2012 IEEE 35th Software Engineering Workshop

An Empirical Study of License Violations in Open Source Projects

Arunesh Mathur*, Harshal Choudhary†, Priyank Vashist‡, William Thies§, Santhi Thilagam¶

* † ‡ ¶ National Institute of Technology Karnataka, Surathkal
§ Microsoft Research India

***Abstract*—The use of Open Source Software (OSS) components in building applications has presented the challenge of integrating them in a way such that the licenses of the individual components do not conflict with each other and if applicable, the overall license of the application. These conflicts lead to violations, with many having far reaching legal consequences. While proprietary software firms are often plagued with the risks of not satisfying the clauses of OSS licenses, we hypothesize that a large degree of code reuse within the OSS community poses similar threats too. Through an analysis of 1423 projects, consisting of approximately 69 million non-blank lines of code from Google Code project hosting, we validate instances of code reuse between projects by comparing their licenses. Our results discover four violations, evaluated by searching for files that share similar content. Additionally, we present statistics on code reuse within the set of projects.**

***Index Terms*—Software reusability, Open source software, Legal factors**

I. Introduction

Over the years, the increasing popularity of the open source movement has resulted in a collaborative environment for software developers to create and share software components and libraries that can be used to provide a variety of functionality. These components usually comprise of projects or parts of projects, that can be plugged into new or existing software, bringing about savings in time and money. Potential users of such components (for example, FFmpeg[1]) include both — proprietary software developers (close source products like Bits on the Run, MovieGate etc.) and developers from the OSS community (open source projects like VLC, MPlayer etc).

Licenses also have a major influence the degree of reuse of such a component. For example, a few pieces of the FFmpeg library are distributed under the GNU Lesser General Public License (LGPL), which while supporting free software, enables a certain degree of reuse in proprietary software as well. Every such license provides certain restrictions and allowances, but due to wide variety of approved open source licenses (69 as of May 2012), legal issues between licenses emerge when components with incompatible licenses are integrated together. This has been characterized as the *license-mismatch* problem [1]. For instance, the GNU General Public License (GPL) has two popular versions that are widely accepted – version 2 and version 3; the latter however, is not backwards compatible with components that are released under version 2 only (i.e., not upgradable to a later version). In situations where they are left with little choice, other than to combine components released under incompatible licenses, developers have known to form a new license that is compatible with each of the licenses of the individual components. The formation of new licenses to combat the license mismatch problem is known as *license proliferation* [2]. License proliferation results in further incompatibilities and has been strongly discouraged by the Open Source Initiative (OSI, http://opensource.org), which has set up a License Proliferation Committee to specifically tackle this problem [3].

Proprietary firms lie potentially at a greater risk of license infringements when trying to incorporate OSS into their products, since all OSS licenses require source code to be made available for everyone to examine; most firms deal with such discrepancies through their legal departments. Perhaps one of the most famous GPL violation involved the use of BusyBox – an amalgamation of Unix utilities for embedded devices – in the proprietary products of Samsung, Westinghouse, JVC amongst others, that resulted in multiple court cases in the United States. In one such case, the developers of BusyBox, with the aid of the Software Freedom Law Center (SFLC), sued Westinghouse for $137865, consisting of damages and lawyer's fees, and ordered all infringing products to be donated to charity [4] [5]. Recently, firms like BlackDuckSoftware (http://www.blackducksoftware.com), OpenLogic (http://www.openlogic.com) and Palamida (http://palamida.com) have begun providing services to clients that plan on using OSS components in their products by helping them analyze the possible legal outcomes. Code search engines like Krugle (http://krugle.org) and Koders (http://koders.com) have also incorporated code search options that allow filtering of results based on license.

Due to the high degree of code reuse within the open source community, we believe that open source developers share similar concerns too. In one such instance, Emacs (http://www.gnu.org/software/emacs), a widely used GPL'ed text editor recently fixed a violation [6]; its developers had failed to make publicly available the sources of a certain grammar which the GPL required them to do. Through this

[1]FFmpeg is a widely used multimedia library (http://ffmpeg.org)

1550-6215/13 $26.00 © 2013 IEEE
DOI 10.1109/SEW.2012.24

empirical study, we aim to discover cases of license violations in a vast array of open source projects by tracking cases of code reuse, and subsequently validating them to ensure fair license use. In order to achieve this, we first retrieve a large repository of open source projects and scan for code clones between projects, using the approach of a plagiarism detection tool – MOSS (Measure of Software Similarity) [7], which has been tried on a variety of programming languages.

The rest of the paper is organized as follows: Section II presents the related work, Section IV briefly describes open source licensing, Section III presents the sample set selection process, Section V presents the approach behind this study, Section VI presents the results and findings, In Section VII, we conclude the paper, with suggestions for future work.

