Social Interactions around Cross-System Bug Fixings: The Case of FreeBSD and OpenBSD

Gerardo Canfora
Dept of Engineering-RCOST,
University of Sannio, Italy
canfora@unisannio.it

Luigi Cerulo
Dept. of Biological and
Environmental Studies,
University of Sannio, Italy
lcerulo@unisannio.it

Marta Cimitile
Fac. of Jurisprudence,
Unitelma Sapienza, Italy
marta.cimitile@unitelma.it

Massimiliano Di Penta
Dept of Engineering-RCOST,
University of Sannio, Italy
dipenta@unisannio.it

ABSTRACT

Cross-system bug fixing propagation is frequent among systems having similar characteristics, using a common framework, or, in general, systems with cloned source code fragments. While previous studies showed that clones tend to be properly maintained within a single system, very little is known about cross-system bug management.

This paper describes an approach to mine explicitly documented cross-system bug fixings, and to relate their occurrences to social characteristics of contributors discussing through the project mailing lists—e.g., degree, betweenness, and brokerage—as well as to the contributors' activity on source code.

The paper reports results of an empirical study carried out on FreeBSD and OpenBSD kernels. The study shows that the phenomenon of cross-system bug fixing between these two projects occurs often, despite the limited overlap of contributors. The study also shows that cross-system bug fixings mainly involve contributors with the highest degree, betweenness and brokerage level, as well as contributors that change the source code more than others.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Corrections*

General Terms

Human Factors

Keywords

Code Migration, Bug Fixing, Social Network Analysis, Empirical Study

MSR'11, May 21–22, 2011, Waikiki, Honolulu, HI, USA

1. INTRODUCTION

Source code cloning among related systems is a common practice, as shown by previous studies [7, 11]. This usually happens among systems belonging to the same application domain—e.g., operating systems or office automation suites.

A recognized problem of code cloning is the need for properly propagating changes across cloned software instances. Various studies show that, within a single system, developers are proficient in maintaining software clones [9, 10, 15]. While managing and propagating changes across similar/related artifacts seems to be successful *within* a single software system, the literature lacks of studies aimed at investigating how bugs, and related changes, are propagated *across* different systems. We call this phenomenon *Cross-System-Bug-Fixings* (CSBFs).

In this paper, we propose a first investigation of CSBFs between two related systems, namely the FreeBSD and OpenBSD kernels[1]. Specifically, the paper mines, from the versioning repository of the two systems, *documented* cases of CSBFs. It is a widespread practice, when performing a CSBF, to annotate in commit notes an explicit reference to the other system where the bug has been previously handled. For example, the following OpenBSD commit note on 1996/08/02 for the file *ip_icmp.h*:

> "*ICMP Router Discovery definitions; from FreeBSD*"

is related to the following commit note reported seven months earlier (1996/01/19) in FreeBSD for a file with the same name:

> "*Add definitions for ICMP router discovery. Reviewed by: wollman*"

Although we are aware that documented CSBFs can represent only a partial view of the phenomenon, in our experience mentioning other systems when fixing a bug is frequent, and FreeBSD/OpenBSD is not the only case we found. As a matter of fact, we found similar CSBF notes between OpenOffice and its Mac OS porting, NeoOffice. On 2008/08/-14, the following commit in NeoOffice adopts a patch submitted three days earlier to OpenOffice:

[1]In the following just referred as "FreeBSD" and "OpenBSD".

"*Use OpenOffice.org patch from the following URL to prevent regcomp from crashing on PowerPC*"

and on 2009/03/26 another NeoOffice commit fixes a bug with a newer version of a source file obtained from OpenOffice that included such fixes before:

"*Fix bug 3434 by using newer version of this OOo file obtained from OOo's svn*"

We experienced that such practices can be observed when parts of a software system are reused in another system by cloning. Those parts are not designed for reuse, for example they are not generalized modules, libraries or packages, but nevertheless they are included in the source-code base of another system, resembling partially what happens in the maintenance of software clones [10, 9, 15].

In this paper we aim at identifying CSBF activities between FreeBSD and OpenBSD, and at understanding the social role of developers performing such activities by means of social network analysis. Specifically, we are interested to understand the communication role played by committers involved in CSBFs, to help understanding how the information related to CSBF propagates between different projects. We rely on information stored in versioning database and mailing lists of both systems and develop methods to reconstruct and integrate different historical database.

In summary, the contributions of this paper are:

1. an approach to mine CSBFs from the commit notes of versioning systems;
2. a case study aimed at applying the proposed approach to FreeBSD and OpenBSD.

The study results indicate that the CSBF phenomenon is not negligible: we found over 430 cases among the commit notes of FreeBSD and 930 among those of OpenBSD, of which 59 (FreeBSD) and 161 (OpenBSD) could be linked to previous commits of the other project. Also, although the cross-system social network highlights a limited overlap of the two projects in terms of contributors, we found that people involved in CSBFs are those (i) having the highest *degree*, i.e., a high number of communication links, (ii) having the highest *betweenness*, i.e., acting as shortest points of transition for many communications, (iii) acting as *brokers* among the two projects, and (iv) performing a higher number of changes than other contributors.

The paper is organized as follows. Section 2 introduces the approach to mine CSBFs, and the approach to extract the cross-system social network. Section 3 describes the empirical study carried out on FreeBSD and OpenBSD. Section 4 reports and discusses results, while Section 5 discusses threats to validity. Finally, Section 6 discusses the related literature, while Section 7 concludes the paper and outlines directions for future work.

2. THE PROPOSED APPROACH

This section describes the proposed method for analyzing CSBFs in two systems, hereby referred as $system_1$ and $system_2$. In the following we refer to the following actors: (i) *committers*, i.e., people who have the right to commit changes in versioning systems, and (ii) *mailing list contributors*, i.e., people exchanging messages on mailing lists. At this stage, we did not explicitly capture project developers—often referred as "contributors"—different from committers, that can often be present is software projects. We plan to consider them in future work using for example the approach by Di Penta and German [6].

