Code siblings: technical and legal implications of copying code between applications

Daniel M. German[†], Massimiliano Di Penta[‡], Yann-Gaël Guéhéneuc[⋆], and Giuliano Antoniol[⋆]

[†] University of Victoria, Victoria, BC, Canada
[‡] RCOST–University of Sannio, Benevento, Italy
[⋆] PTIDEJ Team–SOCCER Lab., DGIGL, École Polytechnique de Montréal, QC, Canada
dmg@uvic.ca, dipenta@unisannio.it, yann-gael.gueheneuc@polymtl.ca, antoniol@ieee.org

Abstract

Source code cloning does not happen within a single system only. It can also occur between one system and another. We use the term code sibling to refer to a code clone that evolves in a different system than the code from which it originates. Code siblings can only occur when the source code copyright owner allows it and when the conditions imposed by such license are not incompatible with the license of the destination system. In some situations copying of source code fragments are allowed—legally—in one direction, but not in the other.

In this paper, we use clone detection, license mining and classification, and change history techniques to understand how code siblings—under different licenses—flow in one direction or the other between Linux and two BSD Unixes, FreeBSD and OpenBSD. Our results show 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. We also discovered that sometimes code is inserted to the kernels from an outside source.

Keywords: Code licensing, software evolution, clone detection.

1 Introduction

A source code fragment (or a whole source code file) can be copied from one system to another for several reasons, including adding features already available in the other system or fixing a bug using a known and robust implementation. Such a copying often happens when a developer works on both systems or migrates from one system to the other. Furthermore, to promote hardware adoption, companies often release and distribute the same code, *e.g.*, a driver, for different operating systems and environments. In all cases, cross-system clones are introduced.

Usually, source code is distributed according to the terms of a software license. Once the developer chooses to distribute her work with a particular license, she explicitly imposes limits on what can be done with the code: if and how it can be used, modified, copied, distributed, and extended.

Software licenses may prevent or favor the migration of code fragments in one or the other direction, or both. Once having migrated, code fragments evolve constrained by the new environment. In the following, we use the term *sibling* to refer to a fragment of code that has been cloned from one file in one system to another file in a different system. In some cases, a sibling may span an entire file.

Then, we propose an analysis process to identify siblings and to locate potential legal issues that affect them. Investigating such issues is relevant because, from a legal point of view, two licenses can be incompatible. With incompatible licenses, code fragments cannot—legally—migrate between systems. The compatibility of one license with another (*e.g.*, the new BSD License is compatible with the GNU General Public License) creates a preferential flow of code with the former license into the system with latter.

The primary contributions of this paper can be summarized as follows: (i) we propose an approach relying on clone detection across systems and license classification to study the impact of software licenses on code siblings; (ii) we provide evidence that a preferential flow exists from FreeBSD/OpenBSD to Linux; (iii) we report unexpected results on the migration of third-party code from outside the kernels into two or more kernels.

This paper is organized as follows. After a discussion of related work in Section 2, Section 3 describes our study and the process followed to extract data from the three kernels. Section 4 presents the empirical study results, while Section 5 provides a qualitative analysis of some examples of

siblings. Section 6 discusses the results along with threats to validity and highlights open issues. Finally, Section 7 concludes the paper and outlines directions for future work.

2 Related Work

In this section we discuss related work related to clone evolution analyses and and legal implications of cloning.

2.1 Studies on Clone Evolution

In recent years, the focus has shifted from the development of algorithms for clone detection to the analysis of clones and clone evolution. Early studies focused on the evolution of clones in the Linux kernel [1, 17]. More recent studies have examined the relationship between clone and defects [16, 19], the origin and stability of clones [7, 13, 14], the impact of clones on code changeability [18, 19], clone maintenance [23], the life duration of clones and their genealogy [12], and whether clones constitutes an harmful phenomenon [11].

We share with the aforementioned works the goal of improving our understanding on clone evolution and management. However, (i) we focus on clones across different systems instead of clones within a same system and (ii) we relate cloning with licensing, to study the impact of software licenses on cloning across different systems and to identify (if any) potential legal issues.

There are studies analyzing the evolution of commonalities among systems belonging to the same family, for example the studies done on the BSDs kernels by Fischer *et al.* [3] and by Yamamoto *et al.* [26]. Our study shares with them the idea of analyzing similar code across different systems. However, we are interested to investigate whether the presence of similar code is influenced in any way by software licenses.

2.2 Legal Implication of Cloning

From a legal point of view, software systems are protected using four different ways: trademarks, trade secrets, patents, and copyright [2, 8, 15, 21]. In particular, copyright protect the expression of an idea. Consequently, many systems can implement the same idea without violating the different implementors' copyright; *e.g.*, there are many different kernels that implement the basic services required to act as a Unix operating system. Every software system is protected by copyright, unless its copyright owner has placed it into the public domain. Anybody can copy and reuse programs in the public domain. Otherwise, only the copyright owner has certain exclusive rights over her creation such as making copies and preparing derivative works [25]. Copyright law allows for the copying and using of portions of copyrighted material under certain circumstances, such as fair use in the United States or fair dealing in the United Kingdom and Canada. However, copying code without the approval of the copyright holder is in general prohibited, *e.g.*, a US court has ruled that even copying 27 out of 525,000 LOCs can be considered copyright infringement [20]. Thus, cloning of code fragments requires the explicit permission from the copyright owner of the fragments.

