JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 9, No. 4, December 2015, pp. 190-203

Crowdsourcing Identification of License Violations

Sanghoon Lee
Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Korea
sanghoon@postech.edu

Daniel M. German
Department of Computer Science, University of Victoria, Canada
dmg@uvic.ca

Seung-won Hwang*
Department of Computer Science, Yonsei University, Seoul, Korea
seungwonh@yonsei.ac.kr

Sunghun Kim
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
hunkim@cse.ust.hk

Abstract
Free and open source software (FOSS) has created a large pool of source codes that can be easily copied to create new applications. However, a copy should preserve copyright notice and license of the original file unless the license explicitly permits such a change. Through software evolution, it is challenging to keep original licenses or choose proper licenses. As a result, there are many potential license violations. Despite the fact that violations can have high impact on protecting copyright, identification of violations is highly complex. It relies on manual inspections by experts. However, such inspection cannot be scaled up with open source software released daily worldwide. To make this process scalable, we propose the following two methods: use machine-based algorithms to narrow down the potential violations; and guide non-experts to manually inspect violations. Using the first method, we found 219 projects (76.6%) with potential violations. Using the second method, we show that the accuracy of crowds is comparable to that of experts. Our techniques might help developers identify potential violations, understand the causes, and resolve these violations.

Category: Human computing

Keywords: Crowdsourcing; Software license; Violation; Clone detection

I. INTRODUCTION

Reusing code is a common practice in software development. However, it may come with a price. According to copyright law, the only individual who is allowed to copy a program (or portions of it) is the copyright owner

Open Access http://dx.doi.org/10.5626/JCSE.2015.9.4.190 http://jcse.kiise.org

Received 02 November 2015; **Accepted** 23 November 2015
*Corresponding Author

and those authorized by the owner such as employees during the course of their work and those who have received a license to make such a copy. Any other copying of the source code by a third party could be heavily penalized. For example, a United States Court has ruled that copying 54 lines out of 160,000 is considered as a copyright infringement or when the copied code is sufficiently important to the operation of the program [1].

To illustrate the importance and complexity of copyright problem in software, in the case Oracle of America vs. Google, a jury has determined that Google has indeed copied a few lines of code (method rangeCheck) from Oracle [2]. It is very important to realize that, even though a lot of codes share similarities (considered as clones by a clone detector), the only code that is found to be a copy is the one that has clear traceability, such as the one between Oracle and Google. In the verdict, a developer admitted that he had made those copies. This reinforces the importance of using clone detection in combination with analysis of the history of the code to determine any potential license violation.

Besides complexity, free and open source software (FOSS) presents challenges in tracing a large pool of source code created daily. In fact, one of the most important features of FOSS is that it can be copied to create new applications as long as the person who makes the copy agrees to conditions imposed by the license of the copied code. Copying source code across FOSS is well documented [3, 4]. Sojer and Henkel [5] have interviewed several hundred developers and discovered that copying FOSS code by commercial enterprises is common. However, in many cases, software developers lack the understanding about the legal risks associated with such activity. Their organizations also lack policies to guide them. As described previously [4], embedding source code from another copyright owner can create several challenges, including the following:

- Provenance: Tracking the copyright owner and license of the copied code.
- License Analysis: Determining if the license of the original code allows the intended use of the copy.
- Evolving the copy as the original changes. Managing the evolution of the copy with respect to the original is difficult.

Manual inspection of these files to find potential violations is time consuming and labor intensive.

Our goal was to achieve scalability of the identification process for potential violations by proposing machine-based tools and crowdsourcing. Specifically, we used the following methods:

Machine-based Framework: We developed a tool that could automatically detect cloned parts and license inconsistencies. Such information can be used to automatically narrow down potential violations to reduce workload. These potential violations can then be reviewed by human workers to guide human decisions (in Section III).

Empirical Validation of Machine-based Approaches: We empirically determined the quantity of potential violations. We found that potential license violations were widespread. A total of 219 out of 286 open source projects (76.6%) had potential license violations (in Section IV).

Crowdsourcing Case Study: Once we narrowed down the potential violations, instead of presenting all potential violations to experts, we proposed to simplify the task by distributing the tasks to general crowds. With information obtained from machine-based algorithms as guidance (e.g., highlighting cloned parts and file level relevance scores), crowds made identical decisions with experts (in Section V).

II. OVERVIEW

This section provides an overview for the decision process in the identification of license violation similar to those described in the literature [6]. The decision process is depicted in Fig. 1. Its input is a file to be verified for potential license violation. It consists of four steps that

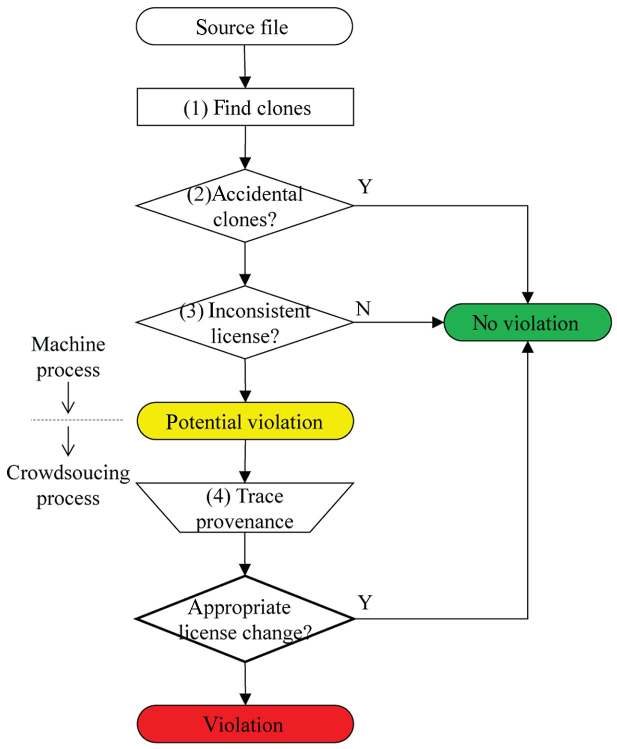


Fig. 1. Decision diagram to identify potential license violations between clones.

we will deal with using either machine-based algorithms or crowdsourcing (marked as [M] and [C], respectively).