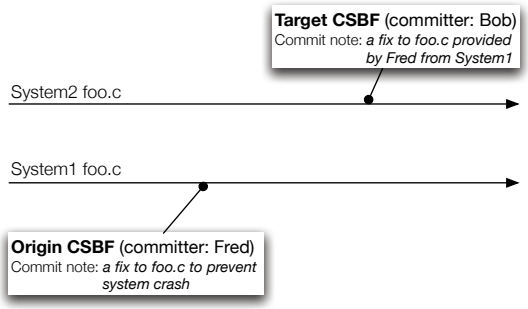


Figure 1: Identifying origins of target CSBFs.

2.1 Extracting and Relating CSBF Commits

First, we extract commit notes from the CVS[2] of the two systems, and for each commit we identify: (i) the file name being changed, (ii) the change date and time, (iii) the committer id, (iv) the number of lines added and removed, and (v) the commit note.

Then, we filter commit notes referring to the other system, e.g., commit notes of $system_1$ referring to $system_2$ and vice versa. To this aim we, preliminary, inspected a wide sample of commit notes to identify the textual patterns, and thus to limit false positives due to the accidental occurrence of the system name in the commit note. Examples of textual patterns we found in OpenBSD are: *"Inspired from FreeBSD"* or *"From FreeBSD"*. From these commit notes, we also extract emails or contributor names, again using regular expressions and heuristics similar to the ones previously applied for mining contributors from source code files [6].

After having identified, e.g., in $system_2$, commits inspired on changes occurred in $system_1$—hereby referred as *Target* CSBFs—we need to identify their origins, hereby referred as *Origin* CSBFs (see Figure 1). This was done using the following heuristics:

1. First, we consider the changes occurred, in $system_1$, on a file having the same name. Since the directory structure of the two systems could be different, we first perform a matching using the file name only, then, if the system contains multiple files with the same name, we use the longest common directory tree suffix between the file in $system_1$ and the file in $system_2$. We know from a previous study [7] that files with the same name in FreeBSD and OpenBSD contain a large percentage of cloned lines. However, it can happen that a file be renamed from one system to the other. In this study we do not handle such cases; we plan to deal with it in our future work by using an approach based on clone detection.
2. Then, we check that the two files have enough common code, by means of clone detection, using CCFinder [8],

[2]In this study the projects were under CVS; however, the approach works with other versioning systems as well.

and checking that the two source code files have at least 10% of cloned lines between each other. Of course we consider, for $system_1$, the file revision after the CSBFs has been performed.

3. Once identified the corresponding file in $system_1$, we take its history of changes occurred before the *Target* CSBF.
4. Among such changes, we consider the one having the highest textual similarity, and at least a textual similarity greater than or equal to a fixed threshold. The similarity is performed using the Information Retrieval *Vector Space Models* [2] as follows:
   - We extract terms from each commit notes, pruning out English stop words and performing a stemming using the Porter algorithm [13].
   - We index terms of each commit note using the term-frequency *tf* metric. We used the *tf* instead of the widely used *tf-idf* since our aim is to find the most similar commit notes, and not discriminating among a large corpus of document (case in which the *idf* would have been useful to penalize non-discriminant words occurring in many documents).
   - We compare the documents using the cosine similarity and prune pairs with similarity below the threshold. In our study we have empirically set the threshold to 0.20, by manually analyzing false positives and negatives obtained with different thresholds.
5. If there are more commit notes with the same similarity, we favor, in order of priority: (i) the change in $system_1$ for which the committer is referred in the commit note of *Target* CSBF, (ii) the change in $system_1$ that mentions at least one name or email mentioned in the commit note of *Target* CSBF, (iii) if none of the above apply, the most recent change.

2.2 Extracting Social Network from Mailing Lists

To analyze the social relations of developers involved in CSBFs, we could mine either bug tracking or mailing lists repositories. In our study, we consider the latter, because FreeBSD and OpenBSD adopt mailing lists instead of bug tracking systems. Specifically, we consider only mailing list archives related to bug fixings (since we are not interested to analyze other mailing lists, e.g., those related to discussing new features to be implemented). Given two mailing list contributors A and B we assume a social interaction $A \rightarrow B$ if the mailing list contributor A sends at least one email to B through the mailing list. Mailing list contributors and their social interactions are extracted from mailing list messages by identifying the sender name and email address, and whenever available the recipient/reply-to name and email address. A preprocessing step resolves inconsistencies and cases of people referring himself differently, e.g., *John Fitzgerald Smith*, *John F. Smith*, *John Smith*, or *J. Smith*. Specifically, we used an approach similar to the one used by Bird *et al.* [3] (although in our case we realized it was not necessary to use string distance on names/emails, as it would have produced too many false positives). The approach is composed of the following steps:

1. **Remove cases and special characters:** names are converted into lower cases, and special characters, including dots ".", are removed, e.g., *John F. Smith* becomes *john f smith*.
2. **Ignore middle names:** if the name is composed of more than two terms, and there are two names matching the first and last term, then we assign them to the same person, e.g., *john f Smith* and *john smith* are considered to be the same person, unless there is another person with the same first and last name, but with a different middle name, e.g., *john paul smith*.
3. **First name referred with initials only:** let us consider a name composed of at least two terms, of which the first one is longer than one letter, e.g., *john smith*. We search for cases where the last word is the same and the first word is the initial of the first word, for example *j smith*. If this is found, the two names are considered to be the same person, unless there is another person with the same last name and a first name, longer than one character, and starting with the same letter, e.g., *jackson smith*.
4. **Last name only:** if the name is composed of one term only, and it corresponds to the last name of one, and only one other, person, the two names are considered to be the same person.
5. **Initials only:** if there is a name composed of terms of one letter only (e.g., *j f k*), and there is one, and only one, name with the same initials (e.g., *john fitzgerald kennedy*) then we assume that two names belong to the same person. This rule is also applied if there is a name composed of one term only (e.g., *jfk*), and these letters correspond to the initials of one and only person.
6. **User ID-like name:** this is a similar, but slightly more complex case than the previous one. Sometimes people refer themselves with IDs composed of concatenated first and last names, or of first (and sometimes middle name) initials concatenated to the last name. For example, *john f. smith* could be referred as *johnsmith*, *jsmith*, or *jfsmith*. In this case we check if the ID could be composed by concatenating names of another person, or her initials and last name.