Within an organization, cloning is not an issue because all the code is typically owned by the organization. However, with the advent of free and open source software (FOSS), it has become easy to copy systems (or portions thereof) and incorporate them into other ones. An important feature of FOSS licenses is that they encourage reuse, either by using their systems as components or by reusing their source code.

Each FOSS license places specific requirements that the licensee should satisfy to be allowed to copy fragments of a system into another one [9]. Given the scope of this paper, we are primarily concerned with two types of licenses: *permissive* (also known as academic), such as the MIT/X11 and BSD licenses, and *reciprocal*, such as the different versions of the GNU General Public License. Permissive licenses place minor constraints on the licensee and allow the inclusion of fragments in a system under a different license; *e.g.*, BSD licensed fragments can be included in proprietary systems. Reciprocal licenses require the system that includes the fragments to be licensed under the same license; *e.g.*, GPL-licensed fragments can only be included in systems licensed under the same version of the GPL.

It is important to highlight that there are several variants of the BSD license. The original BSD, also known as 4-clauses BSD, the new BSD, also known as 3-clauses BSD, and the 2-clauses BSD. Code licensed under the original 4-clauses BSD cannot be included inside systems licensed under the GPL. (See [22] for a detailed discussion of each of these licenses). Most parts of the ⋆BSD kernels use a 2-clauses BSD license while Linux uses primarily the GPL version 2.

3 Empirical Study

The *goal* of the study is to understand to what extent code siblings between different FOSS exist and whether these siblings satisfy the conditions imposed by their open source licenses. The *quality focus* of the study is the proper and consistent usage of open source licenses when code fragments are copied from one system into another. The results are of interest in the *perspectives* of (1) researchers who want to understand the extent of the siblings phenomenon; (2) organizations participating in open source development that are concerned with possible licensing issues;

Table 1. Characteristics of the three kernels.

	Linux	FreeBSD	OpenBSD
Snapshot/release dates	Jan, 18 2009	Jan, 23 2009	Jan, 27 2009
# of .c files	10,343	13,699	3,175
# of .h files	9,422	7,855	3,468
KLOC	7,262	8,456	2,301

and, (3) lawyers who want to advance the cause of intellectual property rights.

The *context* of the study are the siblings occurring between kernels of BSD operating systems (OpenBSD and FreeBSD) and the Linux kernel. For FreeBSD, we downloaded from one of its CVS mirrors the latest snapshot available on Jan, 23 2009 and for OpenBSD the latest snapshot available on Jan, 27 2009. For the Linux kernel, we downloaded the release 2.6.27.12 of Jan, 18 2009. Table 1 reports the number of `.c`, `.h` files and the KLOC for the three kernels. To track back the introduction of siblings, the CVS repositories of FreeBSD and OpenBSD contained information since 1993 and 1995 respectively, while the first Linux kernel release analyzed (1.0.0) dates back to 1994.

3.1 Research Questions

This study aims at answering the following research questions:

- **RQ1:** *What kinds of open source licenses are used in the three kernels?* Answering this question is necessary to understand in what cases a sibling from a kernel to another could potentially create licensing issues.
- **RQ2:** *How many potential siblings exist between the BSD kernels and the Linux kernel?* Answering this question should confirm the importance of studying and tracking clones across systems and their licenses. In particular, we are interested to know how prevalent is the (potential) copying of code fragments between kernels.
- **RQ3:** *What licenses are used by siblings and, if they are different, why?* When a code fragment is copied from one kernel to another, we expect that it is done without breaking its license. Thus, if siblings have different licenses, it is important to quantify how often and understand why this happens because different licenses could lead to licensing issues.

3.2 Data Extraction and Analysis Process

We describe the steps followed to extract and analyze the data necessary to answer the research questions in Section 3.1.

3.2.1 RQ1: What kinds of open source licenses are used in the three kernels?

RQ1 aims at identifying what are the licenses used in the three kernels. To classify file licenses, we rely on the open source license classifier FoSSology version 1.0.0 [6]. FoSSology uses a pattern matching algorithm called the Binary Symbolic Alignment Matrix (bSAM), inspired by an algorithm used to detect protein changes in biomedical research. FoSSology is capable of detecting 78 different license variants, classified in a hierarchy of licenses (for example, there exist several kinds of Corporate licenses belonging to different companies, as well as different versions of the GPL).

We randomly checked some classifications of FoSSology and discovered that it had two limitations. First, FoSSology reported some files as without a license when in fact they had one; when FoSSology failed to detect a license, it labelled the file as "None", resulting in many false positives. Second, some instances of the GPL were not detected[1]. To deal with the first limitation, we extracted comments from the files that FoSSology classified as "None" and, for those containing the string "license", we performed a manual inspection. Using this heuristic, we were able to discover 5 more licenses and add them to FoSSology set of patterns. To deal with the second limitation, we searched the comments of each file for the the terms "General Public License", "GPL License", "GNU License" and, in case of matching, we classified them as GPL. Finally, we marked the remaining files as "Unknown".

