Journal of Information Processing Vol.31 2–12 (Jan. 2023)

[DOI: 10.2197/ipsjjip.31.2]

**Regular Paper**

# Automating License Rule Generation to Help Maintain Rule-based OSS License Identification Tools

Yunosuke Higashi[1,2,a)] Masao Ohira[1,b)] Yuki Manabe[3,c)]

Received: April 11, 2022, Accepted: October 4, 2022

***Abstract:*** Many license identification tools have been proposed to support OSS reuse. License identification tools automatically identify OSS licenses declared in source files. Ninka is one of the most accurate license identification tools. Because OSS licenses are often newly created or inherited, rules built into license identification tools need to be created and updated on a regular basis. However, when a large number of unknown licenses are detected in large OSS products, it is not easy to manually create new rules. In our previous studies, we proposed a method for clustering license statements that Ninka determined to be unknown. In this paper, we propose a method to automatically generate license rules from the clustered license statements. Our approach further filters the license statements from the created clusters to extract sequential patterns and converts the extracted patterns into regular expressions. We conducted conduct a case study where our method is applied to 1,821, 3,561 and 2,838 unknown license statement files respectively collected from FreeBSD v10.3.0, Linux Kernel v4.4.6, and Debian v7.8.0, to confirm the usefulness of our method. As a result, we confirmed that our method successfully generated license rules that take into consideration the orthographical variants and that our method also efficiently identified licenses with a small number of license rules. Furthermore, we found that adding the license rules generated by our method to Ninka improves the licensing rule performance.

***Keywords:*** OSS License, License Identification, License Rules Generation, Sequential Pattern Mining

## 1. Introduction

In modern software development, open source software (OSS) components are often used via application programming interfaces (APIs) [4]. An OSS component can be reused as part of a commercial software product by strictly complying with the OSS license specified for each source file [1]. Currently, 115 OSS licenses [*1] have been approved by Open Source Initiative (OSI). In general, OSS licenses are declared in the header part of a source file. Developers need to identify and understand the terms of each OSS license which is appropriate for their commercial software products, in order to avoid license violations [4].

Several license identification tools have been proposed to help developers identify OSS licenses. The license identification tools output a list of names OSS license names, using license rules predefined by regular expressions [1], [5], [6], [7]. Of the existing tools, Ninka [1] identifies OSS licenses with the highest accuracy. Ninka reduces false positives (e.g., incorrectly judging an unknown license as a known license) by distinguishing between unknown and known licenses. Because of the need to avoid license violations when incorporating OSS components into commercial products, it is heavily important to reduce false positives of license identification tools. The biggest weakness of rule-based license identification tools such as Ninka is that license rules (e.g., regular expressions) must be updated manually, each time the tools encounter new OSS licenses (e.g., the licenses are unknown to the tools).

The creation of license rules is a time consuming manual task, since it involves visually inspecting unknown licenses and creating regular expressions for each sentence while considering orthographical variants. The simplest way to automate the creation of license rules is to add license statements as-is. However, such an approach would generate a huge number of license rules when a large number of unknown licenses are detected. Consequently, the effort required for rule-naming makes this approach impractical.

The goal of this study is to construct a method to automatically generate candidate license rules to be incorporated into rule-based license identification tools in order to address the issue above. There are two requirements that the generated license rules should meet: (1) metacharacters with regular expressions should be used and (2) a large number of license statements should be identified with a minimum number of license rules. To achieve these requirements, we are building a method which consists of the following three steps for generating license rules automatically.

**(Step 1) Grouping source files with unknown licenses:**
Group source files that cannot be identified by the license identification tools by OSS license.

[1] Graduate School of Systems Engineering, Wakayama University, Wakayama 640–8510, Japan
[2] The Japan Research Institute, Limited, Shinagawa, Tokyo 141–0022, Japan
[3] Faculty of Informatics, The University of Fukuchiyama, Fukuchiyama, Kyoto 620–0886, Japan
[a)] higashi.yunosuke@g.wakayama-u.jp
[b)] masao@sys.wakayama-uc.ac.jp
[c)] manabe-yuki@fukuchiyama.ac.jp

[*1] https://spdx.org/licenses/, accessed on April 1, 2022.

**(Step 2) Checking orthographical variants for a single license:** Each group of source files with a single license is checked to extract expressions patterns for a single license. Check each group of source files and extract expression patterns for a single license.

**(Step 3) Generating license rules:** Tokenize license statements as regular expressions and generate license rules that can be matched to new licenses.

In our previous study [2], [3], we proposed a clustering method to classify the license statements of detected unknown licenses to automate **Step 1**. In this paper, we focus on **Step 2** and **Step 3** to automatically generate license rules from each cluster created by the clustering method. First, filtering by the Levenshtein distance is applied to license statements belonging to each cluster to extract the license statements for which license rules should be generated. Next, sequential pattern mining is used to extract patterns of the license statements. Finally, the output sequential patterns are converted to regular expressions.

In this paper, we address the following research questions to confirm the usefulness of our method and conduct a case study where our method is applied to 1,821, 3,561 and 2,838 unknown license statement files respectively collected from FreeBSD v10.3.0, Linux Kernel v4.4.6, and Debian v7.8.0, to answer research questions.

**RQ1:** How many license rules could be aggregated by metacharacters with regular expressions?

**RQ2:** Is the number of license rules generated by our approach less than the AS–IS method?

**RQ3:** Does our approach generate licensing rules with better performance than the AS–IS method?

As a result, we confirmed that our method successfully generated license rules that take into consideration the orthographical variants and that our method also efficiently identified licenses with a small number of license rules. Furthermore, we found that adding the license rules generated by our method to Ninka improves the performance of the license rules.

The rest of the paper is organized as follows. Section 2 describes existing license identification tools and our motivation for this study. Section 3 presents our approach and Section 4 shows a result of our case study. In Section 5, we discuss the result and threats to validity in our case study. Section 6 introduces related work and Section 7 summarizes this paper.

## 2. License Identification Tools and Limitations

In this chapter, we describe the mechanism of existing license identification tools and the technical challenges for automating license rule generation.

### 2.1 License Identification Tools

To reuse OSS as a part of software, the developer must comply with the OSS license. **Figure 1** shows an example of license statements for the BSD3 license.

In general, license statements is written as source code comments in the source file. To reuse OSS, it is necessary to read license statements in all source files to identify OSS licenses. In order to help identify OSS licenses, license identification

```
Copyright (c) <year> <owner>.  All rights reserved.
Redistribution and use in source and binary forms, with
or without modification, are permitted provided that the
following conditions are met:

1.  Redistributions of source code must retain the
above copyright notice, this list of conditions and the
following disclaimer.

2.  Redistributions in binary form must reproduce the
above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or other
materials provided with the distribution.

3.  Neither the name of the copyright holder nor the names
of its contributors may be used to endorse or promote
products derived from this software without specific prior
written permission.

THIS SOFTWARE IS PROVIDED BY [...]
```

**Fig. 1** An example of license statements (BSD3).

tools [1], [5], [6], [7], [8] have been proposed. The rule-based approach for identifying OSS licenses is predominant currently since accurate identification is essential for avoiding legal risks involve in reusing OSS with misidentified licenses.

