# Identifying Licensing of Jar Archives using a Code-Search Approach

Massimiliano Di Penta*, Daniel M. German**, Giuliano Antoniol***
* Dept of Engineering, University of Sannio, Italy
** Dept. of Computer Science, University of Victoria, Canada
*** SOCCER Lab. – DGIGL, École Polytechnique de Montréal, Québec, Canada
dipenta@unisannio.it, dmg@uvic.ca, antoniol@ieee.org

***Abstract*—Free and open source software strongly promotes the reuse of source code. Some open source Java components/libraries are distributed as jar archives only containing the bytecode and some additional information. For whoever wanting to integrate this jar in her own project, it is important to determine the license(s) of the code from which the jar archive was produced, as this affects the way that such component can be used.**

**This paper proposes an automatic approach to determine the license of jar archives, combining the use of a code-search engine with the automatic classification of licenses contained in textual files enclosed in the jar.**

**Results of an empirical study performed on 37 jars—from 17 different systems—indicate that this approach is able to successfully infer the jar licenses in over 95% of the cases, but that in many cases the license in textual files may differ from the one of the classes contained in the jar.**

## I. Introduction

The advent of free/open source software (FOSS) has created a large ecosystem of software components that are readily available for download and reuse. Searching for components to reuse is becoming a common activity. This is especially true for Java components. Indeed, Java has a peculiar advantage over many programming languages: a Java application runs virtually on any known software and hardware architecture without the need of recompilation. Java promotes the reuse adoption via componentization and classes bundled into jars (pre-compiled versions of the components). However, one of the major challenges of reusing a component (in source code or binary form–such as a jar) is to know its provenance: who its copyright owner is, and more important, under which license it is made available.

Some code repositories (such as *MVNrepository*[1]) provide access to jar files of thousands of open source components, but not always to their corresponding source code nor any information about where to find it. Thus, the bytecode files contained in the jar represent the only reliable source of information to determine the jar provenance and license.

In essence, when a jar file is downloaded, it might not be clear where its corresponding source code can be found, and under which license the component is made available. Thus it might not be easy to determine if such jar can or cannot be legally used. This paper proposes an approach to automatize the license identification process supporting developers in the cumbersome mining activities required to discover the license of the source code included in a jar.

To understand the nature of the problem consider the following example. A developer in search of a Java component for decoding and playing sound files might stumble into *sonicplayer*, a component to play audio files in various formats. Within the jar file of version 0.8 of *sonicplayer*[2] the only information regarding its origin and copyright owner is found in the file `about.properties`:

---

blurb=This is a project of Eclipsedesktop.org

(c) Copyright Jordi Böhme López and Leif Frenzel 2004-2006. All rights reserved.
Visit http://eclipsedesktop.org for more information.

---

This URL (http://eclipsedesktop.org/) redirects to a "file not found error"). The jar contains no explicit licensing information either, except for a file called `epl-v10.html`, which contains the text of the Eclipse Public License version 1.0 (EPLv1.0). The user is left to imply that this is the license under which *sonicplayer* is made available.

The problems do not stop there. To simplify distribution, a jar file of a component might include inside itself the jar files of other components that it requires. *sonicplayer*'s jar contains on it own four other jar files it requires: `jl0.4.jar`, `j-ogg-all.jar`, `mp3spi1.9.1.jar` and `tritonus_share.jar`. Neither one of these files contain any documentation stating their license or copyright owner, or even where they can be downloaded from. After inspecting the class names of `jl0.4.jar`, and using a Web search engine, a developer will discover it corresponds to *JLayer*, a component to play mp3s[3]. The source code distribution of *JLayer* version 0.4 states that its license is the General Public License v2.0 (GPLv2)—all its files are indeed licensed under the GPLv2.0 (or any version after).

[1] http://mvnrepository.com/

[2] org.eclipsedesktop.sonicplayer_0.8.0.jar

[3] http://www.javazoom.net/javalayer/javalayer.html

This poses an important question: from a legal point of view, is the use of *JLayer* (licensed under the GPLv2) by *sonicplayer* (licensed under the EPLv1.0) appropriate? As described in [1] this is not a trivial question to answer. Nonetheless, it is an important issue that should be resolved by anybody wanting to further distribute a product based on *sonicplayer*. More precisely, the problem faced by the user, integrator or distributor of a bytecode jar is: ***what is the license (or licenses) of the classes included in a bytecode jar file***?

As we have illustrated above, using Web search engines and code inspection one can manually determine the licenses of jar classes. But this is a tedious process, and, considering that a single Java application might contain dozens or even hundreds of different jar components, each containing other jars and hundreds of classes, this is time consuming or even impractical. Moreover, it is not guaranteed that all files contained in the jar can be found on the Web. Finally, for each file to be searched, one has to instantiate a proper query, using information (e.g., class names, package names) sometimes contained in the bytecode.

Our automatic license mining approach infers the license of jar archives and their contained classes. This approach combines various sources of information. First, it uses information decompiled from the bytecode of its classes to query a code search engine, with the objective of finding the classes and packages contained in the jar, and retrieving the license as classified by code search engines (we call this the *inferred* license). Also, it mines the textual files contained in the jar, automatically classifying licenses contained into these files using an existing classifier [2], in order to verify whether the license of these textual files (which we call the *declared* license) is consistent with the licenses inferred for the class bytecode.

The approach has been validated over **37** jars belonging to 17 different projects (various Apache projects, some jar archives distributed with Eclipse, and an MP3 layer). Results indicate that the proposed approach ensures a high percentage—on average 95%–of correct license classification, although it has limitations in the capability of retrieving licenses for specific versions of a class, which can be a cause of imprecision when licenses change over the time [3], and that, very often, the textual license provided with the jar is not fully consistent with those of the source code of the of classes contained in the archive.