To deal with cases where the email sender/recipient contains an email address only, without specifying any name we use a set of heuristics, mostly derived from the above ones, to associate emails to names:

1. **Previously occurred email address:** if there are other emails where the same email address occurs together with a name, then the email is linked to that name. For example, if the recipient is *dipenta@unisannio.it* and there is another email containing, *Massimiliano Di Penta <dipenta@unisannio.it>*, then the email belongs to *Massimiliano Di Penta*.
2. **Extract name:** first, we extract the name from the email address, i.e., anything preceding the "@" and split it into terms considering special characters (e.g., ".") as separators.

3. **Link email address to a name:** if the heuristic "Previously occurred email address" does not apply, we try to map the name extracted from the email to full names occurred in other emails. For example, *jsmith@google.com* is mapped to *John Smith*, even if he was previously associated with a completely different email address.

4. **Link multiple email addresses of the same person:** we map multiple email addresses applying—on the name extracted from the email address—the same heuristics defined for names, e.g., *massimiliano.di.penta @unisannio.it*, *dipenta@unisannio.it* and *m.di.penta@ unisannio.it* belong to the same person. In our study, we applied the same heuristic even when the email suffix is different i.e., *dipenta@unisannio.it* and *dipenta@ gmail.com* belong to the same person, although we are aware that with larger mailing lists this could lead to many false positives.

Overall, it is worthwhile to point out that the adopted approach for unifying names and email is a conservative one, i.e., it performs an unification only when there are no multiple (ambiguous) possibilities of unification for the same name.

2.3 Unifying Cross-System Mailing List Contributors and Committers

We link mailing lists (or bug tracking system) contributors with committers, and person names and emails mentioned in the commit notes. The process is similar to the above mentioned process to unify names and emails in mailing lists.

For committers, we use the same heuristics adopted for person names, primarily the "User ID-like name" or the "Initials only" heuristics, as CVS IDs are short strings referring to initials only or first/last name initials, plus last name. Sometimes committer IDs could be email addresses—e.g., Mozilla, not considered in this study, uses email addresses where "@" is replaced by "%", thus heuristics for email addresses can be applied as well.

3. CASE STUDY: FREEBSD AND OPENBSD

The *goal* of this study is to analyze the phenomenon of CSBFs, with the *purpose* of understanding its relevance with respect to the social characteristics of developers involved such kind of changes. The context consists of changes—recorded in CVS repositories—and mailing lists archives of two open source operating system (OSs) kernels, FreeBSD[3] and OpenBSD[4]. Both OSs derive from work originated at the University of Berkeley with the aim of developing a free, open source Unix OS. Despite the common origin, the two systems evolved independently, in terms of internal characteristics, user applications and interfaces[5]. Table 1 reports a summary of the main characteristics of the two systems, i.e., the time interval analyzed, the systems size range in terms of KLOC and source code files, the number of commits and change sets (computed using the approach by Zimmermann *et al.* [17]) extracted from CVS, and the number of mailing list messages exchanged. It is worthwhile to mention that both systems have several mailing lists aiming at managing different kinds of discussions. In this context, we used the mailing lists related to bugs reports only (*freebsd-bugs* and *OpenBSD Bug reports*).

[3]http://www.freebsd.org
[4]http://www.openbsd.org
[5]http://www.freebsd.org.in/2010/03/29/comparison-of-freebsd-and-openbsd

Table 1: System history characteristics

	FreeBSD	OpenBSD
Time range	1993–2009	1998–2009
KLOC (min–max)	76–3,298	1,268–2,298
# of Source files (min–max)	211–6,797	4,434–6,440
# of Commits	119,259	70,895
# of Change sets	45,979	27,939
# of Mailing list messages	66,352	22,518

3.1 Research Questions

The research questions this study aims at addressing are the following:

- **RQ1:** *How do the source code committers and contributors of the two systems overlap?* This is a preliminary research question to the study, and aims at understanding the context where CSBFs occur. In fact, we want to understand to what extent committers mailing list contributors overlap between the two projects.

- **RQ2:** *How frequent is the phenomenon of CSBFs?* This research question aims at providing an idea of how relevant is the phenomenon of CSBFs, and whether it mainly occurs in one direction, e.g., from FreeBSD to OpenBSD, or in both directions.

- **RQ3:** *Who are the contributors involved in CSBFs?* This research question aims at characterizing the social characteristic of developers involved in CSBFs, i.e., how they interact, through mailing lists, with other contributors.

- **RQ4:** *Are mailing list contributors involved in CSBFs more active than others?* While **RQ3** aims at characterizing developers involved in CSBFs from a "social" point-of-view, this research question aims at understanding how active they are, based on the number of change commits performed.

3.2 Variable Selection and Analysis Method

This section describes the variables measured to address the research questions stated in Section 3.1.

For **RQ1** we describe how contributors to versioning system (committers) and of mailing list archives overlap within a system and across the two systems considered.

For **RQ2**, we report, for both projects, the number and percentage of commits related to (explicit) CSBFs, and compare them.

For **RQ3** we analyze different social network metrics, extracted from the joint network of committers and contributors of both projects. In particular, we are interested in the following metrics (we refer developers as "actors" using the standard social network analysis terminology [14]):

- *degree:* this is the most obvious measure of the importance of an actor, and it is defined as the number of connections a node has. Actors with a high degree have a higher potential of being influential than those with a lower degree. Specializations of the degree are

the *in-degree* and *out-degree*, indicating the number of incoming and outgoing edges respectively. They distinguish whether an actor receives (in-degree) or provide (out-degree) information from/to a wide range of other actors.

