How the FLOSS Research Community Uses Email Archives

Megan Squire
Elon University, USA

ABSTRACT

Artifacts of the software development process, such as source code or emails between developers, are a frequent object of study in empirical software engineering literature. One of the hallmarks of free, libre, and open source software (FLOSS) projects is that the artifacts of the development process are publicly-accessible and therefore easily collected and studied. Thus there is a long history in the FLOSS research community of using these artifacts to gain understanding about the phenomenon of open source software, which could then be compared to studies of software engineering more generally. This paper looks specifically at how the FLOSS research community has used email artifacts from free and open source projects. It provides a classification of the relevant literature using a publicly-available online repository of papers about FLOSS development using email. The outcome of this paper is to provide a broad overview for the software engineering and FLOSS research communities of how other researchers have used FLOSS email message artifacts in their work.

Keywords: Open source, free software, OSS, FOSS, F/OSS, FLOSS, literature review, software engineering, artifacts, email, mailing list.

INTRODUCTION

The FLOSS research community studies the specifics of how software systems with free, libre, and open source software (FLOSS) licenses are developed and used, and how they evolve. To study this phenomenon, FLOSS researchers may choose to study the artifacts of the development process. The artifacts most frequently studied include archives of group communication (such as IRC logs or email mailing lists), the histories and reports of bug tracking systems, documents produced as a byproduct of its development (documentation, manuals, requirements documents), and the source code itself. These software development artifacts are available from every individual FLOSS project, either at their own web site, or, if the project is hosted at a third-party code forge (for example Github or Sourceforge), from its project page on that forge. Researchers may perform both computational and semantic analyses on the artifacts and metadata of the FLOSS software development process.

This paper is an attempt to classify the ways in which one particular artifact (email message archives) is being used by researchers to understand the FLOSS development process better. Specifically, this paper searches the large body of FLOSS literature to

determine whether and how email is collected for use as a research artifact. Relevant papers were read in order to determine where the emails have been stored, how they have been cleaned, processed, and analyzed, and how the results have been saved or published.

Outcomes of this work are two-fold. First, it will be useful to gather in one place the vast body of literature from the FLOSS community that relies on email artifacts. Earlier work by Crowston, *et al.* (2012) is an examination of the FLOSS literature in the large, covering nearly every aspect of empirical research in this area. Our study will be a more in-depth examination of exactly one type of research artifact or data source, and how it has been used. Second, the results of this email artifact survey can guide the leaders of FLOSS data repositories. Three such repositories are FLOSSmole (Howison, *et al.,* 2006), SRDA (Van Antwerp & Madey, 2008), and FLOSSMetrics (Herraiz, *et al.* 2009). Results here can indicate which artifacts are being used as data in the field, as reflected in its literature, so that these repositories are collecting and storing the emails in a way that is most helpful to the research community.

To achieve these outcomes, the Motivation section outlines in more detail the reasons for doing this survey, and lists the four research questions. The Background section describes the technical foundations for inquiring about email as an artifact of the development process. The Method section explains the structure for the classification process, including an outline of the broad categories of papers and their relation to each other. Following this is a section that includes the Results of the classification, followed by the Limitations and Future Work section.

MOTIVATION

In this paper we examine the literature to determine how emails are used by researchers who study FLOSS. Such a survey does not yet exist for either the software engineering community or the FLOSS research community. Researchers can use this survey to get a handle on the current state of the art in using email archives for FLOSS analysis. For example, questions researchers might ask would be:

- Who has used Method X or Project Y in a study already?
- What techniques have already been used, and what were the results? Can I cross-apply these to Project Y or refine their results by adding Method X?
- Are there any studies that would be interesting to replicate?

Also, as mentioned in the Introduction, the results of this email artifact survey can be used to guide the leaders of FLOSS data repositories (which many times include collected artifacts of the development process, as in Howison, *et al*.) as to how to proceed with email collection for use by the entire research community. To use their resources wisely, the administrators of these repositories need to understand what data is desired by the research community. Then they can write collection scripts or request donations of the appropriate data. These data sets are then curated and made available to the community at large. Sharing data sets streamlines the data collection process for the entire research community and allows people to refer to common data sets, reproduce results from each others' work, etc.

In order to know which emails to collect, how to store the emails, and how to present the emails as re-usable data, the repository administrators will ask the following:

- Which emails should be collected? Which projects should emails be collected from, and which can be ignored?
- Are there particular analyses that are performed again and again?
- Can we pre-process the data for you in a useful way to make your analysis work go faster?
- Are there techniques that should be applied to thousands of projects simultaneously as part of the data repository collection process?
- Are there analysis techniques that have only been performed on small email collections but could be applied to larger collections?

Research Questions

This broad survey and classification is the first step in answering a few of these very practical questions about how the FLOSS community uses email as an artifact of study. The following four research questions are the focus of our classification.

Q0: In the literature, when emails are used as a data source, what types of analyses techniques are performed on the data? Are the same techniques used over and over?

As a researcher, knowing the answer to the question "who is doing something similar to what I am doing?" is a basic preparatory procedure for conducting research. This study will provide a broad base from which to launch a more specific investigation. From a repository administrator point of view, the rationale for asking about commonly-used analysis procedures is because the process of analyzing emails may actually be able to be automated and added to the data repository itself. It is possible to build in an analysis and/or cleaning process into the data repository itself. Researchers could then download raw email data, cleaned email data, or cleaned and processed data.

Q1: In the literature, when emails are used as a data source, are the researchers typically interested in the CONTENT of the email or the HEADERS, and for what purposes are each of these being used?

Differences in analysis techniques between these two email parts are significant: very different skills and techniques are needed to analyze highly structured email headers and unstructured email content. Researchers interested in applying a particular analysis technique will want to find work that is similar to what they are proposing. Additionally, knowing the answer to the content/header split will allow FLOSS data collectors, such as the FLOSSmole or FLOSSMetrics repositories, to collect and store the emails in the correct format for researchers, and to correctly focus the cleaning and processing procedures on the emails themselves. For example, if researchers complain that they are constantly writing and re-writing routines to take care of email aliasing (e.g. as described in Appendix A reference 60 [A60] and elsewhere), it may be possible for a repository administrator to add this cleaning logic into one version of the downloadable repository data.

Q2: In the literature, when emails are used as a data source, how many projects or mailing lists are in the study? For the papers that use a small number of project email lists in their study, what are the most popular projects to study? Are the same projects being studied often?

Again, the optimal choice of research techniques may differ depending on the target size of the data being analyzed. Has anyone ever tried doing *X* technique on *Y* years worth of all the emails stored in *Z* forge? Did anyone get meaningful results from studying emails of just one project? What analyses were performed on this project, and could it be replicated? Note that this question is not attempting to place a value on a higher or lower number of projects or messages in a study. In other words, a study with more projects is not better or worse, it simply points to a method that can potentially be generalizable (extended to more than just a single mailing list). Additionally, knowing how many projects were involved in each study would help the FLOSS data repository administrators and data collectors focus their collection efforts on getting the most useful or popular projects. Knowing which methods have been applied to entire forges full of projects (hundreds or thousands of projects, or more) will help repository administrators understand which goals are realistic for storing and curating re-usable corpuses of FLOSS email.

Q3: Given a particular categorization (for example "single-project studies that use email headers to build a social network"), what specific papers are included in that group? And the opposite question: given a particular paper, how was it categorized?

Knowing the answer to this question will help researchers find similar papers, or to show results occurring from the different approaches between papers.

The next section describes a few of the technical background features of FLOSS email, its collection, and its cleaning.

BACKGROUND

At the most basic level, FLOSS is simply defined: it is software released under a free or open source license. However, largely because of the license requirements (that the source code is to be publicly available and that users should be able to make changes to it), in practice FLOSS is usually developed using tools that support frequent, transparent source code releases and fluid, decentralized, globally-distributed teams. The result is that in addition to the source code being publicly available for a project, there are also many other available artifacts of the FLOSS development process: public communication archives, web-based bug databases and task lists, online wikis and documentation, and the like. In this section, we describe email as an artifact, and discuss why it is critical to have a broad understanding of its use in research.

FLOSS Development Artifacts

Artifacts of the development process, usually source code, are a cornerstone data source for empirical software engineering research (Basili, *et al.*, 2007). By studying the source code of a project, we can attempt to answer a whole host of questions about the software,

for example about its quality (Hassan, 2009; Tarvo, 2009) team structure (de Souza, *et al.*, 2005), and maintenance procedures (Zimmerman, *et al.*, 2005), to name a few. Studying how source code changes over time can yield additional insights into the evolution of the project, e.g. (Nakakoji, *et al.,* 2002).