II. Related work

Reasons and motivations for code reuse in OSS have been studied previously. Through a case study involving 15 open source projects, von Krogh *et al.* [8] show that there is active reuse of code, algorithms and methods in the open source community. Haefliger *et al.* [9] describe the behavior of open source developers – comparing them with their counterparts in corporate firms based on incentives to reuse code by examining a set of 6 open source projects. The authors point out that OSS developers reuse code to mitigate development costs, to avoid working on mundane problems and instead focus on the difficult ones or quickly release production code.

There is a notable lack of large scale analysis of code repositories that track reuse of OSS. Audris Mockus [10] quantifies large scale code reuse in popular and large open source projects and confirms the existence of more than 50% of the files in more than one project, by finding directories of source code files that share several file names and only selecting those cases where the fraction of files was greater than a threshold. While this may seem as a reasonable heuristic, comparing the content of source code files would seem to provide a tighter bound than by just comparing their file names. The performance of both these techniques is captured in a study of code reuse in the FreeBSD project by Chang and Mockus [11]. The authors report that comparing files based on their content produces results with fewer false positives than file name based comparison, which also fails to detect the same file with a different name.

The legal ramifications of code reuse in the context of open source licenses has been a lesser explored topic. German and Hassan [1] develop patterns and models to help developers solve conflicts and compatibility issues between open source licenses. Sojer *et al.* [12] analyze the risks professional software developers face when reusing code in an ad-hoc fashion from the Internet. Based on a survey of 869 professional developers, the authors conclude that ad-hoc code reuse from the Internet is common and that most developers are oblivious of the legal implications of such code reuse. Recently, tools that can detect open source license violations at the binary level – Fingerprint Generator/Detector (FiGD) [13] and Binary Analysis Tool (BAT) [14] – have been developed. We, however, are looking for license violations on the source code level, rather than the binary level.

The vast field of code clone detection tools that operate at the source level have been surveyed in [15]. Despite the presence of a large number of such tools, there have been no reports of their performance on a large scale. A large number of these tools are used for finding clones in smaller sets and focus on improving the quality of results rather than scaling.

III. Sample set selection

A study of this magnitude requires a source of large open source repositories containing projects released under any popular open source license(s), thereby enabling clear compatibility checks. We examined popular code hosting services like Google Code project hosting (http://code.google.com/hosting), GitHub (http://github.com) and SourceForge (http://sourceforge.net), which have a wide variety of open source projects managed by developers over the web. Google Code offers a set of 10 popular open source licenses to choose from, while SourceForge offers a choice of over 80 licenses. GitHub encourages developers to indicate the license in a COPYING/LICENSE file – however, the absence of such a file conceals the license of the project. All the three services allow project files to be browsed online or cloned to a local drive.

We chose Google Code Hosting over the GitHub and SourceForge due to the cogent set of licenses it offers, thus making it easier to identify license violations. To get a good mix of projects, we started by selecting projects with programming languages tags such as C, CPlusPlus, Java, Python, JavaScript, ObjectiveC etc. We then added projects tagged with Database, Game, Web, Google, Linux, Windows, MacOSX, iPhone, Android, Graphics etc. During this phase, for each project, we pinned down its license, repository URL and *Activity* level. The *Activity* level describes the degree of contribution of the developers over time, and can take values *High*, *Medium*, *Low* or *None*. A *High* level indicates that the project is in active development and is contributed to frequently, while *None* indicates it has had very little/no activity.

After forming the list of projects, we retrieved snapshots of their version control repositories using the URLs stored previously. For projects having no source code in their git/svn/hg branch, we first checked to see if they been had migrated to a different location (usually mentioned on the project homepage) and in all such cases, we retrieved code from the new location. In all other scenarios, we scanned the *Downloads* tab for potential source code files. To simplify this process, we wrote special tools and scripts to automate and ignore any non-text files that could be present in the project. In total, we managed to gather a set of 1423 projects, all retrieved between January and March 2012.

IV. Open source licensing

Licenses provide copyright holders, a means of delegating permissions to distribute, modify or build derivative works to potential users of their software. The OSI lists a set of requirements that any license must fulfill on order to be recognized as a valid open source license, called the Open Source Definition (OSD) [16]. The requirements of the OSD include: (i) Allow free redistribution of and modification of the code; (ii) Make the source code available to the public (via. the Internet); (iii) Allow derived works to be distributed under the original license; (iv) Not discriminate against any group or individual; (v) Must not be specific to a technology and not restrict any software. Open source licenses are broadly classified as Copyleft/Restrictive and Permissive licenses. Permissive licenses do not add constraints on the licensing of the derivative code, except for a reference/citation and that the license text be untouched in the modified/distributed code. Copyleft licenses on the other hand, influence the license of the derived/modified code by necessitating it to be released under the license of the original software.