- *betweenness:* this is a generalized concept of "centrality" for an actor, and it is defined as the percentage of all geodesic (shortest) paths from neighbor to neighbor that pass through the actor. Betweenness is computed by determining, for each pair of actors (a_1, a_2), the fraction of shortest paths between a_1 and a_2 that pass through the actor in question, and by summing such percentages over all pairs.

- *brokerage metrics:* these metrics characterize more specifically actors involved in transferring information between nodes belonging to different clusters of the network. In our case, clusters are represented by the mailing list discussions of FreeBSD and OpenBSD. Specifically, in this context we consider the following brokerage metrics:

  - *coordinator:* an actor is coordinator if s/he lies in between nodes within the same cluster;
  - *gatekeeper:* an actor is a gatekeeper if s/he lies in the path between nodes of other clusters and nodes of the same cluster. In other words, the actor acts as a filter/dispatcher for the incoming information;
  - *representative:* it is the dual of the gatekeeper, i.e., the actor lies in the path between nodes of the same cluster and nodes of other clusters. In other words, the actor acts as a proxy for the outgoing communication.

  It is important to mention that brokerages levels are not Boolean, they rather indicate to what extent an actor is in a path between other nodes. For example, a node with high gatekeeping level may not be directly connected to nodes of the other clusters, but may be in a path traversed by many incoming communications. Specifically, the brokerage score for a given actor is computed as the number of ordered pairs of actors, including that actor, in which each actor plays a the role requested by the brokerage metrics (i.e., coordinator, gatekeeper, representative).

Further information about social network analysis can be found in related books such as the one by Scott [14]. To address **RQ3**, we compare the above described metrics for developers participating or not to CSBFs, using the unpaired Mann-Whitney test and the Cohen d effect size measure [5].

For **RQ4** we compare, using Mann Whitney unpaired test and Cohen d effect size, the number of changes performed and the number of lines added/removed (i) by committers involved in CSBFs and (ii) by other committers.

All analyses have been performed using the statistical environment R[6], and in particular the packages *igraph* and *sna* for social network analysis.

[6]www.r-project.org

Table 2: FreeBSD and OpenBSD committers, mailing list contributors, and their intersection.

	FreeBSD	OpenBSD	Both
# of committers	383	211	26
# of mailing list contribs	8,035	3,843	359
# of committers ∩ mailing list contribs	213	122	17

4. RESULTS

This section reports results of our empirical study to answer the research questions formulated in Section 3.1. Data for verification/replication are available on-line[7].

4.1 RQ1: How do the Source Code Committers and Contributors of the Two Systems Overlap?

Table 2 reports the number of committers and mailing list contributors for the two projects, as well as their intersection, i.e., the set of committers that are also mailing list contributors. As expected, the number of mailing list contributors is much greater than the number of committers. This is because mailing list contributors may be end-users, testers, or people who contribute code to the project but do not have commit permissions. The table also reports the number of committers who could be mapped on mailing list names, and those who are committers/mailing list contributors for both systems. 56% and 39% of FreeBSD and OpenBSD committers could be mapped on mailing list names. Instead, the set of committers and mailing list contributors that work on both projects is small: only 26 (out of 383 for FreeBSD and 211 for OpenBSD) committers work on both projects, and mailing lists have 359 common contributors, out of 8035 and 3843 in total for the two projects. Finally, there are 17 persons who are committers and mailing list contributors for both projects.

To understand how committers participated to the mailing list discussion, we built a network of mailing list communications that was a subset of the overall cross-project network, consisting of all messages for which at least one of the peers was a project committer. We did this because we wanted to focus our analysis on committers that, as shown by Bird *et al.* [3], exhibit higher values for social network properties than other mailing list contributors. Table 3 reports characteristics of this network, i.e., (i) the number of nodes pertaining to committers and non-committers, (ii) and the number of communication links between committers themselves and between non committers and committers.

In summary, as far as **RQ1** is concerned, we can conclude that *the two projects have less than 10% of committers, mailing list contributors, and committers ∩ contributors.* This suggests that, although the two projects have a common origin, they really have a different development team.

4.2 RQ2: How Frequent is the Phenomenon of CSBFs?

Table 4 reports data related to the phenomenon of CSBFs. The table reports, for both systems, the number of commits referring the other system, as well as the number of change sets. We found a total of 439 and 933 commits and a total

[7]http://www.rcost.unisannio.it/mdipenta/msr2011-rawdata.tgz

Table 3: Characteristics of the contributors' subset of the cross-system mailing list network.

Nodes	
# of committers	318
# of non-committers	1,272
Total	1,590
Links	
# of committer ↔ committer	168
# of committer → non-committer	1,417
# of non-committer → committer	1,002
Total	2,587

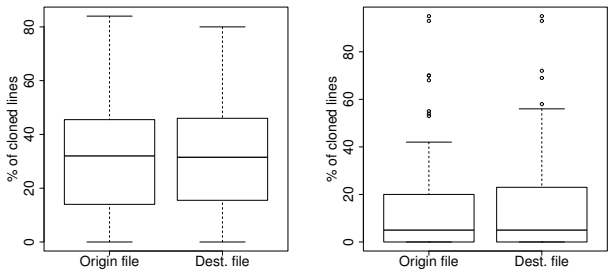
Table 4: Characteristics of the CSBF phenomenon.