The additional artifacts available from FLOSS projects provide another valuable source of data to be mined for patterns. For example, researchers may be able to answer questions about a FLOSS project by studying the way its participants communicate, or by comparing multiple projects for communication patterns (Scacchi, 2010). If communication artifacts are publicly available, as they usually are in FLOSS projects, this makes them easy to study. (Though this ubiquity may not equate to quality, as Wright, *et al.* (2010) points out.) Common communication artifacts in FLOSS projects may include: messages sent to email mailing lists, IRC chat logs, Twitter streams, social media page updates, and the like (Robles, *et al.*, 2009).

How FLOSS Uses Email Artifacts

Email is the oldest and most-used of these communication techniques among the FLOSS projects. Typically, a FLOSS project will have at least one email-based electronic mailing list that helps to streamline communication between developers, users, or both. The centralized mailing list server software keeps track of who is subscribed to receive all messages sent to the list. Mailing lists can be either announcement-based (one-way communication, outbound only, sent to list subscribers) or discussion-based (back-and-forth communication among the subscribers). Some common uses for mailing lists in FLOSS projects include: technical discussions between developers, bug reporting by users or developers, announcements of new releases, asking questions by users, etc. If the list is public, as many FLOSS lists are, the mailing list messages will also be public. Most mailing list server software will provide some sort of ability to search or browse the old messages. This message archive can serve as a valuable tool for project participants both as a socio-technical resource (welcoming and educating new users, answering technical questions, giving insight into the community norms and practices), and as an "institutional memory" of what decisions have been made on the team, etc. (Feller & Fitzgerald, 2002).

Collecting the Email Archives

To use an email mailing list archive as the object of a research study, the researcher will typically begin with identifying the list(s) she is interested in, and then she will use some automated software to download and store all the messages that have been sent to that list. Depending on what mailing list server software is being used, the old messages may be available through a web interface, or as downloadable text files, or as emails from the mailing list server itself. There are also third-party services that scrape and store copies of email traffic sent to public FLOSS projects. Examples of these third party sites include Gmane

[1], MarkMail[2], and MARC[3].

Another location for retrieving mailing list archive data is at a software development forge, such as Sourceforge or Launchpad. If the FLOSS project is hosting its code on one of these forges, many times the forge will also provide mailing list services, which makes the archives for all projects on the forge (or some arbitrary subset) available for download (Squire & Williams, 2012).

A final option that has been less viable in the past but is growing in importance more recently is the use of a centralized FLOSS data repository to collect the emails on behalf of the researcher, and store the results for others to use. Examples of this type of open data collection and sharing infrastructure in the FLOSS community include FLOSSmole (Howison, *et al.*, 2006) and SRDA (Van Antwerp & Madey, 2008) both of which have some historical email data, and FLOSSMetrics (Herraiz, *et al.*, 2009). FLOSSMetrics in particular has done a good job of collecting email for its 2800 projects, and making the files and list statistics available to researchers on a project-by-project basis. The metadata on projects and source code are also available, making this a richer source of data than a list-only approach. Recall that part of the motivation for the research questions in this paper was to investigate how frequently-used or very interesting email data could be cleaned and stored in one of these repositories.

Cleaning the Email Archives

Once the messages have been collected, the researcher - depending on her research objectives - may need to clean these messages so that they are in a particular order, or may need to organize them so that they are associated with each other in a particular way. For example, the messages may need to be organized by thread or by date or by sender, depending on the target analysis the researcher wants to perform. Robles *et al.* (2009) review some tools that are available for this purpose.

Parts of an Email: Content

Email can be thought of as having two parts: the content and the headers. See Figure 1. The content is what people normally think of as the "body" of the email. It is written by the sender (either human or an automated email bot), and includes anything normally found in the body of the email, including text and all attachments. The email content can be encoded or expressed in different character sets and may include markup languages like HTML.

Parts of an Email: Headers

The email headers are defined as part of the email specification (Internet Engineering Task Force, 2008), and include several different fields, some required, some recommended, and some optional. An example of a required field is the From: field (the email address and optional name that sent the message), and the Date: field (telling when the email was sent). An example of a recommended field is Message-ID: which is a numeric identifier for the individual email message, and In-Reply-To: which tells which other identifier this message is replying to. All other fields commonly seen in email message headers are optional, including very common and useful fields like Subject: or Reply-To: or even To:.

When reading emails or analyzing them as an artifact of the software engineering process, it is important that conversation threads can be reconstructed accurately (Yeh & Harnly, 2006; Zawinski, 1997). A thread is a hierarchical grouping of messages according to topic, sending time, and reply history.

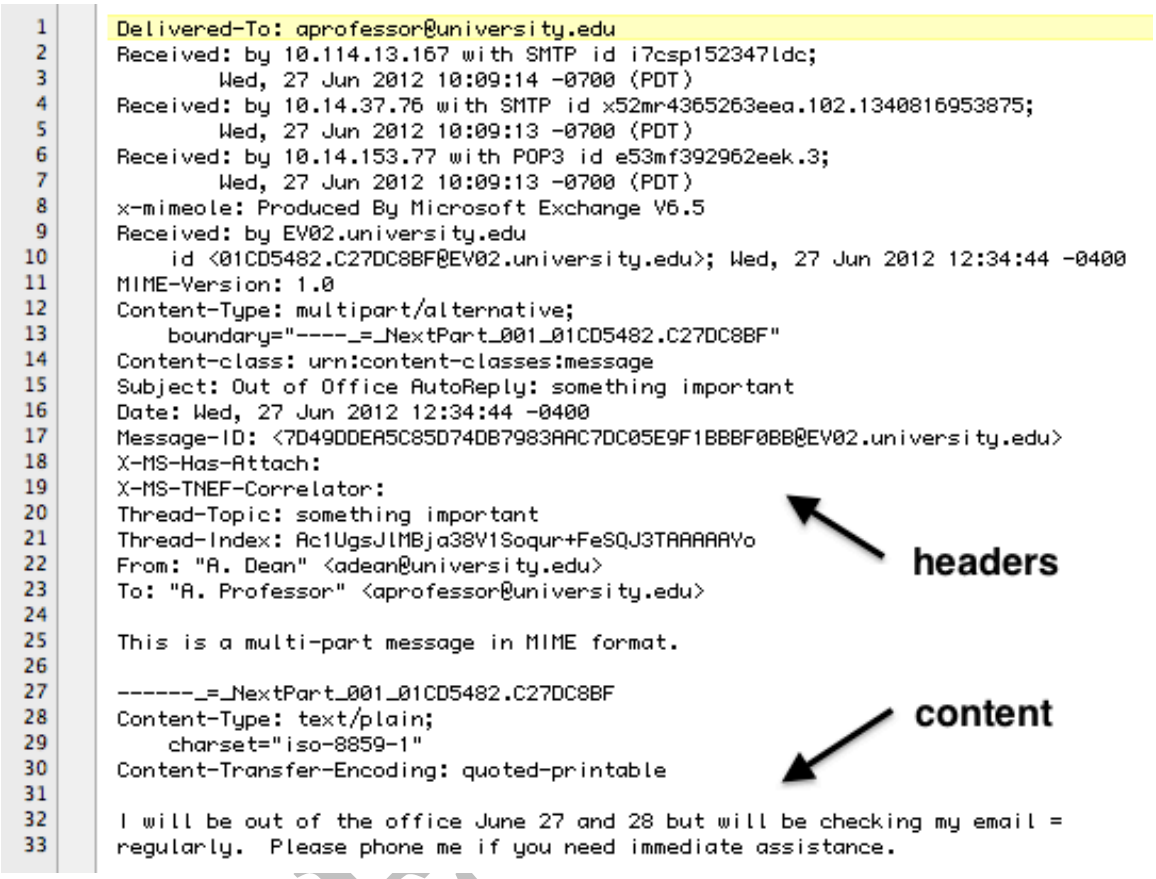

Figure 1. Email messages have two parts: headers and content

The next section outlines the structure for the review, including an explanation of how papers were selected, how the categories were designed, and how the papers were read and put into categories.

METHOD OF SURVEY

This section outlines the methodology used for this broad literature survey. The following list of resources was the starting point for seeking papers on our topic.