The remainder of the paper is organized as follows. Section II presents key concepts on intellectual properties and software licenses while Section III presents related work. Section IV describes in details the approach to mine license information. Section V presents an empirical study, results and some discussions. Section VI concludes the paper and outlines directions for future work.

## II. Licenses and their consequences in open source projects

Almost everywhere in the world, software is protected by copyright legislation [4]. While copyright laws vary from country to country, they are based on common guidelines set by the World Intellectual Property Organization (WIPO). Any fragment of code is protected by copyright at the moment it is created.

Copyright provides its owner several exclusive rights, such as the ability to make copies of the work, and to prepare and distribute derivative works based upon it. Anybody wanting to reuse a software component as part of another software system must possess a software license from its copyright owner that allows such use. A license is a legal mechanism used by the copyright owner (the licensor) to grant permission to others (the licensees) to use and exploit her intellectual property in ways that would otherwise be forbidden by law [5], [6], [7].

Any FOSS component is licensed under one or more open source licenses. The Open Source Initiative lists 65 different open source licenses, but many more exist (see opensource. org). One of the major challenges of reusing open source components is making sure that its license is compatible with the intended use of the component (see [1] for a detailed discussion of the legal issues related to the integration of open source components in either proprietary or open source software). As a consequence, it is paramount for the licensee to know the license of any FOSS component that is to be integrated as part of a bigger product.

FOSS licenses are primarily classified into two types: permissive and reciprocal. Permissive licenses are those that allow the licensor to reuse the software (whether it is a software component, a file, or a fragment of code) under other licenses (which includes proprietary). The best known permissive licenses are the three different variants of the BSD license (original–also known as four clauses, the 3-clauses, and the 2 clauses), the MIT/X11 license and the different versions of the Apache Public License (APL). Reciprocal licenses, on the other hand, are designed to constraint the creator of a derivative work that uses the licensed software (again, a component, a file, or a fragment of code) and require her to license the resulting software under the same license; for example, every version of the GNU General Public License (GPL) requires the derivative work to be licensed under the exact same version of the GPL. Some licenses, such as the Eclipse Public License (EPL), the Mozilla Public License (MPL) and the GNU Lesser General Public License (LGPL) sit somewhere in the middle: they can be permissive under certain circumstances, but might be reciprocal under others.

## III. Related Work

This section describes related literature and tools, concerning three topics (i) code search approaches and engines,

(ii) analysis of software licensing, and (iii) the state of practice in the field of mining code ownership and licenses.

### A. Code Search Approaches and Engines

Code searching is the process of retrieving source code artifacts relevant to a particular query. Thus, overall, it is comparable to any Information Retrieval (IR) activity, although the techniques being used are relatively different, often consisting of a combination of pure IR techniques with some code analysis and comparison activities. In code search, typically, the query consists of terms relevant for the code artifact to be searched: for example, if one wants to search for a function/method that performs sorting, examples of queries could be "sort", "bubble sort", "quick sort" etc. Advanced queries—available in some engines—allow for using regular expressions, specifying the language of the code being searched, etc. The output is a ranked list—ordered by relevance—of the source code artifacts found. As it done in general-purpose search engines, the code is indexed using automatic crawling of Web sites and source code repositories, although it is also possible for a user to submit specific source code for indexing.

Notable examples of code search engines are Google Code Search[4], Krugle [5] and Black Duck™ Koders[6]. All of them report, together with the list of relevant code artifacts found, their license, classified according to an internal license classifier.

In recent years, more sophisticated code search approaches, sometimes combining existing code search engines with other methods, have been proposed in literature. The relevance of this topic in the software engineering community can also by perceived by the presence of a specific workshop—the workshop on search-driven development - users, infrastructure, tools and evaluation (SUITE) [8]—co-located with ICSE.

Holmes *et al.* [9] developed an approach to automatically derive queries for code repositories from the context obtained by analyzing the code the developer is writing through the IDE. Thummalapenta and Xie [10] proposed *ParseWeb*, a tool able to find source code fragments that, obtaining as input a given (set) of object(s), return a target object. Lemos *et al.* [11] used a test-driven development approach to support code search, *i.e.*, the objective of their search was to find source code artifacts that pass a set of test cases. A test-driven approach is also used by *Code Conjurer*, proposed by Hummel*et al.* [12]. The approach proposed by Reiss [13] combines various techniques, *i.e.*, simple keyword matching, slicing and static analysis, test-driven and contract verification techniques; it also encompasses the use of code transformations to modify the found code to produce the desired output from the available inputs.

[4]http://www.google.com/codesearch
[5]http://www.krugle.org/
[6]http://www.koders.com/

Recently, Bajracharya *et al.* [14] performed an empirical study aimed at characterizing code search activities performed by Koders' users, by mining a large portion of the Koders log. Specifically, they investigated on the length of the queries performed, the topic being searched for, or the number of queries performed before a code download.

Overall, although code search is a hot topic, its current use is mainly related to promote reuse rather than—with the exception of Koders—for licensing purposes, as we propose in this paper.

### B. Analysis of Software Licensing

Software licensing patterns have been recently studied by German *et al.* [1] using a formalization of licenses as logical formulae constraining what can and cannot be done with a system. They introduced several legal patterns, along with examples of occurrences of these patterns. In a previous work [15], we presented a study of the influence of software licenses on code migration between the FreeBSD, Linux, and OpenBSD kernels. Our findings support the hypothesis of a preferential code flow induced by permissive licenses from FreeBSD and OpenBSD (BSD licenses) towards Linux (General Public license).

