these cards by redefining some categories. Subsequently, they analyzed the

remaining cards to completely describe the catalogue of thread categories (see Section IV).

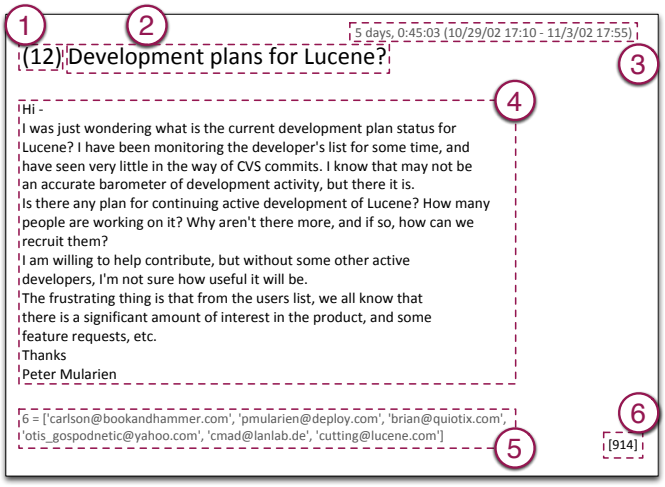

Fig. 2. Card Sort: Example Card

We conducted a validation to verify whether the catalogue was written in a clear and understandable way that was capturing all the facets of each category (see Section IX).

E. Aliasing and Identification of Developers

Resolving multiple identities (aliases) is fundamental to prepare mailing list data for the statistical analysis of the participants [9]. Although a number of approaches to solve aliasing have been proposed (*e.g.,* [8], [15]), this task cannot be fully automatized. To avoid bias in our statistical results, we manually resolved aliasing in our data. We started by aggregating on email addresses, to resolve cases with multiple author names. Then, we manually inspected all possible combinations of names and email addresses. One challenge we encountered regards a handful of participants using distinct names and addresses (*e.g.,*'John: `johns@address1.com`', and 'spacej: `spacej@address2.co.uk`'). To resolve these cases, we read the emails sent from these addresses. To answer the research question regarding developers communicating in the mailing list participants (*i.e.,* RQ3), we also identified the official committers of the project: We matched names and addresses in our sample with the official list of committers[5]. We also extracted developer user names from the versioning system log. Matching developers was time-consuming, as only few developers use their *[user-name]*`@apache.com` address listed on the Lucene website.

IV. What Are Mailing List Participants Talking About?

We extracted email data from the Lucene development mailing list,[6] from its inception (Sep 2001) to Nov 2012, totaling 111,366 emails. We aggregated them into threads and removed automatically generated messages. From the resulting 13,019 discussion threads we randomly sampled 1,000 threads and printed the corresponding cards for the card sort. After sorting the first ca. 300 cards, the new threads started merging in the same groups, reaching a saturation effect [13]. To add confidence that the saturation point was reached, and to improve the significance of the subsequent statistical analysis, another 200 cards were sorted, reaching a sample of 506 threads. The remaining cards were discarded.

Through the card sort 34 groups emerged. During the sorting process we iteratively gave explanatory names to groups and reflected on how they could be clustered into higher level themes. At the end of this phase, we had clustered the 34 groups into 6 categories and 24 subcategories. We now describe each category and the corresponding subcategories.

A. Implementation

The implementation category covers the threads related to the implementation of source code artifacts. It comprises topics spanning from proposing new features to be implemented, to discussing implementation details, to contributing with patches. It also includes emails aimed at understanding the system's implementation, or the rationale behind an implementation choice. It comprises four subcategories:

(A.1) **Comprehension:** Participants start comprehension threads to understand (parts of) the implementation, to verify if their knowledge is correct and up-to-date, and to request clarifications on the rationale behind a particular choice (*e.g.,* a used pattern or a threshold).

(A.2) **Discussion:** Participants initiate discussion threads to ask the opinion of others (*e.g.,* "*what do you think about* [this]"), or to propose one or more possible solutions or ideas (*e.g.,* "*we could do it like* [this]*, or like* [that]"). Usually, discussions revolve around improving an existing code artifact, and start from the comments on a recent feature implementation, bug fix, or submitted patch.

(A.3) **Feature suggestion:** Participants initiate this kind of threads to describe new features from a high-level perspective. Often participants requesting a feature on the mailing list are not directly volunteering to do it: They mostly propose something for others to do.

(A.4) **Code contribution:** Participants start these threads to let the community know that they have working source code ready to be merged in the system. The code may implement new features; or it may tackle issues that were found by the email author or that were reported in the official bug repository. Contributions are in the form of patches, pull requests, external links, or attached code.