- *Empirical Software Engineering* journal (ESE),
- *Empirical Methods in Software Engineering* journal (EMSE),
- *Transactions on Software Engineering* journal (TSE),
- *Information and Software Technology* journal (IST),
- *International Journal of Open Source Software and Processes* (IJOSSP),
- Foundations of Software Engineering (FSE) conference,

- International Conference on Software Engineering (ICSE),
- Open Source Systems (OSS) conference,
- Mining Software Repositories (MSR) working conference,
- FLOSS workshops and tracks held at the Hawai'i International Conference on Systems Sciences (HiCSS)
- ACM digital library
- IEEE digital library

Using each of these sources, we sought papers using keywords and full-text search for "email", "mailing list" and similar variants. We then removed any paper that was not about FLOSS, or which was not about the FLOSS phenomenon in some way (papers that simply described an open source email-based tool, for example, were excluded). We further removed any paper that did not use email as a software development artifact (papers that just mentioned email in passing, for example).

We then sought additional papers by following the citations listed as references for this initial set. This activity uncovered an all-new set of papers that had been published in other venues not included in our original list, for example in special issues of more general disciplinary journals, and the like. We then performed the same task of following the citations on every new paper, until no new papers could be uncovered.

From this search task, 72 papers were identified that used email artifacts (specifically mailing list archives) as a data source for the FLOSS research.[4] Each of these papers was read in its entirety, with the intention of classifying them into groups based on data used (email), projects studied, and analysis technique used. Uncovering these classes will help to answer our four research questions introduced earlier.

(As we completed this review, we contributed the papers we found to the FLOSShub.org[5] paper repository for the general use of the FLOSS development community. While adding each paper to the repository, we included basic bibliographic information, abstracts, DOI links, and if available, preprints in PDF format for download.)

Next is a description of the categories the papers were divided into after reading them, and an explanation of how these categories help to answer each of our research questions.

Q0: In the literature, when emails are used as a data source, what types of analyses techniques are performed on the data? Are the same techniques used over and over?

To answer this question, it is necessary to read each paper in its entirety, with special attention paid to the data and methods sections. The papers can be divided into several categories representing common analysis types for the email portion of the paper (keeping in mind that in some cases, emails were used as confirmatory or secondary sources of data):

- Descriptive Statistics: The paper uses emails to calculate of basic metrics or descriptive statistics about an individual or a team (for example, counting messages per thread, using email headers to establish user identities or to geo-

locate developers, using timestamps to measure elapsed time between messages and follow-ups, etc.);
- Social Network Analysis: The paper uses emails to build a social network (for example, between developers who email each other);
- Non-Automated Content Analysis: The paper applies non-automated content analysis (coding/classification) to messages (for example, to read messages and classify developer actions into categories or timelines, or reading messages to discover how a project applies the concept of "peer review" in its operation);
- Automated Text Analysis: the paper applies automated text analysis techniques (for example, applying linguistic analysis of content in messages to make predictions about future developer behavior, or classifying the intent of a message into one of several categories based on the language used in the message);
- Linking: The paper describes how to automatically link content in emails to other artifacts (for example, using source code in an email discussion to link to file release or a bug report);
- Confirmatory/Secondary Analysis: The paper uses emails to confirm or supplement the data from another artifact or another analysis (for instance, the paper uses email headers to confirm developer identities found in a bug database).

Q1: In the literature, when emails are used as a data source, are the researchers typically interested in the CONTENT of the email or the HEADERS?

To answer this question, papers were divided into groups according to what part of the email message was the focus of the research:

- Message Content: the paper primarily studies the content of the email (word choice, sentiment, tone of discussion, sequence of events described therein, etc.);
- Message Headers: the paper primarily studies the headers of the email (senders, recipients, timestamps, etc.);
- Both: the paper uses both parts of the emails equally.

Q2: In the literature, when emails are used as a data source, how many projects or mailing lists are in the study? What are the most popular projects to study, and are the same projects being studied often?

To answer this question, each paper was read to find the total number of projects in the study, as well as the names of those projects. The following categories emerged:

- Single-project: the paper studies the email archive of a single FLOSS project;
- Small: the paper studies email archives from 2-15 projects;
- Medium-large: the paper studies email archives from 15+ projects (NOTE: no papers were found in this category);
- Forge-based: the paper uses email archives from an entire forge of FLOSS projects (which could be many hundreds or thousands of projects).

Again, the purpose of knowing the number of projects in the study is to give a full picture of the size and scope of the effort to future research teams who want to reproduce or

extend the work. It is also helpful to data repository administrators so that they know which techniques could be rolled out to multiple projects at once, versus which ones are more applicable to one or two projects. (The purpose is *not* to pass any statistical judgments based on the size of the study.)

Thus, with a matrix of classifications for these four questions in place, the next section details how the papers were selected, and which papers were put into which categories.

Q3: We end that section with an answer to Q3, which was "Given a particular categorization (for example "single-project studies that use email headers to build a social network"), what specific papers are included in that group? Given a particular paper, how was it categorized?"

RESULTS

From the FLOSShub repository, 72 papers were identified that used email artifacts as an object of study in the work. Each paper was read in its entirety, with special attention paid to the data and methods sections. This section provides a variety of visualizations for how the papers can be categorized.

Figure 2 uses a visualization type called a parallel set[6] to show how many papers can be grouped into the various categories. Based on the discussion given previously in the *Method* section above, the categories we used are as follows:

Q0 (bottom line in Figure 1): What method of analysis is primarily described in this paper? (From left to right, the categories are: descriptive statistics, social network analysis ("SNA"), non-automated content analysis, automated content analysis, confirmatory or secondary to another method, linking.)

Q1 (middle line in Figure 1): Does the method described in the paper primarily use email headers, email content, or both equally?

Q2 (top line in Figure 1): Does the paper study one project at a time ("Single"), a few projects ("Small"), or hundreds/thousands projects ("Forge")?

(Note that Q3 will not appear on the parallel set, as it requires us to list specific papers by number. Instead, we cover Q3 in Tables 2, 3, and 4 which appear later in this paper.)

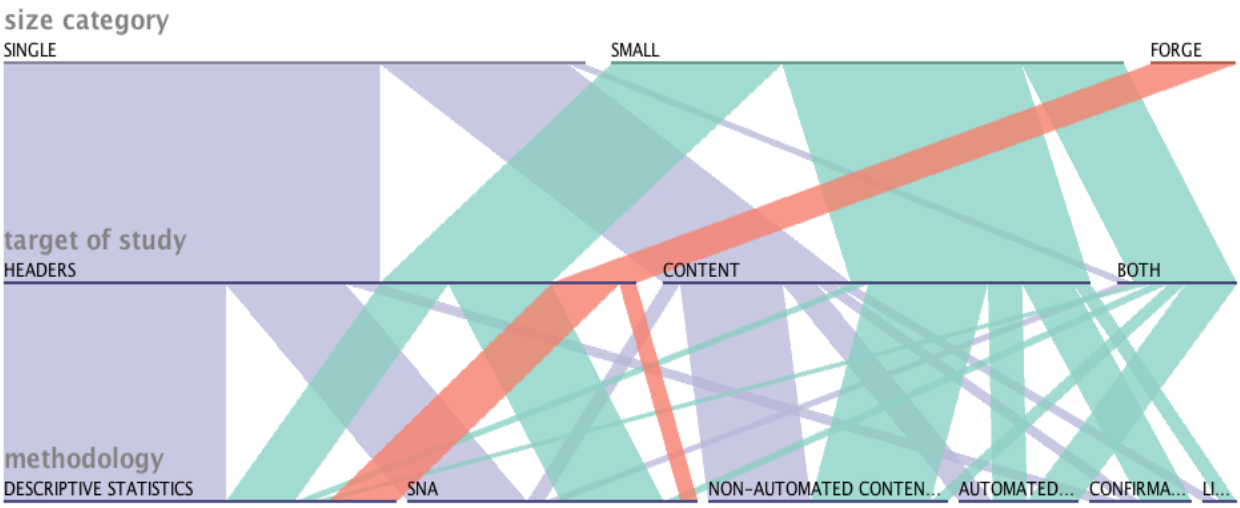

Figure 2: Papers in each category: size, target, and methodology

This parallel set visualization gives a high-level view of the collection – by category - and shows some patterns right away, a few of which are especially interesting:

- The most common type of FLOSS email paper is one that generates *descriptive statistics* using the *headers* of a *single* project.
- No *forge*-based studies (or very large studies) have been conducted using email *content*. With respect to Q1 and Q2, this is an important finding. We discover that when message content is the primary data source for the study, no papers used more than 15 projects. Larger studies (such as forge-based studies) are restricted to message header information, presumably because it is more easily digestible via automated methods (less "messy"), such as would be required with an entire forge of hundreds or thousands of projects.
- In terms of Q0 and Q1, we find that nearly all *SNA* papers are based on email *headers*. Similarly, nearly all papers generating *descriptive statistics* are also using *headers*.
- And in looking at Q0 and Q2, we see that only *descriptive statistics* and *SNA* papers have covered the entire range of *sizes* (single, small, and forge).
- Some categories are obviously mutually exclusive (*headers* alone are not used to perform *automated text analysis* or *non-automated content analysis*, for example) and the diagram reflects this.