	FreeBSD	OpenBSD
# of referring commits	439	933
# of referring change sets	200	476
# of referring commits between cloned files	133	296
# of linked commits (total)	59	161
# of linked commits (.c files)	37	93
# of linked commits (.h files)	22	68
# of linked change sets	46	120
# of CSBF committers	71	52
# of CSBF committers ∩ mailing list contribs	31	30
# of CSBF referred mailing list contribs.	36	59

of 200 and 476 change sets referring the other system for FreeBSD and OpenBSD respectively. Thus, in general, we can tell that the phenomenon is, at least, not negligible. This is not surprising: if, on the one hand the two projects had a common origin, on the other hand, their evolution has been independent, as explained in Section 3. For all referring changes, we were able to find a cloned file (with the same name) in the other system. In 133 cases (where the *Target* CSBF was in FreeBSD) and 296 cases (where the *Target* CSBF was in OpenBSD) corresponding files had the same name and at least 10% of cloned code lines.

Then, the table reports, out of the found references between cloned files, how many commits of each system we were able to link to a commit of the other system using the heuristics described in Section 2.1. We were able to link 59 commits for FreeBSD and 161 commits for OpenBSD, which are almost equally partitioned among header (*.h*) and C source files. This number might appear small if compared with the initial set of referring commits. However, it must be clear that, although file revisions in the two systems are similar, this does not guarantee that both files underwent a common change. In all cases where no link was found, it could be the case that: (i) developers simply used a completely different description for the commit note, although the adopted similarity threshold, 20%, allows to match commit notes that were relatively different, or (ii) the related change could origin from a file having a different name.

The linked commits involved 71 FreeBSD committers and 62 OpenBSD committers. Out of them, 31 and 30 were also involved in the mailing list discussions. Actually, the intersection between the FreeBSD CSBF committers/contributors and OpenBSD CSBF committers/contributors is one person (*peter@freebsd.org*), who was actually involved in CSBF for OpenBSD and participated to mailing lists of both projects.



(a) C Files (b) H Files

Figure 2: Percentage of cloned lines in C and H files involved in CSBFs.

Finally, 36 FreeBSD and 59 OpenBSD mailing list contributors were referred in the commits.

Figure 2 shows, for header (*.h*) and C files, the distribution of cloned line percentages between *Origin* CSBF and *Target* CSBF source files. It can be noticed that (i) the percentage of cloned lines is, on average, similar between origin and target files (the two files were not much different in terms of size), (ii) the percentage is significantly smaller (p-value < 0.001) for header files (median 5%, mean 15%) than for C files (median=32%, mean=33%). While the CSBFs for C source files involved the cloning of entire portions of functions (if not entire functions), for header files the percentage of cloned lines if lower as they often contain OS/specific declarations: for example FreeBSD headers often contain the preprocessor conditional:

```
#if  defined(__FreeBSD__).
```

Table 5 reports some examples of commit notes pairs for CSBFs. In all cases they refer the other system, and in some cases the person who provided the suggestion, e.g., *"suggested by peter@freebsd.org"* in the second example. While in some cases the time interval between the two changes is a few months (or as for the fifth one, within the same day, and within a week as for the sixth one), there are cases (third and fourth) where the related change occurs after years. In some cases, the commit notes (almost) match, while in other cases (second and third) they contain some differences, i.e., they also describe some changes not performed in the other system.

In summary, as far as **RQ2** is concerned, we can conclude that *the two projects contain a non-negligible number of CSBFs, although only about 10% of the target changes is explicitly traceable to the source change based on the content of commit notes.*

4.3 RQ3: Who are the Contributors Involved in CSBFs?

Figure 3 shows an excerpt of the committers/contributors network described in **RQ1** and whose characteristics are reported in Table 3. For the sake of understandability, the figure shows: (i) all committers involved in CSBFs (shown as big circles, blue for OpenBSD, and red for FreeBSD), (ii) all contributors participating to both mailing lists (shown as yellow pentagons), and (iii) a random sample of almost 300 committers/contributors directly interacting with (i) and

Table 5: Examples of linked CSBFs.

Date	Project	Committer	Note
1999/04/08	FreeBSD	wpaul	Make ASIX driver work on FreeBSD/alpha add to GENERIC.
1999/08/14	OpenBSD	aaron	Driver for ASIX88140A/88141 Ethernet; from FreeBSD
2001/02/15	FreeBSD	jlemon	Extend kqueue down to the device layer. Backwards compatible approach suggested by: peter
2001/03/01	OpenBSD	provos	port kqueue changes from freebsd plus all required openbsd glue. okay deraadt@ millert@ from jlemon@freebsd.org: extend kqueue down to the device layer backwards compatible approach suggested by peter@freebsd.org
2002/04/01	FreeBSD	joe	Merge from NetBSD: usb_port.h (1.33) usbdi_util.c (1.32) usbdi_util.h (1.22): date: 2000/06/01 14:37:51; author: augustss; Improve some portability items.
2004/07/21	OpenBSD	dlg	from freebsd ugen.c 1.68 usbdi_util.c 1.27 usbdi_util.h 1.15 log message: Implement outgoing interrupt pipes. It is part of the USB 1.1 spec. The Lego Infrared Tower use it. ok deraadt@
2003/12/26	OpenBSD	markus	use 1/2 space for rijndael context in ipsec - rijndael_set_key_enc_only() sets up context for encryption only - rijndael_set_key() always sets up full context - rijndaelKeySetupDec() gets back original protoype - uvm: use _enc_only() interface with hshoexer@ ok deraadt@
2005/03/11	FreeBSD	ume	use 1/2 space for rijndael context in ipsec - rijndael_set_key() always sets up full context - rijndaelKeySetupDec() gets back original protoype Reviewed by: sam Obtained from: OpenBSD
2004/03/17	FreeBSD	cperciva	Adjust the number of processes waiting on a semaphore properly if we're woken up in the middle of sleeping. PR: misc/64347 Reviewed by: tjr MFC after: 7 days
2004/03/17	OpenBSD	millert	Adjust the number of processes waiting on a semaphore properly if we're woken up in the middle of sleeping; cperciva@freebsd.org. OK deraadt@
2008/03/28	FreeBSD	rpaulo	Add Qualcomm ZTE CMDMA MSM modem to the list of supported modems. MFC after: 1 week
2008/04/02	OpenBSD	fkr	attach the ZTE CMDMA MSM modem from qualcomm. from freebsd ok jsg@