Google Code project hosting offers a set of eight different licenses to choose from: (i) GNU General Public License version 2 (GPL v2); (ii) GNU General Public License version 2 (GPL v3); (iii) GNU Lesser General Public License version 3 (LGPLv3); (iv) Apache License version 2.0 (APLv2); (v) MIT License; (vi) New BSD License/3-Clause (New BSD); (vii) Artistic License/GPL (AL/GPL); (viii) Mozilla Public License version 1.1 (MPLv1.1). The GPL is a strong copyleft license, while the New BSD/MIT/APLv2 licenses are non-copyleft and permissive. All other licenses lie in between. Apart from these eight choices, Google Code provides the *Other Open Source* option to use all other licenses. Table I describes compatibility amongst these licenses.

More recently, multi-licensing – the practice of offering a choice of licenses to the licensee – has to a certain extent sorted license compatibility, but it presumes familiarity with a wide set of licenses. Examples of Multi-licensed software include or example, the PERL license, which offers a choice between the Artistic License and the GPL and JQuery, which offers using the GPL or the MIT license. The Mozilla projects were tri-licensed (MPLv1.1 or later, GPLv2.1 or later, LGPLv2.1 or later) initially, but have recently migrated to MPLv2.0.

V. Research Method

This section describes two topics: (i) Our definition of license violations and how code reuse assists us in detecting them; and (ii) The procedure to detect code reuse in the sample set of open source projects (Section III).

A. Defining reuse and violations

Before describing the ideas behind this study, it is crucial to establish the definitions of code reuse and license violations. We are searching for cases where one project incorporates a set of source code files (or a part of the set) from another project in the same corpus. Such source code files are those that belong to the provider project and are not, for example, a third party library (outside of the corpus) that both projects may coincidentally use. We are not interested in accredited lines of code that may be reused between projects, since such reuse is highly granular and difficult to detect.

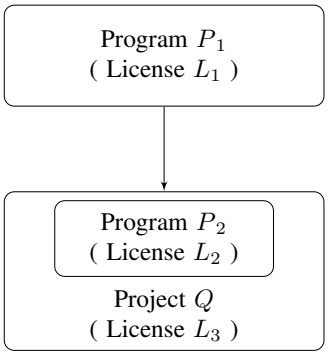

Fig. 1. Definition of a license violation

Figure 1 illustrates the definition of a license violation. Let Program P_1, licensed under L_1, be reused in the form of Program P_2 licensed under L_2, which includes P_1 and all derivative works, if any. Q, the project which contains P_2 may by itself, have an overall license L_3. Based on these notations, we define violations as the following rules:

1) **Type 1**: L_1 and L_2 are incompatible (for instance, L_1 is the MPLv1.1 and L_2 is the GPL) or, L_1 and L_2 are compatible, but L_1's copyleft nature is not honored by L_2 (for instance, L_1 is the GPL and L_2 is the MIT license).
2) **Type 2**: Similar to the violations of first type, but with checks between L_2 and L_3 instead.

B. Architecture

Plagiarism detection tools find similarity between small pieces of text/code and are widely used in academic settings, usually for checking assignments turned in by students and submissions to workshops/conferences. Our approach is borrowed from the popular plagiarism detection tool MOSS, developed by researchers at Stanford University, which is used to detect similarities in programming assignments and supports a variety of languages. MOSS begins the process of detecting code by building hashes of k–grams of source code files, and then selecting those pair of files that have the most common hashes further comparison. This pipeline helps in scaling the comparison process efficiently, while keeping it fundamentally, independent of the programming language.

Figure 2 depicts the architecture of this system. It consists of three phases – Preprocessing, Fingerprinting and Comparing. The Preprocessing stage removes all superfluous features of the source content such as whitespace, capital letters, new lines etc., which are undesirable to determine similarity between files. Since MOSS has primarily been used in academic settings, it replaces all instances of variable declarations with a common symbol before it begins matching files, as students

	GPLv3	EPLv1.0	MPLv1.1	LGPLv3	MIT	New BSD	APLv2
GPLv2	No	No	No	No	Yes	Yes	Yes
GPLv3	–	No	No	Yes	Yes	Yes	Yes
EPLv1.0	–	–	Yes	Varies	Yes	Yes	Yes
MPLv1.1	–	–	–	No	Yes	Yes	Yes
LGPLv3	–	–	–	–	Yes	Yes	Yes
MIT	–	–	–	–	–	Yes	Yes
New BSD	–	–	–	–	–	–	Yes

TABLE I
COMPATIBILITY AMONGST GOOGLE CODE'S CHOICE OF LICENSES

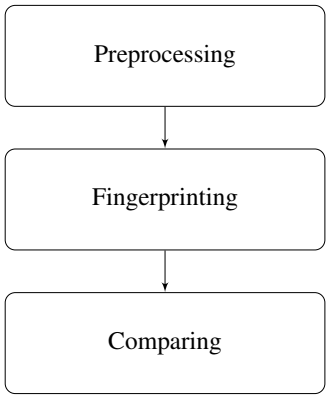

Fig. 2. Architecture of MOSS

may choose to modify variable names to evade such tools. However, Haefliger et al. [9] observed that in the open source world, code reuse is largely, *black–box* (unmodified). Hence, we do not modify or replace any variable declarations with holders.