Given these observations, a researcher can identify some under-developed areas ripe for new research, or some papers that would be fruitful for replication using desired techniques or data sets. Likewise, administrators of data repositories will know what email data sets are being used and how they are being used.

However, one sub-question from Q2 remains:

- What are the most popular projects to study, and are the same projects being studied often?

Answering this requires each paper to be read in order to discern which FLOSS project mailing lists were used. The following word cloud (Figure 3) and data table (Table 1) summarize the results:

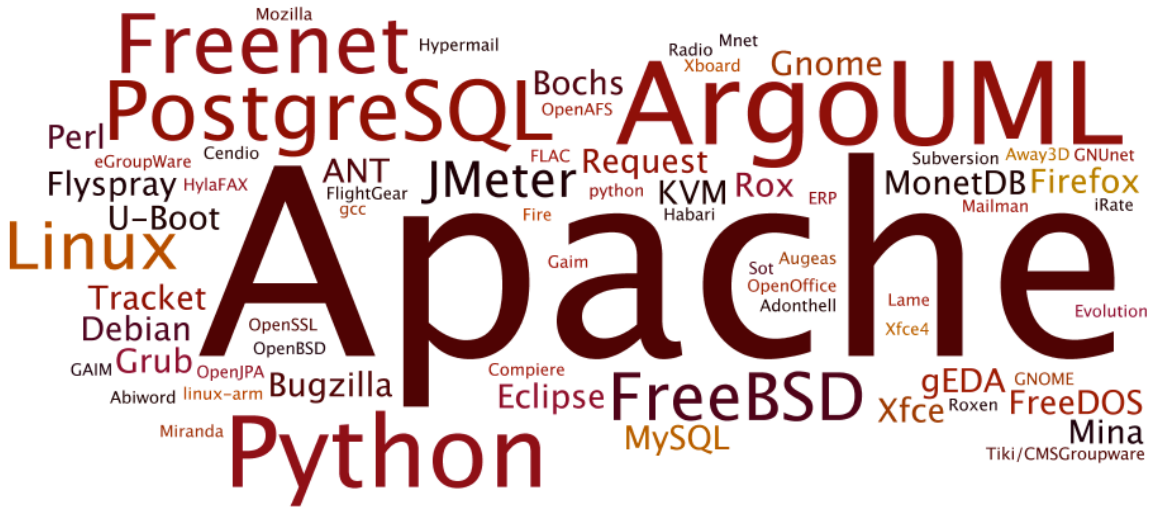

Figure 3: Most commonly studied projects in the FLOSS email literature

FLOSS Project	Count of Times It Was Studied
Apache	17
ArgoUML	7
Python	6
PostgreSQL	5
Freenet	5
Linux	4
FreeBSD	4
JMeter	3
KDE	3

Table 1: Projects that have been studied at least three times in the FLOSS email literature

We find that the most common emails to study are from Apache project (usually the http-server mailing lists). Likely factors leading to Apache's relative ubiquity on this list include its longevity, its popularity, its use by previous authors, its high levels of activity on its list, and the transparency within the list. We suspect the reason the numbers for ArgoUML and PostgreSQL are higher is because those projects were the topic of the 2006 Mining Software Repositories "Mining Challenge". The MSR Mining Challenge call for papers explains: "The MSR Mining Challenge brings together researchers and practitioners who are interested in applying, comparing, and challenging their mining tools and approaches on software repositories for two common open source projects: PostgreSQL and ArgoUML. The input data sources for the challenge comprise: source

code releases, source control data, bug data, mailing lists, execution traces, design and project documentation." The popularity of Python on this list can be partially explained by the fact that Barcellini, *et al.* published three papers on this project [A56-58].

Paper Categories and Summaries

Next we will list each paper and briefly summarize its contribution to the literature and how it was categorized. All papers are included in the Appendix A, and are numbered for reference, for example as [A1] or [A2] to differentiate them from the regular references for this paper. To organize this summary, the papers are first divided into "headers" or "content", and then by the size and methods used by the papers in the discussion.

Papers that focus on content

The rationale for papers that concentrate on using email for the content (rather than the headers) is usually that the researcher wishes to gain understanding about the complex processes of a FLOSS development team, for example about its decision-making, conflict resolution, or establishment of hierarchy and status within the team. Email is a primary vehicle for communication in FLOSS development teams, so it stands to reason that many of these processes would be explained or revealed in email archives. Much of the current research has thus focused on such a content-based strategy. Table 2 summarizes the categorizations for "content" papers in this study.

	Single-Project	Small (2-15 projects)
Automated Text Analysis	[A26] [A30]	[A64] [A65]
Non-Automated Content Analysis	[A10] [A34] [A39] [A56] [A57] [A58]	[A1] [A33] [A35] [A36] [A50] [A67] [A68]
Confirmatory/Secondary Source	[A7]	[A11] [A15] [A28]
Descriptive Statistics	-	[A27]
Linking	[A63]	[A62]
Social Network Analysis	[A14]	-

Table 2: Methodologies and sizes of FLOSS email papers in the "content" category

In looking at Table 2, it becomes clear that email content has most commonly been analyzed in a non-automated fashion, for example manually coding emails or by reading each message individually. By far the most common email archives studied in this way were those from the Apache development team. For example, Jensen, et al. [A1] read email archives to reveal how the developers of Apache and Netbeans made decisions regarding their choice of open source licenses. The paper is structured as case studies with the email archives serving as the paper trail for the team's decision to change licenses. O'Mahoney in [A50] was also concerned with licenses: for six unspecified projects, she uses content of email messages to learn what tactics open source project participants are using to protect their work and defend their licenses. Another paper that was oriented toward case study methodology was Dahlander [A35], which used the

content of mailing lists of four FLOSS projects (MySQL, Roxen, Cendio, SOT) in order to understand the relationship between companies (firms) and their related open source communities.

Freeman [A34] and von Krogh, et al. [A39] use similar techniques for OpenOffice and Freenet respectively. Freeman reads emails (and a few other sources) to understand developer motivations and reasons for contribution, while von Krogh et al. use the content of email to determine the way developers join the community. In [A39], emails are part of the data that the authors are using to create a "joining script" to describe what it is like to join a floss community.

Noll [A33] reviews content of emails for the Firefox and Gnome Metacity projects to determine the timeline of features proposed and implemented. Crowston, et al. in [A36] were concerned with developer task assignment mechanisms in use for three projects (Gaim, eGroupware, and Compiere ERP). Kuk [A10] reads messages to figure out what developers on the KDE project are talking about and how they are re-using code.

Some automated approaches have also been used in analyzing email content. In Rigby and Hassan [A26], the authors analyzed Apache archives looking for personality traits and emotional content in the highest committing Apache developers. They used a linguistic inquiry word count methodology to automatically discover developer personality traits. Similarly, Junior, et al. [A30] use neuro-linguistic theory on the contents of the Apache email messages to put developers into categories based on how they prefer to communicate. Bachelli, et al. in [A64] and [A65] devised a classifier for emails from four different projects (ArgoUML, JMeter, Mina, and Freenet). Their classifier uses five categories based on content of the messages.

In contrast, sometimes the researchers are using the email content as secondary evidence for some other data about the FLOSS developers themselves. In Roberts, et al. [A7] and Shah [A11] the emails are paired in a confirmatory fashion with survey or interview data to provide additional insight into developer behaviors. In the former, the Apache list is reviewed for evidence of the career advancement path and potential for developers (paired in a confirmatory fashion with survey data). In the latter, the messages from two unspecified FLOSS project mailing lists serve as background on the project before doing the interviews that were the bulk of this study. Sometimes the confirmation is for another quantitative source, for example bug databases or the like. Weisgerber et al. [A28] looks at developer emails from two projects (FLAC and OpenAFS) for existence of patches, and then compares these patches introduced in the email to those that made it into the code in CVS version control system. The goal is to determine whether patches introduced in email discussions actually make it into the release or not. Von Krogh et al. [A15] wished to understand the levels of knowledge re-use by developers on 15 different projects. They used email content to supplement a survey distributed to developers on these email lists.

