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 these components such that, the licenses of the individual components do not conflict with each other and if applicable, the overall license of the application. Such license incompatibilities lead to violations, having far-reaching legal consequences. While proprietary software developers are at the risk of not satisfying the terms of OSS licenses, a large degree of code reuse with in the OSS community poses similar threats too. By examining a set 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. To help evaluate such instances, we look for the reuse of files having the same content, which represents an upper bound on the reuse. We discover 6 violations, which we characterize based on the rules set by the licenses involved. In addition, we present statistics on code reuse with in the set of projects.**

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

I. Introduction

Over the years, the increasing popularity of open source software over the Internet has created a collaborative environment consisting of software components, which can be used to provide a variety of functionality. These components may include projects or parts of projects, that can be plugged into new software 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).

The degree to which such a component may be reused is generally defined by the license that accompanies it. For example, some components of the FFmpeg library are distributed under the GNU Lesser General Public License (LGPL) version 2.1 or later, which while supporting free software, enables a certain degree of reuse in proprietary software. Each license imposes certain restrictions and allowances; but due to wide variety of approved open source licenses available (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, but the latter is not backwards compatible with components that are released under version 2 only (not upgradable to version 3 or a later version). In cases where the developers have no choice, other than to combine components released under incompatible licenses, they may choose to resolve such an impasse by forming a new license, that is compatible with the each of the licenses of the individual components. The writing of new licenses to combat the license mismatch problem is known as *license proliferation* [2]. License proliferation leads to further incompatibilities and is thus discouraged strongly by the Open Source Initiative (OSI, http://opensource.org). The OSI has set up a License Proliferation (LP) Committee to specifically tackle this problem [3].

Corporate firms especially are at a big risk of using OSS without complying with the terms of their licenses, which is why most have a legal department that deals with any such discrepancies. Perhaps the most famous GPL violation involved the use of BusyBox, which provides Unix utilities for embedded devices, in Monsoon Multimedia Inc.'s proprietary software, which resulted in a US court case with BusyBox being supported by the Software Freedom Law Center (SFLC). The case resulted in a compensation fee being paid by Monsoon Media Inc. to the developers of BusyBox [4]. In 2009 many software companies were found using BusyBox in their software without complying to the terms of the GPL which again resulted in a lawsuit [5]. In order to prevent such scenarios from arising, companies like BlackDuckSoftware (http://www.blackducksoftware.com/), OpenLogic (http://www.openlogic.com/) and Palamida (http://palamida.com) have started providing consulting, tools and services to firms that plan on using OSS in their products. Code search engines like Krugle (http://krugle.org) and Koders (http://koders.com) provide advanced code search options, which include filtering down search results by licenses, for code that is available out in the open.

Due to the high degree of code reuse in the open source community, we believe that open source developers too have similar concerns. For example, Emacs (http://www.gnu.org/software/emacs/), a widely used GPL'ed

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

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 empirical study, we aim to discover cases of license violations in vast array of open source projects by tracking cases of code reuse, and subsequently validating them by checking for license compatibility. Comparing such a large code base can potentially be time and resource consuming. 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 III briefly describes open source licensing, Section IV 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 researched previously. Through a case study involving 15 open source projects, von Krogh *et al.* [8] show there is active reuse of code, algorithms and methods in the open source community. Haefliger *et al.* [9] examine 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.

The vast field of code clone detection tools have been surveyed in [10]. 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.

There is a notable lack of large scale analysis of code repositories that track reuse of OSS. Audris Mockus [11] 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 [12]. The authors report that comparing files based on their content produces results with fewer false positivies 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 that could be used by developers to solve conflicts and compatibility issues between open source licenses. Sojer *et al.* [13] 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) [14] and Binary Analysis Tool (BAT) [15] – have been developed. We, however, are looking for license violations on the source code level, rather than the binary level.

III. 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* (OOS) option to use all other licenses.

We use notations similar to

IV. Sample set selection

We started by identifying sources of large open source project repositories. Of particular importance to us was that every project in such a corpus be released under any popular open source license(s), allowing clear compatibility checks. We evaluated 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 that are managed by developers over the web. Google Code offered a set of 10 popular open source licenses to choose from, while SourceForge offered over 80 license options. GitHub encouraged developers to indicate the license in a COPYING/LICENSE file. However, when such a file was absent the license of the project remained unknown. All three services allowed the project files to be browsed online or cloned to a local drive.