3.2.2 RQ2: How many potential siblings exist between the BSD kernels and the Linux kernel?

RQ2 aims at understanding how many files in a kernel contain code fragments similar or matching fragments in a different kernel. First, we identified, between file pairs, copied code fragments in different systems using a clone detection tool. We adopted the *CCFinder* clone detector [10] (in particular the new *CCFinderX* tool) as it allowed tuning the clone detection parameters, recovering token sequence for inspection, and accessing its intermediate representations. Then, we used a series of heuristics to reduce the set of possible siblings:

1. We configured the clone detection tool to detect only clones with a minimum size of 100 tokens (using the `-b 100` option) because we are interested in substantial code siblings rather than small clones.
2. We focused our attention on `.c` files only, discarding the header files because we are interested in implementation siblings rather than in finding modules having a similar interface.

[1] We reported this bug and it will be fixed in the next release.

3. We restricted our attention to file pairs having a large percentage of common code fragments. Given two systems s_1 and s_2 (*e.g.*, FreeBSD and Linux) and given the set of file pairs of s_1 and s_2 indicated as $f_{1,i} - f_{2,j}$, where the first index indicates the system and the second the file; for each pair $(f_{1,i}, f_{2,j})$, we computed the percentages $p_{1,i}$, $p_{2,j}$ of $f_{1,i}$ cloned in $f_{2,j}$ and vice versa. Given the distribution of $p_{1,i}$ and $p_{2,j}$, we focused our attention on file pairs for which both $p_{1,i}$ and $p_{2,j}$ are above a given threshold t. The threshold was computed based on average file size and percentiles of code duplication distribution. Details regarding the choice of the thresholds for our study are provided in Section 4.

4. Starting from candidate files, we discarded cases where one file in s_1 has clones in several files in s_2. This often happens when code is first cloned from one system to another and then further cloned in the second system (or vice-versa). In that case, we can consider that the sibling occurs on one file only, and then the latter has several clones in the target system. For example, the file `sfadd.c` is paired to `sfsub.c`, `sfmult.c`, `fsmult.c`, and `sfsqrt.c`. If a file is paired to too many files, it is likely to be cloning code that is too generic to be relevant in our study. Therefore, we removed any file that were paired to more than five files. If a file was paired to five or less others, we kept the pair with the file that contained the largest proportion of clones. For instance, of the pairs in which `sfadd.c` from OpenBSD was matched to files from Linux, we selected `sfadd.c` because it has the largest clone proportion (with 95.9%). In all cases where a file was paired with more than five files, the resulting pair had the same file name in both systems (without its path).

After applying these heuristics, we can say that the remaining clones are siblings because they exist in two different systems.

3.2.3 RQ3: What licenses are used by siblings and, if they are different, why?

Once having identified licenses for each file (RQ1) and siblings (RQ2), we determined in which direction the code fragment was likely to have been copied, *e.g.*, whether the fragment was copied from Linux (where a file could have a GPL license) to FreeBSD/OpenBSD (where the file could have a BSD license). We focused first on siblings for which the detected license was different, *i.e.*, cases where a fragment was likely cloned from one system to another (or shared between the two systems) under different licenses. Then, we identified the time when the cloning happened by applying the following method on all pairs of siblings.

1. We extracted all the siblings in the current revision of the files (on which we ran *CCFinder* to address RQ2, hereby indicated as r_c). *CCFinder* returns the line range of every sibling.

2. We retrieved previous revisions of the file $r_{c-1}, \ldots, r_1$. For BSDs, we checked them out from their CVS repositories considering only the HEAD development trunk. For Linux, we only considered stable releases because its history before the introduction of Git in April 2005 appears to lack details. Linux previous history was imported into the Git repository `old-2.6-bkcvs` but we have found this history to be too incomplete for our study: large numbers of changes are clumped into single commits (probably the result of a merge in BitKeeper that lost detailed history from branches). In essence, because we used versions for Linux, our analysis of Linux is coarser than the one from the BSDs.

3. We ran *CCFinder* between each cloned fragment contained in a file and the previous revisions of the file to assess the revision in which it was introduced: for each clone, we stopped once we found a file revision r_{c-k} where the cloned fragment did not exist and assumed that it was introduced at revision r_{c-k+1}. To speed up the process, the search was performed using a binary search algorithm.

Consequently, we knew for each siblings the revision in which the cloned fragment was introduced. It happened that a sibling for a file pair $(f_{1,i}, f_{2,j})$ consisted of several code fragments, introduced in different revisions. In such a case, we considered the oldest revisions among the ones when these fragments were introduced. Consequently, by knowing in which system the fragment was present first, we were capable of tracking the direction in which the sibling was introduced. When a siblings consisted of multiple cloned fragments, we considered that the sibling has been introduced when the oldest cloned fragment was introduced.

4 Results

This section reports results of our empirical study, answering the questions in Section 3.1.

4.1 RQ1: What kinds of open source licenses are used in the three kernels?

Figure 1 shows, for the three kernels, the numbers of `.c` files having different licenses. It can be immediately noticed