1. **Find clones [M]**: First we identified the fragment level clones using existing machine- based algorithms. We exploited characteristic vector based clone detection tools [7, 8]. This method matched some part of the input code to others in our code corpus.
2. **Are they accidental clones? [M]:** Since not all clones result from copying, we proposed a machine learning approach to classify clones as *accidental* clones and intentionally copied files.
3. **Identify license inconsistency [M]**: We identified licenses of each clone file by using existing machine-based tool FOSSology [9]. If two clones had different licenses, we raised a flag based on the tool.
4. **Determine the provenance and evolution of the files [C]**: Based on *potential* violations identified by machines, human experts could look into potential problem areas to determine which file is the source, which is the copy, at what moment the copy was made, what were the license statements of the files at the moment of copying, how were they evolved, and whether the clones were involved in copyright violation.

In Section III, the first machine-based approaches to identify potential violations, through Sections III-A, III-B, and III-C, are discussed. In Section IV, each method of machine-based process was evaluated. In Section V, how crowdsourcing can contribute to the fourth step and validate its effectiveness in case studies is discussed.

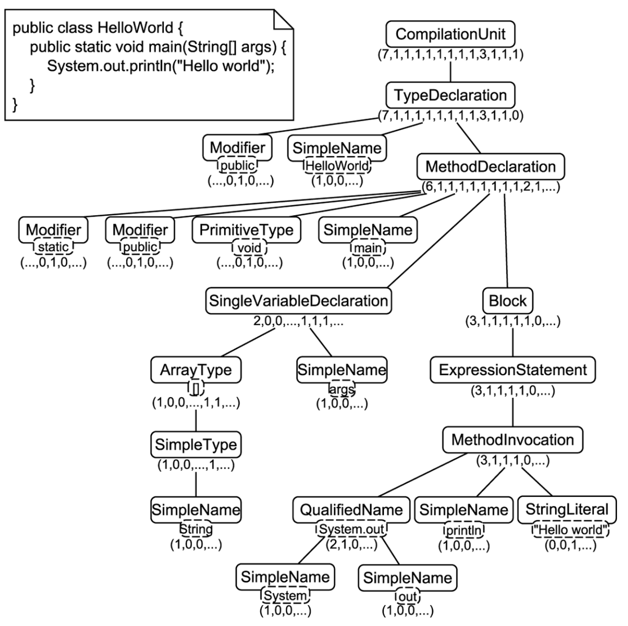
III. MACHINE IDENTIFICATION OF POTENTIAL LICENSE VIOLATIONS

In this section, the detailed process of three machine-based steps are discussed. Methods are introduced in line with steps in decision rules described in Section II. First, we found clones (Section III-A). If the clones are intentionally copied codes (Section III-B), licenses of files containing the clones (Section III-C) are then identified.

A. Finding Clones

Code fragments that are textually, syntactically, or semantically similar are called code clones. Various and efficient clone detection techniques have been proposed by different researchers [7, 8, 10, 11].

We leveraged efficient clone detection tools [7, 8] for large corpus of source files. We fist represented code using *characteristic vectors.* That is, we generated abstract syntax trees (ASTs) (Fig. 2) of the source code and approximated the trees into vectors to find clones with high tree similarity. Vectors were tagged at the bottom of the nodes. The vector in root node indicated that



```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

Fig. 2. Example of vector extraction.

the example code had seven SimpleName syntax units, three Modifier, and one for the rest of each syntax unit. If there was another code whose characteristic vector was the same as the example, the two codes may be clones with a high probability. Therefore, we found clones based on vector similarity.

Once the source code was vectorized, we built an index-based search mechanism by adopting published techniques [8] to detect clone candidates efficiently. We then employed the index in a filter and refinement strategy to find similar vectors. For each vector, we queryed the index to retrieve similar vectors. We computed the exact similarities of result vectors with the query vector and selected only vectors whose similarities were large enough as clones.

B. Classifying Copied Clones

All clones that we found were not results from copying other code. Some of the same codes might have been accidentally created by independent developers. These clones may look alike. However, they are not copied explicitly from each other. We refer to these clones as *accidental clones.* In this case, differences in licenses of accidental clones do not matter since the code is independently written without copying each other. Examples of accidental clones are shown in Fig. 3(a)-(c). Most accidental clones include commonly used code idioms such as for- loops, getter and setter methods, constructors, and exception handling.

On the other hand, *copied clones* are results from intentionally copying of files. An example of copied clones is shown in Fig. 3(d). One of the two clones is cre-

```
public void setScope(String scope) {  | public void setId(String id) {
    this.scopeName = scope;            |     this.id = id;
}                                      | }
```

(a)

```
public Review(String summary,         | public Address(String street,
                Integer vote,          |                   String city,
                Book book) {           |                   String state) {
    this.summary = summary;            |     this.street = street;
    this.vote = vote;                  |     this.city = city;
    this.book = book;                  |     this.state = state;
}                                      | }
```

(b)

```
} catch (Throwable t) {                       |
  ProtocolDecoderException pde;               | } catch (Exception e) {
  if (t instanceof                            |   BuildException be;
          ProtocolDecoderException) {         |   if (e instanceof BuildException) {
    pde = (ProtocolDecoderException) t;       |     be = (BuildException) e;
  } else {                                    |   } else {
    pde = new                                 |     be = new BuildException(e);
          ProtocolDecoderException(t);        |   }
  }                                           |   throw be;
  throw pde;                                  | }
} finally {                                   |
```

(c)