The tools identify OSS licenses (e.g., license names) using rules which we call "license rules". A license rule is defined by tying license statements to a license name. The license rules are mainly based on regular expressions [1], [5], [6], [7] or similarity of license statements [8]. By using regular expressions, the tools can distinguish tiny differences among strings such as license versions. This is critically important for practitioners since different versions of the same license are sometimes incompatible each other (e.g., different meanings from a legal perspective). Using similarity of license statements [8] contributes to reducing costs in creating license rules but leads to low identification performance (55% of precision [1]) because the similarity-based approach cannot identify tiny differences among license statements.

However, since new licenses and orthographical variants appear every year, it is necessary to create and add license rules manually on an ongoing basis. The most accurate license identification tool, Ninka (precision 96.6%) [1], can identify new licenses and orthographical variants of existing licenses as unknown licenses, but the license names can only be confirmed manually.

**Table 1** summarizes license rules creation tasks: (1) classification of license statements, (2) checking orthographical variants, (3) generating rules and (4) Naming licenses. All tasks should be automated to reduce the cost for keeping a license identification tool up-to-date. As shown in Table 1, Ninka [1] and FOSSology [8] need to be maintained manually for all tasks except for (2) of FOSSology. On the other hand, our method aims to automate the process of (1) through (3) to support maintain regular expression-based license identification tools (especially for Ninka in this study). For the remaining task (4), our method does not intend to automate, since it requires a high level of legal knowledge for naming licenses.

### 2.2 The License Rules of Ninka

In order to be able to identify unknown licenses, new license rules need to be created and added. This section details two types of licensing rules used in Ninka as an example.

**Rules for identifying license sentences** This type of license rule is a regular expression to identify each sentence (period-

**Table 1** License rule creation tasks to be automated by our study.

| | Ninka [1] (regular expression) | FOSSology [8] (similarity threshold) | Our Method (regular expression) |
|---|---|---|---|
| 1. Classification of license statements | Manual | Manual | **Automated** |
| 2. Checking orthographical variants | Manual | None (※) | **Automated** |
| 3. Generating rules | Manual | Manual | **Automated** |
| 4. Naming licenses | Manual | Manual | Manual |

(※) FOSSology has no process to reflect tiny orthographical variations in license rules, since FOSSology uses text-similarity to find a target license statement similar to a license rule. This mechanism facilitates the rule creation task, but reduces the accuracy of the license identification.

separated) in license statements. For example, the BSD3 license has a total of six different license sentences. Therefore, six regular expressions must be added to identify each sentence. In addition, metacharacters of the regular expressions are used to take into account any orthographical variants of the license sentences. The following is an example of a rule to identify one of BSD3's license sentences as "BSDcondSource" in the format <license sentence name>:<regular expression>.

*BSDcondSource : Redistributions? of source code must retain the (above )?copyright notice,[...]:*

**Rules for identifying license name** This type of license rules is rules to identify license names expressed as sequences of one or more license sentence names. If the specified sequence of license sentences matches the rules in Ninka, the license name is output. At this time, If the corresponding license name rule is not added, Ninka outputs that it is an unknown license. The following is an example of a rule to identify the BSD3 license in the format <license name>:<license sentence name>.

*BSD3 : BSDpre, BSDcondSource, BSDcondBinary, BSDcondEndorseRULE, BSDasIs, BSDWarr*

The reason why there are two types of license rules is to minimize the number of license rules to be implemented. For example, the BSD2 license omits one clause from the BSD3 license, and the rest of the license sentences is redundant. Therefore, Ninka has been devised to provide rules to identify license sentences so that duplicate regular expressions do not have to be added. However, it is also a fact that more rules are needed to identify license statements than to identify license names. For BSD licenses, 5–6 regular expressions are required. In addition, each rule to identify a license sentence needs to be named by the tool authors. In the following, we define a "license rule" as regular expression for identifying a license statement and aim to automatically generate license rules.

### 2.3 Technical Challenges

Ninka is designed to minimize the number of license rules, it takes a lot of effort to manually create license rules themselves. The goal of our study is to automatically generate candidate license rules to support the creation of license rules. In our previous study [3], we proposed a clustering method to classify unknown license statements by license name. In this paper, we extract expression patterns from clusters classified by our clustering method and generate regular expressions. In order to automatically generate license rules, the following two technical challenges need to be addressed.

**C-1: Dealing with orthographical variants** Unknown licenses are output by Ninka not only because of new licenses, but also because of orthographical variants in a license statement. Orthographic variants should be caught heuristically by manual visual inspection of the license statement. However, when a large number of unknown licenses are output, it is very labor-intensive to visually inspect each sentence and investigate the orthographical variants. Therefore, it is desirable to output license rules in which orthographical variants are taken into account by metacharacters of regular expressions.

**C-2: Generating as few rules as possible** The simplest way to create license rules is to directly add license statements as-is to Ninka. However, when a large number of unknown licenses are detected, a large number of license rules are generated, and it requires a great deal of effort to name the rules. Therefore, simply adding license statements as-is would be time-consuming and cannot be maintained on an ongoing basis. In this study, we focus on the automatic generation of license rules for identifying license sentences, and aim to build a method to identify as many license sentences as possible with as few rules as possible.

## 3. Proposed Method

This section describes our proposed method for automating license rule generation. **Figure 2** shows an overview of the proposed method consisting of the following 5 major processes.

3.1 First, the clustering method of our previous studies [2], [3] is used to classify the unknown license statements according to the similarity of the word vectors (Bag-of-Words).

3.2 Next, the license statements that cannot be classified due to slight differences in minor versions are flittered out as outliers by the similarity based on the Levenshtein distance.

3.3 In the second half of the process, license rules are constructed from frequently-used license statements in order to prevent the generation of a large number of license rules. First, sequential patterns of words are extracted from sentences in license statements using BIDE [9] which is one of the sequence pattern mining algorithms.

3.4 Next, short patterns that cannot be license rules such as simply "GPL" or "License" are removed from extracted sequential patterns.

3.5 Finally, in order to generate license rules that consider orthographical variants, word sequential patterns are represented using metacharacters of regular expressions.