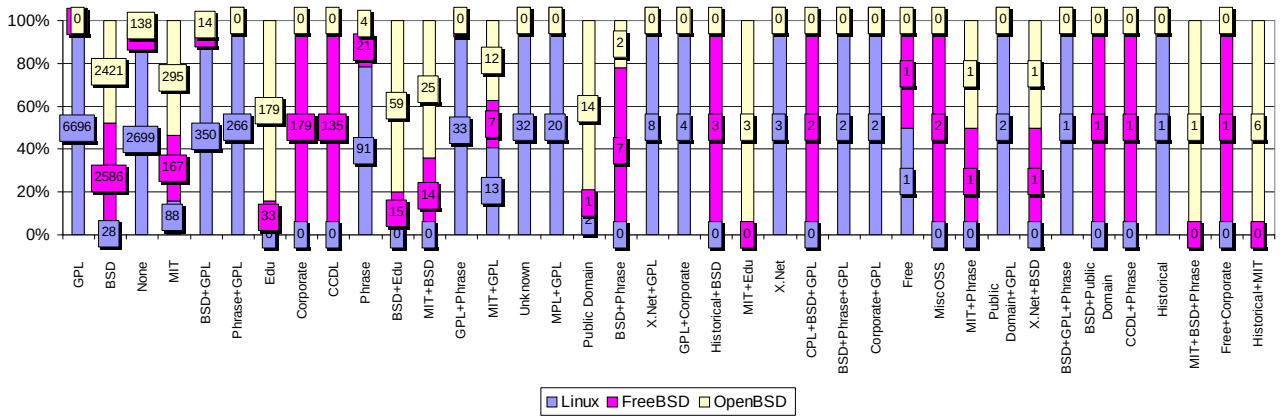

Figure 1. Frequency of licenses in files containing clones. (Bars are normalized to 100%, numbers are the actual numbers of files with a certain license, licenses are sorted from largest numbers of files to smallest. Occluded numbers are always close to zero.)

that, for Linux, most of the files (6,696, 65% of the total) have a GPL license and that there is also a high number of files (2,699, 25%) without any license (Linus Torvalds has stated that files without a license are under the GPL [24], by default). Linux also contains files with multiple licenses (among others, 350 with BSD and GPL). In FreeBSD, as expected, the most widely used license is the BSD license (2,586 files, 75%) and only 189 files (5%) do not have a license. FreeBSD contains a number of files (179) with a Corporate license (Intel licenses) and MIT (167). There are a few cases of files with multiple licenses, *e.g.*, 19 with BSD and GPL, 15 with BSD and Educational, 14 with MIT and GPL. In OpenBSD, BSD licenses dominate (2,421, 76% of the total), followed by 295 files (9%) with a MIT license, 179 with an educational license and only 138 (84%) without license. There are however some files with multiple licenses, in particular 59 with BSD and Educational, 25 with MIT and BSD, and 14 with BSD and GPL.

4.2 RQ2: What is the number of potential siblings between the BSD kernels and the Linux kernel?

Table 2 reports the results of (i) the clone detection between the different kernels and (ii) the filtering process described in Section 3.2 to detect potential siblings between Linux and the BSDs. It shows that 2,208 clone pairs were found between Linux and FreeBSD files, while only 749 between Linux and OpenBSD files. It also shows (1) the number of files containing at least one sibling in Linux and in the BSDs respectively, (2) the number of file pairs for which there is at least one sibling between Linux and BSD (412 between Linux and FreeBSD, 161 between Linux and OpenBSD), and (3) the number of these pairs where the file name is the same in the two kernels (189 between Linux and FreeBSD, 47 between Linux and OpenBSD).

Table 2. Number of files and file pairs with clones between Linux and the BSDs.

CCFinder Outputs		
Metrics	FreeBSD vs. Linux	OpenBSD vs. Linux
# of Clone pairs	2208	749
# Files Linux	296	100
# Files BSD	289	94
# File Pairs	412	161
# File Pairs (same name)	189	47

Filtering Results		
Metrics	FreeBSD vs. Linux	OpenBSD vs. Linux
# Files Linux	224	63
# Files BSD	226	61
# File Pairs	238	73
# File Pairs (same name)	185	45

After having detected siblings, we first pruned out file pairs for which the percentages of cloned tokens in one system or the other was below the threshold t. The threshold value was chosen by observing the distribution of cloned token percentages, shown in Table 3, and obtained by ranking files according to the percentage of cloned tokens that they contain. In all cases, if we pruned out below $t = 33\%$ percentile, we obtained for each file a percentage of cloned tokens equal or greater than 4%. On average, 4% of a file corresponds, in the systems we analyzed, to about 20 SLOCs. This value could appear low, however it should be considered that, in the past, even copying 27 source code lines has been considered a potential copyright violation.

We observed cases where a file in one system had a large number of siblings in another. We only found one file

Table 3. Distribution of percentages of cloned tokens. (The table shows average percentages of cloned tokens for sets of files ranked according to cloned token percentages and grouped in percentiles.)

Comparisons	Percentiles	Proportions Linux	Proportions BSD
Linux vs. FreeBSD	25%	2.9%	2.5%
	33%	4.7%	4.6%
	50%	16.2%	16.1%
	75%	66.3%	65.6%
Linux vs. OpenBSD	25%	2.7%	3.2%
	33%	5.2%	4.2%
	50%	9.3%	8.6%
	75%	59.3%	60.2%

Table 4. Linux vs. FreeBSD siblings with license change.