We chose Google Code Hosting over the other two options due to the concise set of licenses it offered, which made it easier to identify cases of 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 with tags like 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 while *None* indicates it has had very little/no activity.

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

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 approach we followed to detect code reuse in the sample set of open source projects (Section IV).

A. *Defining code reuse and violations*

Before describing the ideas behind this study, it is crucial to establish our 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.

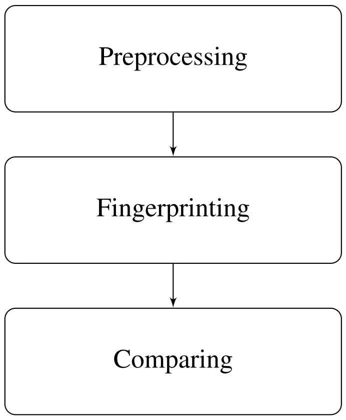

Fig. 1. Architecture of MOSS

Let X and Y denote two distinct open source projects from our corpus, with Y (the acceptor) reusing/deriving code that exists in X (the provider). Let $L1$ denote the license of the original code in X as and $L2$ denote the license of the derived/reused code in Y. When $L1 \neq L2$, we categorize the following cases as violations:

1) $L1$ and $L2$ are incompatible,
2) $L1$ and $L2$ are compatible, but $L1$ is of copyleft nature.

In cases where $L2$ was *Other Open Source*, and the author of the code failed to explicitly specify a license or vice versa, we considered them as special cases of Type 1. A hypothetical example for violations of Type 1 could be: $L1$ is the MPLv1.1 and $L2$ is the GPL (v2 or v3) as these licenses are incompatible. A similar example for Type 2 would be: $L1$ is licensed under the GPL (v2 or v3) and $L2$ is MIT licensed. Although the GPL (v2 or v3) is compatible with the MIT license, the former requires the latter to be covered under the GPL as well.

Plagiarism detection tools are often used to find small pieces of text/code similarity and are used in academic settings for checking assignments turned in by students and conference submissions. Our approach is based on the plagiarism detection tool MOSS, which starts by building hashes of k–grams of source code files, and then selecting files that have the most common hashes to compare. This helps in scaling the comparison process efficiently, while keeping the it fundamentally, independent of the programming language. MOSS is available as a Internet service, and since we couldn't use it in its current form, we built a modified version to suit our purpose of detecting clones.

B. *Architecture*

The architecture of this system is shown in Figure 1 and consists of three phases – Preprocessing, Fingerprinting and Comparing. The Preprocessing stage removes all superfluous features of the text such as whitespace, capital letters, new lines etc., which are not desirable differences between text 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 may choose to modify variable names to evade such tools. However, it is believed that in the open source world, developers do not resort to such techniques and code reuse is mostly, *black–box* [9]. For this reason, we do not modify or replace any variable declarations.

Once the source code is preprocessed, the Fingerprinting stage starts by dividing it into k–grams, which are continuous substrings of size k. The total number of k–grams generated for a string of size n are $n-k+1$. These k–grams are hashed, and subsequently, a subset of these are selected as the fingerprint of a file. Assuming the hashes are collision free, 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:

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

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

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

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

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

Thus, calculating the hash of $H(i+1)$ from $H(i)$ requires two additions and two multiplications, which makes hashing successive k–grams 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.

As stated before, 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 reduce efficiency. In order to counter this, it is preferred to select a subset of these hashes, and store that 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 fingerprint of the file in a relational database, with the schema of the first table being, (file_id, project_name, file_name) and the second table being (file_id, hash, line_numbers). The file_id attribute from the first table is its primary key and also serves as a foreign key for the second table.

The Comparing 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 make sure 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 user to place legal boilerplate at the beginning of a file, which may lead to increasing the number of matched hashes between files. To avoid this, we hash the headers of all licenses that Google Code allows and ignore all such hashes when matching files.

Finally, to highlight sections of a pair of files that were matched, we used the line_numbers attribute to aid us in visually classifying the match as a genuine case of code reuse or a false positive.

VI. Results

A. Repository statistics

Through our sample selection process, we retrieved 1,423 projects equaling 340,164 text files, consisting of about 69 million lines of code.