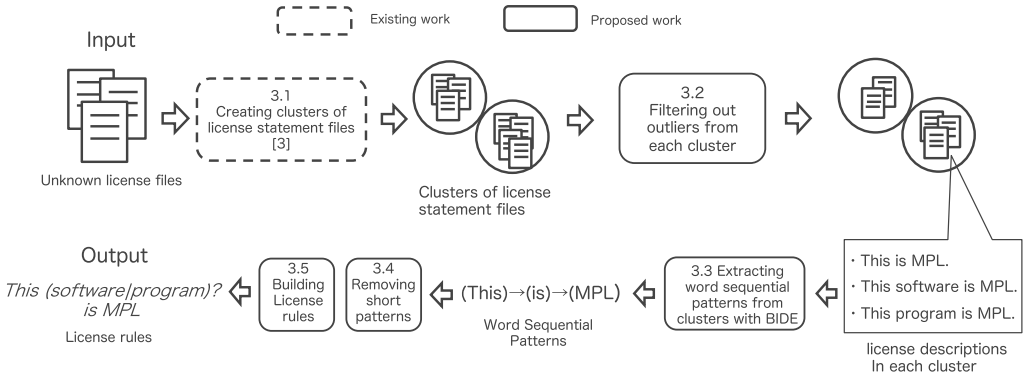


**Fig. 2** An overview of the automated license rule generation process.

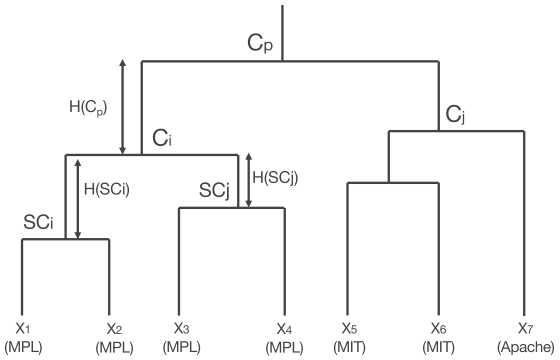


**Fig. 3** An example of a dendrogram: $C_i$, $C_j$, $C_p$ are candidates for a cluster. $SC_i$ and $SC_i$ are single (minimum) clusters that only have two data. $H(C_x)$ and $H(SC_x)$ are dissimilarities between candidate clusters, based on the the Euclidean distance.

### 3.1 Hierarchical Clustering of OSS License Statements

In our previous study [2], [3], we proposed a clustering method to automate the grouping of license statements in Step (I) described in Section 1. It consists of the three parts as follows.

#### 3.1.1 Extracting License Statements for Unknown Licenses

We exploit Ninka to extract license statements from source files with unknown OSS licenses. It outputs license statements as unknown license files when it cannot identify OSS licenses.

#### 3.1.2 Filtering GPL/BSD Family Licenses

Before clustering, our method classifies license statements into GPL family licenses, BSD family licenses or others. GPL family or BSD family licenses have version numbers and some derived licenses. Even if there is a difference of just one word between two license statements, it may mean there are two distinct different licenses. This characteristic is not suitable for machine learning approaches. We use key phrases and remove license statements for GPL/BSD family licenses in advance.

#### 3.1.3 Hierarchical Clustering

At first, each license statement is converted to a Bag of Words (BoW) vector. Then, Ward's method is applied to the BoW vector to create a dendrogram based on the Euclidean distance among license statements. **Figure 3** shows an example of a dendrogram. Finally, based on the dendrogram, clusters are generated. If two clusters $C_i$ and $C_j$ will be not merged into a cluster $C_p$, the condition for pruning the two clusters is as in Eq. (1).

$$H(C_p) > H(SC_i) \ \ and \ \ H(C_p) > H(SC_j) \tag{1}$$

In the previous study [2], [3], we calculated and evaluated the percentage of clusters consisting of a single license. The ratio of the single license clusters was high enough for FreeBSD v10.3.0 (91.7%) and Linux Kernel v4.4.6 (90.7%) respectively, but it was not so high for Debian v7.8.0 (69.1%). Since Debian is a Linux distribution which consists of many packages with many kinds of OSS licenses.

### 3.2 Filtering Out Outliers from Each Cluster

Since our clustering method is not perfect as described above, some percentage of the created clusters contain license statements of multiple OSS licenses. This is mainly due to slight differences in minor versions of a license statement. In this step, such minor versions are filtered out as outliers by the similarity based on the Levenshtein distance. Levenshtein similarity is defined by the following Eq. (2), where $\mathrm{sim}(d, d')$ is the similarity between two license statements $(d, d')$ belonging to a cluster, $L(d, d')$ is the Levenshtein distance between $d$ and $d'$, and $len(d)$ and $len(d')$ are lengths of strings.

$$sim(d, d') = \frac{\mathrm{L}(d, d')}{max\{\mathrm{len}(d), \mathrm{len}(d')\}} \tag{2}$$

The selection for representative license statements $d$ is conducted based on the following Eq. (3), where $D$ is a set of license statements in a cluster and $s$ is threshold for filtering license statements.

$$\{d \in D \mid max\{n \ \{d' \mid \mathrm{sim}(d, d') > s\}\}\} \tag{3}$$

After $d$ is selected, license statements whose similarity to $d$ is greater than a threshold $s$ are extracted as outliers from the cluster. Since there can be multiple $d$ in a cluster, the extraction process is repeated for the same cluster until there are no more license statements with a similarity higher than the threshold. License statements excluded by this filter will not generate the license rules described in Section 3.5.

### 3.3 Extracting Statement Patterns with BIDE

In order to generate license rules that consider orthographical variants, fundamental expression patterns are extracted as a word sequence of license sentences in each cluster. The sequential pattern mining algorithm BIDE [10] is used to extract the word sequence. In general, a sequential pattern mining algorithm prefixes each element of sequence data, repeatedly searches for the

**Table 2** Examples of word sequences in license statements.

| No. | Sentence |
|---|---|
| 1 | (This), (is), (MPL) |
| 2 | (This), (software), (is), (MPL) |
| 3 | (This), (programs), (is), (MPL) |

**Table 3** Closed sequential patterns extracted with BIDE.

| No. | Pattern | Support |
|---|---|---|
| 1 | (This) → (is) | 3 |
| 1 | (This) → (is) → (MPL) | 3 |

subsequent elements, and extracts sequential patterns that satisfy the frequency (hereafter referred to as "support"). Among sequential pattern mining algorithms, BIDE is one of the fastest algorithms to extract sequential patterns. If there are three sentences as showed in **Table 2**, two sequential patterns in **Table 3** will be extracted. In this Table 3, sequential pattern "(This) → (is) → (MPL)" and "(This) → (is)" have support 3. Then, "(This) → (is)" is not extracted as a sequential pattern because this sequential pattern is a subsequence of "(This) → (is) → (MPL)" by BIDE. BIDE achieves fast extraction of sequential patterns by extracting only closed sequential patterns such as "(This) → (is) → (MPL)".

### 3.4 Removing Short Statement Patterns

Before building license rules, our method removes short statement patterns. These statement patterns are not suitable for constructing license rules because they tend to match multiple licenses. In our preliminary study, we found short license statements such as "GPL" and "License". When applying BIDE to clusters including these short license statements, BIDE will produce "(GPL)" or "(License)" as statement patterns. In this phase, our method removes statement patterns with lengths shorter than the threshold are deleted.