Siblings introduced first in FreeBSD and then in Linux				
License FreeBSD	License Linux	# of Files	Before 2002/01/01	After 2002/01/02
BSD	GPL	8	1	7
BSD	MIT	2	–	2
BSD	None	2	–	2
Corporate	BSD+GPL	89	79	10
GPL	None	1	1	–
Phrase	BSD+GPL	1	1	–
X.Net+BSD	MIT	1	–	1
TOTAL		104	82	22
Siblings introduced first in Linux and then in FreeBSD				
License Linux	License FreeBSD	# of Files	Before 2002/01/01	After 2002/01/02
BSD+GPL	Corporate	8	–	8
GPL	BSD	17	–	17
GPL	BSD+GPL	1	–	1
GPL	CPL+BSD+GPL	1	–	1
MIT	BSD	1	–	1
MIT+GPL	None	2	–	2
None	BSD	1	–	1
Phrase+GPL	MIT	2	–	2
TOTAL		33	–	33

in Linux with a substantial percentage (above the threshold t) of siblings with at most two files in FreeBSD. For this reason, we decided not to further prune the results for FreeBSD. In the case of OpenBSD, we found cases where a file in Linux had a substantial percentage of clones with at least five files in OpenBSD; the same happened in the opposite direction. We decided to prune these pairs. Results of pruning are shown in the bottom part of Table 2

4.3 RQ3: How many siblings occur between file pairs with different kinds of open source licenses?

Tables 4 and 5 count siblings with a different license in two systems. The tables also distinguishes where the sibling was introduced first, *i.e.*, in Linux or FreeBSD/OpenBSD, which could represent two different phenomena: (1) the

Table 5. Linux vs. OpenBSD: siblings with license change.

Siblings Introduced First in OpenBSD and Then in Linux				
License OpenBSD	License Linux	# of Files	Before 2002/01/01	After 2002/01/02
BSD	BSD+GPL	1	1	–
BSD	MIT	2	–	2
BSD	Unknown	1	1	–
BSD+GPL	GPL	1	1	–
BSD+Phrase	Phrase+GPL	1	–	1
MIT	GPL	23	23	–
TOTAL		29	26	3
Siblings Introduced First in Linux and Then in FreeBSD				
License Linux	License OpenBSD	# of Files	Before 2002/01/01	After 2002/01/02
GPL	BSD	3	–	3
TOTAL		3	–	3

code is introduced in one system first and then copied in the other one, changing the license; or (2) the code is introduced—often by the same developer or organization—in both systems at different times. Our change history analysis cannot tell whether it is the former or the latter. Only a qualitative, detailed analysis—we report some examples in Section 5—would be able to pinpoint the real reason.

There are 35 siblings between OpenBSD and Linux (17 occurring first in Linux and 18 first in OpenBSD) and 91 between FreeBSD and Linux (62 occurring first in Linux and 29 first in FreeBSD), for which the license did not change.

Results for FreeBSD indicate that, in most cases, siblings with different license appear first in FreeBSD and then in Linux (104 cases). This could be explained in two ways:

1. A higher activity in FreeBSD in early periods of our analysis time frame: as shown in the rightmost columns of the table, sibling occurring first in FreeBSD are, in most cases, introduced before 2002. Instead, all siblings occurring first in Linux are introduced starting from 2002;

2. Cases where the code fragment is actually copied from one system to the other can be due to the different license schemes that the two kernels adopt largely. In fact, FreeBSD mostly uses the less stringent licenses (*e.g.*, BSD) while Linux tend to use the most stringent one (GPL).

Also interesting to notice is the one case of two files with substantial siblings, the files *gnu/fs/ext2fs/ext2_linux_ialloc.c* in FreeBSD and *fs/ufs/cylinder.c* in Linux: while the file in FreeBSD contained a GPL license, the one in Linux did not have any license. This discrepancy could be due to licensing issue if the code was actually copied from FreeBSD (where it appeared first) towards Linux.

Sometimes siblings appeared first in Linux and later in FreeBSD. In 17 cases the Linux file has a GPL license and the FreeBSD file has a BSD license, which is a potential legal issue. In two cases, the Linux file has a Phrase+GPL license and the FreeBSD a MIT license, and, finally, in two cases, the Linux files have a MIT+GPL and the FreeBSD file has no license. In other two cases, the Linux files (where the cloned code fragments appeared first) have a GPL license and the FreeBSD files kept the GPL license, however with the addition of BSD and CPL+BSD respectively.

Results for OpenBSD clearly indicate that, in presence of siblings with different licenses, the sibling almost always appeared first in OpenBSD and then in Linux, for the same reasons as for FreeBSD. Only in three cases, the sibling appeared in Linux first and then in OpenBSD, with a license change from GPL to BSD. In all other 29 cases, the sibling was introduced first in OpenBSD and the license changed from MIT to GPL. Siblings occurring first in OpenBSD often occurred before 2002, while those occurring first in Linux are dated from 2002 beyond.

5 Qualitative Analysis

In this section, based on the quantitative information obtained by answering the research questions and on other sources of information (*e.g.*, commit logs, mailing lists), we provide a qualitative interpretation for some interesting cases of siblings with inconsistent license usage. Only a qualitative analysis could indicate whether the code was copied from a system to another or whether, instead, third-party code was made available, at different times, in two systems (*e.g.*, a driver coming from industry).

5.1 Drivers for SCSI AIC7xxx

The implementation of the drivers for the Adaptec AIC7xxx series SCSI adapters dates back to 1994. It was originally started in Linux. In 1995, some of the Linux code is incorporated into an OpenBSD driver, shortly before NetBSD forks from OpenBSD. In 1996, the NetBSD driver is also ported to FreeBSD. For some time, its author maintains the differences required by the two BSD kernels via `#if defined(__FreeBSD__)` and `#if defined(__NetBSD__)`. In 1997, a mailing list is created in FreeBSD to unify the efforts of people in the different kernels and the major development of the driver seems to happen in FreeBSD. Then, development propagates to Linux, NetBSD, and OpenBSD in 2000. To this day, the core development of these four drivers has been happening in Linux and FreeBSD, under the supervision of the same developer who started the FreeBSD port.