Indeed, many of the studies contain such hybrid approaches, where email messages are just one data source among many. In Patterson, et al. [A27] the authors compile counts of when certain words are used in Apache and Python project emails. They explain that they "...study the frequency with which software entities (functions, methods, classes, etc) are

mentioned in the mail", and then this is compared to the source code to see if the discussions are actually reflected in the work. The subject of Bachelli, et al in [A62] and [A63] is explicitly to link project source code with email content. They explain that it is important to link these because "[t]hrough e-mails, developers discuss design decisions, ideas, known problems and bugs, etc. which are otherwise not to be found in the [software]."

Duchenaut, et al. [A14] also combined email with source code. In that study, the authors used Python developer emails, this time looking for progression and acculturation to the group, with the email content ultimately being compared to source code contributions in order to draw a social network for describing development team structure.

Papers that focus on headers

When message headers are used in the literature, which is often, there are a few header fields that are typically the most commonly used: the sender field and the timestamp field. By collecting the sender information, it is possible to assess the participation level of developers, to figure out where a developer is physically located geographically, or to create graphs of email senders and recipients (which results in a social network for a FLOSS project). And by collecting timestamps, it is possible to make some generalizations about the health or vigor of a community, or about the proximity of email posts to commits into a version control system. Table 3 summarizes the different types of analyses done on "header" papers in this study.

	Single-Project	Small (2-15 projects)	Forge-based (or very large)
Descriptive Statistics	[A2] [A4] [A8] [A12] [A13] [A22] [A23] [A42] [A44] [A45] [A46] [A48] [A51] [A55] [A61]	[A5] [A38] [A40] [A43] [A37] [A41] [A47] [A49] [A52] [A70]	[A18] [A19] [A20] [A24]
Social Network Analysis	[A6] [A16] [A21] [A25] [A31] [A32] [A66] [A71]	[A69]	[A17]

Table 3: Methodologies and sizes of FLOSS email papers in the "header" category

Mockus in [A40] and [A44] had some of the earliest papers that tried to assess levels of developer participation via message headers from an email archive. In these papers, they used email message headers from Apache (and in one case, Mozilla) in concert with CVS and Bugzilla data to quantify aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution intervals. Hann, et al. in [A42] and [A46] gathered email headers from Apache to be used as confirmatory evidence to a survey of Apache developers for why they participate in open source. Koch, et al. in [A48] also uses email sender lists, this time from the Gnome project, as a secondary source to determine the identify of developers with entries in the version control system (CVS). Similarly, German in [A51] uses emails and some CVS data to learn about how the Gnome Evolution project evolved.

Studer [A4] also uses mailing lists to supplement the main data source, which is commit histories. From the mailing lists, they counted messages sent by each developer. These counts serve as a secondary metric (in addition to commit histories) of a developer's "participation trajectory". Cerulo [A43] does something similar by using email headers along with data from a bug system to determine patterns of contribution by developers from two different but related communities of users (FreeBSD and OpenBSD). Dinh-Trong and Bieman [A55] used email headers to gather metrics about the communication patterns for bug reporting (number of people reporting bugs, and number of bugs reported by each) for the FreeBSD project. This work was a replication study based on [A40]. Jergensen, *et al.* [A61] were curious how developers move between layers of the commonly-used onion model. They attempted to craft a "joining script" for the Gnome project by using email headers in concert with bug tracking data.

Ye and Kishida [A8] describe a system that establishes a count of how many people sent different amounts of email to the Gimp project list, and compares this to different numbers of code contributions made by project participants. Similarly, in another early work, Lee and Cole [A13] calculate simple counts of developers participating on the Linux mailing list to try to determine participation levels. Koch in [A18] does something similar with an entire forge of projects, counting messages to produce confirmation about levels of developer participation and effort estimation.

Tsunoda in [A22] examines the work habits of PostgreSQL developers based on their email sending patterns. Weisgerber [A23] use message counts to see if refactoring of software results in additional email communication between ArgoUML developers, and whether the refactoring leads to additional bugs. And Schilling, et al. [A45] collects email archive metadata such as date of a developer’s first post and date of joining the mailing list to determine the familiarity a student-developer has with the KDE project before joining its Google Summer of Code team.

Sometimes the email headers are used to assess some measure of quality or health of the project. Sowe, et al. [A38] use email headers to count sender posting and replying activities, in order to determine some patterns of knowledge sharing between developers, and Sowe, et al. [A5] use sender data from emails to match to CVS commit data, analyzing the relationship between commits and email posts for 14 projects. In a similar vein, Gamalielsson, et al. [A2] collect email headers in order to measure the health of the community, which they define as the response time between an initial message and its replies.

There are a few papers that use the email headers to determine the identity of developers, or the geographic location of developers. [A19], [A20], and [A24] are a group of papers that attempt to discern the identity of developers who often use multiple email addresses, usernames, and other identifiers. They also describe how to glean geographic data about developer location based on the sending email address (for example, the top level domain and country code) and/or time zone setting for the message. In [A70] the authors construct a list of male names and female names, then use the message headers combined with this list of names to determine mailing list posting patterns for males and females on six FLOSS projects.

Building a social network of developers is the subject of numerous papers in FLOSS research generally, and we find that the use of email archives to supplement this work is common. The earliest example is Oh and Sangyong [A41] who use email message headers to build a social network of the Linux and Hypermail projects, which is then used to study membership dynamics and stability. Wagstrom, et al. [A47] do something similar for an unspecified Web browser project, Roberts, et al. [A32] work on building social networks from Apache mailing list data, and Valverde, et al. [A17] accomplish this same task for 120 projects on Sourceforge. Conaldi [A6] builds a social network of Gnome Epiphany using email sender data, Sowe, et al. [A37] do this for KDE, Debian-user, and Debian-mentor lists, while Nia, et al. [A49] look at the validity of the social networks generated for Apache, MySQL, and Perl projects.

Weiss, et al. [A31] uses email senders to show change in a community over time through Apache Agora-generated social networks. They are particularly interested in finding communication dyads (who emails whom). Oezbek, et al. [A69] use message headers, specifically the reply structure of messages, to build social networks for fifteen different open source projects. In [A66], Toral et al. use email senders to draw social networks in the Linux-ARM project, using these to find developers playing a "middle man" role. [A71] uses email headers to build a social network for Mozilla, specifically to understand what sort of quality assurance (QA) work is happening on the project, and who is involved.

Sometimes the social networks are combined with other project artifacts. In Bird, et al. [A21] and [A25], they use both source code and mailing lists to try to construct a developer social network for the PostgreSQL and Apache projects. Kidane and Gloor [A16] collect message senders to build a social network, and then compare this network to known code contribution data (bugs and enhancements) from Bugzilla database for the same project. Crowston, et al. [A52] use email headers in concert with bug tracker information to build social networks of developers, for the purpose of understanding centralization in project leadership.

Papers that focus on both content and headers

Table 4 summarizes which papers use both the content and headers from email messages.

	Single-Project	Small (2-15 projects)
Automated Text Analysis	-	[A9] [A29] [A54][A72]
Non-Automated Content Analysis	-	[A53]
Descriptive Statistics	-	[A59]
Social Network Analysis	[A3]	[A60]

Table 4: Methodologies and sizes of FLOSS email papers in the "content and header" category

Barcellini, et al. [A59] use a hybrid approach to understand the community structure of a project, specifically in terms of developer roles. In this work, they examine message text quoting patterns (how developers reply and quote one another in email). They use these patterns to determine the status/role and level of involvement (number of messages) of each conversation participant.

McLean, et al. [A3] constructed lists of topics from email messages on the Apache http-server mailing list, then constructed social network of senders to determine if the developers congregate around certain topics. Bird, et al [A60] also uses content and headers to build social networks for five projects. This work uses the social networks to find sub-communities within a larger development team.

Yamauchi, et al. [A53] used content analysis paired with header counts and sender counts to determine how FLOSS developers coordinate their tasks on the FreeBSD Newconfig team and the GCC team.

In [A9], Porcini, et al. use process mining techniques, similar to log file mining, to match disparate events or entities that occur in the same development artifact. In this case the development artifact is email, and the events they are trying to match are changes to a file, a bug report, and a mention of this change in an email conversation.

Ibrahim [A29] also takes a hybrid approach, using past emails to build a Naïve Bayesian classifier for how likely a developer will be to contribute to an email thread, based on past contributions to the source code. This work uses Apache, PostgreSQL, and Python.