Once the source code is preprocessed, the Fingerprinting stage starts by dividing it into k–grams, which are continuous substrings of size *k*. These are hashed, and subsequently, a subset of these are selected as the fingerprint of a file. Assuming collision free hashes, if two files share the same hash, then it is very likely that they share the same k–gram. For a large set of files, hashing can be a very computationally intensive process for large values of *k*. Rabin-Karp's rolling hash function reduces this complexity by computing the hash of the i^{th} k–gram from the hash of the $i - 1^{th}$ k–gram. For example, consider the k–gram (c_1 c_2 . . . c_{k-1} c_k), with each c_i representing the i^{th} character. Given a base b, the k–gram's hash $H(i)$ is calculated as:

$$H(i) = c_1 * b^{k-1} + c_2 * b^{k-2} + ... + c_{k-1} * b + c_k \quad (1)$$

Similarly, the hash $H(i+1)$ of the k–gram (c_2 c_3 . . . c_k c_{k+1}) is:

$$H(i+1) = c_2 * b^{k-1} + c_3 * b^{k-2} + ... + c_k * b + c_{k+1} \quad (2)$$

Writing $H(i+1)$ in terms of $H(i)$:

$$H(i+1) = (H(i) - c_1 * b^{k-1}) * b + c_{k+1} \quad (3)$$

Thus, calculating the hash of $H(i+1)$ from $H(i)$ requires two additions and two multiplications, which makes hashing successive k–grams extremely fast. Since for arbitrary b, the value of $H(i)$ may exceed the largest number that can be stored on a machine, $H(i)$ is stored as $H(i)$ % m, where m is a prime number. The choice of m is crucial to this computation, since a poor selection could lead to an increase in collisions.

For a file of length n, a total of $n - k + 1$ hashes are generated. When computed for a large number of files, the hashes require a lot of storage space and thus reduce efficiency for the later stages. To counter this, it is preferred to select a subset of these hashes, and store them as the fingerprint of a file. This is achieved through the Winnowing algorithm defined as follows:

Let a window of size w be a series of w continuous hashed k–grams (h_i, h_{i+1}, . . ., h_{i+w-2}, h_{i+w-1}). From each window, a hash is selected as follows:

1) Select the smallest hash in a window
2) In case of a tie, select the rightmost smallest hash

We store the hashes that form the fingerprints of files in a relational database as two tables. The first table (schema: $file\_id$, $project\_name$, $file\_name$), holds the details of each file, and the second table (schema: $file\_id$, $hash$, $line\_numbers$) holds the fingerprints. The $file\_id$ attribute from the first table, the primary key, serves as the foreign key for the second table.

The Comparison phase starts once the fingerprints for all the files in every project have been generated. To find files similar to any given source code file d in project p, we select those files that have the highest number of hash matches with d and are outside of p. Files that have the matched hash count greater than a given threshold t, are those that have a very high probability of being similar to d. To ensure that the matches obtained as a result of this phase are not false positives, it is important to ignore boilerplate text. Open source licenses usually require the license user to place legal boilerplate at the beginning of every file, which may lead to increasing the number of matched hashes between files. To avoid this,

we hash the headers of all licenses offered by Google Code and ignore all such hashes when matching files. Finally, we pretty print all pairs of matched files to aid us in discarding all remaining false positives.

VI. Results

A. Repository statistics

Through our sample selection process, we retrieved 1,423 projects translating to 340,164 text files, consisting of about 69 million non-blank lines of code. Figure 3 shows the count of projects for each license. The GNU GPLv3 and GPLv2 constituted nearly 46% of all licenses and this is in accordance with their popularity in the open source space [17]. The EPLv1.0 and MPLv1.1 were the least used licenses, given their are incompatibility with the popular GPL and prohibitive reuse. Both these licenses are specific to the Eclipse and Mozilla community and generally, are rarely used outside of those communities.

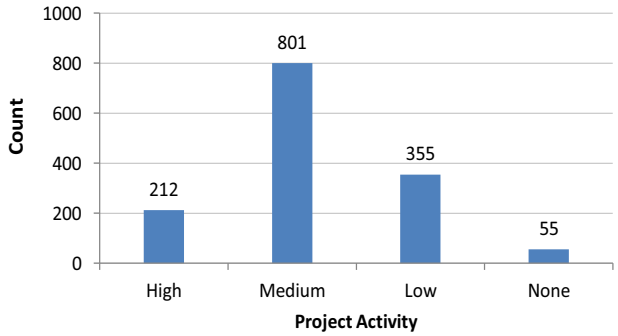

Fig. 4. Distribution of projects based on Google Code activity.

