Development of a Code Clone Search Tool for Open Source Repositories

Pei Xia Yuki Manabe Norihiro Yoshida Katsuro Inoue

Finding code clones in the open source systems is important for efficient and safe reuse of existing open source software. In this paper, we propose a novel search model, open code clone search, to explore code clones in open source repositories on the Internet. Based on this search model, we have designed and implemented a prototype system named *OpenCCFinder*. This system takes a query code fragment as its input, and returns the code fragments containing the code clones with the query. It utilizes publicly available code search engines as external resources. Using *OpenCCFinder*, we have conducted several case studies for Java code. These case studies show the applicability of our system.

1 Introduction

Nowadays reusing existing source code to build up new software systems becomes common. Developers can easily get source code of various projects that hosted in open source repositories on the Internet such as Google code, sourceforge, github and so on. Some previous studies show that even industrial software products are also increasingly reusing open source code due to their reliability and cost benefits [4].

However, when reusing source codes, some problems about software compliance may happen.

1. When we find a useful source code file, can we reuse it safely?
2. Are our own open source projects illegally reused by other people?

For the first question, before reusing the source code, developers should make sure that they will not violate the license. A license violation may take them to court and cost them a lot. However, to tell the license of a source file is not easy because there are many code clones among open source projects [13]. That means they also copy and modify source code from other projects. In extremely cases they even change or remove the original license statement in the source files [1]. Reusing such source files is risky. For the second question, even though some other projects had reused source code while violating its license, the original code owner would hardly know it, since it is hard to check other projects by hand.

In order to answer such questions, one solution is to find out all the cloned code in the world, and compare the related information about them to tell the reuse relationship between those codes. In these days, many code clone detection tools as well as code search tools have been developed [5][7]. However, none of them can search for code clones from open source repositories.

Basing on the code clone detection and code search technologies, we proposed a novel approach for open source code clone search, and also implemented a prototype tool named *OpenCCFinder* [12]. *OpenCCFinder* takes a code fragment as its query input, and returns a list of files from open source repositories that contain cloned code with the queried one, along with extra information. This

オープンソースレポジトリに対するコードクローン検索ツールの開発.

夏沛, 眞鍋雄貴, 井上克郎, 大阪大学大学院情報科学研究科コンピュータサイエンス専攻, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University.

吉田則裕, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

コンピュータソフトウェア, Vol.29, No.3 (2012), pp.181–187.
[研究論文 (レター)] 2011 年 12 月 12 日受付.

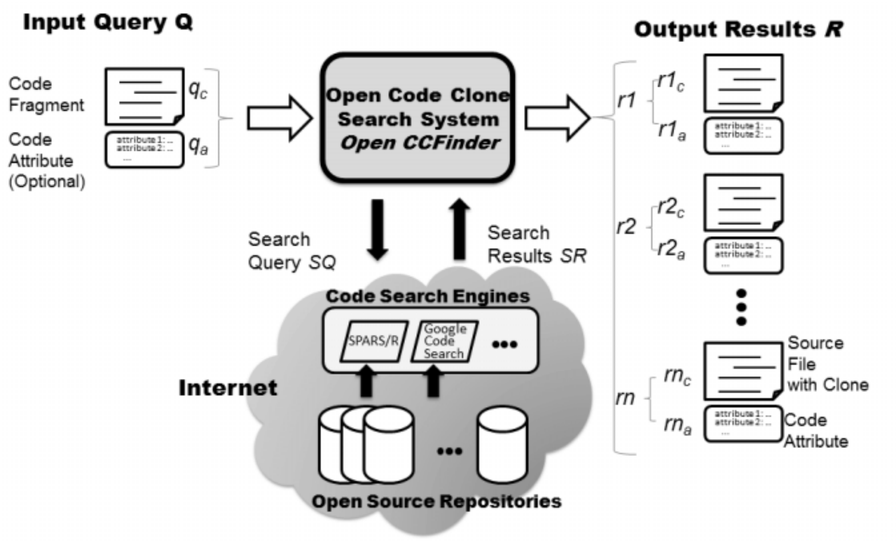

Fig. 1 Architecture of *OpenCCFinder*.

tool can support us to study the raised problems.