B. Technical Infrastructure

Most OSS software projects rely on a technical infrastructure to support development, maintenance, and the building process, and to facilitate the communication among project contributors [12]. This category covers email threads related to such an infrastructure; the topics of discussions are (B.1) **building system** (*e.g.,* notification of problems with the building system), (B.2) **documentation** (*e.g.,* decisions on the javadoc), (B.3) **issue tracking** (*e.g.,* move to a new tracking system), (B.4) **mailing**

[5]http://lucene.apache.org/whoweare.html
[6]org.apache.lucene.java-dev

LISTS (not only the development mailing list itself, but also *e.g.,* the user mailing list), (B.5) PROGRAMMING LANGUAGE (*e.g.,* the version of [programming language] to use), (B.6) TESTING (*e.g.,* how to use the continuous testing system), (B.7) VERSIONING (*e.g.,* discussions on branches), and (B.8) WEBSITE (*e.g.,* threads on what content to put in the website). Authors of infrastructure threads write to the list for different reasons, such as sending notifications, discussing problems, and posing questions.

C. Project Status

As described in previous work (*e.g.,* [9], [19], [23]), development mailing lists are also used to raise awareness on the status of the project and to discuss future steps. This category regards these kind of topics, and includes two groups of threads: those about (C.1) PLANNING the future development of the project, and those about (C.2) RELEASES. Authors of PROJECT STATUS threads write to the mailing list to announce a new release, to decide which issues to fix for a milestone, or to discuss the ongoing activity on the project.

D. Social Interactions

Socializing is an essential ingredient in the long-term survival of OSS projects [10], and mailing lists play an important role in this context [12]. Participants write to the mailing list about the norms, values, and perspectives that are part of the community's operational structure, and to coordinate with others. This category revolves around these social interactions, and threads are about (D.1) ACKNOWLEDGEMENT of efforts (*e.g.,* replying to a code commit to thank the author), (D.2) COORDINATION (*e.g.,* raising awareness about an issue in the bug repository, or notifying a participant's absence),greetings and suggestions to (D.3) NEW CONTRIBUTORS, and (D.4) SOCIAL NORMS governing the behavior of mailing list participants (*e.g.,* advices on successfully submitting a patch). Authors of such threads notify their absence, welcome new members, thank someone for a bug fix, and tell everyone about newly submitted issues.

E. Usage

The USAGE category comprises threads with questions and problems about the usage of the software being developed by the programmers enrolled in the development mailing list, and it also includes threads related to external projects. It comprises three subcategories:

(E.1) **PROBLEMS AND BUGS:** Authors ask advice on how to solve issues they have operating the project, or report a general problem they have found. Participants may also bring up a discussion about a problem by forwarding emails sent to other mailing lists, or by answering automatic messages from the issue tracking system.

(E.2) **INFORMATION SEEKING:** Authors write to ask advice on how to complete an operation (*e.g.,* "*How to do* [this]*?*"), on where to find usage related resources (*e.g.,* documentation, examples), and on the right approach to choose among different usage options (*e.g.,* "*What is the proper means to do* [this]*?*").

(E.3) **EXTERNAL PROJECTS:** Participants write, for example, to raise awareness about their own, external, software project. They ask to be included among the online list of applications using the main project (*e.g.,* "*Powered by*"). Participants developing other systems also ask about including their work as part of the main project.

F. Discarded

This category groups the threads that do not fit into the categories previously described. They are of three kinds:

(F.1) **AUTO-GENERATED:** Auto-generated threads, such as emails from the continuous building system or the wiki, that were not filtered out by our heuristics.

(F.2) **TRASH:** Threads exclusively composed of unreadable emails (*i.e.,* due to formatting problems), and spam emails that are not pertaining to the content of the mailing list (*i.e.,* unsolicited commercial emails).

(F.3) **TURTLE:** Email threads that are unrelated to any other thread, or very difficult to classify due to the nature of their content (*e.g.,* meaningless because out of context).

V. How Often Do Participants Talk About Each Topic?

Figure 3 shows the distribution of the threads among the different categories (see also column 'threads' in Table I).