Figure 4 shows the count of projects based on Google Code activity. While the activity of a project can change over time, depending on numerous factors, we capture the activity status at a particular instance for the purpose of this study. 56.29% of the selected projects were *Medium*-active compared to the 3.87% of *None*-active projects.

B. Code Reuse

Our initial experiments dealt with choosing values for k, m and t judiciously, as they directly influence the results of the procedure described in Section V-B; a poor choice leads to multiple false positives between files. The value of k and t primarily depend on the nature of the document and the strings it contains – in practice however, we observed that values of 40 and 45 respectively, work sufficiently well, even on source code files written in a variety programming languages. To determine a suitable value for m, we conducted two tests on about 10 MB of text – first, with the largest 32-bit prime, which led to 128 collisions and second, with the largest 64-bit largest prime number, which led to no collisions at all. A total of 31,187,119 hashes were generated at the end of V-B. While we did find false positives, they were largely mitigated by ignoring hashes of license headers at the beginning of source files.

We discovered 103 instances of code reuse in the set of projects, listed in Table III. Figure 5 presents the activity levels of the reused projects. Although *High* and *Medium* active projects were reused equally (16 each), it is worth nothing that *High* active projects constitute only 14.90%, where as *Medium* active projects constitute 56.29% of the total set of projects. Consequently, the reuse rate for the former (7.56%) is higher than than the latter (2.00%). This is in conformity with the observations made in [9] – ratings and certification influence the popularity of code, as poorly written code can be detrimental to any system. Projects that were actively developed and updated were reused more frequently and this is true for both corporate firms, as well as the open source world.

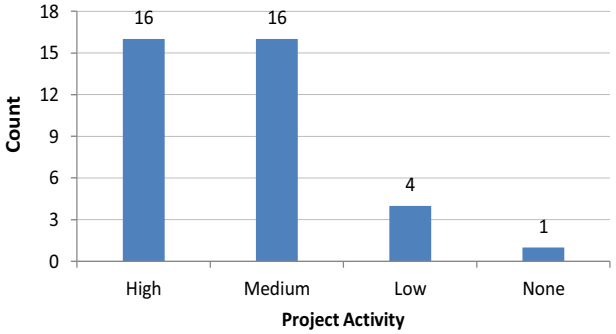

Fig. 5. Distribution of reused projects based on Google Code activity. Although *High* and *Medium* active projects are equally used, the fraction of the former is nearly 3.8 times higher.

C. License violations

Table II details the license violations out of the instances of code reuse listed in Table III. The *Type of violation* column indicates the category of license violation discussed in Section V-A. We observed a lack of proper use of the acceptor license in 3 out of the 4 cases of violations. For instance, to apply the GPL to a project, the Free Software Foundation lists the following requirements [18] :

- Add a copyright statement to each source code file along with the copyright permission text:

```
This file is part of Foobar.

Foobar is free software: you can redistrib-
ute it and/or modify it under the terms of
the GNU General Public License as publishe-
d by the Free Software Foundation, either
version 3 of the License, or (at your opti-
on) any later version.

Foobar is distributed in the hope that it
will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERC-
HANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU General Public Licen-
se for more details.

You should have received a copy of the GNU
```

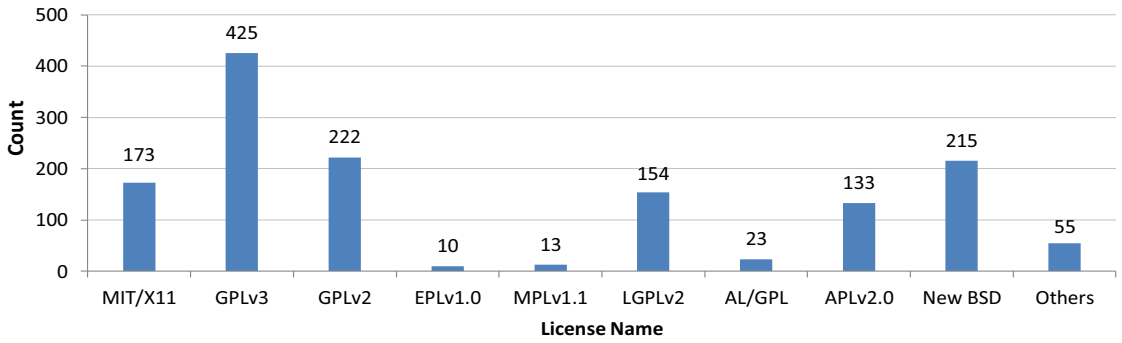

Fig. 3. Distribution of projects based on license. The General Public Licenses (GPL) together constitute nearly half of all licenses.