Hindle *et al.* [16] discovered that many of the largest commits of a software project correspond to changes to the licenses or copyright owners of files. In a recent paper, Di Penta *et al.* [3] investigated to what extent licensing changes occur in software systems because of requirements due to the way software is used, or because of the evolution of software licenses themselves. Finally, a related paper by Di Penta and German [17] studied changes occurring to copyright owner names, finding that explicit contributors and copyright owners are not necessarily the most frequent committers, although they are often added during larger changes than average.

### C. State of Practice

The relevance of the problem is also supported by the existing code search engines supporting mining activities similar or leading to results closed to what proposed in this paper. All of them, in fact, are able to report license information for the source code files relevant to a query. More relevant to this works is Koders, as it was also conceived to assess whether some source code one is using/integrating is violating any copyright or license. This makes Koders very related to our work, with the following differences:

1) we propose a method to identify the licensing of a jar archive using code search tools. To this extent, our approach is complementary to Koders—*i.e.*, Koders could be used in place of Google Code Search to get the license of each class part of the package once classes have been extracted from the archive and queries to be performed (class and package names) have been instantiated. Then, we use the heuristics described in

Section IV to identify the likely license(s) of the archive by aggregating licensing information from all the classes, and also text files—contained in it;

2) at least in this version of the approach, our search is performed using very simple queries, *e.g.*, class and package names, rather than performing a thorough code comparison. This makes the proposed approach quite lightweight and, as shown in Section V-A, accurate enough without the need for performing code comparison, which would be however part of our future work.

Overall, we share with previous works and code search engines the idea that software licenses are key to the software development and evolution. However, this paper focuses on a different perspective, aiming at supporting the developers license mining manual activity conceptually improving over existing code search engines as with our approach the entire mining process is automated.

## IV. The proposed approach

This section details the various steps of the proposed approach for inferring licenses of Java classes through code search.

*Step 1: Extracting files from the Jar archive:* The first step of the approach consists in extracting, from the jar archive, (i) textual (often .txt, XML or HTML files) and (ii) class files, which will be used in the following steps as a source of information to identify licenses.

*Step 2: Identifying and classifying licenses contained in textual files: the declared license of each file:* This steps aims at analyzing textual files found in jar archives to determine whether they contain licenses. We want to know if such licenses—hereby referred as the "declared license" of the jar—is consistent with the license inferred from its class files (described below). We use the license identification tool FOSSology [2] for this purpose. FOSSology identifies licenses using the Binary Symbolic Alignment Matrix (bSAM) pattern matching algorithm, matching the text files against license templates contained in a database. FOSSology is able to detect a wide variety of specific licenses (version 1.1 is capable of recognizing 360 different ones). Specifically, we use FOSSology's `fosslic` to identify the license in any file that does not have the a `.class` extension.

*Step 3: Retrieving licensing info for Java files from Google Code Search: the inferred licenses:* This step constitutes the core of the proposed approach. It aims at inferring the licenses for class files contained in the jar archive. First, we extract, from the bytecode, the class and package name[7] using the ASM bytecode analysis library[8]. Qualified package and class names are then used to query Google Code Search;

[7]The same information could be extracted by looking at the path name, however we prefer to make our approach as resilient and generic as possible, and not dependent on the path name of the files within the jar archives.

[8]http://asm.ow2.org/

Table I
COMMONLY USED LICENSES

| Name | Version |
|---|---|
| Apache | 1.0, 1.1, 2.0 |
| Common Public License (CPL) | 0.5, 1.0 |
| IBM Public License (IPL) | 1.0 |
| Eclipse Public License (EPL) | 0.5, 1.0 |
| GNU Affero General Public License (AGPL) | 1, 3 |
| GNU General Public License (GPL) | 1, 2, 3 |
| GNU Lesser General Public License (LGPL) | 2, 2.1, 3 |
| MIT License | Many versions |
| BSD License | Many versions |
| Mozilla Public License (MPL) | 1.0, 1.1 |
| Netscape Public License (NPL) | 1.0, 1.1 |

the output of Google Code Search consists of a ranked list of entries containing, among other fields:

- the name of the file being matched;
- the file's author, if available;
- the file's license;
- a hyperlink to a page showing a cached version (on Google Code Search) of the source code of the file entry being found;
- one or more links to sources where the original file can be found. These sources might be SVN or CVS repositories, or FTP or Web addresses.

The file's license is clearly the most important piece of information we retrieve.

When reporting the license of a file, Google Code Search takes a cautionary approach. Frequently, it only reports the name of the license but not its version; for example, it might report that a file is licensed under the GPL, but not under the GPL version 3. Some well-known licenses, such as the Apache version 1 are reported as BSD (the Apache version 1 is a variant of the BSD license). From a legal point of view, every version of a license is a different license (i.e. version 2.0 and version 3.0 of the GPL are two independent licenses, each with its own language, and set of grants and conditions). Knowing the precise version under which a component is licensed is important. For instance, a person wanting to reuse a component that is licensed under the old BSD license (also known as 4-clauses BSD) will have to satisfy requirements crediting the authors in any advertisement of the resulting product (regardless of its license), but the 3-clauses BSD does not contain such requirements. Similarly, a person developing software under the GPL version 2 might not be able to integrate a component licensed under the GPL version 3.

Google Code Search querying is performed through a Java application relying on the GData API[9], a suite of APIs to build applications that interact with various features provided by Google, ranging from the search engine to the Google Maps and Google Code Search. For the purpose of this work, the tool sends two queries to Google Code Search for each `.class` file. The first query contains the fully qualified class

[9]http://code.google.com/apis/gdata/

name (i.e., the class name containing its package name), while the second contains the package name only.