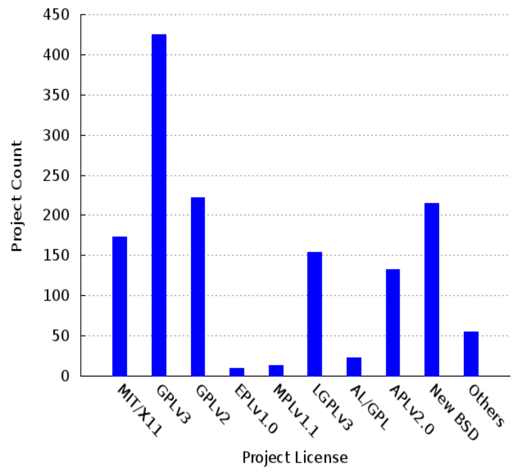

Fig. 2. License statistics in the sample set

Figure 2 shows the number of projects in each license category. The GNU GPLv3 and GPLv2 were among the most widely used licenses and this is in accordance with their popularity in the open source space. The EPLv1.0 and MPLv1.1 were the least used licenses, since they are incompatible with the GPL and prohibit further reuse. Both these licenses are specific to the Eclipse and Mozilla community and generally, are rarely used outside of those communities.

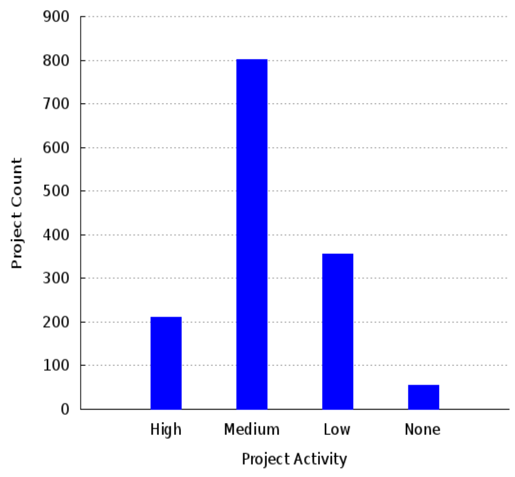

Fig. 3. Activity of projects in the sample set

Figure 3 shows the count of projects, activity wise. While the activity of a project can change over time, depending on numerous factors, we capture the activity status at a particular instance during the course of this study. Most projects were *Medium*-active and only a few were *None*-active.

B. Code Reuse

Our initial tests were to minimize collisions when hashing source code files using the procedure described in Section V-B. The value of k is important to report matches that are not spurious and depends on the type of document. In practice, setting the value of k to 40 seemed to detect cases across most programming languages. In our first experiment, we hashed about 10 MB of text and selected m to be a 32-bit prime number. We observed a total 128 collisions; switching to a 64 bit prime number led to zero collisions. A total of 31,187,119 hashes were generated as a result of the Fingerprinting phase. Independently testing the threshold size t, we found that a lower bound value of 45 works reasonably well in detecting similar files, even across different programming languages. While we did find false positives, it is worth noting that ignoring hashes of legal license headers helped reduced them to a minimum.

We discovered a total of 103 cases of code reuse between projects, the details of which are presented in Table II. Like corporate firms, which have ratings and certification for code that can be reused within the firm, open source developers too look for the quality of the code, because poorly written code can be detrimental to any system; popularity of code can serve as a proxy certification in the open source world [9], that is, projects that were actively developed and updated were reused more. This is supported by Figure 4, which shows the activity levels of projects that were reused. Although *High* and *Medium* active projects were almost equally used (16 each), it is worth nothing that *High* active projects constitute only 14.8% , but *Medium* active projects constitute 56.3% of the total projects. Subsequently, the reuse rate for *High* active projects (7.6%) is considerably higher than *Medium* active projects (2.0%).