Code provider [provider license]	Code acceptor [Acceptor license]	Type of violation	Acceptor license used correctly?
Flvplayer [MPLv1.1]	Khan Academy [Other Open Source]	1	no
Arduino [GPLv2]	Micropendous [MIT]	2	yes
Miranda [GPLv2]	Toptoolbar [LGPLv3]	1,2	no
Miranda [GPLv2]	Wi2geoplugin [MIT]	2	no

TABLE II
INSTANCES OF LICENSE VIOLATIONS

```
General Public License along with Foobar.
If not, see <http://www.gnu.org/licenses/>.
```

- Include the GPL license in a text file in the project tree.
- Place the Copyright notices of all copied GPL code at the beginning of every file.

Therefore, these 3 cases cannot be truly be considered as cases of violations, as the license of the acceptor project in each of these cases is technically, unclear. We classify such violations as *violations of recommended practice*, where *recommended practice* refers to meeting the requirements listed by the license. In other words, had the developers of the acceptor project applied all of the clauses of the license they intended to use (mentioned on the Google Code project homepage), the would have violated the provider project's license in the process. The *Acceptor license used correctly?* column indicates whether the acceptor license was applied correctly.

In the following sections, we present the details of each license violation and suggest possible steps that can be taken to help erase the violation, such as using alternate libraries or choosing an alternate license, wherever applicable.

1) Flvplayer and Khan Academy: Flvplayer is a flash player library that can be plugged into websites for streaming multimedia and is licensed under the MPLv1.1 (http://code.google.com/p/flvplayer). The MPLv1.1 allows a limited amount of copyleft by requiring all modifications to MPLv1.1 licensed files and files that borrow MPLv1.1 licensed code, to be released under the same license. This library is used by Khan Academy, an online e-learning platform that provides video tutorials for a variety of subjects on its website (http://code.google.com/p/khanacademy), seemingly to stream flash content. The developers of the Khan Academy repository however, fail to mention its license explicitly, as the *Other Open Source* option advises them to. This violation falls under *Type 2*, as the MPLv1.1 may conflict with the possible choices of the overall license of the project. For example, a choice of any of the GPL licenses (v2 or v3) would make the release incompatible. Most of the reuse by Khan Academy is without any modification; all derived work has been credited according to the requirements of the MPLv1.1.

There exist other alternatives to Flvplayer – OSFlv player (http://www.osflv.com), f4player (http://gokercebeci.com/dev/f4player) and flowplayer (http://flowplayer.org) – all licensed under the GPLv3, that could serve as potential replacements for the less compatible MPLv1.1 licensed Flvplayer. However, we are unsure if the integration of these alternatives would be feasible from a technological perspective.

To notify the developers of Khan Academy of this violation, we opened an *issue* on their new GitHub repository. One of its developers acknowledged the lack of of a license; unfortunately the repository has been moved or deleted at the time of writing this paper.

2) Arduino and Micropendous: Arduino is a GPLv2 licensed (or a later version) software suite for programming for microcontroller boards with the same name (http://code.google.com/p/arduino). Micropendous, like Arduino, is a suite for programming hardware boards with the provision to run Arduino specific software (http://code.google.com/p/arduino). Micropendous contains the Arduino board firmware as a part of its distribution, but

is licensed under the liberal MIT license. The GPLv2 being copyleft in nature, requires the distribution to be to be under a license that does not violate its norms. Micropendous is however, not required to be GPLv3 licensed, since Arduino is only *distributed* and not modified or linked. Therefore, this violation can be rectified by changing its license to *Other Open Source*, and choosing a combination of the MIT and GPLv2 license, as the overall project license, i.e., the libraries (Arduino) remain under their original license (GPLv2) and the Micropendous specific code uses the MIT license. In order to have the overall projects license as the MIT license, the developers of Micropendous would have to find an alternative Arduino library that released under a license not more restrictive than the MIT license.

We posted the details of this violation to the Micropendous discussion forum. The project license has since then, been changed to *Other Open Source*.

3) Miranda and Toptoolbar: Miranda is a multi-protocol instant messenger licensed under the GPLv2, or a later version (http://code.google.com/miranda). The GPL, by its copyleft virtue requires all modified and derivative works to be released under the GPL. Toptoolbar (http://code.google.com/p/toptoolbar) is a plugin/extension that adds a toolbar for quick access of functions in the Miranda IM client and is released under the LGPLv3 (or a later version). This violation falls under the both the categories. *Type 1*, because the GPL requires all derivative works to be released under the same license, but the derived works in the repository are not licensed. At the same time, the developers of Toptoolbar have the option to choose from either the GPLv2 or a later version, namely the GPLv3 for the reused Miranda SDK code. Choosing the GPLv2 can be ruled out, since the GPLv2 and the LGPLv3 are incompatible; whereas, although the GPLv3 and LGPLv3 are compatible, choosing the GPLv3 would require Toptoolbar to be conveyed under the GPLv3. This leads to a violation of *Type 2*. This impasse can however, still be solved by releasing Toptoolbar under the GPLv2 (or a later version) and licensing all derived works under the same license. The code reused from the Miranda SDK consists of user interface components that are Miranda specific and thus results in non-availability of alternatives.