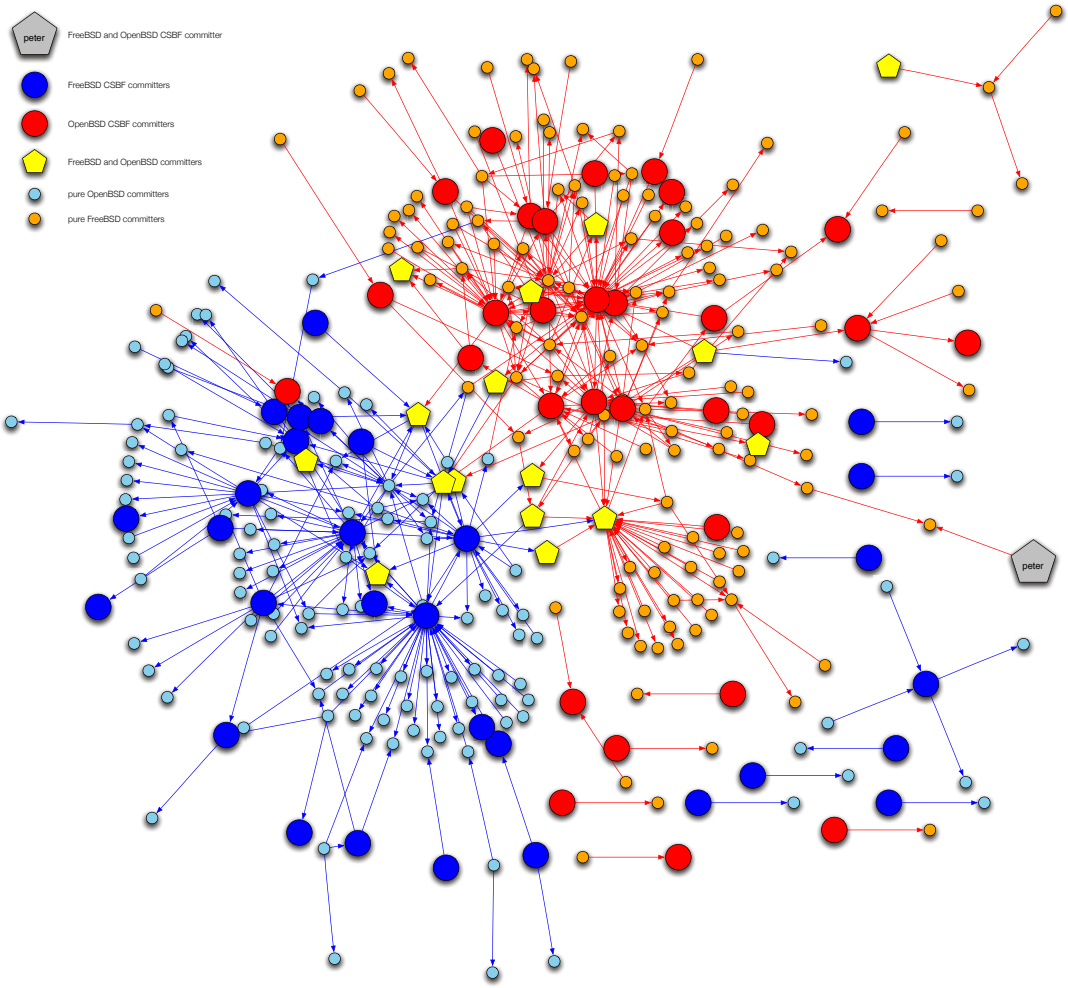


Figure 3: Excerpt of the social network of OpenBSD and FreeBSD committers.

Table 6: Statistical comparison of degree, betweenness and brokerage metrics for contributors involved and not in CSBFs.

Metric	Contributors involved in CSBFs Mean	Median	σ	Other contributors Mean	Median	σ	p-value	Cohen d
Degree	22.52	5.00	39.47	2.47	1.00	7.26	< 0.01	1.79
In degree	9.42	2.00	16.72	1.31	1.00	3.46	< 0.01	1.63
Out degree	13.10	2.50	23.46	1.16	1.00	4.38	< 0.01	1.78
Betweenness	20,248.50	1,938.90	45,880.18	1,220.75	< 0.01	9,363.52	< 0.01	1.42
Coordinator	455.92	4.50	1,481.99	8.33	0.00	186.66	< 0.01	1.26
Gatekeeper	11.45	0.00	34.22	0.28	0.00	5.06	< 0.01	1.30
Representative	8.61	0.00	37.03	0.18	0.00	3.72	< 0.01	1.02

Table 7: Top 10 Contributors with highest values of degree, betweenness and brokerage (names in bold face are involved in CSBFs).

Degree	In Degree	Out Degree	Betweenness	Coordinator	Gatekeeper	Representative
todd c miller	remko lodder	todd c miller	**poul henning**	**poul henning**	**poul henning**	**otto moerbeek**
otto moerbeek	**poul henning**	**otto moerbeek**	**john mark gurney**	**otto moerbeek**	todd c miller	**poul henning**
poul henning	**otto moerbeek**	**poul henning**	kenneth r westerback	todd c miller	**henning brauer**	todd c miller
remko lodder	todd c miller	**miod vallat**	todd c miller	**bill fenner**	**bill fenner**	**henning brauer**
bill fenner	**bill fenner**	**henning brauer**	remko lodder	**miod vallat**	remko lodder	theo de raadt
miod vallat	**robert watson**	**bill fenner**	**otto moerbeek**	**henning brauer**	**otto moerbeek**	brooks davis
henning brauer	**miod vallat**	remko lodder	**henning brauer**	**mark kettenis**	**miod vallat**	**john mark gurney**
mark kettenis	**henning brauer**	**mark kettenis**	**bill fenner**	**gary palmer**	**robert watson**	**robert watson**
gary palmer	mark murray	brooks davis	**miod vallat**	**robert watson**	angelos d keromytis	alex
robert watson	bill paul	**gary palmer**	theo de raadt	mark murray	kenneth r westerback	**reyk floeter**

(ii), shown as small cyan and orange circles for OpenBSD and FreeBSD respectively.