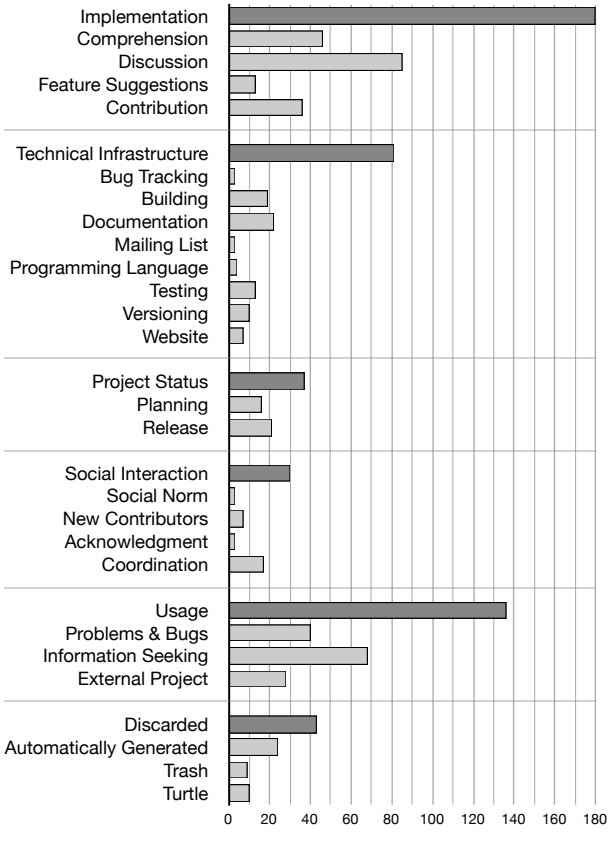

Fig. 3. Distribution of threads per category.

A. How Are Topics Distributed Among Threads?

IMPLEMENTATION is the most frequently occurring category, comprising 36% of the threads. Since the declared aim of the *development* mailing list of LUCENE is to be where "*participating developers* [...] *meet and discuss issues concerning* LUCENE [...] *internals, code changes/additions, etc.*", we were surprised that—in reality—IMPLEMENTATION threads only count for just a little more than a third of the total threads. This is different from the Linux kernel mailing list (often used for studying developers' interaction), where IMPLEMENTATION threads "*form the large majority of the traffic on the list.*" [14]

In comparison, we found the ratio of USAGE threads in the mailing list to be surprisingly high (27%). In particular, half of these threads regard INFORMATION SEEKING (13% overall), in spite of a note on the LUCENE website exhorting participants to "*not send mail to this list with usage questions or configuration questions and problems*". Moreover, threads regarding PROBLEMS AND BUGS account for 8%. Even considering sampling limitations (see Section IX), in LUCENE these threads would corresponds to less than half of the bugs reported in JIRA (up to one fifth, when considering other types of issues), meaning that in LUCENE the mailing list may not be the primary channel for discussing and reporting problems and bugs.

Threads on TECHNICAL INFRASTRUCTURE total 16%. The less frequent categories are SOCIAL INTERACTION and PROJECT STATUS. It was surprising to us that, despite the mailing list always having been considered the hub for OSS project communication [12], [23], only 7% of the threads regard the project status, and just 6% regard social interactions among participants.

Finally, there is a not negligible portion (8%) of threads DISCARDED during the card sorting. Besides 10 threads with no clear meaning (TURTLE), and—despite the fact that we performed an rigorous pre-processing and data cleaning phase (Section III)—a substantial amount of noise (7% of the total threads, from AUTOMATICALLY GENERATED and TRASH threads) was still present our sample. We also notice that these threads cannot be clearly distinguished from the other categories: a third of them are replied (*e.g.,* there are threads automatically generated from the wiki, which all have the same subject and thus get threaded), almost a third include developers in the emails (*e.g.,* svn commits initially sent to the mailing list results as sent by the developer author of the commit), and finally almost a fourth of these threads contain code (*e.g.,* svn commits, and change logs from the wiki pages).

B. How Prominent Are Implementation Details?

To better understand how prominent implementation details are, we analyzed the distributions of threads containing code entities (*e.g.,* class names). Are mentioned code artifacts an indication of discussion about implementation details?

In previous work, Bird *et al.* reported that the mailing list is made of more than implementation. They distinguish between *process* and *product*, and use the presence of source code names, such as class names, as classifiers: "*Messages that include these source code names are classified as product and the rest are classified as process*" [9]. We also apply this distinction to our data and verify whether and how it fits to our categories. We considered the entities mentioned in all the releases of LUCENE, and we analyzed threads to determine whether they contained code entities. Results from our analysis can be seen in the column 'with code entities'.

Our results show that 57% of all the analyzed threads contain code entities, and at least a third of threads in each category contains code entities (except DISCARDED threads, 28%). Of IMPLEMENTATION threads, 77% contains code.

To verify to which degree Bird *et al.*'s classification fits to our data, we first need to define which of our own categories are part of *product*. According to the description, these would correspond to our IMPLEMENTATION category alone. However, USAGE and DISCARDED threads would not fit in either definition: we decided to include USAGE as *product* (since LUCENE is an API, many usage questions regards its code artifacts), while we consider DISCARDED as *process*.

Our data shows that when only considering threads containing code entities, only 76% of these threads would be regarding *product* (*i.e.,* IMPLEMENTATION and USAGE), while the remaining 24% would actually be about *process*. Moreover, we would only select 70% of all the IMPLEMENTATION+USAGE threads. This is in contrast with Bird *et al.* findings, where they estimated a correct classification in 90% of the cases.