4) Miranda and Wi2geoplugin: Wi2geoplugin is another plugin/extension that enables location based sharing in the Miranda IM client and is licensed under the permissive MIT license (http://code.google.com/p/wi2geoplugin). By using and linking against the GPL'ed code of Miranda, Wi2geoplugin forms a *derivative work*, which is required to be released under the GPLv2 (or a later version) and thus is a violation of *Type 2*. Like toptoolbar, Wi2geoplugin borrows code from the Miranda SDK to build and extend the user interface and hence, makes it difficult to locate alternatively licensed code. However, by licensing Wi2geoplugin under the GPLv2 (or a later version), its developers can avoid such a violation.

We contacted the developers of both, Toptoolbar and Wi2geoplugin describing these violations and seeking their opinion, but did not receive any correspondence in return till the time of writing this paper.

VII. Conclusion & Future work

With a large number open source components just a click away, license compatibility is quickly turning into an intricate scenario, that needs to be dealt with diligence. The legal complications involved in using open source licenses is imperative to the success of any project. Crucial to the core of this study is the collection of open source projects from project hosting websites; intuitively, one may not expect mature GNU projects to be in violation.

It is important to emphasize the validity of our results. While we have focused on reuse as a metric to detect violation, we are unsure of the manner in which the code is actually used inside the project. Examining the projects for source comments, commit history and documentation may offer further insight into license use.

We believe that there is scope for automation in detecting violations and offering possible solutions to these problems. In this study, we suggest two possible solutions to counter violations, either by tweaking the overall license of the project, or by suggesting replacements for the reused libraries under a different license. Both these solutions can be integrated into the existing programmer productivity toolchain, with advancements in code search engines that now enable filtering of code on license. We are currently working on a tool to achieve this.

Acknowledgments

We are grateful to Alwyn Roshan Pais for his comments and feedback on detecting code reuse. We would also like to thank Gervase Markham and Clint Adams for useful discussions on open source licensing. This work is supported by a Microsoft Research India travel grant.

References

[1] D. M. German and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 188–198. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070520

[2] "License proliferation," accessed April, 2012. [Online]. Available: http://www.opensource.org/proliferation

[3] "License proliferation report," accessed April, 2012. [Online]. Available: http://www.opensource.org/proliferation-report

[4] "Busybox and the gpl prevail again - updated 4xs," accessed April, 2012. [Online]. Available: http://www.groklaw.net/article.php?story=20100803132055210

[5] "Best buy, samsung, westinghouse, and eleven other brands named in sflc lawsuit," accessed April, 2012. [Online]. Available: http://www.softwarefreedom.org/news/2009/dec/14/busybox-gpl-lawsuit/

[6] "Emacs license violation," accessed April, 2012. [Online]. Available: http://lists.gnu.org/archive/html/emacs-devel/2011-07/msg01155.html

[7] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 76–85. [Online]. Available: http://doi.acm.org/10.1145/872757.872770

[8] G. v. Krogh, S. Spaeth, and S. Haefliger, "Knowledge reuse in open source software: An exploratory study of 15 open source projects," in *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07*, ser. HICSS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 198.2–. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2005.378

[9] S. Haefliger, G. von Krogh, and S. Spaeth, "Code reuse in open source software," *Manage. Sci.*, vol. 54, no. 1, pp. 180–193, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1287/mnsc.1070.0748

[10] A. Mockus, "Large-scale code reuse in open source software," in *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, ser. FLOSS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 7–. [Online]. Available: http://dx.doi.org/10.1109/FLOSS.2007.10

[11] H.-F. Chang and A. Mockus, "Evaluation of source code copy detection methods on freebsd," in *Proceedings of the 2008 international working conference on Mining software repositories*, ser. MSR '08. New York, NY, USA: ACM, 2008, pp. 61–66. [Online]. Available: http://doi.acm.org/10.1145/1370750.1370766

[12] M. Sojer and J. Henkel, "License risks from ad hoc reuse of code from the internet," *Commun. ACM*, vol. 54, no. 12, pp. 74–81, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043174.2043193

[13] C. Brown, D. Barrera, and D. Deugo, "Figd: An open source intellectual property violation detector," *Proceedings of the 21st International Conference on Software Engineering Knowledge Engineering SEKE2009*, pp. 536–541, 2009. [Online]. Available: http://scs.carleton.ca/ cbrown7/papers/seke09-figd.pdf

[14] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra, "Finding software license violations through binary code clone detection," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 63–72. [Online]. Available: http://doi.acm.org/10.1145/1985441.1985453