```
final RegExp parseRepeatExp()             | RegExp parseRepeatExp()
    throws IllegalArgumentException {     |     throws IllegalArgumentException {
  RegExp e = parseComplExp();             |   RegExp e = parseComplExp();
  while (peek("?*+{")) {                  |   while (peek("?*+{")) {
    if (match('?'))                       |     if (match('?'))
      e = makeOptional(e);                |       e = makeOptional(e);
    else if (match('*'))                  |     else if (match('*'))
      e = makeRepeat(e);                  |       e = makeRepeat(e);
    else if (match('+'))                  |     else if (match('+'))
      e = makeRepeat(e, 1);               |       e = makeRepeat(e, 1);
    else if (match('{')) {                |     else if (match('{')) {
      int start = pos;                    |       int start = pos;
      while (peek("0123456789"))          |       while (peek("0123456789"))
      ...                                 |         ...
```

(d)

Fig. 3. Clone examples: (a) setter methods, (b) constructors with member field initialization, (c) exception handlings, and (d) copied clones.

ated by copying the other. Unlike accidental clones, if copied clones have different licenses, it can be a potential violation.

For efficient license violation detection, it is important to identify accidental clones and filter them out. Automatic detection of copied clones is generally challenging. However, these clones can be distinguished by experienced developers as shown in Fig. 3(a)-(d).

Our approach to classify accidental and copied clones is by using a machine learner based on manually marked copies and accidental cloned benchmark set. First, we constructed a benchmark set by labeling clones as accidental and copied clones by a human assessor. Then we built a classification model based on machine learning with labeled characteristic vectors. Once a prediction model was built based on the benchmark set, it could classify clones as accidental or copied ones.

C. Identifying License Inconsistency

After identifying clones and filtering out accidental clones, we could simply check the license information of copied clones. When the licenses of copied clones are not identical, they might have potential violations. In this section, we discussed how to detect non-identical license pairs and identify inconsistent combinations.

Licenses were assigned to each file. Licenses in a file did not affect other files with different licenses. When identifying license inconsistencies, it does not matter if there are many source files with different licenses in a project. Therefore, we detected licenses in each file and handled it independently from other files, although these files were in the same project. To identify the license of each copied clone, we used the state-of-the-art license detection tool FOSSology [9].

We compared the identified licenses in the copied clones to find license differences. We then classified them into two main groups according to the degree of violation risk: low and high.

In the low risk group we have:

- **No License**: One clone had a license, but the other did not. There might be three reasons why a file with no license might have a clone with one: 1) the clone was made when the original had no license; the original later changed its license, but the clone still has no license (no potential violation); 2) the original still has no license, but a license was added to the clone (a potential violation); and 3) there was a license in the original file, but it was removed from the clone (a potential violation too).
- **License Upgrade**: One file had one license and the other had a newer license (e.g., Apache v1.1 and Apache v2.0). By manually sampling some of these pairs we found that in all cases it was because the clone was made when the file used the previous version of the license.
- **Added License**: The license of one clone was a superset of the original file. From the point of view of copyright law, a person who copies and then modifies a copyrighted work is the owner of such modifications only. As such, this person can attach a license to such changes or add another license to the cloned file. For example, we observed that the original files came from Apache Foundation projects. The projects that made the copy had added other licenses.
- **License Upgrade and License Added.** One combination was a mixture of an upgrade and an added license. In this case, the clone happened when the older version of the license was in use (Apache v1.1). The original file changed its license (to Apache v2.0), while the clone file had two licenses added to it (CDDL and GPL v2).

In the high risk group we have:

- **Different Licenses**: Both clones had different and unrelated licenses.

According to the above taxonomy of license inconsistency with risk of license violation, we focused on fatal conflicts to resolve. If we know how to change the license in a code, we can classify some license inconsistency into the above risks. In addition, when the code creation time is given, we can judge which code has the possibility of violation. We will explain the process of tracing code provenance by crowdsourcing in Section V.

IV. EVALUATION OF MACHINE-BASED APPROACHES

A. Setting

In order to present distinguishing features about clones with respect to license issue empirically, we first built a large corpus of source code against which we searched for clones. We collected 212,657 Java source files of 286 free and open source software projects that are popularly and actively managed in major large FOSS repositories such as the Apache Software Foundation, Google Code, Java.net, and SourceForge.net as shown in Table 1. Among the projects we gathered, no project was released into two or more repositories. Every project was unique in our corpus.

B. Find Clones

To extract characteristic vectors from each code, we used the parser in the Eclipse Java development tools (JDT; http://www.eclipse.org/jdt/) to build an AST. We generated 83 dimensional vectors representing 83 different AST node types provided by JDT. Each value represented the frequency of each element. We only considered vectors with frequency sum of 50 or above to prune out code fragments of insignificant size. Finally, we linearly normalized each vector to build a unified tool for clones of various sizes.

In our corpus, we generated 1,960,360 vectors from 212,657 files. Once the vectors were extracted, we built an index structure to query similar vectors. We used a relational database management system MySQL to store and index vectors. Although finding clones for a single code fragment using the index could be finished within a second, it took more than ten days to find clones for all code fragments in our dataset consisting of 2 million vec-

Table 1. Statistics of collected FOSS projects

Community	Project	#subproj.	#files	#LOC	#vectors
Apache Software Foundation	Hadoop	8	5,905	1,327,985	81,013
	Xerces	1	823	256,493	13,022
	Jakarta	8	2,717	468,118	20,087
	…	…	…	…	…
	Total	99	97,077	16,929,668	872,740
Google Code	Google Web Toolkit	1	3,556	564,598	28,467