The application we developed for retrieving licenses from Google Code Search takes as input a list of class file names, and produces an XML output containing:

- a tag *class*, indicating the class being searched for;
- a tag *classpackage*, indicating the class package (as said, two queries are sent to Google Code Search, one with the qualified class name, another with the package name)
- a set of *file* items, containing results found. In turn for each item it is indicated:
  - the file *name*,
  - whether it is a Java source (*javasource* tag),
  - whether the file name matches the qualified class name (*classmatch*),
  - or whether the file name matches the package name (*packagematch*) and,
  - finally, the file *license*.
- In addition, the *file* tag encloses other information regarding the file, such as *author*, and about where the file can be retrieved from (the *uri* and the *archive* name).

*Step 4: Combining the different sources of information:* The final step of the proposed approach aims at determining the license of class files contained into the jar archive, based on the results obtained from Google Code Search, and, finally, determining the likely license for the whole jar archive. Specifically, for each class file, its license is determined as follows:

1) if, when searching through Google Code Search exactly the same class was found, and the Google Code Search entry contains a license for that class, then it is assumed that the license indicated by Google Code Search is the license of the class.
2) if Google Code Search was not able to find any class with the same qualified class name, or reported no license available for that class, then it is assumed that the license for the class is the most frequently used license among all entries belonging to the same package. For instance, if a jar file is composed of 10 files, of which 7 are under the GPL v2, one under another license, and for 2 we could not retrieve the class or its license, then it is assumed that these two files are under the GPL v2 (the most commonly found license).

Finally, based on the inferred licenses (those associated to each class of the jar), and based on the declared licenses (those found into textual files–Step 2 or the approach) we determine:

1) the distribution of inferred licenses,
2) which is the most frequent inferred license. Such a license is deemed to be the likely license of the whole archive;
3) when the jar archive contains a declared license, it is reported if such a license matches the most frequent license found in the step above or not.

Table III
COMPARISON OF THE PRECISION OF LICENSE CLASSIFICATION TOOLS.

| | Ninka | Fossology |
|---|---|---|
| Accuracy | **96.6%** | 55.0% |
| Execution Time | **22s** | 923s |

## V. EMPIRICAL STUDY

The *goal* of this study is to analyze the proposed method to infer licenses for jar archives. The *quality focus* is the capability of the method to correctly infer licenses, and to check their conformance with those declared in textual files within the jars. The *perspective* is of researchers, wanting to evaluate the performance of the proposed approach, and of system integrators, who want to use jars archives in their systems, but want to know under what licensing conditions these jars can be used. The *context* consists of 37 jars coming from (i) 10 different projects from *www.apache.org*, (ii) 5 third-party jars included in the Eclipse[10] bundle, and (iii) 2 different releases of a Java MP3 player library, named *JLayer*[11]. Table II provides a thorough overview of the jars used in this empirical study.

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

- **RQ1:** *How does the inferred license compare to the licenses stated in the source code?*
- **RQ2:** *Are licenses inferred for class files consistent with those declared in the documentation of the jar?*

To address **RQ1**, we use the approach described in Section IV to retrieve, from Google Code Search, the licenses of *.class* files contained in jar archives. At the same time, we take the source code from which the jars were produced, and classify the licenses contained in it using the automatic license identification tool Ninka [18]. Ninka was created to identify the license of source code files only and outperforms FOSSology in both accuracy (reporting the correct license name and version in a source code file) and speed (runs an order of magnitude faster) as illustrated in Table III[12]. While Ninka performs a detailed classification, Google Code Search returns, most of the times, only the license family (as explained in Section IV), thus we consider the classification as correct if the license family returned by Google Code Search is the same as the family of the license returned by Ninka. Once a classification of source code licenses has been performed, we compare the *.class* licenses with

[10]http://www.eclipse.org

[11]http://www.javazoom.net/javalayer/javalayer.html

[12]Ninka was not designed for finding licenses in text files, and for this reason we used FOSSology when searching for licenses in text files.

Table II
JAR ARCHIVES USED IN THE STUDY.

| Application | Description | Jars | Classes |
|---|---|---|---|
| APACHE | | | |
| commons-attributes-2.1 | Enables C#/.Net-style attributes in Java | 3 | 36 |
| commons-beanutils-1.8.2 | Wrapper for Java introspection API | 5 | 137 |
| commons-betwixt-0.6 | Maps Java beans into XML | 1 | 135 |
| commons-chain-1.2 | Implementation of Chain of Responsibility | 3 | 61 |
| commons-cli-1.0 | Parsing of command-line options | 3 | 22 |
| commons-codec-1.1 | Encoders and decoders (*e.g.*, Base64, Hex, Phonetic and URLs) | 1 | 14 |
| commons-logging-1.1.1 | Library to log various information | 6 | 83 |
| commons-pool-1.5.4 | DB Connection Pooling | 3 | 52 |
| collections-3.2.1 | Collection of data structures | 4 | 497 |
| apache-log4j-1.2.15 | Manages execution logging | 1 | 259 |
| ECLIPSE BUNDLE | | | |
| com.jcraft.jsch_0.1.28 | Java Secure CHannel | 1 | 95 |
| com.sun.syndication_0.9.0 | Feed management | 1 | 120 |
| javax.servlet.jsp_1.2.0 | Servlet/JSP API | 1 | 25 |
| javax.xml.ws_2.0.0 | Java Web services library | 1 | 41 |
| org.mozilla.javascript_1.6.2 | JavaScript engine (Rhino) | 1 | 199 |
| JLAYER | | | |
| JLayer 0.4 | Java MP3 library | 1 | 142 |
| JLayer 1.0 | – | 1 | 142 |
| TOTAL: | | 37 | 2060 |

the source code licenses, and report the percentage of correct classifications. We compute the percentage of correct classifications considering both licenses for cases where a class with exactly the same name was found on Google Code Search—in the following we refer these as *Found* licenses—and licenses *Inferred* considering, in case it was not possible to find on Google Code Search exactly the same file, the majority of licenses for files belonging to the same package. We statistically compare these two proportions using the Fisher's exact test [19], which is a non-parametric test used to compare proportions.