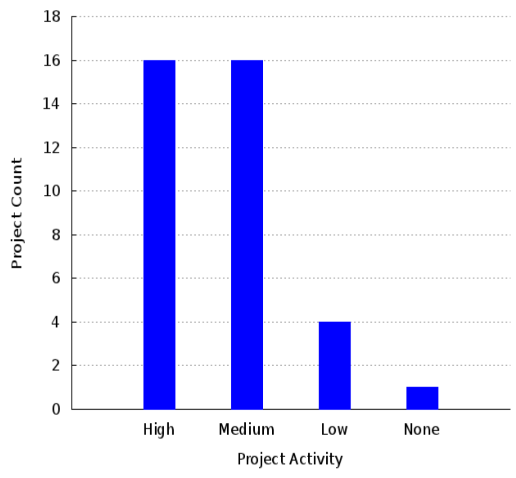

Fig. 4. Activity of reused projects

C. License violations

Table I lists the cases of license violations out of the cases of code reuse presented in Table II. A total of 6 license violations were observed, with the the GPLv2 being violated five times. The *Type of violation* column denotes the category of license violation discussed in Section V. It is important to note that the license mentioned on the project home page (selected while creating the project) does not necessarily reflect the license of the source code. It is only when all conditions of the license are met, the project can claim to be licensed under it.

For example, to apply the GPLv3 to any project Foobar, the minimum actions required are [17]:

- Provide a copy of the GPLv3
- Include the following GPLv3 header template at the top of all the source files of that project:

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

In 4/6 cases of violations, we found that the required steps of the acceptor license were either not followed correctly or

Code provider [provider license]	**Code acceptor [Acceptor license]**	**Acceptor license used correctly?**	**Type of violation**
flvplayer [MPLv1.1]	khanacademy [OOS]	not applicable	1
miranda [GPLv2]	toptoolbar [LGPLv3]	no	2
miranda [GPLv2]	wi2geoplugin [MIT]	no	2
miranda [GPLv2]	miranda-twitter-oauth [GPLv3]	no	1
mockcpp [GPLv3]	test-ng-pp [LGPLv3]	no	2
siphon [GPLv2]	csipsimple [GPLv3]	yes	1

TABLE I
LICENSE VIOLATIONS CASES

were incomplete. Therefore, these 4 cases cannot be truly be considered as cases of violations, as the license of the acceptor project in each of these cases is unclear. We classify such violations as *violations of recommended practice*, where the *recommended practice* refers to meeting the requirements specified by the license. In other words, had the developers of the acceptor project followed the conditions of the license they intended to use, the provider project's license would have been violated in the process.

In the following sections, we describe each case in detail along with the licenses in violation. We also present if using alternate libraries could sort out the violations wherever applicable.

1) Flvplayer: Flvplayer (http://code.google.com/p/flvplayer) is a flash player library that can be embedded into websites for streaming purposes, and is released under the MPLv1.1, which provides a limited amount of copyleft by requiring modifications to MPLv1.1 files and files that copy MPLv1.1 code to be released under the same license. It is used by the Khan Academy, an online e-learning portal that provides video lectures for a variety of subjects on its website (http://code.google.com/p/khanacademy) to stream the flash content. By choosing *Other Open Source* as its license choice, the developers of the Khan Academy website are required to specify the license explicitly, which they fail to do [18](issued by us on their active GitHub repository), resulting in a violation of Type 1. Thus, the Flvplayer library is linked with the unlicensed (essentially copyrighted) website source files and this makes it crucial to understand the options available to license the files that *use* the Flvplayer code. Such files cannot be released under licenses that are incompatible with the MPLv1.1, for example, the GPLv2 or GPLv3.

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, but they would require the developers of Khan Academy's website to release the linked files under the GPLv3. However, we are unsure if the integration of these players would be ideal from a technology perspective.

2) Miranda: Miranda (http://code.google.com/miranda) is a multi-protocol instant messenger released under the GPLv2 with an option for the licensee (in this case, the developers that reuse the code) to choose a later version – the GPLv3. We discover the following three cases of violations:

- 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). Since the licensee can choose either of the GPLv2 or a later version, namely the GPLv3 for code from the Miranda SDK, this leads to two different kinds of violations. The GPLv2 is incompatible with the LGPLv3, and hence leads to a violation of Type 1, where as the GPLv3 is compatible with the LGPLv3, but requires Toptoolbar to be conveyed under the GPLv3 and leads to a violation of Type 2. The code used from Miranda consists of GUI components that are Miranda specific and thus leads to the non-availability of alternatives. This impasse can be solved by releasing Toptoolbar under the GPLv2 (or a later version).