	jMonkeyEngine	1	1,139	270,202	14,267
	FEST	1	3,202	255,836	8,654
	…	…	…	…	…
	Total	36	23,380	3,670,310	199,965
Java.net	GlassFish	1	15,437	2,994,385	142,247
	JavaCC	1	195	34,928	2,552
	LG3D	2	1,005	195,223	5,635
	…	…	…	…	…
	Total	147	91,144	16,965,554	874,667
SourceForge.net	CUBRID Cluster	1	89	18,120	845
	Guacamole	1	36	3,707	249
	…	…	…	…	…
	Total	4	1,061	178,807	12,988
Total		286	212,657	37,572,912	1,960,360

tors. After finding clones, we could reuse this information to detect potential license violations.

We queried the index for each vector to find clones in the same corpus. A total of 639,789 clones were detected. Among them, 168,167 clones were related to more than two projects, and 82,950 files (39%) had cross-project clones.

Due to approximation of source code into characteristic vectors, this approximation may return false answers. Based on [8], both precision and recall of clone identification were around 0.75.

C. Copied Clones

We employed classifier to distinguish copied clones learned from benchmark set. The benchmark set was constructed by labeling clones as accidental and copied clones by a human assessor who had more than 5 years of programming experience. After labeling clones as accidental or copied, we used characteristic vectors [7] of each clone as features, namely the frequency of syntactic elements in the code. For the classification model, we used Support Vector Machines [12, 13], a state-of- the-art classification algorithm that has been shown to be empirically optimal for several learning tasks.

To evaluate our proposed classifier, we designed two simple experiments. First, we learned an SVM classifier using randomly sampled 428 clones. Among the 428 clones, 210 were manually labeled as accidental clones, and 218 were labeled as copied clones. The classifier was first evaluated using 10-fold cross validation with the labeled data. As shown in Table 2, the average accuracy of the classifier was 91.2%.

Unfortunately, it is possible that these samples are not representatives. It may not work properly for new clone inputs. Therefore, we also sampled additional 219 test clones and applied the same classifier to the new test clones. We then manually verified the classification results. Of the 219 samples, 197 clones were correctly classified, i.e., the accuracy was 89.9% (Table 2). The accuracy for the new clones was not significantly decreased compared to the accuracy of cross validation. Many clones were found. Selecting representative clones among them will require too much effort. Instead, we showed that the performance of learner was stable for the new test clones and that the learner was robust.

Our experimental results showed that the proposed classifier was accurate enough to eliminate accidental clones for license violation detection. We applied this classifier and filter out accidental clones. Out of a total of 639,789 clones, 484,460 clones were classified as accidental clones, i.e., more than 75% of clones were accidental clones.

Table 2. Evaluation of the copied clone classifier

Validation model	Accuracy (%)
10-fold cross validation	91.2
219 test set	89.9

D. Identify Licenses

To identify the license of each copied clone, we used the state-of-the-art license detection tool FOSSology [9] version 1.1.0. FOSSology is based on Binary Symbolic Alignment Matrix (bSAM) algorithm [14] for textual comparison of license statements.

Using FOSSology, 92 different licenses were identified from our 286 subjects. The top 10 most used licenses in our subjects are shown in Table 3. Apache Software License v2.0 is the most common one found in our collected subjects. A disjunctive combination of GPL v2, CDDL and LGPL v2.1 was also widely used. A total of 25,367 files (11.9%) and 45 projects (15.7%) from 212,657 files and 286 projects used them.

Approximately 25% of files did not have any license, which was noted as ‘NO LICENSE’ in Table 3. This is consistent with results of other studies. For example, 31.5% and 19% of files had no license [15, 16]. However, FOSSology could not identify licenses in 1.8% of total files, which we marked as ‘UNKNOWN’.

Table 4 shows the most frequent different licenses for clone files found in our subjects. More than half of the files related to license conflicts were caused by empty license notices. We also found many added licenses on Apache licenses. Moreover, there was a significant difference in the number of licenses which might be at high risk of violation.

Results of filtering out the low risk license inconsistent clones are shown in Table 5. A total of 5,095 clones had different licenses, of which 1,104 from 132 different projects were at high risk of potential violations.

Overall, these results show that more than 132 projects

Table 3. Top-10 licenses used in collected FOSS projects

License	#files (%)	#projects (%)
Apache v2.0	113,572 (53.4)	171 (59.8)
NO LICENSE	53,478 (25.1)	227 (79.4)
CDDL, GPL v2	16,595 (7.8)	15 (5.2)
GPL v2	5,547 (2.6)	27 (9.4)
UNKNOWN	3,847 (1.8)	53 (18.5)
INRIA-OSL	3,751 (1.8)	25 (8.7)
MIT Free with copyright clause	1,957 (0.9)	14 (4.9)
LGPL GNU C Library	1,827 (0.9)	14 (4.9)
LGPL v2.1+	1,556 (0.7)	9 (3.1)
Apache v2.0, CDDL, GPL v2	1,338 (0.6)	7 (2.4)

Table 4. Inconsistent license combinations found frequently

Risk of violation	Type	License 1	License 2	#files	#projects
Low	No License	NO LICENSE	Apache v2.0	1,799	178
		NO LICENSE	CDDL, GPL v2	635	61
		NO LICENSE	INRIA-OSL	283	38
		NO LICENSE	GPL v2	217	47
		NO LICENSE	UNKNOWN	163	39
		NO LICENSE	GPL v3	131	15
		NO LICENSE	LGPL v2.1+	87	31
		NO LICENSE	Apache v2.0, CDDL, GPL v2	71	21
		NO LICENSE	MIT	67	17
		NO LICENSE	LGPL GNU C Lib	60	22
	Upgrade	Apache vl.l	Apache v2.0	230	28
	Add	Apache v2.0	Apache v2.0, CDDL, GPL v2	1,224	42
		Apache v1.1	Apache v1.1, CDDL, GPL v2	150	2
	Upgrade & Add	Apache v1.1, CDDL, GPL v2	Apache v2.0	202	15
High	Different	Apache v2.0	CDDL, GPL v2	1,030	86
		Apache v2.0	GPL v2	259	79
		Apache v2.0	LGPL v2.1+	155	36
		Apache v2.0	INRIA-OSL	133	62
		Apache v2.0	UNKNOWN	128	59
		Apache v2.0	LGPL GNU C Lib	89	38

Table 5. Number of files and projects involved in potential license violations

Domain	#files (%)	#projects (%)
Clones	82,950 (39.0)	280 (97.9)
Copied clones	24,566 (11.6)	264 (92.3)
Different licenses	5,095 (2.4)	219 (76.6)
Low risk inconsistent clones	4,548 (2.1)	216 (75.5)
High risk inconsistent clones	1,104 (0.5)	132 (46.2)
Total	212,657 (100)	286 (100)

might be at high risk of potential license violations.

V. CROWDSOURCING FOR FINAL DECISION

A. Expert Task

To make the final decision about the correct licenses with potential violation, experts were asked to identify the provenance of conflict files, such as when the copy was made, in which direction, and what other modifications the clone suffered.

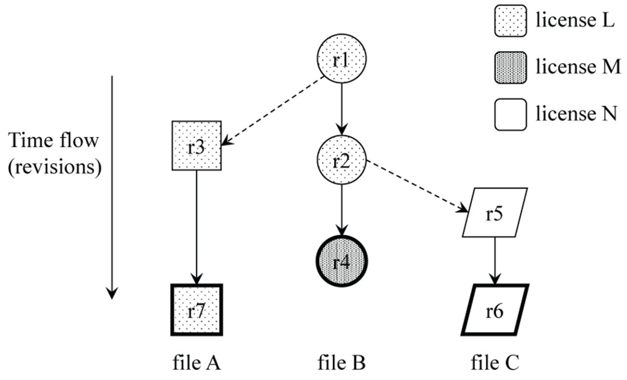


Fig. 4. Example provenance diagram of 3 cloned files. The hashed line denotes the potential copy of the file from the original project to another one. Each revision is colored with its license at that moment.