**RQ2** aims at investigating whether the license (if any) declared in textual files contained in the jar archive reflects the licenses of all *.class* files, of the majority of them, or if at least is compatible with the license of these files. To this aim, we compare the declared license obtained in *Step 2* with the set of *.class* licenses obtained in *Step 3*.

### A. Results

This section reports results of the empirical study above defined, with the objective of addressing research questions **RQ1** and **RQ2**. Raw data used for our analyses are available for replication purposes[13].

*1) RQ1: How does the inferred license compare to the licenses stated in the source code?:* Table IV shows, for each jar, the number (and percentage) of classes for which it was possible to find a license on Google Code Search, by exactly matching the fully qualified class name. As explained in Section IV, the proposed approach infers the licenses using a two-stages approach, *i.e.*,(i) searching from the same class and (ii) inferring the license of the class by determining the license of the majority of classes belonging to the same package. The purpose of Table IV is, thus, to just provide an idea of how many classes can be found on Google Code Search, and what their license is. As it can be seen, such a percentage oscillates between 7% only for *common-betwixt*, up to 62% for *commons-logging*. Also, the table shows the number and percentages of different licenses found: the projects from the Apache foundation used Apache licenses including few BSD licenses; *org.mozilla.javascript* used primarily the MPL; *JLayer* with an almost equal number of GPL and the LGPL files; the other Eclipse jars used a mixture of Apache, BSD with some GPL files.

Table V is similar to Table IV, however it shows the number and percentages of licenses inferred by finding classes belonging to the same package of the class being analyzed. This is basically the final result we rely on. As it can be seen, in this case the percentage of resolved classes is very high, always higher than 91%, and 100% for 13 out of 16 projects. Overall, the percentages for different licenses remain consistent with Table IV, with some exceptions (i) the percentage of non-Apache licenses for Apache applications is lower, even for *commons-codec* where it decreases from 75% to 43%; (ii) for *JLayer* the percentage of GPL (59%) is higher than LGPL (41%), while the other way around (47% and 53%) was found in Table IV.

While Table V indicates the ability of our approach to infer licenses for .class files, it is now necessary to evaluate the correctness of such a license inference. Figure 1 shows the percentage of correctly classified licenses for (i) cases where it was possible to find a file matching the class name, referred as *Found* and (ii) cases where the license was *Inferred* from the license of the majority of classes belonging to the same package of the class being analyzed. For the *Found* cases, the percentage oscillates from 29% of *commons.codec* to 100% of *commons-attributes*, *commons-*

[13]http://www.rcost.unisannio.it/mdipenta/licsearch-data.tgz

Table IV
NUMBER AND PERCENTAGES OF FILES FOUND ON GOOGLE CODE SEARCH.

| Application | Overall found (%) | Found Licenses (%) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AGPL | GPL | LGPL | BSD | Apache | MPL | MIT |
| commons-attributes | 5 (14%) | – | – | – | – | 5 (100%) | – | – |
| commons-beanutils | 62 (45%) | – | – | – | 6 (10%) | 56 (90%) | – | – |
| commons-betwixt | 9 (7%) | – | – | – | 3 (33%) | 6 (67%) | – | – |
| commons-chain | 18 (30%) | – | – | – | – | 18 (100%) | – | – |
| commons-cli | 6 (27%) | – | – | – | 2 (33%) | 4 (67%) | – | – |
| commons-codec | 8 (57%) | – | – | – | 6 (75%) | 2 (25%) | – | – |
| commons-logging | 71 (62%) | – | – | – | 1 (1%) | 70 (99%) | – | – |
| commons-pool | 26 (50%) | – | – | – | 3 (12%) | 23 (88%) | – | – |
| collections | 81 (16%) | – | – | – | 9 (11%) | 72 (89%) | – | – |
| apache-log4j | 102 (39%) | – | – | – | – | 102 (100%) | – | – |
| com.jcraft.jsch | 22 (23%) | – | – | 2 (9%) | 20 (91%) | – | – | – |
| com.sun.syndication | 28 (23%) | – | – | – | – | 28 (100%) | – | – |
| javax.servlet.jsp | 14 (56%) | – | 1 (7%) | 2 (14%) | – | 11 (79%) | – | – |
| javax.xml.ws | 22 (54%) | – | 16 (73%) | 2 (9%) | – | 4 (18%) | – | – |
| org.mozilla.javascript | 51 (26%) | 1 (2%) | – | – | – | 1 (2%) | 47 (92%) | 2 (4%) |
| JLayer 0.4 | 86 (61%) | – | 40 (47%) | 46 (53%) | – | – | – | – |
| JLayer 1.0 | 86 (61%) | – | 40 (47%) | 46 (53%) | – | – | – | – |

Table V
NUMBER AND PERCENTAGES OF LICENSES INFERRED USING GOOGLE CODE SEARCH.