[15] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, May 2009. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2009.02.007

[16] "Open source document," accessed April, 2012. [Online]. Available: http://opensource.org/docs/osd

[17] "Open source license data," accessed April, 2012. [Online]. Available: http://osrc.blackducksoftware.com/data/licenses/

[18] "How to use the gpl licenses for your own software," accessed April, 2012. [Online]. Available: http://www.gnu.org/licenses/gpl-howto.html

Code provider [provider activity]	Code acceptor [acceptor activity]
Lufa-Lib [High]	Micropendous [High], Embedded-Projects [High], Usb-Travis [High], Hiduino [Medium]
Arduino [High]	Pushpak [Medium], Micropendous [High], Wireplant [Low], Easyrobot [Medium]
Libsquish [Medium]	Libhplasma [Medium], Nvidia-Texture-Tools [Medium]
Guichan [Low]	DB-Tins07 [Medium], DB-Speedhack07 [Medium], Naruto-Hand-Signs-Fighting [Low]
Upp-Mirror [High]	Boxvivd [Medium], Upp-Mac [Low]
Portableproplib [Medium]	Xbps [High]
Chipmunk-physics [Medium]	Chipmunk-Space-Manager [Medium], Cocos2d-x [Medium], Cocos2d-iPhone [High]
Box2d [Low]	Quickanoid [Low], Emo-Framework [High], Upp-Mirror [High], Cocos2d-iPhone [High], Cocos2d-x [Medium], Party-Family[Medium], Cocos2d-Android [Medium]
Skia [High]	Cocos2d-x [Medium]
Cocos2d-iPhone [High]	CCjoystick [Medium], Cocos2d-x [Medium], chipmunk-spacemanager [Medium]
Kissxml [Medium]	Parallax-Scrolling-Videogame [Low], Xmppframework [High]
Cocoahttpserver [Medium]	Runtimebrowser [High], Xmppframework [High]
Cocoaasyncsocket [High]	Mjpeg-iPhone [Medium], Cocoahttpserver [Medium]
Cocoalumberjack [Medium]	Cocoahttpserver [Medium]
Syphon-Framework [Medium]	Syphon-Implemenatations [Medium]
Miranda [High]	Miranda-Twitter-Oauth [Medium], Mirandaimplugins [Low], Dbmmapmod [Medium], Pboonplugins [Medium], Pescuma [Medium], Toptoolbar [None], Wi2geoplugin [None]
Mirandaimplugins [Low]	Dezeath [Medium]
Juced [Medium]	Ugen [Medium]
Gwen [High]	Party-Family [Medium]
Msinttypes [Low]	Omega-Cronus [Low], Mockcpp [Medium], Soar [Medium], Networkpx [Medium], Ossbuild [High] Sacd-Ripper [Medium], Foxpilot [Medium], Test-NG-PP [Medium], 3ceamu [High], Wagic [High]
Libjingle [High]	Pescuma [Medium], Gtalkbot [High], Ipcamera-For-Android [Low]
Growl [High]	Growlmail [High], Quicksynergy [Low], Sequel-Pro [High], Kaincode [Medium], Welly [High]
Gtm-Oauth [Medium]	Etsycocoa [High]
Google-Toolbox-For-Mac [Medium]	Precipitate [Medium], Update-Engine [Low], Mocean-Sdk-Ios [High], Blazingstars [Medium]
Codesuppository [Medium]	Meshimport [Medium]
Gtm-Http-Fetcher [Medium]	Gtm-Oauth [Medium], Etsycocoa [High], Gtm-Oauth2 [Medium], Google-Api-Objectivec-Client [Medium], Gdata-ObjectiveC-Client [High]
Gtm-Oauth2 [Medium]	Google-Api-ObjectiveC-Client [Medium], Gdata-ObjectiveC-Client [High]
Gdata-Objectivec-Client [High]	Update-Engine [Low], Precipitate [Medium], Google-Email-Uploader-Mac[Medium], Vidnik[Low]
Mockcpp [Medium]	Test-NG-PP [Low]
Effocore [None]	Effogpled [Low]
Googletest [Medium]	Cpp-Library-Project-Template [Low], Easyrobot [Low], Party-Family [Low], Slimdx [High]
Support [High]	Winx [Low], Adlaird [Low], Avbin [Low], Postgres-Kit [High], Libdgnsc [Low], Duplicate-Windows [Low], Doom-Android [Low]
Android-Wifi-Tether [High]	Android-Wired-Tether [High]
Jmonkeyengine [High]	Jme-Glsl-Shaders [Medium], Jmonkeyplatform-Contributions [Medium]
Jbox2d [High]	Plar [Medium], Angry-Food [Low]
Siphon [High]	Csipsimple [Medium]
Flvplayer [Medium]	Khanacademy [High]

TABLE III
INSTANCES OF CODE REUSE IN THE SET OF PROJECTS