To understand code evolution over time, experts could look at SCM repositories and trace provenance. They retrieved all revisions for each file containing the clone snippets. An example of provenance that an expert discovered is shown in Fig. 4. When the machine-based process identified that the three files−A (r7), B (r4), and C (r6)—were clones, experts tried to find out where the copied codes were derived from by investigating license statements. For example, experts would investigate dotted arrows as potential derivations and conclude one of the arrows as an actual violation. However, the whole process of identifying potential violations and investigating related statements is very labor intensive. Furthermore, this task needs background knowledge about software and licenses as well as the intuition to understand the collected histories, making it hard to perform it in a mechanical way.

B. Case Studies

We proposed an alternative method to solve such challenge by using wisdom of the crowd. Provenance of codes can be traced by many individuals with their collective intelligence rather than an expert's knowledge. In order to show that crowds can be substituted for an expert, we asked crowds to identify the origin of clones. To assist the crowds, we provided commit date, committer, commit message, license, different codes of other revisions, and similarity scoring to guide crowds in decision making. We will present four case studies in the following sections.

1) Case Study 1: XMLChar.java (I)

The first case was built for a well-known open source project Xerces, a widely adopted module in other projects to process XML documents. XMLChar.java of Xerces project defines the XML character properties. It is utilized to parse XML data. Many other projects copied and modified its source file which may cause license issue. We found several projects with XMLChar.java of Xerces copied without knowing the origin of clone files. We asked the crowd to find the origin of XMLChar.java file in FastInfoset project that was actually copied from the Xerces project. This file has been changed to another package location, modified to comments after copied. In addition to FastInfoset, we found clone XMLChar.java files in five other projects. We collected a total of 44 source files with different revisions. To alleviate the heavy work load for participants caused by many revisions, we computed textual similarities to FastInfoset's first revision of copied file. Only top-10 similar candidates were presented with question. The similarity was computed based on the number of lines they share as shown in the following:

$$similarity\ (r, s) = 1 - \frac{diff\ (r, s)}{|r| + |s|}$$

where r and s were source files, $|r|$ was the number if lines of file r, and *diff*() was the number of different lines between two files.

The commit date, committer, license, and similarity for each revision are shown in Table 6. In addition, we gave clone snippets and different lines between files (not depicted in Table 6 due to the lack of space).

Individuals made their decision about the origin by looking at the difference in each revision of XMLChar.java. We asked four non-experts to answer the question. They had programming experience. However, they have less knowledge about license or open source projects. There were three files tied for the best similarity score. However, files in Jaxp-Sources were created later than those of FastInfoset. If we considered the commit dates, we could identify the correct origin by revision r319746 of Xerces. Commit message of FastInfoset also revealed that it came from Xerces. Although these individuals were not experts, all participants successfully found the origin of FastInfoset by the commit date and message. This case study showed that the crowd has the ability to trace the history of codes.

2) Case Study 2: XMLChar.java (II)

In this study, we designed a tracing question with the same XMLChar.java. We found that some copied XMLChar.java had been copied again to other projects. As mentioned earlier, the origin of XMLChar.java is Xerces. However, Sling project did not copy it directly from Xerces. It was brought from Tomcat-Jasper. As usual, Tomcat-Jasper was copied from Xerces. We collected a total of 42 revisions from seven projects. Among them, the top-11 similar candidates to the Sling were given to crowd (Table 7).

As in the previous case study, the crowd tried to find the origin by comparing the difference among the candidates. However, in this case, similarities were high in general. Therefore, it was difficult to choose the origin only based on the score. In addition, it was not easy for non-experts to understand commit messages in this case. Participants had to check the difference between Sling file and origin candidates in the manual. Among them, the last three revisions of Jasper were almost identical to the Sling. The only changes were several blank lines. Individuals had to decide the origin with their own eyes in this case. Three of four participants identified the origin correctly. Therefore, the majority of the four non-experts could correctly identify the origin of license.

3) Case Study 3: StringUtil.java

This case study was designed with clones used in many projects but built by only one developer. StringUtil.java was created and managed by ericmacau who is involved in five open source projects (Reales, PutoWeb, SharpPOS, Simflet, and Kudocs). The same named files were used in all five projects. However, all files were different in content and license. Therefore, if we were not aware that those files were written by the same developer, we might have thought that there is a license violation. We asked the crowd to determine the license violation issue and to find the origin of Reales's StringUtil.java. The

Table 6. Version history for XMLChar.java clones for finding origin of FastInfoset

(a) XMLChar.java of FastInfoset project

Revision	Commit date	Committer	Commit message	License
r1.1	2005-04-18	sandoz	Xerces XMLChar class renamed and included	Apachev1.1, Apachev2
r1.2	2006-03-16	sandoz	Started cleaning and adding JavaDoc	Apachev1.1, Apachev2

(b) Four projects of the same committer containing clones

Project	Revision	Commit date	Committer	License	Similarity
Xerces	r319746	2004-02-04	mrglavas	Apachev1.1	0.976
	r319806	2004-02-25	mrglavas	Apachev2	0.932
GlassFish	r1.1	2005-05-27	dpatil	Apachev2	0.942
Jasper	r389146	2006-03-27	remm	Apachev2	0.940
	r423920	2006-07-20	mturk	Apachev2	0.940
	r466609	2006-10-21	markt	Apachev2	0.940
	r467222	2006-10-24	markt	Apachev2	0.940
	r593649	2007-11-09	markt	Apachev2	0.940
Jaxp-Sources	r1.1.2.1	2005-08-01	jeffsuttor	Apachev1.1	0.976
	r1.2	2005-08-16	jeffsuttor	Apachev1.1	0.976

Table 7. Version history of XMLChar.java clones for the origin of Sling

(a) XMLChar.java of Sling project

Revision	Commit date	Committer	Commit message	License
r599941	2007-12-01	fmeschbe	SLING-116 Reintroduce compilation to repository…	Apachev2
r746681	2009-02-22	cziegeler	SLING-865: Move to bundles	Apachev2
r768268	2009-04-24	jukka	SLING-941: Lots of svn:eol-style settings…	Apachev2
r785979	2009-06-18	fmeschbe	Move Sling to new TLP location	Apachev2