| Application | Overall inferred (%) | Inferred Licenses (%) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AGPL | GPL | LGPL | BSD | Apache | MPL | MIT |
| commons-attributes | 36 (100%) | – | – | – | – | 36 (100%) | – | – |
| commons-beanutils | 268 (98%) | – | – | – | 6 (2%) | 262 (98%) | – | – |
| commons-betwixt | 135 (100%) | – | – | – | 5 (4%) | 130 (96%) | – | – |
| commons-chain | 61 (100%) | – | – | – | – | 61 (100%) | – | – |
| commons-cli | 22 (100%) | – | – | – | 2 (9%) | 20 (91%) | – | – |
| commons-codec | 14 (100%) | – | – | – | 6 (43%) | 8 (57%) | – | – |
| commons-logging | 115 (100%) | – | – | – | 1 (1%) | 114 (99%) | – | – |
| commons-pool | 52 (100%) | – | – | – | 3 (6%) | 49 (94%) | – | – |
| collections | 495 (100%) | – | – | – | 10 (2%) | 485 (98%) | – | – |
| apache-log4j | 236 (91%) | – | – | – | – | 236 (100%) | – | – |
| com.jcraft.jsch | 95 (100%) | – | – | 2 (2%) | 93 (98%) | – | – | – |
| com.sun.syndication | 120 (100%) | – | – | – | – | 120 (100%) | – | – |
| javax.servlet.jsp | 25 (100%) | – | 1 (4%) | 2 (8%) | – | 22 (88%) | – | – |
| javax.xml.ws | 41 (100%) | – | 35 (85%) | 2 (5%) | – | 4 (10%) | – | – |
| org.mozilla.javascript | 198 (99%) | 1 (0.5%) | – | – | – | 1 (0.5%) | 194 (98%) | 2 (1%) |
| JLayer 0.4 | 142 (100%) | – | 84 (59%) | 58 (41%) | – | – | – | – |
| JLayer 1.0 | 142 (100%) | – | 84 (59%) | 58 (41%) | – | – | – | – |

*chain*, *apache-log4j*, and *com.sun.syndication*, with an average of 82% and a median of 89.5%. For the *Inferred* cases, the percentage oscillates between 62% of *JLayer 1.0* to 100% of 12 different projects, with an average of 95% and a median of 100%. Overall, it can be noticed that the idea of inferring licenses by looking at those of the same package of the class works very well in terms of classification correctness—other than in terms of completeness of classes analyzed—and, excepts than for one case (*com.sun.syndication*), it overcomes the correctness for *Found* licenses alone. Comparing the number of correct and wrong classifications over the *Found* and *Inferred* licenses, the Fisher's exact test indicates a significant improvement when using the *Inferred* licenses (p-value $< 0.001$).

Finally, Table VI shows, for both kinds of classifications, the occurrences of various kinds of misclassifications. A comparison between inferred licenses and licenses found in the source code is also shown in Table VII. As it can be noticed, most of them are between LGPL and GPL (and vice versa) and, especially for *Found* licenses, between *BSD* and *Apache*. The mismatching between Apache and BSD is mainly due to *commons-codec*, where files are licensed under the Apache version 1.1 license and this license is derived from the BSD (and hence can be considered a variant

Table VI
CASES OF INCORRECT CLASSIFICATIONS.

| Source lic. | Inferred lic. | Num. occ. |
|---|---|---|
| FILE NAME FOUND | | |
| LGPL | GPL | 48 |
| GPL | LGPL | 40 |
| BSD | Apache | 29 |
| Apache | GPL | 4 |
| LGPL | Apache | 2 |
| LGPL | BSD | 2 |
| MIT | MPL | 2 |
| Apache | MPL | 1 |
| AGPL | MPL | 1 |
| GPL | Apache | 1 |
| INFERRED FROM PACKAGE | | |
| GPL | LGPL | 52 |
| LGPL | GPL | 42 |
| BSD | Apache | 7 |

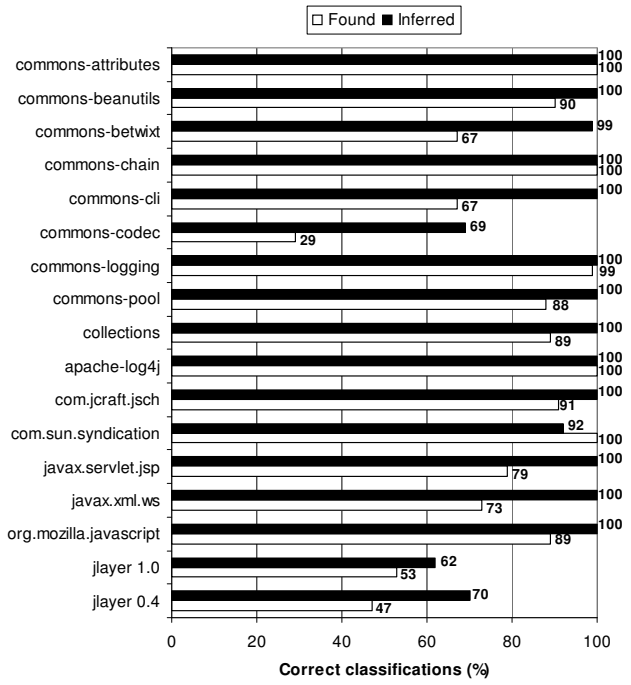


Figure 1. Percentage of correctly classified licenses for files (F)ound and for licenses (I)nferred from the package.

of the BSD). The mismatching between GPL and LGPL is mainly due to *JLayer*. As it can be noticed in table V, the inferred licenses are exactly the same in both releases (0.4 and 1.0). However, as shown in Table VII, licenses in the source code were all GPL in release 0.4 and all LGPL in release 1.0. The partially wrong classification was due to the fact that the application changed license between the two releases, and the current implementation of the code search (and our approach) was not able to distinguish between files belonging to different releases. When our method retrieved files for JLayer, it did not distinguish which version they came from.