VI. Is the Development Mailing List Only for Developers?

Once our categories were stable, and after performing several card sort iterations to ensure the integrity of our categories, we resolved aliasing and determined which participants were project developers (*i.e.,* those with commit privileges). Table I shows the statistical information we collected on the sample of threads categorized in the card sorting process. We include email granularity for completeness.

A. What Do Developers Focus On?

The overall ratio of threads in which at least one developer participated (column 'with developers') is quite high: Developers are present in more than 75% of the treads in each category, except in USAGE (55%) and DISCARDED (35%). In PROJECT STATUS and TECHNICAL INFRASTRUCTURE threads, developers are present in more than 90% of these threads.

Our results also show that in some categories there is a prevalence of threads 'started by developers'. However, overall, only half of all the analyzed threads have been started by a developer. Developers start the majority of threads in PROJECT STATUS (89% of the threads in this category), TECHNICAL INFRASTRUCTURE, (78%), and SOCIAL INTERACTION (70%). Only 54% of the IMPLEMENTATION threads are started by a developer. This may seem surprising, but, if we look at the subcategories, we can see that only a third of CONTRIBUTION threads were started by developers. This is also due to the OSS structure in general, where a person can be a contributor without committing rights. Participants write to the mailing list *offering* their contributions, hoping that a developer might integrate it in the project. Moreover, users occasionally write to the development mailing list with program comprehension questions or feature requests.

TABLE I
CATEGORIZATION OF EMAIL THREADS.

	categories	threads	replied	with developers	started by developers	with code entities	unique participants	developers	emails	from developers	with code entities
A.1	Comprehension	**46**	*74%*	*74%*	*43%*	*78%*	**66**	*39%*	**208**	*60%*	*72%*
A.2	Discussion	**85**	*80%*	*86%*	*68%*	*78%*	**87**	*39%*	**551**	*70%*	*70%*
A.3	Feature Suggestion	**13**	*54%*	*77%*	*54%*	*62%*	**19**	*53%*	**35**	*66%*	*51%*
A.4	Contribution	**36**	*75%*	*81%*	*33%*	*81%*	**56**	*39%*	**135**	*59%*	*62%*
A	**Implementation (36%)**	**180**	**76%**	**81%**	**54%**	**77%**	**155**	**26%**	**929**	**66%**	**69%**
B.1	Bug Tracking	**3**	*100%*	*100%*	*67%*	*0%*	**8**	*88%*	**24**	*92%*	*0%*
B.2	Building	**19**	*84%*	*95%*	*53%*	*37%*	**25**	*56%*	**54**	*72%*	*24%*
B.3	Documentation	**22**	*59%*	*95%*	*86%*	*45%*	**33**	*73%*	**78**	*83%*	*37%*
B.4	Mailing List	**3**	*33%*	*67%*	*67%*	*0%*	**4**	*75%*	**4**	*75%*	*0%*
B.5	Programming Language	**4**	*100%*	*100%*	*75%*	*50%*	**27**	*52%*	**100**	*54%*	*18%*
B.6	Testing	**13**	*77%*	*92%*	*92%*	*62%*	**21**	*81%*	**71**	*94%*	*42%*
B.7	Versioning	**10**	*80%*	*90%*	*80%*	*20%*	**28**	*61%*	**76**	*78%*	*4%*
B.8	Website	**7**	*86%*	*100%*	*100%*	*0%*	**13**	*85%*	**32**	*94%*	*0%*
B	**Technical Infrastructure (16%)**	**81**	**75%**	**94%**	**78%**	**36%**	**76**	**43%**	**439**	**77%**	**21%**
C.1	Planning	**16**	*88%*	*94%*	*88%*	*56%*	**48**	*54%*	**233**	*84%*	*19%*
C.2	Release	**21**	*71%*	*90%*	*90%*	*38%*	**34**	*56%*	**126**	*85%*	*28%*
C	**Project Status (7%)**	**37**	**78%**	**92%**	**89%**	**46%**	**63**	**48%**	**359**	**84%**	**22%**
D.1	Social Norm	**3**	*33%*	*100%*	*100%*	*0%*	**4**	*75%*	**6**	*83%*	*0%*
D.2	Contributors	**7**	*71%*	*86%*	*71%*	*29%*	**15**	*80%*	**26**	*85%*	*19%*
D.3	Acknowledgment	**3**	*0%*	*100%*	*100%*	*33%*	**3**	*100%*	**3**	*100%*	*33%*
D.4	Coordination	**17**	*35%*	*65%*	*59%*	*47%*	**17**	*47%*	**29**	*41%*	*38%*
D	**Social Interaction (6%)**	**30**	**40%**	**77%**	**70%**	**37%**	**30**	**57%**	**64**	**66%**	**27%**
E.1	Problems & Bugs	**40**	*58%*	*70%*	*35%*	*80%*	**53**	*34%*	**128**	*45%*	*81%*
E.2	Information Seeking	**68**	*68%*	*47%*	*4%*	*60%*	**99**	*24%*	**210**	*37%*	*61%*
E.3	External Project	**27**	*59%*	*52%*	*41%*	*30%*	**45**	*36%*	**86**	*52%*	*24%*
E	**Usage (27%)**	**135**	**63%**	**55%**	**21%**	**60%**	**164**	**20%**	**424**	**43%**	**60%**
F.1	Automatically Generated	**24**	*33%*	*25%*	*21%*	*29%*	**6**	*50%*	**156**	*5%*	*19%*
F.2	Trash	**9**	*33%*	*44%*	*44%*	*11%*	**16**	*63%*	**30**	*77%*	*20%*
F.3	Turtle	**10**	*60%*	*50%*	*30%*	*40%*	**19**	*42%*	**27**	*44%*	*33%*
F	**Discarded (8%)**	**43**	**40%**	**35%**	**28%**	**28%**	**34**	**44%**	**213**	**20%**	**21%**
	Total	**506**	**67%**	**73%**	**50%**	**57%**	**315**	**16%**	**2428**	**63%**	**46%**