Figure 2 compares the same function in the drivers of Linux and FreeBSD. As it can be seen, the functions are almost identical except for the names of the functions that they call (including the names of the functions themselves). Also, one `if` statement is different in both versions.

It is interesting that the automatic tags added by CVS, `$Id$`, are maintained across kernels, helping to pin-point the direction of the flow of code. For instance, revision 1.19 of OpenBSD originates in revision 1.40 of FreeBSD and contains the following log: “new ahc driver. Adds support for newer Adaptec controllers. This represents two months of work.”. Its `$Id$` comments are:

```
FreeBSD: [...]aic7xxx.c,v 1.40 2000/01/07 [...]
OpenBSD: aic7xxx.c,v 1.19 2000/03/22 [...]
```

while the new driver for NetBSD comes from one revision later and contains a similar log: “New ahc driver, a port of Justin Gibbs’ FreeBSD driver. This adds support for the U2W chips, and U160 controllers.”:

```
NetBSD: aic7xxx.c,v 1.42 2000/03/15  [...]
FreeBSD: [...]aic7xxx.c,v 1.41 2000/02/09 [...]
```

Similar Id tags can be found in the Linux driver too.

5.2 GPL Code in FreeBSD

xfs (`xfs.org`) is a file system developed by Silicon Graphics. It was integrated into Linux in Sept 11, 2002 (located in `fs/xfs/`). Code siblings of its source code (a total of 48 files located in `/sys/gnu/fs/xfs/`) appeared in FreeBSD in Dec 12, 2005. The license of *xfs* is GPL. Including *xfs* as part of the BSD kernel would require it to be under the GPL too, which appears a contradiction to the licensing terms of FreeBSD (licensed under the 2-clause BSD license).

We found that FreeBSD has established a mechanism to deal with this licensing issue. Compiling GPL-licensed code into the kernel makes it “RESTRICTED”, hence it can no longer be distributed in binary-only form nor its source code be made available for mirroring (see “Ports with distribution restrictions” [4]).

5.3 A “Defect” in the License

The files `sys/contrib/rdma/rdma_cma.c` and `drivers/infiniband/core/cma.c` are examples of siblings, the first in FreeBSD and the second in Linux. In FreeBSD, the sibling appeared on May 05, 2008, and underwent two changes only in the main trunk; in Linux, it appeared on Jun 17, 2006, with 64 changes respectively, including 8 changes after it appeared in FreeBSD. Surprisingly, their current licenses are different. The Linux sibling is licensed under the terms of the GPL v2 and the 2-clause BSD licenses; the FreeBSD sibling, however, is licensed under the terms of the new BSD license, the GPL v2, and

```
/* FreeBSD */
struct ahd_pci_identity *
ahd_find_pci_device(aic_dev_softc_t pci)
{
 ...
 ...    ahd_pci_identity ...
 ...
 ... aic_pci_read_config ...
 ... aic_pci_read_config ...
 ... aic_pci_read_config ...
 ... aic_pci_read_config ...
 ... ahd_compose_id ...
/* If we are configured to attach to HostRAID
 * controllers, mask out the IROC/HostRAID bit
 * in the  */
 if (ahd_attach_to_HostRAID_controllers)
         full_id &= ID_ALL_IROC_MASK;
 [rest of function identical except
  for naming conventions]
}
```

```
/* Linux */
const struct ahc_pci_identity *
ahc_find_pci_device(ahc_dev_softc_t pci)
{
 ...
 const ... ahc_pci_identity ...
 ...
 ... ahc_pci_read_config ...
 ... ahc_pci_read_config ...
 ... ahc_pci_read_config ...
 ... ahc_pci_read_config ...
 ... ahc_compose_id ...
 /* If the second function is not hooked up, ignore it.
  * Unfortunately, not all MB vendors implement the
  * subdevice ID as per the Adaptec spec, so do our best
  * to sanity check it prior to accepting the subdevice
  * ID as valid. */
 if (ahc_get_pci_function(pci) > 0
  && ahc_9005_subdevinfo_valid(vendor, device, subvendor,
    subdevice)  && SUBID_9005_MFUNCENB(subdevice) == 0)
         return (NULL);
 [rest of function identical except
  for naming conventions]
}
```

Figure 2. Excerpt from code siblings in the aic7xxx drivers of Linux and FreeBSD. (For the sake of brevity identical code has been replaced with "...". Note how both functions use different naming conventions and one section of the function, the `if` statement, is very different in both.)

the Commons Public License. In both cases, the licensee is allowed to choose the license that applies (new BSD for FreeBSD and GPL for Linux, presumably).

We explored the logs of both version control systems and discovered that the original license (when the siblings were introduced) was the one still present in FreeBSD. After the code sibling appeared in FreeBSD, the license in its Linux counterpart was changed, with the following log:

```
commit a9474917099e007c0f51d5474394b5890111614f
Author: Sean Hefty <sean.hefty@intel.com>
Date:   Mon Jul 14 23:48:43 2008 -0700

 RDMA: Fix license text

 The license text for several files references
 a third software license that was inadvertently
 copied in. Update the license to what was intended.
 This update was based on a request from HP.
 [..]
```

Not only the Commons Public License was dropped but the BSD license was also changed. It originally pointed to a URL that listed the new BSD. The new version was included verbatim, but was the 2-clauses BSD instead. The log seems to imply that there was an error in the license, and that such commit fixed it. Unfortunately, this fix has not been propagated to FreeBSD.