(b) Five projects of the same committer containing clones

Project	Revision	Commit date	Committer	License	Similarity
Xerces	r319864	2004-03-25	mrglavas	Apachev2	0.963
	r447241	2006-09-18	mrglavas	Apachev2	0.970
GlassFish	r1.1	2005-05-27	dpatil	Apachev2	0.964
Jasper	r389146	2006-03-27	remm	Apachev2	0.968
	r423920	2006-07-20	mturk	Apachev2	0.968
	r466609	2006-10-21	markt	Apachev2	0.975
	r467222	2006-10-24	markt	Apachev2	0.975
	r593649	2007-11-09	markt	Apachev2	0.975
Saxon	r2	2006-09-05	mhkay	Apachev2	0.964
	r4	2006-09-05	mhkay	Apachev2	0.964
Jaxp-Sources	r1.3	2005-09-26	sunithareddy	Apachev2	0.963

crowd was referred to the meta data of the files as shown in Table 8. We presented all histories of every revision in this study. The fact that the all revisions were committed by one developer is interesting. It may give an intuition to

Table 8. Version history of StringUtil.java clones for the origin of Reales

(a) StringUtil.java of Reales project

Revision	Commit date	Committer	Commit message	License
r1.1	2007-04-25	ericmacau	*** empty log message ***	LesserGPLv2.1+
r1.2	2007-04-25	ericmacau	*** empty log message ***	LesserGPLv2.1+
r1.3	2007-05-11	ericmacau	*** empty log message ***	LesserGPLv2.1+
r1.4	2007-05-13	ericmacau	*** empty log message ***	LesserGPLv2.1+
r1.5	2007-05-30	ericmacau	*** empty log message ***	LesserGPLv2.1+

(b) Four projects of the same committer containing clones

Project	Revision	Commit date	Committer	License	Similarity
Kudocs	r1.1	2005-12-26	ericmacau	X	0.152
Simflet	r1.1	2006-06-23	ericmacau	GPLv2+	0.862
	r1.2	2006-07-04	ericmacau	GPLv2+	0.873
	r1.3	2006-07-04	ericmacau	GPLv2+	0.873
	r1.4	2006-07-06	ericmacau	GPLv2+	0.868
	r1.5	2006-07-06	ericmacau	Apachev2	0.872
	rl.6	2006-07-06	ericmacau	Apachev2	0.846
SharpPOS	r1.1	2007-02-26	ericmacau	LGPL, LGPL v2.1+	0.996
	r1.2	2007-02-26	ericmacau	LGPL, LGPL v2.1+	0.999
PutoWeb	r1.1	2008-06-28	ericmacau	Apachev2	0.840
	r1.2	2008-07-01	ericmacau	Apachev2	0.833
	r1.3	2008-07-01	ericmacau	Apachev2	0.703
	r1.4	2008-07-04	ericmacau	Apachev2	0.798
	r1.5	2008-07-08	ericmacau	Apachev2	0.767
	r1.6	2008-07-11	ericmacau	Apachev2	0.759
	r1.7	2008-07-18	ericmacau	Apachev2	0.548
	r1.8	2008-08-12	ericmacau	Apachev2	0.535
	r1.9	2009-02-21	ericmacau	Apachev2	0.550
	r1.10	2009-02-23	ericmacau	Apachev2	0.550

the crowd to answer those questions.

StringUtil.java has been changed slightly over projects. The first revision of Reales's file was almost identical to the last revision of SharpPOS's file, indicating that SharpPOS was the origin of Reales. Although there was no commit message in Reales, a majority of participants agreed that the r1.2 of SharpPOS was the origin. It had the highest similarity to the Reales committed in close time. Other projects had different licenses, which might have caused license violation. However, StringUtil.java of Reales had the same license with its origin. Therefore, Reales was safe from license violation. All participants also correctly judged that this project followed the license policy.

4) Case Study 4: getImageComponents method

In this case study, we designed a trace with a clone snippet method. Wonderland-Incubator project was composed by copying Lg3d-Wonderland. We found the same getImageComponents method in both projects. Lg3d-Wonderland had the method in two files: TextureFixer.java and ModelCompiler.java. Wonderland-Incubator had the same method in GzScenePacker.java. As we inspected the history of the two projects, the method was created in Lg3d-Wonderland project for the first time. We asked crowd to find the origin of getImageComponents method and determine whether there was license violation.

We present the history of three files in Table 9. File level similarities were relatively small because only getI-

Table 9. Version history of clone getImageComponents method snippets

(a) GzScenePacker.java of Wonderland-Incubator project

Revision	Commit date	Committer	Commit message	License
r1.1	2008-09-18	morrisford	*** empty log message ***	X

(b) Two files of Lg3g-Wonderland project containing clones

Filename	Revision	Commit date	Committer	License	Similarity
TextureFixer.java	r1.1	2008-05-20	paulby	GPL v2	0.136
	r1.2	2008-05-21	paulby	GPL v2	0.167
	r1.3	2008-05-28	paulby	GPL v2	0.167
ModelCompiler.java	r1.1	2008-05-22	kaplanj	GPL v2	0.142

mageComponents method parts were identical among files. In this test case, we focused on judging the potential license violation rather than finding the correct origin of clones. GzScenePacker.java of Wonderland-Incubator project did not have any license statement, indicating a high risk of license violation. All participants agreed that GzScenePacker.java of Wonderland-Incubator violated the original license.

In summary, tracing repositories to find origin of copied codes can be addressed by the crowd. When meta-data for the clone files are given, the crowd can understand the provenance of clones by linking the relationships between clones. With the help of machine-based analysis and meta-data for clone files, a few non-experts could replace an expert without compromising the accuracy. Considering the fact that experts with deep understanding of open source projects are rare and that tracing provenance demands lots of time, this is a promising application of crowdsourcing.

VI. RELATED WORK

Crowdsourcing is a task to solicit contributions from a large number of people. It has been actively studied as a means to integrate human intelligence with machine computations based on the survey of Doan et al. [17]. It focuses on how to compose or distribute questions to the crowd that machine cannot answer. We categorized crowdsourcing tasks according to the characteristic of the problem. First, the tasks were small ones so that it was easy for individuals to reply with little effort, such as tagging and voting products. The goal of crowdsourcing is to minimize the total elapsed time, error rate [18], or payment [19]. Second, some problems require huge tasks that it is even hard for experts to solve, such as protein folding problem [20]. A large set of answers from non-experts are aggregated and presented as solution relying on the wisdom of the crowd [21]. Our work successfully unleashed the power of crowds in addressing the license violation problem that has been solely determined through manual inspection by experts.

Many researchers have warned about the difficulties in including FOSS components in commercial software [22-24]. IBM's *Ariadne* appears to be the only software environment that incorporates the management of intellectual property in software development [25]. However, it is restricted to tracking the provenance of software assets that are created in-house. Alspaugh et al. [26] and Alspaugh and Scacchi [27] have modeled how licenses can (or cannot) interact between each other. They have described how software licensing is an important concern in requirements analysis of systems built from components with different licenses. They have also studied the importance of licensing in ecosystems of FOSS applications [28]. Sojer and Henkel [29] have interviewed several hundred developers and discovered that copying FOSS code by commercial enterprises is now common. However, in many cases, software developers lack the understanding of legal risks associated with this activity. In addition, their organizations lack policies to guide them [30].