### 3.5 Building License Rules

Extracted expression patterns in the previous BIDE process are used to build license rules. Each license rule is written in regular expressions. In this process, our method finds words between adjunct two words in each sequential pattern. For each pair of two adjacent words in the sequential pattern, sequences from one word to another word are found from license sentences including the sequential pattern.

If a word in a license sentence is "$(W_n)$", a sequential pattern is represented by "$(W_1), (W_2)$". "$(X_n)$", indicates a word added between two elements in the sequential pattern. Therefore, a license rule is expressed as "$W_0X_0\ W_1X_1\ W_2X_2\ \ldots W_{n-1}X_{n-1}\ W_nX_n\ W_{n+1}$". For example, in 2, if "$(W_n)$" is *"(This) → (is) → (MPL)"*, then "$(X_n)$" corresponds to *"(software)"* or *"(program)"*. $W_0$ and $W_{n+1}$ mean the beginning and the end of a license sentence respectively. Each $X_i\{i = 0, 1, 2, 3, \ldots, n, n+1\}$ is determined in the following rules. At first, the position of "$W_i$" and "$W_{i+1}$" is identified. Then, if no phrase appears between them, $X_i = \phi$. If a specific phrase "$P_1$" in all the license sentences, $X_i = P_1$. If $P_1, P_2, \ldots, P_m$ appear in the license sentences, $X_i$ = "$(P_1|P_2|..|P_m)$". In addition, If $P_n \in P_1, P_2, \ldots, P_m$ and $P_n$ are empty, $P_n$ is converted to ".+" and then $X_i$ = "$(P_1|P_2|..|P_m)$?".

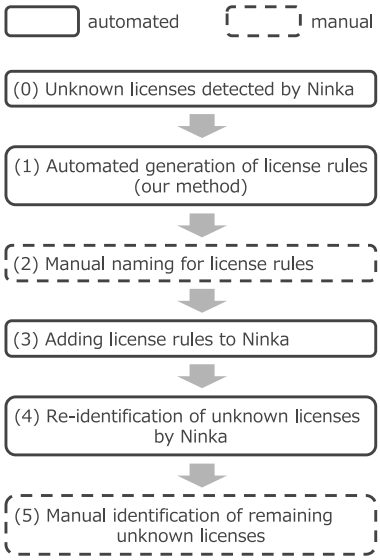


**Fig. 4** Use scenario for license rule generation with our method.

### 3.6 Use Scenario

Here, we describe a usage case of our proposed method. The **Fig. 4** illustrates the use scenario. The proposed method assumes a scenario where (0) a developer runs a license identification tool (Ninka) for OSS that the user wants to reuse and needs to know licenses which cannot be identified by Ninka. First, (1) the developer applies our method to unknown licenses to generate candidate license rules. Next, (2) the developer names the license rule candidates manually by looking at the SPDX license list[*1] and other information, and then (3) adds the named license rules to Ninka. The named license rules are added to the end of Ninka's rule file. Next, (4) the developer runs Ninka again to identify license names of unknown licenses, including unknown licenses which are filtered out in the license rule generation process described in Section 3.2. Finally, (5) the developer manually identifies unidentified licenses remained even after adding the license rules in (3).

We believe that the performance of license identification tools can be continuously maintained through the above five tasks. However, (2) and (5) are manual tasks and involve a trade-off relationship. The more license rules generated, the more license statements are matched (i.e., the effort for naming will increase). If either (2) or (5) is a very time-consuming task to complete, it would be difficult to maintain license identification tools. Therefore, license rules generated by our method should contribute to identifying as many license sentences as possible with as few rules as possible.

## 4. Case Study

We conducted a case study to address our research questions.

### 4.1 Dataset

Thiscase study uses license statements which are identified by

**Table 4** Summary of dataset used in this case study.

| Project | Release | #files | #licenses | The way to extract unknown license files |
|---|---|---|---|---|
| FreeBSD-10.3.0 | Apr, 2016 | 1,821 | 69 | Apply Ninka to Source files of FreeBSD-10.3.0 [*2] and then extract source files classified into unknown license. |
| Linux-4.4.6 | Mar, 2016 | 3,561 | 33 | Apply Ninka to Source files of Linux-4.4.6 [*3] and then extract source files classified into unknown license. |
| Debian-7.8.0 | Jan, 2015 | 2,838 | 194 | Sampled one file from each package managed in Debian-7.8.0 [*4], apply Ninka to the set of source files and then extracted source files classified into unknown by Ninka. |

Ninka-1.1 (2013, July) [*5] as unknown. These are the same dataset in our previous study [3] as shown in **Table 4**. The source files are written in C, C++, Java, Python or Lisp and are extracted from FreeBSD-10.3.0 (FB), Linux-4.4.6 (LI) and Debian-7.8.0 (DE). These projects were released 2-3 years after the release of Ninka-1.1. The target files for the detection of unknown licenses are different for each project. FB and LI projects have their own set of source files as the target of detection. Note that target files for DE are randomly selected from each software packages in DE. The reason for targeting software packages is that a software package consists of a large number of software and we want to create a dataset consisting of a larger number of licenses. The first author of the paper manually reviewed all the extracted unknown license statement files and identified OSS licenses in the unknown license statement files.

### 4.2 Thresholds

Our approach requires thresholds for removing license statements from a cluster and for extracting statement patterns. In what follows, we explain how to set these thresholds.

#### 4.2.1 Maximum Similarity for Removing License Statements Regarded as Outliers

In this case study, we use 94% as the maximum similarity to remove license statements from a cluster. To set this threshold, we conducted a preliminary study. In the study, we calculated similarities of 1,000 pairs of license statements in which each license statement corresponds to different licenses in Debian v7.8.0. The result showed that the maximum of similarity was 93.5% in case of a pair of license statements for LGPLv3 and for LGPLv2.1. Thus, we use 94% as the threshold.

#### 4.2.2 Minimum Length of License Statement Patterns Suitable for License Rules

To determine the minimum length of the number of elements, we counted words in Ninka's license rules. As a result, we have identified a rule with the lowest number of words ("This program is free software"). The number of words in the rule is 5 but we set the minimum length of the number of word elements to 3, because we anticipated variations without adjective words such as "This" and "free" could be included in the dataset.

### 4.3 Result

#### 4.3.1 RQ1: How Many License Rules Could be Aggregated by Metacharacters with Regular Expressions?

**Motivation:** This question addresses the effect of using regular expressions (C-1). Proposed approach aggregate license rules corresponding to one license rule with metacharacter. If metacharacters are used effectively, the number of license rules will be reduced with keeping the accuracy.