Furthermore, we notice that only 21% of the USAGE threads were started by a developer, and, in particular, only 4% of the INFORMATION SEEKING threads. It is not very surprising that these threads are not started by LUCENE developers. However, developers also start EXTERNAL PROJECT threads: They often have side projects, built on top of LUCENE, they want to mention in the mailing list (*e.g.,* announcing a new release).

B. Dynamics of Interactions

By analyzing the population of mailing list participants, we found that only 16% of the participants are official committers (column 'developers'). Thus, the vast majority of participants in the development mailing list are *not* LUCENE developers. We asked ourselves: How are participants interacting via the mailing list? Do developers have a particular position?

The column 'Unique participants' indicates the number of individual people participating to discussions threads. When the number of participants is lower than the number of threads, this means that people are participating in more than one thread (*e.g.,* this is the case in the IMPLEMENTATION category). Similarly, a higher number of participants than the number of discussion threads indicates "one-timers" (we observe this in the USAGE and PROJECT STATUS categories).

To better understand where participants interact, we counted threads that are *replied to* (*i.e.,* with more than one email). The analysis of the replied threads by category gives an idea of the responsiveness of the mailing list and the "rhytm" of talks within each category. Interestingly, the threads that are responded the least are those about the SOCIAL INTERACTION (40% overall). We also analyzed multi-email threads in terms of first-response rate (*i.e.,* how long before the first reply). Threads are answered within a day: TECHNICAL INFRASTRUCTURE and SOCIAL INTERACTION threads get faster reactions (first reply within two hours), while IMPLEMENTATION and USAGE threads might take up to 21 hours to be replied to. We also measured if there was a difference in responsiveness depending on who sent the first email (*i.e.,* a developer or not): We did not find a statistically significant difference.

C. The Overall Picture

Figure 4 puts all the threads in our sample in a nutshell: It shows, by category, how many threads are with *vs.* without code entities (left vs right side), with *vs.* without participating developers (top vs bottom bar), and how many of the latter have been initiated by a developer (light vs dark color). The exact amount of threads is reported for each "type".

We see a large amount of threads without developers in the USAGE category compared to other categories, a prevalence of developers on IMPLEMENTATION and TECHNICAL INFRASTRUCTURE, and a large amount of threads without developers in the USAGE category compared to other categories.

VII. WHAT IS THE ROLE OF THE DEVELOPMENT MAILING LIST?

By answering the previous research questions, we found that the official description of the aim of the development mailing list does not correspond to its real usage. Our fourth research question seeks to understand the role of development mailing lists for the communication in OSS at large. We attempt to achieve this by triangulating the information that we obtained by reading the 506 threads during the card sort, by analyzing the statistical data on the categories, and by searching more facts in the rest of the mailing list.

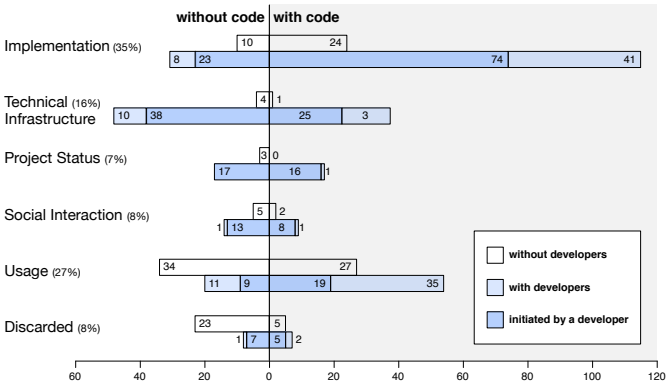

Fig. 4. Thread types distribution: gives an idea of the distribution of threads and the different "shapes" of our categories.