The FOSSology Project has been creating infrastructure to identify and visualize licenses in software packages to aid in the identification of possible license mismatches [9]. It does not, however, track the copying of components. Based on German et al. [16], we demonstrated that license auditing was a complex and difficult process. Di Penta et al. [31] have studied how the license of open software changed over time along with its implications.

German et al. [4] have studied the legal implications of copying source code amount in open source kernels. They used `ccfinder` to identify copied code and used different versions of the software to determine in which direction the copy was made. Their focus was on the migration of code from one kernel to another and the influence of licenses in these migration. Our work is an extension of their paper. We defined a general method to identify clones, including removal of false positives and the identification of license inconsistencies among copied code. In addition, we developed a graphical representation to visualize the evolution of the clones and their licensing.

There are many studies on the origin, maintenance, and evolution of clones [32-40]. Others have concentrated on their lifespan and genealogy [41]. Our study is different from those studies in that we studied cloning across applications with focus on their licensing. More recently, Ossher et al. [42] have used clone detection to quantify cloning across applications. However, they did not study the licensing implications of such copies. Machine algorithms [8, 10] for clone detection have been used for various applications such as API finding [43, 44], bug triage [45], and code search engine [46-48].

German et al. [4] have reported the cloning in the kernels focusing on the legal issues around it. Our paper was influenced by that paper. We extended the work by looking at a larger number of applications in a different domain (Java). We also extended their method by creating an accidental clone classifier and formalized their method in a decision diagram, a taxonomy of risk of violation, and the provenance diagrams. We also extended the work of Di Penta et al. [31] by demonstrating that licensing evolution was also a potential problem for those who copied code before the license was upgraded. Ossher et al. [42] have studied cloning in a large group of Java software projects to quantify the amount of cloning and to find reasons behind it. Our work extended their work by trying to identify the source of the copy and by incorporating licensing and trying to determine when the cloning might be inconsistent with the licensing or the source of the system.

VII. CONCLUSIONS

We presented machine-based algorithms to systematically determine potential license violations and proposed crowdsourcing as a scalable alternative to manual inspection by experts for those potential violations. Our machine-based approach found 5,095 potential violations from 219 projects among 286 subjects, suggesting that license violation was a widespread problem. Our crowdsourcing framework enabled non-experts to identify actual violations without compromising the accuracy. We anticipate that the results herein will bring awareness of this problem to researchers and practitioners.

ACKNOWLEDGMENTS

This work was supported by a grant of the Institute for Information & Communications Technology Promotion (IITP) funded by the Korea government (MSIP) (No. 10041244, SmartTV 2.0 Software Platform).

REFERENCES

1. N. J. Mertzel, “Copying 0.03 percent of software code base not ‘de minimis’,” *Journal of Intellectual Property Law & Practice,* vol. 9, no. 3, pp. 547-548, 2008.
2. K. Taylor, “Oracle Am., Inc. v. Google Inc. 750 F. 3d 1339 (Fed. Cir. 2014),” *Intellectual Property Law Bulletin*, vol. 19, no. 2, pp. 221-223. 2014.
3. J. Krinke, N. Gold, Y. Jia, and D. Binkley, “Cloning and copying between gnome projects,” in *Proceedings of 7th IEEE Working Conference on Mining Software Repositories (MSR)*, Cape Town, South Africa, 2010, pp. 98-101.
4. D. M. German, M. Di Penta, Y. G. Gueheneuc, and G. Antoniol, “Code siblings: technical and legal implications,” in *Proceedings of 6th IEEE International Working Conference on Mining Software Repositories (MSR)*, Vancouver, Canada, 2009, pp. 81-90.
5. M. Sojer and J. Henkel, “License risks from ad hoc reuse of code from the internet,” *Communications of the ACM*, vol. 54, no. 12, pp. 74-81, 2011.
6. M. B. Jensen, *Does Your Project Have a Copyright Problem? A Decision-Making Guide for Librarians*. Jefferson, NC: McFarland & Company, 1996.
7. L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “Deckard: scalable and accurate tree-based detection of code clones,” in *Proceedings of the 29th international conference on Software Engineering (ICSE’07)*, Minneapolis, MN, 2007, pp. 96-105.
8. M. W. Lee, J. W. Roh, S. W. Hwang, and S. Kim, “Instant code clone search,” in *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, Santa Fe, NM, 2010, pp. 167-176.
9. R. Gobeille, “The fossology project,” in *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR),* Leipzig, Germany, 2008, pp. 47-50.
10. T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: a multilinguistic token-based code clone detection system for large scale source code,” *IEEE Transactions on Software Engineering,* vol. 28, no. 7, pp. 654-670, 2002.
11. I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” in *Proceedings of the International Conference on Software Maintenance*, Bethesda, MD, 1998, pp. 368-377.
12. C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery,* vol. 2, no. 2, pp. 121-167, 1998.
13. C. C. Chang and C. J. Lin, “*LIBSVM: a library for support vector machines,”* https://www.csie.ntu.edu.tw/~cjlin/libsvm/.
14. N. Krawetz, “Symbolic alignment matrix,” 2008; http://www.fossology.org/projects/fossology/wiki/Symbolic_Alignment_Matrix.
15. D. M. German, Y. Manabe, and K. Inoue, “A sentence-matching method for automatic license identification of source code files,” in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, 2010, pp. 437-446.
16. D. M. German, M. Di Penta, and J. Davies, “Understanding and auditing the licensing of open source software distributions,” in *Proceedings of 18th International Conference on Program Comprehension (ICPC),* Braga, Portugal, 2010, pp.

84-93.
17. A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Communications of the ACM,* vol. 54, no. 4, pp. 86-96, 2011.
18. M. S. Bernstein, D. R. Karger, R. C. Miller, and J. Brandt, "Analytic methods for optimizing realtime crowdsourcing," in *Proceedings of Collective Intelligence 2012,* Cambridge, MA, 2012, pp. 1-8.
19. P. Welinder and P. Perona, "Online crowdsourcing: rating annotators and obtaining cost-effective labels," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW),* San Francisco, CA, 2010, pp. 25-32.