In this paper, we first describe the overview of *OpenCCFinder* including architecture and search process in Section 2. Section 3 shows our case studies. Section 4 discusses our approach and Section 5 shows the related works. In Section 6, we conclude our discussions with some future works.

2 Overview of *OpenCCFinder*

2.1 Architecture

Figure 1 shows the architecture of *OpenCCFinder*. It takes an input query Q and returns an output results set R. Input query Q is composed of code fragment q_c and code attribute q_a. q_c may be a complete source file or a part of a source code file, which is in question. q_a is a set of associated information characterizing q_c, such as the file name. q_a is optional and could be added to improve the quality of the output results. Given an input Query Q, *OpenCCFinder* extracts useful information from it and generates queries for external code search engine (e.g. Google code search, SPARS/R etc.), and then analyze the returned candidate files from external search engines, at last form a final result as output R. The detail of this search process will be introduced in section 2.2.

Output result R is composed of results $r1$,$r2$ ... rn. Each result ri is composed of a code file ri_c and its code attribute ri_a. ri_c is a code file which is returned by external code search engines, and ri_a a set of associated information about ri_c, including URL, file path, LOC, license, copyright, last modified time, clone cover ratio and clone detail.

For the external code search engines, we use Google code search and SPARS/R in our tool implementation. Google code search is a famous code search engine. It provides search service API to the user, so we can easily integrate it to our tool; SPARS/R is a Java component search engine with the keyword input and component rank mechanism developed by our research group [9]. The Java class repository of SPARS/R is kept updated by us.

2.2 Search process

Search process of *OpenCCFinder* can be devided into 6 steps, as shown in Figure 2.

(a) **Word Extraction.** At the beginning, code fragment q_c in input query Q is tokenized, and the words from source code and comments are separated. User can choose whether to extract words from source code or from comments, or from both.

(b) **Keyword Ranking.** Next, the keywords used for query generation are selected from the extracted words. In this step, first *OpenCCFinder* filters out the words that con-

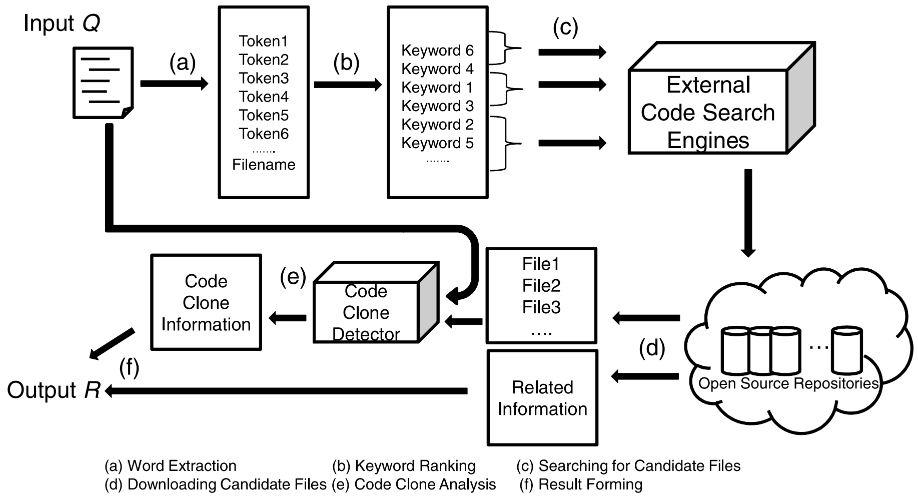

Fig. 2 Search process of *OpenCCFinder*.

sidered being featureless. For example, the reserved words of each source code language, the words in very short length, and the words included in customized filter are filtered out. After the filtering, a simple words importance ranking strategy is applied on the remaining words. Currently, there are two strategy implemented in the tool for ranking the words: frequency strategy and random strategy. Frequency strategy is to rank the keywords by the times they appear in the source codes or comment, while random strategy is just to rank the words randomly.

(c) **Searching for Candidates Files.** As the search engines, here we choose SPARS/R and Google Code Search. We use different combination of high ranked keywords to generate search queries for each search engine to get appropriate candidate files. For each query, the returned results set should not be very large, so as not to include too many irrelevant results. If the returned results set are too large, we will add one more keyword from the ranked keywords list to the query to narrow the results set. At last, we merge the returned results of several queries as the analysis candidate files.