6 Discussion and Open Issues

Code siblings exist. Code is moved from one system to another, and, in some cases, from the outside to many systems simultaneously. As shown in our study, the analysis process that we presented in this paper provides an effective means to find siblings between systems.

Perhaps one of the most interesting issues that we have uncovered is the impact of licensing in the creation of siblings. Most files in the kernels come with their own license, and, as we have seen, their licensing terms vary dramatically (from the tongue-in-cheek "Beer license" to the complex GPL). We have found evidence that the developers of the kernels are concerned to honor these licenses when they create siblings—in particular when they are not the copyright owner. This an area that requires further work: what is the impact of the license of a file in the creations of siblings from it? Does the license of a file evolve due to the need to create siblings from it? How does the licensing terms of the file evolve?

We have found evidence also that copyright owners are also interested in the existence of siblings. For instance, Intel contributes code directly to both FreeBSD and Linux. In this case, the copyright owner takes the legal steps necessary, either by contributing the different siblings under different licenses (BSD for FreeBSD and GPL for Linux) or using a dual license (each file is licensed simultaneously as BSD or GPL [5]). This is another area that is totally unexplored: how developers are affected by the licenses of their systems and vice-versa? How do the licenses affect the work of the developers?

We also observed that the flow of siblings has changed over the years. In the early days, siblings were flowing from

FreeBSD to Linux, now the tide seems to have changed direction. This is perhaps not surprising. Today Linux is a larger, more vibrant community. The rate of commits to Linux is significantly larger than FreeBSD. OpenBSD appears to have an even smaller community than FreeBSD. During January 2009, FreeBSD had—in the HEAD trunk—937 file revisions, and OpenBSD 564, while Linux had 10,909. Further research should study whether developers have moved from one kernel to the other.

We were surprised to find that the source code of FreeBSD contains files under the GPL license (*e.g.*, the implementation of the *xfs* file system). The FreeBSD maintainers justify their decision by stating that, if a developer compiles the kernel with *xfs* support, then the resulting kernel must be distributed under the terms of the GPL. By default, *xfs* is not compiled into FreeBSD (there exists a BSD-licensed implementation of read-only support for *xfs* too).

Managing siblings appears to be difficult. We have found evidence that shows that developers have tried to maintain a single source for two or more siblings, relying heavily on `#if` C preprocessor constructs to support FreeBSD and NetBSD (*e.g.*, the driver for the SCSI AIC7xxx series). Over time, this method was abandoned and, instead, several sources are currently maintained (still under the supervision of the same individual).

Another challenge is to recognize that some bugs are due to the generic algorithm and functionality that the code implements (*e.g.*, support to a particular type of equipment) and some are due to the environment in which they operate (*e.g.*, the OS calls that the driver needs to call to function within such OS or particular services they should provide to that specific OS). Bugs in the inherent nature of the driver should be propagated to the other siblings, but not those germane to its native OS. We have observed that code siblings are currently evolving faster in Linux than in the BSDs, but research is needed to understand why, and if, when, and how these changes make it to the other kernels. Other open issues include studying: when does a bug that appears in one sibling should be propagated to the other siblings? How can bugs be tracked across systems belonging to different organizations?

Downstream developers might be well aware of the source of the siblings (they copied it), but upstream developers might not (they might not know that the sibling exists). Bug corrections might never be propagated to the sibling, no matter how critical they are. The example in Section 5.3 is a good example where the license of the upstream code sibling was properly changed, but never the one of the downstream one. We believe that these challenges make the management of siblings a topic worth researching.

6.1 Threats to validity

This section briefly describes threats to validity that can affect our study. Threats to *construct validity* concern the relation between theory and observation. This kinds of threat are mainly related to the errors introduced by our measurement instruments. First, we rely on the effectiveness of the widely used clone detector *CCFinderX*. To limit the presence of false positives, we use a set of heuristics (see Section 3.2) to restrict our focus on the most substantial siblings. As reported in [6], the license classification performed by FoSSology could have some imprecision. Finally, the way we trace back clone fragments to determine their introduction depends on the effectiveness of clone detectors and, although they were already used for similar tasks, *i.e.*, to perform origin analysis [7], we cannot guarantee the result accuracy. In particular, tracing siblings in time is more imprecise for the Linux kernel wrt. FreeBSD and OpenBSD, as in the first case it was done at release level rather than at file level. Also, we did not handle file renaming or refactoring, we plan to do it in future work.

This is mainly an exploratory study, thus we are not particularly concerned with threats to *internal validity*. This means that, although we found some relations between licenses and code siblings across projects, we cannot claim causation. Also, to answer our research questions we do not need to test particular hypotheses using statistical tests, thus we are not concerned with threats to *conclusion validity*.

External validity threats concern the generalization of the findings. Our study focuses (i) on siblings between specific systems, *i.e.*, two BSD operating system kernels (FreeBSD and OpenBSD) and Linux. Other systems, in particular systems belonging to different domains, are worthwhile of being considered to increase the external validity of our findings and to established firmly causality.