**Approach:** To answer this research question, we evaluate the effect of metacharacters with regular expressions for reducing the number of license rules. The effect is represented as a consolidation ratio defined as follows:

$$ConsolidationRatio = \frac{RulesNum - RegexpRulesNum}{RulesNum} \quad (4)$$

where *RulesNum* means the number of license rules before consolidating with metacharacters and *RegexpRulesNum* means the number of license rules after consolidating with metacharacters.

**Result:** **Table 5** shows that the consolidation ratio for each dataset. 3.8%, 5.9% and 4.8% of license rules in FreeBSD-10.3.0, Linux-4.4.6 and Debian-7.8.0 respectively are consolidated. **Table 6** shows some instances of license rules with metacharacters. The instance of RSA-MD (RSA-Message-Digest License) is extracted from FreeBSD-10.3.0 and the instance of GPL is extracted from Linux-4.4.6 and the instance of CeCILL-C License is extracted from Debian-7.8.0. From these results, we can answer for RQ1 as follows:

**Table 5** Consolidating ratio using metacharacters.

| Project Name | # license rules before | # license rules after | Consolidation Ratio |
|---|---|---|---|
| FB | 475 | 457 | **3.8%** |
| LI | 286 | 269 | **5.9%** |
| DE | 440 | 419 | **4.8%** |

**Table 6** Instance of license rules with metacharacters.

| | License | License rule |
|---|---|---|
| FB | RSA-MD | License( is also granted)? to( copy\| make) and use( this software is granted\| derivative work) provided[...] RSA Data Security, Inc |
| LI | GPL | the( terms of the)? GNU General Public License |
| DE | CeCILL-C | software( is governed by\| under) the ( terms of the)? CeCILLC license ( under French law and abiding\| as circulated)[...] |

Answer to RQ1:

By using metacharacters of regular expressions to generate license rules, we were able to aggregate 3.8% for FreeBSD-10.3.0, 5.9% for Linux-4.4.6, and 4.8% for Debian-7.8.0.

#### 4.3.2 RQ2: Is the Number of License Rules Generated by Our Approach Less than the AS–IS Method?

**Motivation:** This question addresses the prevention of the generation of many license rules (C-2). The license generation

[*2] https://github.com/freebsd/freebsd/tree/release/10.3.0
[*3] https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.6.tar.xz
[*4] http://ftp.riken.jp/pub/Linux/debian/debian-cd/7.8.0/source/iso-dvd/
[*5] https://github.com/dmgerman/ninka/releases/tag/1.1

**Table 7** Number of generated rules.

| Project | #sentences | Approach | #generated rules |
|---|---|---|---|
| FB | 9,435 | Proposed Method | **457** |
| | | AS–IS | 1,422 |
| LI | 8,661 | Proposed Method | **269** |
| | | AS–IS | 2,916 |
| DE | 9,807 | Proposed Method | **419** |
| | | AS–IS | 5,932 |

method preferably generates a small number of license rules. The more rules are generated, the greater the workload for the latter task of naming the license rules by manual. In our approach, we build license rules from frequently appearing license sentences to prevent the generation of a large number of license rules.

**Approach:** We compare the number of license rules generated by our method to the number of license rules when license statements are added to a license identification tool as is. We call a method for adding license statements as license rules as the AS–IS method. AS–IS is the simplest way to create a license rule, break down a license statement into sentence units and eliminating duplication.

**Result:** **Table 7** shows the number of rules generated by the proposed method and AS–IS. The number of license rules generated for FreeBSD-10.3.0 is 457 by the proposed method and 1,422 by AS–IS, and for Linux-4.4.6 the proposed method is 269 and AS–IS is 2,916. This indicates FreeBSD-10.3.0 and Linux-4.4.6 have many duplicate license statements. For Debian-7.8.0, AS–IS generated 5,932 license rules while the proposed method generated 419. This means that Debian-7.8.0 has fewer overlapping license statements (5,932/9,807) compared to other two projects but our method successfully reduce the number of generated license rules for Debian-7.8.0. From these results, we can answer for RQ2 as follows:

Answer to RQ2:

The number of license rules generated by the proposed method was 457 for FreeBSD-10.3.0, 269 for Linux-4.4.6, and 419 for Debian-7.8.0, indicating that the proposed method generates fewer license rules than AS–IS method. In addition, compared to AS–IS, the proposed method is less sensitive to the duplication of license sentences.

#### 4.3.3 RQ3: Does Our Approach Generate Licensing Rules with better Performance than the AS–IS Method?

**Motivation:** In Section 2.3, we discussed the importance of not generating a large number of license rules. At the same time, identifying a larger number of license statements is important for supporting the license identification process. In this research question, we address the performance of license rules generated by the proposed method from the two factors in the trade-off relationship.

**Approach:** The license rules generated by the proposed method and the AS–IS method are respectively combined with default rules of Ninka and matched with the license sentences in each dataset used for the rule generation. We evaluate the increase in the number of matched license sentences by adding the generated license rules by our method and the AS–IS method.

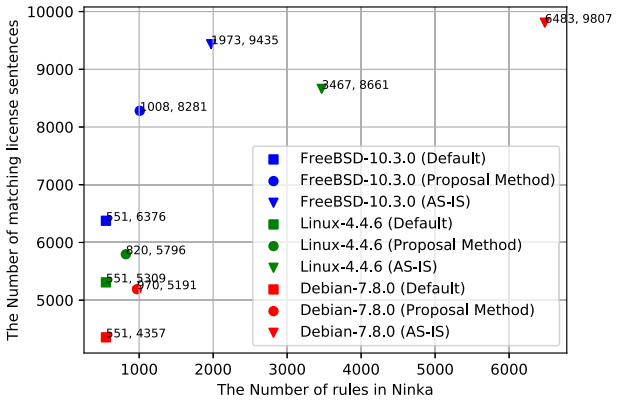


**Fig. 5** The number of license rules used to identify OSS licenses and the number of matched license sentences.

$$ER = 1 - \frac{\#\ rules\ for\ sentence\ matching}{\#\ matched\ sentences} \tag{5}$$

$$MR = \frac{\#\ matched\ sentences}{\#license\ sentences} \tag{6}$$

$$PR = \frac{2 * ER * MR}{ER + MR} \tag{7}$$

We define the following metrics and evaluate the harmonic mean of the two factors for each method. The Efficiency of Rules (ER), shown in Eq. (5), is the normalized proportion of license sentences that the rules match. The higher the ER value, the fewer the rules that are generated. The Matching Rate (MR), shown in Eq. (6), is the percentage of matched license sentences in the total. The higher the MR value, the more license sentences are matched. The Performance of Rules (PR), shown in Eq. (7), represents the harmonic mean of the trade-off ER and MR values. We calculate the PR value and evaluate whether the proposed method is higher than the AS–IS or not. The performance of the default rules of Ninka (Default) is also included for comparison. Ninka already has 551 license rules by default.