(d) **Downloading Candidate Files.** All the candidate files in step (c) are downloaded from Internet. While downloading the file, the tool is also crawling the web to extract useful information for the code attributes such as file path, URL, LOC, License, Copyrights, and last modified time if available.

(e) **Code Clone Analysis.** The code clones between the input query code fragment q_c and each source code file obtained at Step (d) are computed. We have used a code clone detection tool CCFinder [17], with its parameter setting for the minimum token length 15.

(f) **Result Forming.** All the candidate files and their code attributes are combined and packed as the output result R of this system, sorted by their similarity to q_c. Here we use *coverratio* to evaluate their similarity, which defined as the percentage of reused q_c's code.

3 Case Study

We have conducted two case studies to explore applicability of *OpenCCFinder* approach[†1].

3.1 Case study 1: *base64.java*

Case study 1 is designed for the first raised question: When we find some useful source code, can we reuse it safely?

Consider such a situation that we have found a

†1 All these case studies have been performed under PC Workstation with dual Xeon X5550 2.66GHz processors and 24 GB memory between Aug.2011 and Sep.2011.

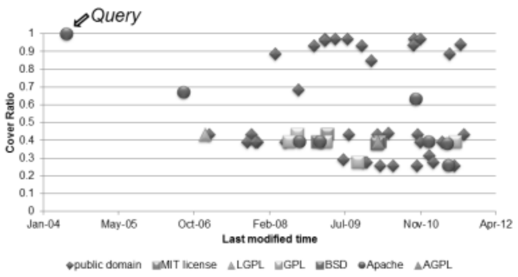

Fig. 3 Files that contain code clones with base64.java found by *OpenCCFinder*.

file named base64.java [3] in Apache ObjectRelationBridge (Apache OJB) open source project [2] and we would like to reuse it. The comments section in the source file represents that this file is under the Apache license. But we wondered if this file is copied from somewhere else that may be under another license. Then we take this base64.java file as input and search for similar files from open source repositories using *OpenCCFinder*.

OpenCCFinder returns 57 other files from open source repositories that contain code clone with the base64.java. For the limited space here, we cannot present all the detailed data of the 57 files. Instead we organized the data and draw a scatterplot view, as shown in Figure 3. In the figure, the files are distributed by their cover ratio and last modified time. Licenses are shown in different icons. From it, we can observe the following:

1. 55 source files from other projects contain code clones of base64.java. The last modified time is varied from 2004 to current. The earliest file we can find is under Apache license.
2. In these files, the cover ratios are not the same, which may indicate these files reused and modified from each other in different ways.
3. Most of the licenses are found as public domain, while several files have been found under MIT, LGPL, GPL, BSD, Apache or AGPL licenses.

After checking the detailed information of each file by hand, we confirmed that it is safe to keep this file as Apache license. Another choice is to reuse the public domain code, which is also safe. Though *OpenCCFinder* cannot tell the answer of the first question directly, it helps us to do the analysis easier.

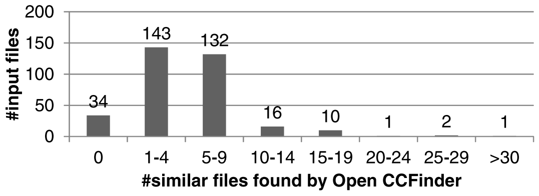

Fig. 4 The histogram of #input files in terms of #similar files found by *OpenCCFinder*.

3.2 Case study 2: *SSHTools*

Case study 2 is designed for the second question: Are our own open source projects illegally reused by other people? In this case study, we investigate a Java project SSHTools [14], to find some files that may illegally reused by other projects.

SSHTools is a Java SSH application providing Java SSH API and terminal, which is under the GPL license. We choose this project because it is widely used by other projects and it is a small project that is easy to analyze.

Ignoring some tiny sized files, we have selected 339 files from SSHTools project (version 0.2.9, last modified time is 6-23-2007), and each of them is used as the input of *OpenCCFinder*. We counted the number of similar files found in open source repositories for each SSHTools' file, and also counted the number of different licenses of these similar files. In this case study, we set up a threshold of the cover ratio to filter out non-similar files. Only those files which cover more than 40% code of the queried file would be identified as similar file. The result is shown as Figure 4. The figure shows the number of similar files found in each of the 339 SSHTools' files.