A. Is in the Mailing List Where All the Communication Occurs?

Previous literature stated that mailing lists are "*the bread and butter of project communications*" [12], and in particular that "*the developer mailing list is the primary communication channel for an OSS project*" [14]. Reading the analyzed emails, however, makes it clear that the development mailing list is just *one* of the communication channels used in a OSS project; in fact, other channels also play an important role:

Issue Repository: Many threads provided evidence that a significant amount of communication takes place in the JIRA issue repository: Participants often reference JIRA issues in emails, or omit details because already mentioned in the issue discussions. Altough project members started using JIRA only in mid 2005, in our entire population of emails (Sep 2001 to Nov 2012), we found 69,632 (63%) messages automatically forwarded from discussions taking place in JIRA, still showing a clear increasing trend in its usage.

IRC: Participants talk about the project and implementation details also on the development IRC channel (created in Apr 2010): "*I propose that we chat on irc at #lucene-dev* [...]. *I'd like to discuss the core elements of the Spatial Strategy API, namely makeQuery,* [...]*, and SpatialOperation.*" The channel was created in Apr 2010.

User Mailing List: The user mailing list also plays a role in the project and developers' communication. Developers monitor it, for example, to understand the usage of the system (*e.g.,* "*I am wondering if TermVectorsWriter is still used* [...]. *The reason I am asking is the java-user* [email subject]"), to improve the documentation (*e.g.,* "*about the exposure of FieldCache in the documentation* [...] *see for instance this discussion in the user list*"), and to forward interesting discussions to other developers.

In person: We found evidence that developers also have a number of in person meetings where they discuss about project details (*e.g.,* "[Developer] *and I talked a little bit about this at the ApacheCon*").

B. Is the Mailing List for Driving Coordination?

Previous work reported that a portion of the communication taking place in the mailing list regards coordination between developers as they work together on the software [9], [23]. Surprisingly we found a very small amount (3%) of COORDINATION threads, with an average of less than two emails per thread; moreover, most of these threads were not for fostering collaboration on the implementation, but for raising awareness on already accomplished work.

By reading emails, we found evidence that developers, instead of using the mailing list, prefer to coordinate through items in the issue tracking system. For example, one developer who sent an email with: "*If you can help, please coordinate here on this thread, so that we don't stomp on each other.*" afterwards corrected himself in a second message: "*Sorry, should have said, please coordinate on the* JIRA *issue*". Another developer, who was guiding a newcomer through the coordination norms in the project, wrote: "*You will not fall out of sync in short order, especially if you work with* JIRA *so others know what you are doing.*"

In addition to the issue tracking system, developers also coordinate in the IRC channel: "*As we discussed on IRC yesterday, the number of people* [...] *qualified to write* [code] *will still be very small*"; or in person: "*I talked about this with* [list of developers] *in Berlin, and they all like this proposal.*" Moreover, developers remain coordinated by keeping track of code changes. They do this by reading emails generated by the versioning system, sometimes forwarding these emails to the development list along with their comments.

C. Is the Mailing List Used for Peer Code Review?

Rigby *et al.* reported that OSS mailing lists are also used for submitting patches and performing peer code reviews [25]. We indeed found that most patches led to a purely technical discussion, while some others also led to a discussion of project objectives, scope, or politics.

The vast majority of threads with patches in our sample was sent earlier than the introduction of the JIRA issue tracking systems: After mid 2005, we saw the number of patches drastically diminishing. Reading emails, we found additional evidence that patches, nowadays, are not sent anymore to the mailing list, but they are sent, discussed, peer-reviewed, and approved/rejected in the issue tracking system. For example, when a contributor asked to go to the issue repository to review a patch: "[issue id] *Did anyone try out or took a look at my redesign* [...]*? I'd love some feedback.*" A senior developer explained: "*You should submit *all* patches you want to commit to* JIRA *first to give others the chance to review and possibly vote against the patch.*" This finding is inline with the project website: "*How to contribute:* [...] *Finally, patches should be attached to a bug report in* JIRA."

D. Is the Mailing List the Hub of Project Communication?

Although other researchers also found that the development mailing list is not the only channel of communication in OSS projects (*e.g.,* [9], [26]), it has always been considered the *hub*

of project communication. For example, Mockus *et al.* reported that developers use "*email lists exclusively to communicate with each other*" and that "*due to some annoying characteristics of the* [issue tracking system]*, very few developers keep an active eye on* [it]." [19].