Table 6 reports descriptive statistics (mean, median, and standard deviation) of social network metrics computed over the inter-project committer-contributors mailing list network for contributors participating (and not) in CSBFs. Also, the table reports results of the Mann-Whitney test and Cohen d effect size, aimed at comparing metrics for the two groups of contributors. It can be noticed that for some metrics, e.g., betweenness and coordinator, there is a high standard deviation as the network comprises individuals playing and not such roles. Nevertheless, tor all metrics, it can be seen that differences are statistically significant and values are higher for contributors involved in CSBFs (the effect size is always *high*, i.e., $d > 0.8$).

By analyzing the metrics, we can make several considerations. First, as it can also be noticed from the yellow boxes in Figure 3, CSBF contributors have a very high degree (absolute degree, in-degree, and out-degree), i.e., they communicate with many other contributors acting as both sources of information and as information collectors. CSBF contributors also have a very high betweenness (on average one order of magnitude higher than other contributors), i.e., they reside in many shortest communication paths. Finally, also brokerage metrics are significantly higher. This is not surprising for the coordinator metric, as we have found that these contributors have a high degree and betweenness. However, also the gatekeeper and representative metrics are significantly higher (though their median is zero for both CSBF and non-CSBF contributors); this was not expected because, as mentioned in **RQ2**, only one CSBF contributor (*peter@freebsd.org*) is actually involved in both mailing lists and thus acts as a direct point of contact between the FreeBSD and OpenBSD mailing list discussions. However, as it can be noticed from Figure 3, most of the CSBF contributors are "in touch" with people involved in both mailing lists. For this reason, they also exhibit relatively high coordinator and representative metrics.

Table 7 reports, for the various social network metrics computed, the top ten contributors, i.e., those having the highest values for that metric. Most of them (highlighted in bold face) are involved in CSBFs. Specifically, 8 (degree), 6 (in-degree), 7 (out-degree), 6 (betweenness), 8 (coordinator), 6 (gatekeeper), and 6 (representative). Noticeably, the person who participated to CBSFs and contributed to both mailing lists (*peter@freebsd.org*) does not appear in this list. The curiosity to inspect such a special case in detail confirmed that his activity is not so different from other CSBF contributors but, indeed, his social interaction, as captured by our analysis, is different. Our conjecture is that the information may flow also through the source code itself and not only through mail exchanges (or telephone calls). In other words, when a developer says *"from peter"*, the developer did not actually exchanged messages—directly or through other persons–with *peter*, but rather s/he looked at *peter* source code in the other system, found it useful, and reused it to make the change.

We therefore conclude **RQ3** stating that *contributors involved in CSBFs have a higher importance in the mailing list discussion and in the flow of communication between different mailing lists than other contributors.*

4.4 RQ4: Are Mailing List Contributors Involved in CSBFs more Active than Others?

Table 8 reports descriptive statistics (mean, median, and standard deviation) of the number of commits and of lines added/removed by contributors involved (or not) in CSBFs. Also, the table reports results of the Mann-Whitney test, and the Cohen d effect size. As it can be seen, for both projects contributors involved in CSBFs performed a significantly higher number of commits, and added/removed a

Table 8: Statistical comparison between change activity of contributors involved and not in CSBFs.

Project	Metric	Contributors involved in CSBFs Mean	Median	σ	Other contributors Mean	Median	σ	p-value	Cohen d
FreeBSD	# of Commits	1,111.55	408.00	1,827.39	135.32	25.50	287.25	< 0.01	1.06
	LOC add/del	35,662.18	15,904.00	48,327.41	4,351.08	380.00	10,266.49	< 0.01	1.20
OpenBSD	# of Commits	1,102.71	404.50	1,987.68	86.85	22.50	172.31	< 0.01	0.93
	LOC add/del	30,170.52	13,082.00	47,175.23	3,534.24	433.00	11,448.81	< 0.01	0.95

significantly higher number of source code lines than other contributors (among those that have access to the project CVS repository). In all cases, the effect size is *high* ($d > 0.8$). It is therefore possible to conclude **RQ4** stating that contributors involved in CSBFs not only play an important role in the mailing list communication and in the brokerage between project mailing lists—as shown by **RQ3**, but *they are also more active than others in terms of change activities.*

5. THREATS TO VALIDITY

This section discusses the main threats to the validity of our study.

Construct validity threats mainly concern with imprecisions in our measurements, especially (i) the capability of the approach for detecting CSBFs explained in Section 2.1, and (ii) the approach for disambiguating and unifying emails and committer IDs. About mining CSBFs, we have chosen an approach that favors precision over recall and, as explained in Section 3, all the detected CSBF links have been manually validated. In future work we plan to use more sophisticated approaches to detect CSBFs not highlighted by similar commit notes. For what concerns mailing list and commit notes, as explained in Section 2.2 we used a conservative approach that unifies names/ids/emails only when there is no possibility of ambiguity. Also, we manually validated the correspondences for CSBF committers.

For what concerns *internal validity* threats, we are aware that we cannot claim any cause-effect relationship between the particular values observed for social network metrics (as well as for the high number of changes) and the role played by contributors in CSBFs. In other words, there could be other factors influencing such values. We are also aware that in many cases the communication between important CSBFs could be hidden from mailing lists [1].

For what concerns *conclusion validity* threats, we used, where appropriate—i.e., for **RQ3** and **RQ4**—statistical tests to answer our research questions. Specifically, we used non-parametric tests (Mann-Whitney and Wilcox) which do not require assumptions on the distribution of the underlying data set. In addition, we used the Cohen d effect size measure to evaluate the magnitude of the differences.