7 Conclusion

This paper proposed an analysis process to study code siblings across systems under different licenses and reported an empirical study on issues related to siblings arising in the OpenBSD, FreeBSD, and Linux kernels.

Results indicate that siblings occur between different kernels and that they often tend to occur mainly in specific directions. While migration from BSDs to Linux is frequent, the opposite was significantly less frequent: this is mainly due to constraints imposed by licenses. In addition, we also show that code fragments migrated from third-parties into Linux and BSDs. A deep, qualitative analysis of some cases of siblings suggested that developers take care of licensing issues when migrating code: often by creating files with dual licenses (*e.g.*, BSD+GPL).

Our findings emphasize the need for automatic tools to check the consistency of sibling licenses to avoid licensing issues, assure that cloning happens between compatible licenses, and make sure that siblings share the same license. Examples, such as the *xfs* file system tainting FreeBSD kernel and preventing code and binary distribution, Adaptec `aic87xxx` driver, development migrated across kernels with multiple license changes, and `rdma_cma.c`, `cma.c`, with incompatible licenses, underline the relevance of the research questions addressed in this paper.

Future work aims at investigating the impact various factors—such as licenses and developers—on siblings and the effect of license change/evolution.

8 Acknowledgment

We thank Bob Gobeille from the FoSSology Project for his invaluable help with using FoSSology license detector. We thank Dr. John Aycock, who was a contributor to the Linux AIC7xxx driver, for his description of its early history. D. German was supported by the National Sciences and Engineering Research Council of Canada. G. Antoniol was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chair in Software Evolution #950-202658).

References

[1] G. Antoniol, U. Villano, E. Merlo, and M. Di Penta. Analyzing cloning evolution in the Linux kernel. *Information and Software Technology*, 44:755–765, Oct. 2002.

[2] A. Becerman-Rodau. Protecting Computer Software: after Apple Computer Inc. v. Frankin Computer Corp., 714 F.2d 1240 (3d Cir. 1983) does copyright provide the best protection? *Temple Law Review*, 57(527), 1984.

[3] M. Fischer, J. Oberleitner, J. Ratzinger, and H. Gall. Mining evolution data of a product family. In *Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005, USA, May 17, 2005*. ACM, 2005.

[4] FreeBSD Documentation Project. Ports with distribution restrictions. *http://www.freebsd.org/doc/en/books/porters-handbook/porting-restrictions.html*. Accessed Apr. 2009.

[5] D. M. German and A. E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *Proceedings of the International Conference on Software Engineering*, 2009. To appear.

[6] R. Gobeille. The fossology project. In *Fith International Workshop on Mining Software Repositories, MSR 2008, Leipzig, Germany, May 10-11, 2008, Proceedings*, pages 47–50, 2008.

[7] M. Godfrey and L. Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering*, 31(2):166–181, Feb. 2005.

[8] P. Goldstein. *International Copyright: Principles, Law, and Practice*. Oxford University Press US, 2001.

[9] Jacobsen v. Katzer, No. 2008-1001 (Fed. Cir. 8/13/2008). U.S. Court of Appeals for the Federal Circuit, 2008.

[10] 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.

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

[12] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. *ESEC/FSE*, 30(5):187–196, 2005.

[13] 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.

[14] J. Krinke. Is cloned code more stable than non-cloned code? In *SCAM 08: Proceedings of the Working Conference on Source Code Analysis and Manipulation*, pages 57–66, Washington, DC, USA, 2008. IEEE Computer Society.

[15] S. Lai. *The Copyright Protection of Computer Software in the United Kingdom*. Hart Publishing, 2000.

[16] Z. Li, S. Lu, and S. Myagmar. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Trans. Softw. Eng.*, 32(3):176–192, 2006.

[17] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Analysis of the linux kernel evolution using code clone coverage. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 22, Washington, DC, USA, 2007. IEEE Computer Society.

[18] A. Lozano. A methodology to assess the impact of source code flaws in changeability and its application to clones. In *ICSM 08: Proceedings of the International Conference of Software Maintenance*, pages 424–427, Washington, DC, USA, 2008. IEEE Computer Society.

[19] A. Lozano, M. Wermelinger, and B. Nuseibeh. Evaluating the harmfulness of cloning: A change based experiment. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 18, Washington, DC, USA, 2007. IEEE Computer Society.

[20] N. J. Mertzel. Copying 0.03 percent of software code base not "de minimis". *Journal of Intellectual Property Law & Practice*, 9(3):547–548, 2008.

[21] M. B. Nimmer and D. Nimmer. *Nimmer on Copyright*. Matthew Bender & Company, 2002.

[22] L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, 2004.

[23] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. How clones are maintained: an empirical study. *Emp. Soft. Engineering*, 2009 (to appear).

[24] L. Torvalds. Re: GPL V3 and Linux - Dead Copyright Holders. *http://lkml.org/lkml/2006/1/27/339*, Jan 2006.

[25] United States Copyright Office. Circular 92 Copyright Law of the United States of America and Related Laws Contained in Title 17 of the United States Code, June 2003.

[26] T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue. Measuring similarity of large software systems based on source code correspondence. In *6th Int. Conf. on Product Focused Software Process Improvement, PROFES 2005, Oulu, Finland, June 13-15, 2005*, pages 530–544. Springer, 2005.