20. C. B. Eiben, J. B. Siegel, J. B. Bale, S. Cooper, F. Khatib, B. W. Shen, F. Players, B. L. Stoddard, Z. Popovic, and D. Baker, "Increased Diels-Alderase activity through backbone remodeling guided by Foldit players," *Nature Biotechnology*, vol. 30, no. 2, pp. 190-192, 2012.
21. J. Lee, H. Cho, J. W. Park, Y. R. Cha, S. W. Hwang, Z. Nie, and J. R. Wen, "Hybrid entity clustering using crowds and data," *The VLDB Journal*, vol. 22, no. 5, pp. 711-726, 2013.
22. M. Bayersdorfer, "Managing a project with open source components," *Interactions,* vol. 14, no. 6, pp. 33-34, 2007.
23. T. Madanmohan and R. De', "Open source reuse in commercial firms," *IEEE Software,* vol. 21, no. 6, pp. 62-69, 2004.
24. C. Ruffin and C. Ebert, "Using open source software in product development: a primer," *IEEE Software,* vol. 21, no. 1, pp. 82-86, 2004.
25. Y. B. Dang, P. Cheng, L. Luo, and A. Cho, "A code provenance management tool for IP-aware software development," in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 975-976.
26. T. Alspaugh, H. U. Asuncion, and W. Scacchi, "Intellectual property rights requirements for heterogeneously-licensed systems," in *Proceedings of 17th IEEE International Requirements Engineering Conference (RE'09),* Atlanta, GA, 2009, pp. 24-33.
27. T. Alspaugh and W. Scacchi, "Heterogeneously-licensed system requirements, acquisition and governance," in *Proceedings of 2nd International Workshop on Requirements Engineering and Law (RELAW 2009),* Atlanta, GA, 2009, pp. 13-14.
28. T. Alspaugh, H. U. Asuncion, and W. Scacchi, "The role of software licenses in open architecture ecosystems," in *Proceedings of 1st International Workshop on Software Ecosystems (IWSECO),* Falls Church, VA, 2009, pp. 4-18.
29. M. Sojer and J. Henkel, "Code reuse in open source software development: quantitative evidence, drivers, and impediments," *Journal of the Association for Information Systems*, vol. 11, no. 12, pp. 868-901, 2010.
30. M. Sojer and J. Henkel, "License risks from ad hoc reuse of code from the internet," *Communications of the ACM,* vol. 54, no. 12, pp. 74-81, 2011.
31. M. Di Penta, D. M. German, Y. G. Gueheneuc, and G. Antoniol, "An exploratory study of the evolution of software licensing," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Cape Town, South Africa, 2010, pp. 145-154.
32. M. Godfrey and L. Zou, "Using origin analysis to detect merging and splitting of source code entities," *IEEE Transactions on Software Engineering,* vol. 31, no. 2, pp. 166-181, 2005.
33. J. Krinke, "A study of consistent and inconsistent changes to code clones," in *Proceedings of 14th Working Conference on Reverse Engineering (WCRE)*, Vancouver, Canada, 2007, pp. 170-178.
34. J. Krinke, "Is cloned code more stable than non-cloned code?," in *Proceedings of 8th IEEE International Working Conference on Source Code Analysis and Manipulation*, Beijing, China, 2008, pp. 57-66.
35. A. Lozano, "A methodology to assess the impact of source code flaws in changeability and its application to clones," in *Proceedings of the International Conference of Software Maintenance*, Beijing, China, 2008, pp. 424-427.
36. A. Lozano, M. Wermelinger, and B. Nuseibeh, "Evaluating the harmfulness of cloning: a change based experiment," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, Minneapolis, MN, 2007.
37. S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Software Engineering,* vol. 15, no. 1, pp. 1-34, 2010.
38. C. Kapser and M. W. Godfrey, "Cloning considered harmful," considered harmful: patterns of cloning in software," *Empirical Software Engineering*, vol. 13, no. 6, pp. 645-692, 2008.
39. R. Tiarks, R. Koschke, and R. Falke, "An assessment of type-3 clones as detected by state-of-the-art tools," in *Proceedings of 9th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, Edmonton, AB, 2009, pp. 67-76.
40. Y. Kashima, Y. Hayase, N. Yoshida, Y. Manabe, and K. Inoue, "An investigation into the impact of software licenses on copy-and-paste reuse among OSS projects," in *Proceedings of 18th Working Conference on Reverse Engineering (WCRE),* Limerick, Ireland, 2011, pp. 28-32.
41. M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 187-196, 2005.
42. J. Ossher, H. Sajnani, and C. Lopes, "File cloning in open source Java projects: the good, the bad, and the ugly," in *Proceedings of the International Conference in Software Maintenance,* Williamsburg, VI, 2011, pp. 283-292.
43. J. Kim, S. Lee, S. W. Hwang, and S. Kim, "Adding examples into java documents," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09),* Auckland, New Zealand, 2009, pp. 540-544.
44. J. Kim, S. Lee, S. W. Hwang, and S. Kim, "Enriching documents with examples: a corpus mining approach," *ACM Transactions on Information Systems*, vol. 31, no. 1, article no. 1, 2013.
45. J. W. Park, M. W. Lee, J. Kim, S. W. Hwang, and S. Kim, "CosTriage: a cost-aware triage algorithm for bug reporting systems." in *Proceedings of 25th AAAI Conference on Artificial Intelligence (AAAI)*, San Francisco, CA, 2011.
46. J. Kim, S. Lee, S. W. Hwang, and S. Kim, "Towards an intelligent code search engine," in *Proceedings of 24th AAAI Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010.
47. M. W. Lee, S. W. Hwang, and S. Kim, "Integrating code search into the development session," in *Proceedings of 2011 IEEE 27th International Conference on Data Engineer-*

ing (ICDE), Hannover, Germany, 2011, pp. 1336-1339.
48. J. W. Park, M. W. Lee, J. W. Roh, S. W. Hwang, and S. Kim, "Surfacing code in the dark: an instant clone search approach," *Knowledge and Information Systems*, vol. 41, no. 3, pp. 727-759, 2014.

Sanghoon Lee

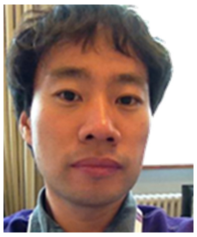
Sanghoon Lee is a Ph.D. student in the Department of Computer Science and Engineering at Pohang University of Science and Technology, Korea. He received his B.S. degree from Kyungpook National University, Korea in 2009. His research areas are instance matching, entity detection, linking in data science, and clone code detection in software engineering.

Daniel German

Daniel German is a Professor in the Department of Computer Science at University of Victoria, Canada. He completed his Ph.D. at University of Waterloo in 2000. His research areas are mining software repositories, open source, and intellectual property in software engineering. For more information, visit http://turingmachine.org

Seung-won Hwang

Seung-won Hwang is a Professor in the Department of Computer Science at Yonsei University, Korea. She completed her Ph.D. at University of Illinois Urbana-Champaign in 2005. Her research areas are data (-driven) intelligence, knowledge graph, search engines, query optimization, and language understanding in data science. For more information, visit http://dilab.yonsei.ac.kr/~swhwang/



Sunghun Kim

Sunghun Kim is an Associate Professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology, China. He completed his Ph.D. at University of California, Santa Cruz in 2006. His research areas are repository data mining, software testing, dynamic and static analysis in software engineering. For more information, visit https://www.cse.ust.hk/~hunkim/