Lanzara and Morner [A54], [A72] are also interested in what they call the "knowledge ecology" of FLOSS, but they concentrate on using email content and headers together to explain how the Apache and Linux development teams introduce and solve problems, or perform other tasks to create organizational knowledge. They both classify threads by topic, then use headers to find the developer names.

Papers that focus on Apache

One final interesting table helps to explicate Q2 (popular projects) a bit more. We noted previously in Table 1 that Apache was the most common project studied. Table 5 shows the classification of projects that study Apache and what techniques are most commonly used to do that.

	Content	Headers	Both
Descriptive Statistics	[A27]	[A12][A40] [A42][A44][A46]	-
Automated Text Analysis	[A26][A30]	-	[A29] [A54] [A72]
Non-Automated Content Analysis	[A1]	[A3]	[A68]
Confirmatory/Secondary	[A7]	-	-
Social Network Analysis	-	[A25][A31] [A32][A49]	[A60]

Table 5: Techniques and sizes for papers that use Apache

LIMITATIONS AND FUTURE WORK

There are two limitations of this literature survey and classification. The most significant possible limitation of this work is that it may have missed inclusion of some papers. This project started with a comprehensive search in a thorough listing of major FLOSS conferences and workshops, as well as the FLOSS-oriented journals, and software engineering conferences that focus on FLOSS or the mining of FLOSS software artifacts. We also used lists of citations to track related works. However, there is still the possibility that some papers may be missing.

The second main limitation to this survey is that because it focuses on FLOSS research (and how FLOSS email archives are used) there could be studies within the non-FLOSS literature that contain applicable techniques, but were not included because they are not really about FLOSS projects. It is our contention that the relative importance of email as a communication medium in the decentralized, geographically-dispersed world of FLOSS development, coupled with the required ease-of-access to FLOSS emails, makes FLOSS projects the appropriate and obvious place to start looking at how email archives can be used in software engineering research more generally. However, it seems reasonable that other, non-FLOSS papers may also use email archives in interesting and useful ways.

One other theme emerges for future work in this area. We are reminded that email is just one of many useful artifacts that are part of a typical FLOSS project footprint. Two other commonly-studied artifacts are bug reports and the actual project source code. These have both been studied extensively and with more than ten years of extant FLOSS research literature, both are ripe for surveying and classifying in the future.

CONCLUSIONS

We can draw several conclusions from this survey and classification of the literature on FLOSS email analysis.

First, with respect to Q0 and Q3 and the use of certain data analysis techniques, the most obvious outcome of this work is that it becomes plain that there are few papers using content analysis or textual analysis techniques (automated or not) on a large scale or on a forge-sized scale. Fifteen seems to be the highest number of projects studied in this way. However, based on the large number of papers that are looking at email content, expanding the ability to conduct content analyses on a large scale could be a very interesting avenue for research. Repositories of FLOSS data, such as FLOSSmole or FLOSSMetrics, could work on this problem. Alternatively, they could host data sets or analyses based on third-party (non-project-based) mailing list repositories such as Markmail.org.

Second, in terms of Q1 and Q3 and the structure of messages, we see that a very large number of papers used email header data exclusively, usually to gather descriptive statistics about the communication patterns in a team. Header data is theoretically quite

manageable as it is already labeled and mostly atomic. Yet upon closer reading we see that many of the papers were very detailed in describing how it was still necessary to clean the headers to prepare them for use (for example [A19],[A20],[A24],[A21]). So it seems clear that even though email headers could be considered less "messy" than email content, there is still quite a bit of work to be done to the raw headers to ready them for mining and analysis. Because the precise cleaning procedure may be dependent upon the choice of analysis method and on the source of the data, this literature review can help researchers decide on an appropriate model to follow, or can indicate where to find similar examples and guidance. In addition, repository administrators or data collectors can learn a few ways that the data should be cleaned before dumping it out for the researchers to use. FLOSS repositories, in addition to collecting the emails as they do now, could provide cleaned header fields for these emails according to the various methods already in use in the literature. This meshes nicely with the mission of repositories of data, which is to centralize and coordinate data collection efforts so that researchers do not have to reinvent the wheel each time they want to use a FLOSS artifact.

Third, in terms of Q2 and project popularity, this study reveals that Apache (particularly the *http-server* list) is far and away the most frequently studied project. Some of the papers studying Apache explain that their choice is due to some combination of the size, popularity or success (variously defined) of that project [A42], [A60]. In some other cases, a research group gets started with Apache and keeps using that list for several years (for example the same core group works on [A12], [A26], [A68]). In terms of replicability, this project would be an ideal candidate for long-term archiving and automation of repeated studies. Data repositories should prioritize storage of prime email archives like Apache.

REFERENCES

Basili, V.R., Zelkowitz, M.V., Sjøberg D.I., Johnson, P. & Cowling, A.J. (2007). Protocols in the use of empirical software engineering artifacts. *Empirical Softw. Eng. 12, 1 (February 2007)*, 107-119. DOI=10.1007/s10664-006-9030-4

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre Open Source Software Development: What we know and what we do not know. *ACM Computing Surveys*, 44(2).

de Souza, C., Froehlich, J., & Dourish, P. (2005). Seeking the source: Software source code as a social and technical artifact. In *GROUP '05: Proc of the 2005 Int. ACM SIGGROUP Conf. on Supp. Group Work*, 197–206.

Feller, J. & Fitzgerald, B. (2002). *Understanding Open Source Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Free Software Foundation. (n.d.) Free Software Foundation available licenses. Retrieved July 1, 2012, from http://www.gnu.org/licenses/

Hassan, A.E. (2009). Predicting faults using the complexity of code changes. (2009). In *Proc 31st Int. Conf. on Softw. Eng. (ICSE '09)*, 78–88.

Herraiz, I., Izquierdo-Cortazar, D., & Rivas-Hernandez, F. (2009). FLOSSMetrics: Free/Libre/Open Source Software Metrics. In *Proc. 2009 European Conf. on Software Maintenance and Reengineering (CSMR '09)*. 281-284. DOI=10.1109/CSMR.2009.43

Howison, J., Conklin, M., & Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *Int. J. of Info. Tech. and Web Eng*. 1, 3, 17–26.

Internet Engineering Task Force. (2008). RFC 5322. Retrieved July 1, 2012 from http://tools.ietf.org/html/rfc5322

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution patterns of open-source software systems and communities. In *IWPSE '02: Proc. of the Int. Wksp. on Prin. of Softw. Evolution*, 76–85.

Open Source Initiative. (n.d.) Licenses by category. Retrieved July 1, 2012, from http://www.opensource.org/licenses/category

Robles, G., Gonzalez-Barahona, J.M., Izquierdo-Cortazar, D., & Herraiz, I. (2009). Tools for the study of the usual data sources found in libre software projects. *Int. J. Open Source Softw. and Proc*, 1 (1), 24-45.

Scacchi, W. (2010). Collaboration practices and affordances in Free/open source software development. In I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead, (Eds.), *Collaborative Software Engineering*. Springer, New York, 307-328.

Squire, M. & Williams, D. (2012). Describing the software forge ecosystem. In *Proc. 45th Hawaii Int. Conf. on System Sciences (Maui, Hawaii, January 4-7, 2012)*. 3416-3425.

Tarvo, Alexander. (2009). Mining software history to improve software maintenance quality: A case study. *IEEE Software*, 26, 1, 34–40.

Van Antwerp, M. & Madey, G. (2008). Advances in the SourceForge Research Data Archive (SRDA). In *Proc. 3rd Workshop on Public Data about Software Development (WoPDaSD'08), (Milan, Italy, September 2008)*. 6 pages.

Wright, H., Kim, M., & Perry, D. (2010). Validity concerns in software engineering research. In *Proc. Future of Software Engineering Research (FoSER'10*). 411-414.

Yeh, J.-Y., & Harnly, A. (2006). Email Thread Reassembly Using Similarity Matching. In *Proc. 3rd Conference on Email and Anti-Spam* (July 27-28, 2006, Mountain View, California USA). 8 pages.

Zawinski, J. (1997). Message threading. Retrieved July 1, 2012 from http://www.jwz.org/doc/threading.html

Zimmermann, T., Zeller, A., Weissgerber, P. & Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Trans. on Softw. Eng*., 31,6, 429–445.

APPENDIX: Papers Reviewed

[A1] Jensen, C. & Scacchi, W. (2011). License update and migration processes in open source software projects. In *Proc. of the 7th Int. Conf. on Open Source Systems (OSS 2011)*. 177-195.