**Result:** **Figure 5** shows the number of license rules used to identify OSS licenses and the number of matched license sentences. 551 Default rules in Ninka matched 6,376 (67.6%), 5,309 (61.2%) and 4,357 (44.4%) license sentences in FreeBSD-10.3.0, Linux-4.4.6 and Debian-7.8.0 respectively. We found that the proposed method matched 8,281 (87.8%), 5,796 (66.9%), 5,191 (52.9%) license sentences in FreeBSD-10.3.0, Linux-4.4.6 and Debian-7.8.0 respectively. On the other hand, we also found that the AS–IS method matched a larger number of license sentences than our method while it also requires a much greater number of license rules than our method.

**Table 8** shows the number of rules when matching license sentences and ER, MR, and PR values calculated from the matched license sentences.

The ER value of Default was 0.91 for Free-BSD-10.3.0, 0.90 for Linux-4.4.6, and 0.87 for Debian-7.8.0, the highest among all methods. This indicates that the default license rules in Ninka, which were created manually, are the rules that match more license sentences.

The MR value of AS–IS was 1.00 for all projects, therefore, the AS–IS is the highest of MR value among all methods, since AS–IS method used all sentences including unknown license statements “as-is”. The MR values of our method were better than

**Table 8** Performance of generated rules.

| Project | Approach | #rules | #matched sentences | ER | MR | PR |
|---|---|---|---|---|---|---|
| FreeBSD-10.3.0 | Default | 551 | 6,376 | **0.91** | 0.68 | 0.78 |
| | Proposed Method | 1,008 | 8,281 | 0.88 | 0.88 | **0.88** |
| | AS–IS | 1,973 | 9,435 | 0.79 | **1.00** | **0.88** |
| Linux-4.4.6 | Default | 551 | 5,309 | **0.90** | 0.61 | 0.73 |
| | Proposed Method | 820 | 5,796 | 0.86 | 0.67 | **0.75** |
| | AS–IS | 3,467 | 8,661 | 0.60 | **1.00** | **0.75** |
| Debian-7.8.0 | Default | 551 | 4,357 | **0.87** | 0.44 | 0.59 |
| | Proposed Method | 970 | 5,191 | 0.81 | 0.53 | **0.64** |
| | AS–IS | 6,483 | 9,807 | 0.34 | **1.00** | 0.51 |

the Default method. This indicates that generating and adding license rules increases the number of license sentences that can be identified.

The PR value of Default was 0.78 for Free-BSD-10.3.0, 0.73 for Linux-4.4.6, and 0.59 for Debian-7.8.0. The PR value of AS–IS was 0.88 for Free-BSD-10.3.0, 0.75 for Linux-4.4.6, and 0.51 for Debian-7.8.0. In AS–IS, the PR value of Debian-7.8.0 is lower than Default, which indicates that AS–IS method is a performance degrading method for Debian Debian-7.8.0. On the other hand, the PR value of our method was the same as AS–IS for FreeBSD-10.3.0 and Linux-4.4.6, and 0.64 for Debian-7.8.0 which is higher than Default and AS–IS. Therefore, the proposed method is the most dominant method in terms of rule performance. From these results, we can answer RQ3 as given below.

Answer to RQ3:

The PR values indicating the performance of the license rules generated by the proposed method were 0.88 for FreeBSD-10.3.0, 0.75 for Linux-4.4.6, and 0.64 for Debian-7.8.0, which were higher than AS–IS and Default. The performance was the highest among all methods.

## 5. Discussion

### 5.1 Improving the Consolidation Rate of License Rules

In RQ1, we evaluated the consolidation of orthographical variants in license statements for Technical Challenges (C-1). We found that we were able to extract license rules with metacharacters of regular expressions such as RSA-MD license and CeCILL-C license as shown in Table 6. However, the consolidation rate was not very high.

As an additional study, we counted the metacharacters in 551 license rules which were manually created and included in Ninka. and investigated the consolidation ratio of the default license rules included in Ninka. We calculated the consolidation rate for the license rules in Ninka and found that at least 133 metacharacters are used in Ninka's license rules with the consolidation ratio of 19.4%. The reason why the results of the consolidation ratio with our method is only a few % is because the threshold for filtering our outliers as described in Section 3.2 is set high (94%).

Furthermore, we inspected the license rules generated in RQ1 and found that many license rules including license statements with orthographical variants were eliminated by filtering out outliers from each cluster.

The number of filtered license statements was 205 for FreeBSD-10.3.0, 1,150 for Linux-4.4.6, and 1,869 for Debian-7.8.0. We found two reasons of the elimination: one is that Ninka extracts not only license statements but also other texts such as explanations of applications and another is that Ninka has some rules including frequently-used words such as "copy" and "modified", which can lead to erroneously extract source code comments. In future work, we will need to review words appearing in explanations of applications and source code and refine the set of license-related words used in Ninka. However, in order to exclude very similar but different license statements, it is basically necessary to set the filtering threshold higher. Since an expert needs to determine whether the legal meaning of such slight differences is actually different or not, we can expect some improvements but it would be difficult to make the consolidation rate equal to that of Ninka's. license rules.

### 5.2 Efficiency of Manual License Rule Creation

In RQ2 and RQ3, we evaluated the performance of the automatically generated license rules for Technical Challenges (C-2). In RQ2, we evaluated the number of generated license rules and found that the number of license rules of the proposed method is less when compared to AS–IS. In addition, the AS–IS method generated a much larger number of license rules for the Debian-7.8.0 dataset, which contains many licenses, compared to FreeBSD-10.3.0 and Linux-4.4.6, while the proposed method generated almost the same number of license rules as other projects. The reason for this is that the proposed method generates license rules based on frequent expressions patterns, and therefore, our method does not generated so many license rules. Even on a dataset with a large number of license types and a small number of overlapping license statements such as Debian-7.8.0, our method is able to output a stable number of license rules. Therefore, this property is desirable in terms of performing continuous maintenance of the license rules, since there is less subsequent manual work involved in naming the rules.

From a practical standpoint, we believe that the number of rules generated by our method will contribute to the continuous maintenance of the license rules. Ministry of Economy, Trade and Industry (METI) in Japan reports [*6] that it takes about ten minutes to identify a single license. For 457 rules for FreeBSD-10.3.0 in RQ2, it would take 76.1 hours with our method (cf. 237 hours with AS–IS). This means our method works in a feasible range if the tasks of visual inspection and naming of license statements are

[*6] Ministry of Economy, Trade and Industry (METI), Direction of the Task Force on Software Management Methods to Ensure Cyber-Physical Security, p.29, March, 2022.
https://www.meti.go.jp/shingikai/mono_info_service/sangyo_cyber/wg_seido/wg_bunyaodan/software/pdf/006_03_00.pdf

performed by multiple persons. Our case study was conducted in the domain of operating systems as a large-scale OSS. It indicates that our method could provide a value not only to developers who produce small applications using OSS but also to manufacturers who use Linux as an embedded OS.