When more communication repositories exist, the policy of most OSS projects is to transfer all the official decisions and useful discussions to the mailing list [12], so that they can be later retrieved. These traceability links between the development mailing list and other communication repositories must be manually created and updated. We found some cases in which the traceability link was established, but, more often and in line with the findings of Sarma *et al.* [26], we found a clear *disconnection* among repositories, which led to coordination issues and duplicated/lost information. For example, because of multiple communication repositories, developers need to raise inter-repository awareness (*e.g.,* "*I submitted a patch for* [JIRA issue] *a month ago,* [...] *it hasn't been picked by anybody yet*"), ask where a discussion takes place (*e.g.,* "*were there emails about it or it has been discussed on IRC?*"), and go back and forth between the same discussion taking place in more venues (*e.g.,* "*We would like to implement* [this]*, which was discussed in* JIRA"). Overall, our study provides evidence that the communication channels work in parallel and they remain disconnected between one another, and that the development mailing list does not play (anymore) the role of a hub.

VIII. IMPLICATIONS

From our investigation, we found that the role of the development mailing list, previously considered as the place for discussing code artifact implementation and as the hub of all project communication, has changed. In the following we describe some of the subsequent implications.

On Communication. The development mailing list is no longer the hub of OSS project communication: communication is scattered among repositories. This once again underlines the importance of adopting a holistic view and considering software repositories as a whole, not only in research but also in practical development. In fact, even project developers have problems in maintaining awareness of each other's work in the current situation. Automatically recovering traceability links among communication repositories would free developers from the task of recovering scattered traces of previous communication, and would help researchers having a more complete picture of the development process. More tools for maintaining awareness would be also necessary to improve developers' productivity. Since the advent of better issue tracking systems led to a shift in the habits of OSS participants toward different communication means, we should investigate the features in issue tracking systems that produced this change of direction.

On Data Quality. We found that a number of different communication topics take place in the development mailing list; to extract valuable information we have to take this into account. First, we have to improve our methods for removing noise (8% of our sample, even after a careful pre-processing phase), then there are the premises for future work on automatic classification of threads of discussions, so that only the relevant categories could be taken into account for analysis. We underlined the importance of a correct aliasing resolution, which still cannot be fully automatized. We provide our complete aliasing and thread categorization to used to benchmark novel automatic techniques. Nevertheless part of the communication data is going to be lost, because communication takes place in unrecorded places, even in OSS systems. We have to take this into account in our statistical analyses.

On Software Development. We found that not only committers respond to the development mailing list, but also other people are very active. We could consider techniques for finding code experts not only among active contributors, but also among active respondents of the mailing list. Moreover, considering the shift to other communication repositories, mailing list may not be the right venue for studying code review anymore. In this context we can further investigate the role of issue tracking systems and social coding websites such as GITHUB.

IX. LIMITATIONS

One potential criticism is that a case study with one project may provide little value. Historical evidence shows otherwise: Flyvbjerg gave many examples of individual cases contributing to discoveries in physics, economics, and social science [11]. To understand mailing list communication we read emails spanning 11 years of mailing list usage, and written by 155 diverse participants. To answer our research questions, we also analyzed data from the code repository, the project website, and email threads external to our sample.

To ensure that the thread categories emerged from the card sort were clear and accurate, and to judge whether our set of category provides an exhaustive and effective way to organize mailing list communication, we conducted a validation phase that involved three people external to the pair-card sort. Three software engineering researchers conducted a closed card sort on 50 cards (10%) randomly selected from our sample. They observed that the 6 main categories were clear and covered all thread topics. We measured inter-rater agreement: The Fleiss' Kappa value for the four ratings of the random sample was 0.657 (*i.e.,* substantial agreement) for the six categories, and 0.505 (*i.e.,* moderate agreement) for the 24 sub-categories (which were more difficult to be all recalled by participants). To verify whether there was a systematic error in our catalogue, we also measured the inter-rater agreement among the three experiment participants. Their agreement was 0.592 for the main categories, and 0.458 for sub-categories (both corresponding to a moderate agreement, suggesting there was no systematic misinterpretation).

Threats to validity—Concerning *internal* threats, the sample size (506) of threads provides a 98% confidence level and 5% error on subsequent estimations of proportions [30]. Concerning *external* threats, other OSS projects use communication tools similar to LUCENE, for example, 87 other Apache projects are also using the JIRA issue tracking system and have IRC channels. However, team dynamics may differ and our research should be repeated in other contexts.

X. Conclusions

Conventional wisdom on OSS communication is that development mailing lists are central to the entire development process and serve as the hub for project communication. This knowledge derives from the first analyses of very successful OSS projects, such as Linux or Apache. Nevertheless, in the years, a number of other successful OSS projects driven by different communities emerged, other communication means appeared, and there has been a shift of interest to new social platforms that support project development. Despite this scenario, most of the research work related to mailing list communication builds on the conventional wisdom generated by early work. Moreover, prior work focused on specific aspects of emails (*e.g.,* code review, traceability concerns, or social networks), which led to insights pertaining to the investigated aspects, but not to a comprehensive view of what developers communicate about.