External validity threats concern the generalization of our findings. For this first study we have chosen two very related systems, having a common origin but evolved almost independently. Although, as we have explained in the introduction, we have noticed that similar mechanisms also occur in other systems, we do not know whether results obtained here could still be considered valid in other cases, thus replication on different system families will be crucial.

6. RELATED WORK

In this section we discuss related work concerned with code provenance, socio-technical congruence among different systems, and the adopted data extraction techniques. Whilst a number of studies exist about cloning within a system, and clone maintenance within a system, to the best our knowledge the literature lacks of specific works related to performing maintenance/bug fixing activities across two ore more systems, and how such kind of activity is related to the social characteristics of system developers.

Krinke *et al.* presented an approach to automatically distinguish the copied clone from the original in a pair of clones to detect the provenance of code among several subprojects of the GNOME Desktop suite [11]. The work revealed a complex flow of reused code between the different sub-projects, giving grounds for the study conducted in this papers which is related to the maintenance of such kind of code.

Germán *et al.* introduced the concept of *code sibling* to refer to a code clone that evolves in a different system than the code from which it originates [7]. In particular, the work analyzed copyright implications when a code fragment is transfered between systems under different licenses. The work showed that, in most cases, this migration appears to happen according to the terms of the license of the original code being copied, favoring always copying from less restrictive licenses towards more restrictive ones. Also this work showed that the phenomenon of cross-system code migration is not so rare and is worth being analyzed.

In software development contexts, socio-technical congruence among different systems has been analyzed in the work of Bird *et al.* [3], where email archives of open source projects and the developers commits are used to trace the communication and coordination activities between participants. Bird *et al.* [4] also adopted communication and development data to detect the network structure of the community of five large open source project and to detect the centrality of individuals. Pohl and Diehl [12] showed how social networks could be used to determine roles in a community of developers within a single project. In this paper we adopt similar social network metrics, however our aim is to study developers involved in CSBFs.

Xu *et al.* [16] performed a topological analysis over the network of developers participating to SourceForge projects. They found several properties of such a network, such as the fact that the community is self-organizing, with small average distance among developers and high-clustering coefficient. As in our study, Xu *et al.* analyzed the network of developers participating to multiple projects. In our case we specifically focus on how properties of such a network are related to CSBFs.

7. CONCLUSIONS AND WORK-IN-PROGRESS

When multiple software systems contain cloned code, it can happen that a change, or more specifically a bug fixing,

is propagated from one system to the other. This might involve communication between the contributors of the two systems to share information about the related bug.

This paper reported a study on cross-system bug fixings (CSBFs) in two open source operating systems, FreeBSD and OpenBSD. We detected explicitly documented CSBFs, i.e., all those for which the commit note explicitly refers the other system, and linked them using an approach combining textual comparison of commit notes with clone detection between the involved files. Then, based on messages exchanged on bug-fixing mailing lists of both systems, we analyzed the cross system social network of committers and contributors directly communicating with committers. We found that committers involved in CSBFs have a higher importance in the communication both for what concerns being source/sink of information, for being in the flow of many communications, and for playing brokerage roles for within-the-project communication, but also for inter-project communication, which serves to propagate information about similar problems the two systems have, thus triggering and supporting CSBF activities. Noticeably, there are also cases where committers involved in CSBFs did not seem to be involved in discussing the issue through mailing lists; this confirms that, sometimes, software repositories do not track all the communication—as pointed out by Aranda and Venolia [1]—or that the communication is implicit in the source code itself.

In summary, this paper represents a first example of CSBF analysis, and highlights the role of project committers involved in CSBFs. Clearly, this work has a limited scope in that it only considers *explicit* CSBFs. In many cases, CSBF could not refer the other project (though we found evidence that such a practice is also followed in other projects besides the BSDs); above all, commit notes might be totally different, or the change could be inherited from files having a different name from the target one.

Future work will aim at developing more sophisticated approaches to identify CSBFs, by (i) combining the analysis, comparison and tracking of changes occurred in both systems, and (ii) analyzing similar posts in bug tracking systems of different projects and the changes they are linked to. Finally, we plan to extend the study on further families of systems that could likely share bug fixing activities.

8. REFERENCES

[1] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 298–308, 2009.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 137–143, New York, NY, USA, 2006. ACM.

[4] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 24–35, New York, NY, USA, 2008. ACM.

[5] J. Cohen. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

[6] M. Di Penta and D. M. Germán. Who are source code contributors and how do they change? In *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, pages 11–20. IEEE Computer Society, 2009.

[7] D. M. Germán, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol. Code siblings: Technical and legal implications. In *Proc. of the 2009 Working Conference on Mining Software Repositories, MSR 2009*, pages 81–90, 2009.

[8] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, July 2002.

[9] C. Kapser and M. W. Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13(6):645–692, 2008.

[10] J. Krinke. A study of consistent and inconsistent changes to code clones. In *WCRE '07: Proceedings of the 14th Working Conference on Reverse Engineering*, pages 170–178, Washington, DC, USA, 2007. IEEE Computer Society.

[11] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Cloning and copying between GNOME projects. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 98–101. IEEE, May 2010.

[12] M. Pohl and S. Diehl. What dynamic network metrics can tell us about developer roles. In *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, CHASE '08, pages 81–84, New York, NY, USA, 2008. ACM.

[13] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[14] J. P. Scott. *Social Network Analysis: A Handbook (2nd edition)*. Sage Publications Ltd, Englewood Cliffs, NJ, 2000.

[15] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, 15(1):1–34, Mar. 2009.

[16] J. Xu, Y. Gao, S. Christley, and G. R. Madey. A topological analysis of the open souce software development community. In *38th Hawaii International Conference on System Sciences (HICSS-38 2005), CD-ROM / Abstracts Proceedings, 3-6 January 2005, Big Island, HI, USA*. IEEE Computer Society, 2005.

[17] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 563–572. IEEE Computer Society, 2004.