*2) RQ2: Are licenses inferred for class files consistent with those declared in the documentation of the jar?:* Table VII reports a summary of (i) licenses found in textual files contained in the jar archives (ii) inferred licenses (as from Table V), and (iii) licenses found in the source code. All jars, except *commons-attributes*, *apache-log4j*, and *JLayer*, contained information about the correct license. The upper part of the table highlights an almost perfect consistency for the Apache-related projects where an Apache v2 license was declared in the jars, and matched the ones in the source code and those inferred (although Google Code Search did not return the exact license version). When looking at the Eclipse jars, we can notice that:

- for *com.jcraft.jsch*, *com.sun.syndication*, and *javax.servlet.jsp*, the Eclipse developers added an EPL license to the ones of the code, which were Apache v2 or BSD 3 found in *com.jcraft.jsch* is, actually, a BSD 3 license);
- *javax.xml.ws* declared a CCDL v.1.0 license, while the code was actually GPL (with Classpath exception), and this was correctly inferred in 85% of the cases. This is likely due to the change of license of *javax.xml.ws*: its latest source code contains the newer license, but the documentation contains the older one;
- *org.mozilla.javascript* (Rhino) declares a Netscape Public License (NPL), while Google Code Search reports a MPL, belonging to the same family.

### B. Lessons learned

This section summarizes some insights we learned from the study we performed.

*1) The textual information does not tell the whole story:* Some text files that accompany the jar archives do not fully describe the licensing of the class files the archive contains. For example, the Rhino's jar file packaged by Eclipse we analyzed (`org.mozilla.javascript_1.6.2.tar.gz`) indicates that its license is NPL v1.1, but does not indicate that, as an alternative, the user is allowed to select the GPL (no version is indicated) instead of the NPL v1.1.

*2) Some licenses are incorrectly classified by the Code Search Engine:* To the best of our knowledge, this is the first empirical study that attempts to evaluate the accuracy of the license reported by Google Code Search. As we have observed, sometimes the license reported for a file is incorrect. The most common error we observed was classifying the Apache License as the BSD. In some cases licenses were simply not reported by Google Code Search.

*3) Code Search engines are not doing fine-grained license identification:* Code search engines are currently restricting their results to license families, and do not report the exact license version. As licenses continue to evolve, this becomes very important. Also, files are frequently licensed using sophisticated licensing schemes, such as disjunctive or conjunctive licenses, special exceptions for particular purposes, etc. Unfortunately, these nuisances are not documented in the results of code search engines.

*4) Some Packagers use "boiler-plate" licensing information:* The jars we downloaded from Eclipse always include a file called `about.html` with licensing information. This file included the following statement: *"The Eclipse Foundation makes available all content in this plug-in ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 1.0"* and further below included a section called *Third Party Content* that listed the original name and source of the code and its license. In several of these jars we analyzed, no single source code file was licensed under the EPLv1.0. While these statements are not incorrect, they might be misleading to automatic license identification tools that are not able to understand the nuisances of the wording.

Table VII
SUMMARY OF LICENSES DECLARED IN TEXTUAL FILES, INFERRED, AND EXTRACTED FROM THE SOURCE CODE.

| Application | Declared | Inferred | Source |
|---|---|---|---|
| commons-attributes | – | Apache (100%) | Apache v2 (100%) |
| commons-beanutils | Apache v2.0 | (BSD 2%), Apache (98%) | Apache v2 (100%) |
| commons-betwixt | Apache v2.0 | BSD (4%), Apache (96%) | Apache v2 (100%) |
| commons-chain | Apache v2.0 | Apache (100%) | Apache v2 (100%) |
| commons-cli | Apache v2.0 | BSD (9%), Apache (91%) | Apache v2 (100%) |
| commons-codec | 'Apache v1.1'-style | BSD (43%), Apache (57%) | Apache v1.1 (100%) |
| commons-logging | Apache v2.0 | BSD (1%), Apache (99%) | Apache v2 (100%) |
| commons-pool | Apache v2.0 | BSD (6%), Apache (94%) | Apache v2 (100%) |
| collections | Apache v2.0 | BSD (2%), Apache (98%) | Apache v2 (100%) |
| apache-log4j | – | Apache (100%) | Apache v2 (100%) |
| com.jcraft.jsch | EPL v1.0, BSD-3 | LGPL (2%), BSD (98%) | BSD 3 (100%) |
| com.sun.syndication | EPL v1.0, Apache 2.0 | Apache (100%) | Apache v2 (100%) |
| javax.servlet.jsp | EPL 1.0, Apache v1.1 | GPL (4%), LGPL (8%), Apache (88%) | Apache v2 (100%) |
| javax.xml.ws | CDDL v1.0 | GPL (85%), LGPL (5%), Apache (10%) | GPL v2 ClasspathException (100%) |
| org.mozilla.javascript | NPLv1.1 | AGPL (0.5%), Apache (0.5%), MPL (98%), MIT (1%) | MPL (99.5%) MIT-old (0.5%) |
| JLayer 0.4 | – | GPL (59%), LGPL (41%) | GPL v2 (100%) |
| JLayer 1.0 | – | GPL (59%), LGPL (41%) | LGPL v2 (100%) |

### C. Threats to Validity

This section discusses the threats to validity that can have affected the study we performed to validate the proposed approach.