In RQ3, we evaluated the performance of the license rules. We found that adding the license rules generated by our method to Ninka improved the performance of the Default license rules. In addition, the performance of our method is comparable or better than the AS–IS method. Since our method contributes to to adding fewer license sentences to Ninka, less effort is required to name the rules. Therefore, even if the PR value is the same as that of AS–IS, our method with the superior ER value is more useful in improving the overall efficiency of the license rule creation process. We also compare the results with Ninka's default licensing rules. The ER value of our method is lower compared to Default. This indicates that Default licensing rules created manually are more efficient. The Default license rule of Ninka is written by developers who are knowledgable about orthographical variants in license statements.

The MR value for FreeBSD-10.3.0, Linux-4.6.0, and Debian-7.80 were 0.88, 0.67, and 0.53, respectively, indicating the degree to which they can actually support the identification of unknown licenses. Since 88% of the unknown licenses were matched for FreeBSD-10.3.0, we believe it is sufficient to identify unknown licenses. However, for the other two projects, the generated rules matched less than 70% of the license statements (i.e., more than 30% of the unknown licenses still remain to be identified). This suggests that there is room for improvement. In the future, we will need to generalize known orthographical variants to improve the ER value of our method.

### 5.3 Threats to Validity

#### 5.3.1 Internal Validity

In this paper we conducted a case study to evaluate the automated license rule extraction approach. This approach heavily depends on results from the hierarchal clustering method proposed in our previous study [3]. Since the clustering method still has room for improvement, the enhanced clustering method in the future might change the result of our case study in the paper.

Furthermore, our approach in this paper relies on the three kinds of thresholds described in Section 4.2: (1) maximum similarity for removing license statements, (2) minimum support for BIDE and (3) minimum length of license statement patterns. Although the result of our case study might change depending on these thresholds, we believe that we have at least set the best thresholds for our dataset.

#### 5.3.2 External Validity

Including our previous study, we tested our approach only with the datasets from the three OSS products: FreeBSD-10.3.0, Linux-4.4.6, and Debian-7.8.0. Although these are representative examples of very popular and large-scale OSS, we have no data for other OSS products. We plan to replicate the case study in this paper for other OSS products to improve the generality of using our approach.

In this paper, in order to prepare the ground truth, we manually identified the license names of unknown licenses detected by Ninka from FreeBSD-10.3.0, Linux-4.4.6, and Debian-7.8.0. The ground truth was created by the authors who are industrial practitioners and academic researchers. Since they are neither lawyers nor practitioners engaged in legal practices, the quality of the ground truth might depends on authors' interpretations of license statements. We will need to work with legal experts in the future to ensure the generality in creating the ground truth for rigidly evaluating our approach.

## 6. Related Work

### 6.1 License Identification Tools

There are many license identification tools presented in existing work. These tools are categorized into similarity-based approaches and a regular expression–based approaches. Similarity-based approaches are FOSSology [8] and OSLC [5]. FOSSology uses bSAM algorithm originally used in matching patterns among base sequences of proteins. This approach does not require so many license rules for variations of license statements because this tool finds similar license rules for the target license statements. This leads to high recall but the tool is very slow. German et al. [1] reports that FOSSology require 923 seconds for 250 files although Ninka require only 22 seconds. On the other hand, the examples of regular expression–based approaches are Ohcount [6], OSLC [5] and ASLA [7]. These approaches achieve high precision because rules written in regular expressions make it easier to distinguish one license from another. However, the performance of this approach depends on the quantity and quality of license rules. German et al. [1] reported that Ohcount [6] and OSLC [5] sometimes fail to identify licenses because of simple regular expressions. For example, "This file is not licensed under the GPL" is erroneously identified as GPL. In this study, we focused on Ninka, a regular expression–based approach, to achieve high precision and quick identification.

Some license identification tools do not address source files but other format files. Di Penta et al. [11] proposed a license identification approach for jar archives. Hemel et al. [12] deal with binary files. These approaches are based on the license identification tool for source files such as Ninka. Our approach also can improve these approaches.

### 6.2 License Violation Detection

The license identification is a fundamental technology for detecting license violations. Mlouki et al. [13] proposed a license violation detection approach for the Android ecosystem. They analyzed 857 Android applications and found license violations in 17 of them. In addition, seven applications have not removed the license violations yet. Lokhamn et al. [14] proposed a tool to detect license conflicts on software architectures. Alspaugh et al. [15] proposed a model for representing a license and detecting license conflicts based on the model. This model represents rights and obligations of a license as a tuple. This tuple consists of Actor, Modality, Action, Object, and reference to the other licenses. German et al. [16] proposed patterns to resolve license conflicts. German et al. [17] also proposed an approach to analyze incompatibilities between licenses of software pack-

ages specified by meta-data in the package and licenses of source files. Di Penta et al. [18] proposed an approach to find license violations automatically from software development history. Our approach enhances existing license identification approaches.

### 6.3 OSS License and Software Development

Several existing studies report that OSS license has an impact on software developments. Kashima et al. [19] studied what licenses of source files have impact on software reuse by copy and paste. Colazo et al. [20] studied the relations between OSS licenses and project activities. They state that the quantity of code and OSS project continuity following copy left licenses such as GPL is higher or larger, and development time is shorter than them of OSS projects following non copy left licenses. Sojer et al. [21] conducted an interview with software developers on software reuse. The result showed that their knowledge on software licenses is insufficinet. Almeida et al. [4] interviewed software developers about OSS licenses. They found that developers have a good understanding of popular licenses but have only a limited understanding when they need to deal with multiple open source licenses. Meloca et al. [22] investigated the impact of non-OSI-approved licenses on software projects from 657,000 open-source projects. Vendome et al. [23] conducted a study to find when and how software developers decide an OSS license. Vendome et al. [24] analyzed 1,200 license reported in the bug tracking system.

These studies show the importance of analyses on the use of the software license to understand software development. Our work also contributes to the progress of these analyses.

## 7. Conclusion and Future Work

In this study, we propose a method for generating license rules (e.g., regular expressions) that identify license statements from clusters of unknown license statements. Our method firstly applies filtering to each cluster of license statements, followed by sequential pattern mining, and convert the extracted patterns into license rules. In addition, it uses metacharacters of regular expressions to aggregate multiple license rules into a single license rule. To evaluate the proposed method, a case study was conducted using 1,821, 3,561, and 2,838 unknown license statements detected from FreeBSD-10.3.0, Linux-4.4.6, and Debian-7.8.0. The results showed that the proposed method identifies more licenses with a minimum number of license rules, and that adding license rules created by the proposed method to Ninka improves the performance of the license rules. Future work includes reviewing keywords used in Ninka for license statements extraction, and generalizing orthographical variants.