[A2] Gamalielsson, J., Lundell, B., & Lings B. (2010). Responsiveness as a measure for assessing the health of OSS ecosystems. In *Proc. of the 2nd International Workshop on Building Sustainable Open Source Communities (BSOSC at OSS 2010)*. 8 pages.

[A3] McLean, A.C., Pratt, L.J., Knutson, C.D., & Ringger E.K. (2011). Knowledge homogeneity and specialization in the Apache HTTP Server project. In *Proc. of the 7th Int. Conf. on Open Source Systems (OSS 2011)*. 106-122.

[A4] Studer, M., Muller, B. & Ritschard, G. (2007). Understanding the KDE social structure through mining of email archive. In *Proc. 2nd Workshop on Public Data about Software Development. (OSS 2007)*. 11 pages.

[A5] Sowe, S., Samoladas, I., Stamelos, I., & Angelis, L. (2008). Are FLOSS developers committing to CVS/SVN as much as they are talking in mailing lists? Challenges for Integrating data from Multiple Repositories. In *Proc. 3rd Workshop on Public Data about Software Development. (OSS 2008)*. 49-54.

[A6] Conaldi, G. (2009). Flat for the few, steep for the many: Structural cohesion as a measure of hierarchy in FLOSS communities. In *Proc. 4th Workshop on Public Data about Software Development. (OSS 2009)*. 3 pages.

[A7] Roberts, J.A., Hann, I-H., & Slaughter, S. (2006). Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects. *Management Science, 52(7)*. 984-999. DOI=10.1287/mnsc.1060.0554

[A8] Ye. Y. & Kishida, K. (2003). Toward an understanding of the motivation Open Source Software developers. In *Proc. 25th Int. Conf. on Software Engineering (ICSE 2003)*. 419-429.

[A9] Poncin, W., Serebrenik, A., & van den Brand, M. (2011). Process mining software repositories. In *Proc. 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*. 5-14.

[A10] Kuk, G. (2006). Strategic interaction and knowledge sharing in the KDE developer mailing list. *Management Science, 52(7)*. 1031-1042. DOI=10.1287/mnsc.1060.0551

[A11] Shah, S.K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science, 52(7)*. 1000-1014. DOI=10.1287/mnsc.1060.0553

[A12] Rigby, P.C., German, D.M., & Storey, M.-A. (2008). Open source software peer review practices: A case study of the Apache server. In *Proc. 30th Int. Conf. on Software Engineering (ICSE 2008)*. 541-550. DOI=10.1145/1368088.1368162

[A13] Lee, G.K. & Cole, R.E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science, 14(6)*. 633-649.

[A14] Duchenaut, N. (2005). Socialization in an Open Source software community: A socio-technical analysis. *Computer Supported Cooperative Work, 14(4)*. 323-368. DOI= 10.1007/s10606-005-9000-1

[A15] von Krogh, G., Spaeth, S., & Hafliger, S. (2006). Knowledge reuse in open source software: An exploratory study of 15 open source projects. In *Proc. 38th Annual Hawaii Int. Conf. on System Sciences (HICSS 38)*. 1-10.

[A16] Kidane, Y. & Gloor, P. (2007). Correlating temporal communication patterns of the Eclipse open source community with performance and creativity. *Computational and Mathematical Organizational Theory, 13(1)*. 17-27. DOI=10.1007/s10588-006-9006-3

[A17] Valverde, S., Theraulaz G., Gautrais J., Fourcassie V., & Sole R.V. (2006). Self-organization patterns in wasp and open source communities. *IEEE Intelligent Systems, 21(2)*. 36-40. DOI=10.1109/MIS.2006.34

[A18] Koch, S. (2008). Effort modeling and programmer participation in open source software projects. *Information Economics and Policy (Empirical Issues in Open Source Software), 20(4)*. 345-355. DOI=10.1016/j.infoecopol.2008.06.004

[A19] Gonzalez-Barahona, J.M., Robles, G., Andradas-Izquierdo R., & Ghosh, R.A. (2008). Geographic origin of libre software developers. *Information Economics and Policy (Empirical Issues in Open Source Software), 20(4)*. 356-353. DOI=10.1016/j.infoecopol.2008.07.001

[A20] Robles, G. & Gonzalez-Barahona, J.M. (2005). Developer identification methods for integrated data from various sources. In *Proc. 2005 Intl. Workshop on Mining Software Repositories*. 106-110. DOI=10.1145/1082983.1083162

[A21] Bird, C., Gourley, A., Devanbu, P., Gertz, M. & Swaminathan, A. (2006). Mining email social networks in Postgres. In *Proc. 2006 Intl. Workshop on Mining software repositories (MSR 2006)*. 185-186. DOI=10.1145/1137983.1138033

[A22] Tsunoda, M., Monden, A., Kakimoto, T., Kamei, Y., & Matsumoto, K. (2006). Analyzing OSS developers' working time using mailing lists archives. In *Proc. 2006 Intl. Workshop on Mining Software Repositories (MSR 2006)*. 181-182. DOI=10.1145/1137983.1138031

[A23] Weißgerber, P., Diehl, S., & Gorg, C. (2006). Mining refactorings in ARGOUML. In *Proc. 2006 Intl. Workshop on Mining Software Repositories (MSR 2006)*. 175-176. DOI=10.1145/1137983.1138028

[A24] Robles, G. & Gonzalez-Barahona, J.M. (2006). Geographic location of developers at SourceForge. In *Proc. 2006 Intl. Workshop on Mining Software Repositories (MSR 2006)*. 144-150. DOI=10.1145/1137983.1138017

[A25] Bird, C., Gourley, A., Devanbu, P., Gertz, M. & Swaminathan, A. (2006). Mining email social networks. In *Proc. 2006 Intl. Workshop on Mining Software Repositories (MSR 2006)*. 137-143. DOI=10.1145/1137983.1138016

[A26] Rigby, P.C. & Hassan, A.E. (2007). What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. In *Proc. 2007 Intl. Workshop on Mining Software Repositories (MSR 2007)*. 23-31. DOI=10.1109/MSR.2007.35

[A27] Pattison, D.S., Bird, C.A., & Devanbu, P.T. (2008). Talk and work: a preliminary report. In *Proc. 2008 Intl. Working Conference on Mining Software Repositories (MSR 2008)*. 113-116. DOI=10.1145/1370750.1370776

[A28] Weißgerber, P., Neu, D., & Diehl, S. (2008). Small patches get in!. In *Proc. 2008 Intl. Working Conference on Mining Software Repositories (MSR 2008)*. 67-76. DOI=10.1145/1370750.1370767

[A29] Ibrahim, W.M., Bettenburg N., Shihab E., Adams B., & Hassan A.E. (2010). Should I contribute to this discussion? In *Proc. 2010 Intl. Working Conference on Mining Software Repositories (MSR 2010)*. 181-190. DOI=10.1109/MSR.2010.5463345

[A30] Junior, M.C., Mendonca, M., Farias, M., & Henrique, P. (2010). OSS developers context-specific preferred representational systems: An initial neurolinguistic text analysis of the Apache mailing list. In *Proc. 2010 Intl. Working Conference on Mining Software Repositories (MSR 2010)*. 126-129.

[A31] Weiss, M., Moroiu, G., & Zhao, P. (2006). Evolution of open source communities. In *Proc. 2nd Int. Conf. on Open Source Systems (OSS 2006)*. 21-32. DOI=10.1007/0-387-34226-5_3

[A32] Roberts, J., Hann, I.-H., & Slaughter, S. (2006). Communication networks in an open source project. In *Proc. 2nd Int. Conf. on Open Source Systems (OSS 2006)*. 297-306. DOI=10.1007/0-387-34226-5_30

[A33] Noll, J. (2007). Innovation in open source software development: A tale of two features. In *Proc. 3rd Int. Conf. on Open Source Systems (OSS 2007)*. 109-120. DOI=10.1007/978-0-387-72486-7_9

[A34] Freeman, S. (2007). The material and social dynamics of motivation: Contributions to open source language technology development. *Science Studies, 20(2)*. 55-77.

[A35] Dahlander, L. & Magnusson M. (2008). How Do Firms Make Use of Open Source Communities? *Long Range Planning, 41(6)*. 629-649.

[A36] Crowston, K., Li, Q., Wei, K., Eseryel Y.U., & Howison J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology Journal, 49*. 564-575.