First, there could be threats to *construct validity*, mainly related to imprecision of the classifications we performed. For what concerns the tools being used to classify source code and text licenses, we have reported the performances in Table III. We do not have exact measures for what concerns Google Code Search license classification, although the inspection we did on a few of its results confirmed the correctness of the classification performed. On the other hand, we are aware that there is no guarantee, right now, that the file being found is the same version of the one in the jar, and that different versions could have different licenses. As we reported in [3], it is not uncommon to see software change its license. As also noticed when reporting results for **RQ2**, this is the case with some of the packages that we analyzed. For instance, *JLayer* (included in sonicplayer) has seen its license change from the GPL v2 (the one included in the jar) to the LGPL v2. .

As the reader can notice, many of the analyzed projects have an Apache license. This would make one wonder whether in this case a constant classifier would have worked fine. However, it must be clear that the purpose of this work is not to do a classification, but to just retrieve the license classified by the code search engine and compare it with the one in the source code. For this reason, we also believe it is sufficient to just report the percentage of correctly inferred licenses, without reporting a confusion matrix. Also, the overall set of projects covers seven different kinds of licenses.

Regarding *conclusion validity*, we mainly report descriptive statistics, plus for comparing the performances of the classification over the *Found* and *Inferred* licenses we use the non-parametric Fisher's exact test.

Finally, for what concerns threats to *external validity*, *i.e.*, the extent to which our results can be generalized, although we analyzed 17 different applications with different licenses, further studies are surely desirable. Above all, we mainly assessed the performances using jars from well-known open source systems. It could happen that, for jars composed of less-common code, the number for files for which it could not be possible to retrieve a license from the code search engine would higher. Similarly, the organizations behind these jars (Apache, Eclipse, Sun) are very good at including a license in each of their files. This might be the case for other jars. Last, but not least, it would be interesting to replicate the study using code search engines different from Google Code Search.

## VI. CONCLUSIONS AND WORK-IN-PROGRESS

Knowing the license of code released in jar archives is a crucial issue when integrating these jars in software systems. This paper proposed an approach to infer licenses of bytecode files contained into jars by using code search engines. Results of an empirical study we performed over 37 jars from 17 different projects indicate a high percentage—95% on average—of correctly inferred licenses. The study also shows that sometimes Java classes are re-packaged into jars—as it happened for jars released with the Eclipse project—and a new license is added to the jar.

Although the proposed approach exhibits good performances, there are a number of improvements we are going to investigate in our work-in-progress. In particular, an improved version of the approach would also download the code found on the code search engine, with the objective of:

- retrieving the same version of the class being analyzed, to avoid wrong classifications due to license changes;
- comparing the downloaded code with the decompiled class to check whether the found class actually corresponds to the analyzed one. This can be done us-

ing clone detection/code fingerprinting approaches and tools; and
- performing a more detailed classification of the licenses contained in the downloaded source code, using tools such as FOSSology or Ninka.

Last, but not least, we intend to perform a more extensive validation of the approach through a larger collection of jars.

## VII. Acknowledgements

We would like to thank Dr. Ian Bull from Eclipse Source for helping us understand the problem of licensing of jars. D. German was supported by Hewlett-Packard to support the FOSSology Project. G. Antoniol was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chair in Software Evolution #950-202658).

## References

[1] D. M. Germán and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. IEEE, 2009, pp. 188–198.

[2] R. Gobeille, "The FOSSology project," in *MSR '08: Proceedings of the 2008 International Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2008, pp. 47–50.

[3] M. Di Penta, D. M. German, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the evolution of software licensing," in *Proceedings of the ACM/IEEE 32rd International Conference on Software Engineering (ICSE 2010) 2-8 May 2010, Cape Town, South Africa*, 2010.

[4] World International Property Organization, "CRNR/DC/94 WIPO Copyright Title," Dec 1996.

[5] A. M. S. Laurent, *"Understanding Open Source and Free Software Licensing"*. O'Reilly, 2004.

[6] D. McGowan, "Legal Aspects of Free and Open Source Software," in *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, Eds. MIT Press, 2005, pp. 211–226.

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

[8] S. K. Bajracharya, A. Kuhn, and Y. Ye, "Suite 2009: First international workshop on search-driven development - users, infrastructure, tools and evaluation," in *ICSE Companion*, 2009, pp. 445–446.

[9] R. Holmes, R. J. Walker, and G. C. Murphy, "Approximate structural context matching: An approach to recommend relevant examples," *IEEE Trans. Software Eng.*, vol. 32, no. 12, pp. 952–970, 2006.

[10] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*. ACM, 2007, pp. 204–213.

[11] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, P. C. Masiero, and C. V. Lopes, "Applying test-driven code search to the reuse of auxiliary functionality," in *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, 2009, pp. 476–482.

[12] O. Hummel, W. Janjic, and C. Atkinson, "Code conjurer: Pulling reusable software out of thin air," *Software, IEEE*, vol. 25, no. 5, pp. 45–52, Sept.-Oct. 2008.

[13] S. P. Reiss, "Semantics-based code search," in *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, 2009, pp. 243–253.

[14] S. K. Bajracharya and C. V. Lopes, "Mining search topics from a code search engine usage log," in *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009, Vancouver, BC, Canada, May 16-17, 2009*, 2009, pp. 111–120.

[15] D. M. German, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol, "Code siblings: Technical and legal implications," in *Proc. of the 2009 Working Conference on Mining Software Repositories, MSR 2009*, 2009, pp. 81–90.

[16] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us? a taxonomical study of large commits," in *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, May 2008, pp. 99–108.

[17] D. M. German and M. Di Penta, "Who are source code contributors and how do they change?" in *16th Working Conference on Reverse Engineering (WCRE 2009)*, 2009, pp. 11–20.

[18] D. M. German, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," Under review, available at http://turingmachine/~dmg/papers/, 2009.

[19] S. D.J., *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.