**Acknowledgments** This work is conducted as part of Grant-in-Aid for Scientific Research: (C) 22K11974 by Japan Society for the Promotion of Science.

### References

[1] German, D.M., Manabe, Y. and Inoue, K.: A Sentence-Matching Method for Automatic License Identification of Source Code Files, *Proc. IEEE/ACM International Conference on Automated Software Engineering* (*ASE '10*), pp.437–446 (2010).

[2] Higashi, Y., Manabe, Y. and Ohira, M.: Clustering OSS License Statements Toward Automatic Generation of License Rules, *Proc. 7th IEEE International Workshop on Empirical Software Engineering in Practice* (*IWESEP2016*), pp.30–35 (2016).

[3] Higashi, Y., Ohira, M., Kashiwa, Y. and Manabe, Y.: Hierarchical Clustering of OSS License Statements Toward Automatic Generation of License Rules, *Journal of Information Processing* (*JIP*), Vol.27, pp.42–50 (2019).

[4] Almeida, D.A., Murphy, G.C., Wilson, G. and Hoye, M.: Do Software Developers Understand Open Source Licenses?, *Proc. 25th International Conference on Program Comprehension* (*ICPC '17*), pp.1–11 (2017).

[5] OSLC: Open Source License Checker, available from ⟨https://sourceforge.net/projects/oslc/⟩ (accessed 2020-07-10).

[6] Ohcount: Ohloh's source code line counter, available from ⟨https://github.com/blackducksw/ohcount⟩ (accessed 2020-07-10).

[7] Tuunanen, T., Koskinen, J. and Kärkkäinen, T.: Automated software license Analysis, *Automated Software Engineering*, Vol.16, No.3-4, pp.455–490 (2009).

[8] Gobeille, R.: The FOSSology Project, *Proc. 2008 International Working Conference on Mining Software Repositories* (*MSR '08*), pp.47–50 (2008).

[9] Wang, J., Dang, Y., Zhang, H., abd Tao Xie, K.C. and Zhang, D.: Mining succinct and high-coverage API usage patterns from source code, *Proc. 2013 10th Working Conference on Mining Software Repositories* (*MSR '13*), pp.319–328 (2013).

[10] Wang, J. and Han, J.: BIDE: Efficient Mining of Frequent Closed Sequences, *Proc. 20th International Conference on Data Engineering* (*ICDE '04*), pp.79–90 (2004).

[11] Di Penta, M., German, D. and Antoniol, G.: Identifying licensing of jar archives using a code-search approach, *Proc. 7th IEEE Working Conference on Mining Software Repositories* (*MSR '10*), pp.151–160 (2010).

[12] Hemel, A., Kalleberg, K.T., Vermaas, R. and Dolstra, E.: Finding Software License Violations Through Binary Code Clone Detection, *Proc. 8th Working Conference on Mining Software Repositories* (*MSR '11*), pp.63–72 (2011).

[13] Mlouki, O., Khomh, F. and Antoniol, G.: On the Detection of Licenses Violations in the Android Ecosystem, *Proc. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering* (*SANER '16*), pp.382–392 (2016).

[14] Lokhman, A., Luoto, A., Hammouda, I. and Mikkonen, T.: Open Source Legality Compliance of Software Architecture, A Licensing Profile Approach, *Proc. 8th International Conference on Software Engineering Advances* (*ICSEA '13*), pp.571–578 (2013).

[15] Alspaugh, T.A., Asuncion, H.U. and Scacchi, W.: Analyzing Software Licenses in Open Architecture Software Systems, *Proc. ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (*FLOSS '09*), pp.54–57 (2009).

[16] German, D.M. and Hassan, A.E.: License Integration Patterns: Addressing License Mismatches in Component-based Development, *Proc. 31st International Conference on Software Engineering* (*ICSE '09*), pp.188–198 (2009).

[17] German, D.M., Penta, M.D. and Davies, J.: Understanding and Auditing the Licensing of Open Source Software Distributions, *Proc. 2010 IEEE 18th International Conference on Program Comprehension* (*ICPC '10*), pp.84–93 (2010).

[18] Di Penta, M., German, D.M., Guéhéneuc, Y.-G. and Antoniol, G.: An Exploratory Study of the Evolution of Software Licensing, *Proc. 32nd ACM/IEEE International Conference on Software Engineering* (*ICSE '10*), pp.145–154 (2010).

[19] Kashima, Y., Hayase, Y., Yoshida, N., Manabe, Y. and Inoue, K.: An Investigation into the Impact of Software Licenses on Copy-and-paste Reuse among OSS Projects, *Proc. 2011 18th Working Conference on Reverse Engineering* (*WCRE '11*), pp.28–32 (2011).

[20] Colazo, J. and Fang, Y.: Impact of License Choice on Open Source Software Development Activity, *American Society for Information Science and Technology*, Vol.60, No.5, pp.997–1011 (2009).

[21] Sojer, M. and Henkel, J.: License Risks from Ad Hoc Reuse of Code from the Internet, *Comm. ACM*, Vol.54, No.12, pp.74–81 (2011).

[22] Meloca, R., Pinto, G., Baiser, L., Mattos, M., Polato, I., Wiese, I. and German, D.: Understanding the Usage, Impact, and Adoption of Non-OSI Approved Licenses, *Proc. 15th International Conference on Mining Software Repositories* (*MSR '18*), pp.270–280 (2018).

[23] Vendome, C., Linares-Vásquez, M., Bavota, G., Penta, M.D., German, D.M. and Poshyvanyk, D.: When and Why Developers Adopt and Change Software Licenses, *Proc. 2015 IEEE International Conference on Software Maintenance and Evolution* (*ICSME '15*), pp.31–40 (2015).

[24] Vendome, C., German, D.M., Di Penta, M., Bavota, G., Linares-Vásquez, M. and Poshyvanyk, D.: To Distribute or Not to Distribute? Why Licensing Bugs Matter, *Proc. 40th International Conference on*

*Software Engineering* (*ICSE '18*), pp.268–279 (2018).

**Yunosuke Higashi** received his M.E. degree in engineering from Wakayama University, Japan in 2017. He is currently engaged in the development of financial systems at Japan Research Institute, Ltd. He has also been a Ph.D. student at Wakayama University since 2020. His research interests include open source software license and open source software engineering.

**Masao Ohira** received his Ph.D. degree from Nara Institute of Science and Technology, Japan in 2003. He is currently Associate Professor at Wakayama University, Japan. He is interested in software maintenance and software repository mining. He is a director of Open Source Software Engineering (OSSE) Laboratory at Wakayama University. He is a member of ACM and IEEE.

**Yuki Manabe** received his Ph.D. degree in Information Science and Technology from Osaka University in 2011. He is Lecturer at the University of Fukuchiyama from 2020. His research interests include open source software license, open source software development and software repository mining.