[A37] Sowe, S.K., Stamelos, I., & Angelis, L. (2006). Identifying knowledge brokers that yield software engineering knowledge in OSS projects. *Information and Software Technology, 48*. 1025-1033. DOI=10.1016/j.infsof.2005.12.019

[A38] Sowe, S.K., Stamelos, I., & Angelis, L. (2008). Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software, 81(3)*. 431-446. DOI=10.1016/j.jss.2007.03.086

[A39] von Krogh, G., Spaeth, S., & Lakhani, K. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy, 32(7)*. 1217-1241. DOI=10.1016/S0048-7333(03)00050-7

[A40] Mockus, A., Fielding, R., & Herbsleb, R.T. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11(3)*. 309-346. DOI=10.1145/567793.567795

[A41] Oh, W. & Sangyong, J. (2004). Membership dynamics and network stability in the open-source community: the ising perspective. In *Proc. Int. Conf. on Information Systems 2004*. 413-426.

[A42] Hann, I.-H., Roberts, J., Slaughter, S., & Fielding, R. (2002). Economic incentives for participating in open source software projects. In *Proc. Int. Conf. on Information Systems 2002*. 365-372.

[A43] Cerulo, L., Cimitile, M., Di Penta, M., & Canfora, G. (2011). Social interactions around cross-system bug fixings. In *Proc. 2011 Intl. Workshop on Mining Software Repositories (MSR 2011)*. 143-152. DOI=10.1145/1985441.1985463

[A44] Mockus, A., Fielding, R., & Herbsleb, J. (2000). A case study of open source software development: The Apache server. In *Proc. 21st Int. Conf. on Software Engineering (ICSE '00)*. 263-272. DOI=10.1109/ICSE.2000.870417

[A45] Schilling, A., Laumer, S., & Weitzel, T. (2012). Who will remain? An evaluation of actual person-job and person-team fit to predict developer retention in FLOSS projects. In *Proc. 45th Hawaii Int. Conf. on Systems Science. (HiCSS 45)* 3446-3455. DOI=10.1109/HICSS.2012.644

[A46] Hann, I.-H., Roberts, J., Slaughter, S., & Fielding, R. (2002). Why do developers contribute to open source projects? First evidence of economic incentives. In *Proc. 2nd International (ICSE) Workshop on Open Source*. 4 pages.

[A47] Wagstrom, P.A., Herbsleb, J., & Carley, K. (2005). A social network approach to free/open source software simulation. In *Proc. 1st Int. Conf. on Open Source Systems (OSS 2005)*. 16-23. DOI=10.1.1.178.4984

[A48] Koch, S. & Schneider, G. (2003). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal, 12(1)*. 27-42. DOI=10.1046/j.1365-2575.2002.00110.x

[A49] Nia, R., Bird C., Devanbu P., & Filkov V. (2010). Validity of network analyses in Open Source projects. In *Proc. 7th IEEE Working Conf. on Mining Software Repositories (MSR 2010)*. 201-209. DOI=10.1109/MSR.2010.5463342

[A50] O'Mahony, S. (2003). Guarding the commons: How community managed software projects protect their work. *Research Policy, 32(7)*. 1179-1198.

[A51] German, D. (2004). Mining CVS repositories, the softChange experience. In *Proc. Intl. Workshop on Mining Software Repositories (MSR 2004)*. 17-21.

[A52] Crowston, K., Wiggins, A., & Howison, J. (2010). Analyzing leadership dynamics in distributed group communication. In *Proc. 43rd Hawaii Int. Conf. on Systems Science (HiCSS 43)*. 10 pages. DOI=10.1109/HICSS.2010.62

[A53] Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. (2000). Collaboration with lean media: How open source software succeeds. In *Proc. Computer Supported Cooperative Work (CSCW '00)*. 329-338. DOI=10.1145/358916.359004

[A54] Lanzara, G.F. & Morner, M. (2005). The knowledge ecology of open-source software projects. *Information Systems Journal*. October 7. 44 pages.

[A55] Dinh-Trong, T.T. & Bieman, J.M. (2005). The FreeBSD Project: A replication case study of open source development. *IEEE Trans. Software Eng. 31(6)*. 481-494. DOI=10.1109/TSE.2005.73

[A56] Barcellini, F., Detienne, F., & Burkhardt, J.-M. (2009). Participation in online interaction spaces: Design-use mediation in an Open Source Software community. *International Journal of Industrial Ergonomics, 39*. 533-540. DOI=10.1016/j.ergon.2008.10.013

[A57] Barcellini, F., Detienne, F., & Burkhardt, J.-M. (2008). User and developer mediation in an Open Source Software community: Boundary spanning through cross-participation in online discussions. *International Journal of Human-Computer Studies, 66. 558-570*. DOI=10.1016/j.ijhcs.2007.10.008

[A58] Barcellini, F., Detienne, F., & Burkhardt, J.-M., Sack, W. (2008). A socio-cognitive analysis of online design discussions in an open source software community. *Interacting with Computers, 20*. 141-165. DOI=10.1016/j.intcom.2007.10.004

[A59] Bird, C. & Nagappan, N. (2012). Who? What? Where? Examining distributed development in two large open source projects. In *Proc. 2012 Working Conf. on Mining Software Repositories (MSR 2012)*. 237-246. DOI= 10.1109/MSR.2012.6224286

[A60] Bird, C., Pattison, D., & D'Souza, R. (2008). Latent social structure in open source projects. In *Proc. 16th ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*. 24-35.

[A61] Jergensen, C., Sarma, A., & Wagstrom, P. (2011). The onion patch: Migration in open source ecosystems. In *Proc. 19th ACM SIGSOFT symposium and the 13th*

European conference on Foundations of software engineering. 20-30. DOI=10.1145/2025113.2025127

[A62] Bacchelli, A., Lanza, M., & Robbes, R. (2010). Linking e-mails and source code artifacts. In *Proc. 32nd ACM/IEEE Int. Conf. on Software Engineering (ICSE 2010)*. 375-384. DOI=10.1145/1806799.1806855

[A63] Bacchelli, A., D'Ambros M., Lanza M., & Robbes R. (2009). Benchmarking Lightweight Techniques to Link E-Mails and Source Code. In *Proc. 16th Working Conference on Reverse Engineering (WCRE)*. 205-214. DOI=10.1109/WCRE.2009.44

[A64] Bacchelli, A., Dal Sasso, T., D'Ambros, M., & Lanza, M. (2012). Content classification of developer emails. In *Proc. 34th ACM/IEEE Int. Conf. on Software Engineering (ICSE 2012)*. 375-385.

[A65] Bacchelli, A., D'Ambros, M., & Lanza, M. (2010). Extracting source code from e-mails. In *Proc. 18th IEEE Int. Conf. on Program Comprehension (ICPC 2010)*. 24-33.

[A66] Toral, S.L., Martinez-Torres, M.R., & Barrero, F. (2010). Analysis of virtual communities supporting OSS projects using social network analysis. *Information and Software Technology, 52*. 296-303. DOI=10.1016/j.infsof.2009.10.007

[A67] Prechelt, L. & Oezbek, C. (2011). The search for a research method for studying OSS process innovation. *Empirical Software Engineering, 16(4)*. 514-537. DOI=10.1007/s10664-011-9160-1

[A68] Rigby, P. & Storey, M.-A. (2011). Understanding broadcast based peer review on open source software projects. In *Proc. 33rd Int. Conf. on Software Engineering (ICSE 2011)*. 541-550. DOI=10.1145/1985793.1985867

[A69] Oezbek, C., Prechelt L., & Thiel F. (2010). The onion has cancer: some social network analysis visualizations of open source project communication. In *Proc. 3rd Intl. Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS 2010)*. 5-10. DOI=10.1145/1833272.1833274

[A70] Kuechler, V., Gilbertson, C., & Jensen, C. (2012). Gender Differences in Early Free and Open Source Software Joining Process. In *Proc. 8th Int. Conf. on Open Source Systems (OSS 2012)*. 78-93.

[A71] Barham, A. (2012). The impact of formal QA practices on FLOSS communities - The case of Mozilla. In *Proc. 8th Int. Conf. on Open Source Systems (OSS 2012)*. 262-267.

[A72] Lanzara & Morner. (2004). Making and sharing knowledge at electronic crossroads: the evolutionary ecology of open source. In *Proc. 5th European Conference on Organizational Knowledge, Learning and Capabilities*. Innskbruck 2-3 April.

[1] http://gmane.org
[2] http://markmail.org

[3] http://marc.info
[4] All the papers are listed in Appendix A. In this paper, when we refer to one of these 72 items, we will use a parenthetical like [A36] to indicate which paper we are discussing. Papers that describe the development of tools to handle email were not included.
[5] http://flosshub.org
[6] Software for generating parallel sets from categorical data can be found at http://eagereyes.org/parallel-sets