We conducted a case study on Lucene to gain a more comprehensive and updated view of OSS development mailing list communication. Our results show that email threads cover a wide range of topics and implementation details are only in a portion of them, that code artifacts are also mentioned in topics not related to implementation, and that project developers are not the majority of the participants. Moreover, our case study provides evidence that the development mailing list is only *one* of the communication channels used in an OSS project, and that there has been a shift in the communication habits toward an increased usage of issue repositories. In addition, some of our findings cast doubts on assumptions made in previous work, for example that development mailing lists are only for programmers discussing implementation details.

Although based on a single OSS system, we hope that the insights we have discovered will lead to a more comprehensive analysis of communication repositories and to improved tools for aiding developers communicate.

The entire dataset used in the experiment, including the cards, the categorized threads, the resolved aliases, and the detailed statistical results, can be found online on the website supporting this paper.[7]

Acknowledgment

Bacchelli gratefully acknowledges the Swiss National Science foundation's support for the project "SOSYA" (SNF Project No. 132175).

References

[1] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? In *Proc. of FASE'10*, pages 59–73, 2010.

[2] A. Bacchelli, M. Lanza, and M. D'Ambros. Miler: A toolset for exploring email data. In *Proc. of ICSE'11*, pages 1025–1027, 2011.

[3] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proc. of ICSE'10*, pages 375–384. ACM Press, 2010.

[4] I. Barker. What is information architecture? http://www.steptwo.com.au/, May 2005.

[5] O. Baysal and A. J. Malton. Correlating social interactions to release history during software evolution. In *Proc. of MSR'07*, page 7. IEEE CS, 2007.

[6] N. Bettenburg, E. Shihab, and A. E. Hassan. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *Proc. of ICSM'09*, pages 539 –542. IEEE CS, 2009.

[7] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in OSS projects. In *Proc. of MSR'07*, pages 26–29. IEEE CS, 2007.

[8] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proc. of MSR'06*, pages 137–143. ACM, 2006.

[9] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proc. FSE'08*, pages 24–35. ACM, 2008.

[10] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *CSCW*, 14(4):323–368, 2005.

[11] B. Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245, 2006.

[12] K. Fogel. *Producing Open Source Software*. O'Reilly Media, 2005.

[13] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory: Strategies For Qualitative Research*. Aldine, 1967.

[14] C. Gutwin, R. Penner, and K. A. Schneider. Group awareness in distributed software development. In *Proc. of CSCW'04*, pages 72–81, 2004.

[15] A. Guzzi, A. Begel, J. K. Miller, and K. Nareddy. Facilitating enterprise software developer communication with cares. In *Proc. of ICSM'12*, pages 527–536, 2012.

[16] D. Kawrykow and M. P. Robillard. Non-essential changes in version histories. In *Proc. of ICSE'11*, pages 351–360, 2011.

[17] K. Kuwabara. A bazaar at the edge of chaos. *First Monday*, 5(3), 2000.

[18] B. Martin and B. Hanington. *Universal Methods of Design*. Rockport, 2012.

[19] A. Mockus, R. T. Fielding, and J. D. Herbsleb. A case study of open source software development: the apache server. In *Proc. of ICSE'00*, pages 263–272, 2000.

[20] T. H. D. Nguyen, B. Adams, and A. E. Hassan. A case study of bias in bug-fix datasets. In *Proc. of WCRE 2010*, pages 259 –268. IEEE CS Press, 2010.

[21] M. Ogawa, K.-L. Ma, C. Bird, P. T. Devanbu, and A. Gourley. Visualizing social interaction in open source software projects. In *Proc. of APVIS07*, pages 25–32, 2007.

[22] D. Pattison, C. Bird, and P. Devanbu. Talk and Work: a Preliminary Report. In *Proc. of MSR'08*, pages 113–116. ACM, 2008.

[23] E. Raymond. *The Cathedral and the Bazaar - Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1999.

[24] P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: a case study of the Apache server. In *Proc. of ICSE'08*, pages 541–550. ACM, 2008.

[25] P. C. Rigby and M.-A. Storey. Understanding broadcast based peer review on open source software projects. In *Proc. of ICSE'11*, pages 541–550. ACM, 2011.

[26] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Proc. of ICSE'09*, pages 23–33. IEEE CS Press, 2009.

[27] A. Schröter, J. Aranda, D. Damian, and I. Kwan. To talk or not to talk: factors that influence communication around changesets. In *Proc. of CSCW'12*, pages 1317–1326, 2012.

[28] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE TSE*, 25:557–572, 1999.

[29] E. Shihab, N. Bettenburg, B. Adams, and A. E. Hassan. On the central role of mailing lists in open source projects: An exploratory study. In *JSAI-isAI Workshops*, pages 91–103, 2009.

[30] M. Triola. *Elementary Statistics*. Addison-Wesley, 2006.

[31] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What makes a good bug report? *IEEE TSE*, 36(5):618–643, 2010.

[7] http://www.st.ewi.tudelft.nl/~guzzi/oss-communication