- Wi2geoplugin (http://code.google.com/p/wi2geoplugin/), is another plugin/extension that enables location based sharing in the Miranda IM client (also for Skype and Quiet Internet Pager [QIP]) released under the permissive MIT license. However, by using and linking against the GPL'ed code of Miranda, Wi2geoplugin forms a *derived work*, which is required to be released under the GPLv2 (or a later version) and is a violation of Type 2. Again, the code borrowed from Miranda consists of GUI components among others, which are essential to create and integrate Wi2geoplugin to Miranda. Since the reused set of code is specific to Miranda, it would be difficult to find alternatives, under a more permissive license that are compatible with the MIT license; it would be easier for the developers of Wi2geoplugin to release the files that link to the Miranda SDK under the GPLv2 (or a later version).

- Miranda-twitter-oauth (http://code.google.com/p/miranda-twitter-oauth) is a plugin that adds a Twitter sidebar to the Miranda IM to read/write tweets. Released under the GPLv3, it is compatible with the Miranda SDK code as long as the licensee chooses to follow the terms of the GPLv3 instead of the GPLv2, which is incompatible with the former and a violation of Type 1. The reused code consists of Miranda's content update and GUI components that cannot be replaced with more permissively licensed ones. Thus, by choosing the GPLv3 instead of the GPLv3 for Miranda's components, the licensees avoid running into conflicts.

3) Mockcpp: Mockcpp (http://code.google.com/p/mockcpp) is a mock C++ object creation framework that is available under the GPLv3 license and is used by Test-ng-pp (http://code.google.com/p/test-ng-pp/) a C++ testing framework which is licensed under the LGPLv3. The GPLv2 and the LGPLv3 are compatible licenses, but using GPLv3 code in a LGPLv3 licensed project requires the project to be conveyed under the GPLv3. While the Test-ng-pp project is released under the LGPLv3. the sources contain the GPLv3 header, which makes it difficult to identify its actual license. Assuming a LGPLv3 license, it will be required to be released under the GPLv3 to be compatible with Mockcpp. However if the developers of Test-ng-pp intended to make a LGPLv3 release, then alternate C++ mock object frameworks like GoogleMock (http://code.google.com/p/googlemock/) – released under the New BSD license and Amop (http://code.google.com/p/amop/) – released under the MIT license provide suitable licenses. Note that we are unsure if these frameworks are suitable to Test-ng-pp's use case.

4) Siphon: Siphon (http://code.google.com/p/siphon) is GPLv2 (or a later version) licensed Session Initiation Protocol (SIP)/ Voice over Internet Protocol (VoIP) application for the iOS platform licensed under the GPLv2 (or a later version). Csipsimple (http://code.google.com/p/csipsimple/) is the equivalent for Android phones which is released under the GPLv3. It uses links files relating to G.729a, and audio data compression algorithm used in VoIP written in C, from the Siphon project. This would potentially lead to a violation if the developers of Csipsimple followed the terms of the GPLv2 (and not the GPLv3) for the G.729a code, since the GPLv2 and GPLv3 are incompatible licenses and a violation of Type 1. Note that although G.729a is originally not a part of Siphon; it was modified by the developers of Siphon to suit their needs and subsequently re-licensed under the GPLv2.

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. Unlike corporate firms which have IP lawyers and attorneys assisting them in using OSS, open source developers have no such comforts. Crucial to the core of this study was the use of open source projects from project hosting websites; intuitively, on may not expect mature GNU projects to be violating licenses. By using comparing files on content, rather than file names, we present an upper bound on the reuse of source code files and manually validate license violations.

While Multi-licensing – where the developer may offer the user choices of licenses to use – has to a certain extent sorted license compatibility, it requires a developer to have knowledge about a variety of licenses and may lead to an increase in license proliferation. 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.

We are working on ways to automate licensing options for OSS, by examining the source code and suggesting the choices of licenses that the software can be released under, or what parts of the software need to be replaced in order to shift to a more acceptable license. Such a tool could be integrated with code search engines to provide substitute packages.

Acknowledgments

We are grateful to Alwyn Roshan Pais for his comments and feedback on the code detection tool. We would also like to thank Gervase Markham and Clint Adams for useful discussions on open source licensing.

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] "On behalf of busybox developers, sflc files first ever u.s. gpl violation lawsuit," accessed April, 2012. [Online]. Available: http://www.softwarefreedom.org/news/2007/sep/20/busybox/

[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] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

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

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

[18] "Khan academy website license unspecified," accessed April, 2012. [Online]. Available: https://github.com/Khan/khan-website/issues/3

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 II
CODE REUSE CASES