From this figure, we can observe that 305 of the 339 files contain code clones with other files from open source repositories. 275 of them have less than 10 similar files found for each; several files have 10-30 similar files found for each; one file has more than 30 similar files. Besides, we also investigated the different licenses appeared in each similar file. SSHTools is under GPL license. However, 285 files in SSHTools have similar files found with 1 different license, 10 files in SSHTools have similar files with 2 different licenses, and 1 file in SSHTools have

Table 1 The case that similar files are under 3 other license (partial results)

File path	Project name	Cover ratio	License	Last modified time
/j2ssh-fork/src/com/sshtools/common/util/BrowserLauncher.java	j2ssh-fork	0.91	GPL	2008/6/17
/de.fzj.unicore.rcp.terminal.ssh.gsissh/.../sshtools/common/util/BrowserLauncher.java	unicore	0.89	LGPL	2010/2/3
/openfire/launcher/BrowserLauncher.java	openfire-tomcat	0.88	Apache	2010/4/19
/dg/hipster/BrowserLauncher.java	hipster	0.84	BSD	2006/10/12
...	...	...	...	...

similar files with 3 different licenses.

We checked the detailed files whose cloned files have 2 or 3 different licenses. There are several unusual source code reuse cases. Here we only state one case of them as an example.

Table 1 shows the similar files with 3 other different licenses, along with extra information about file path, project name, cover ratio and last modified time. For the space limitation, we only present a small part of the results here.

From this table, we can see 4 files named BrowserLauncher.java with very high cover ratio. Beside the GPL license, there are LGPL, Apache and BSD licenses. It is reasonable for us to suppose that they share the same origine. However, sometimes it is not allowed to release the reused source code under a different license. Some reuse activities in this case are suspicious.

This case study shows that *OpenCCFinder* is helpful for answering the second raised question. We can find candidates of the suspicious files easily and effectively by using this tool.

4 Discussion

4.1 Usefulness

As shown in the case studies, *OpenCCFinder* is helpful to analyze how source code is reuse. In case study 1, from the open source repositories we search out many similar files of Apache OJB's file base64.java. With extra information about each file, we can know how the searched code is used in different projects. Then developer's reuse activity that we focus on becomes easy and efficient. In case study 2, we have found out files from open source repositories containing code clones with SSHTools files, among which there are several suspicious cases.

However, although this tool provides some clues to get evidence, the final judgment on the legality issue should be made by human after all.

And due to the keyword based search model, *OpenCCFinder* can only find out Type 1 clones [5].

4.2 Performance

It takes about 1-2 minutes for *OpenCCFinder* to complete one search task, including keyword extracting, downloading, collecting information and running CCFinder. Most of the time is spend at downloading step. Network traffic and the size of file to be downloaded also affect the executing time. Case study 1 took about 2 minutes, while case study 2 took about 7-8 hours in total.

4.3 Recall and precision

OpenCCFinder searches for code clones in open source repositories using external code search engines. So the recall and precision of this tool depends on those search engines. It is difficult to evaluate recall of *OpenCCFinder* quantitatively because we could never know all the files in open source repositories, and *OpenCCFinder* cannot download all the files stored in open source repositories due to the limitation of space and time cost. The precision of *OpenCCFinder* has been calculated. In this discussion, it is defined as the ratio of files containing code clones to all the downloaded files. In case study 1, 62 files have been downloaded, of which 55 files contain code clone with the queried file. The precision can be simply calculated as 0.887; in the case study 2, overall we have downloaded 17054 files from the Internet, and 2480 of those files have been detected as containing code clone with the files in SSHTools. So the average precision is 0.145. The precision in case study 2 is low, but non-cloned files are filered out by the later process.

5 Related work

There are many research studies on analyzing and tracing code origin, provenance, evolution, genealogy through code clone analysis [10][11][15][16]. Duala-Ekoko et. al propose Clone Tracker to trace and manage code clone history [6]. They have developed a tool for supporting clone tracking, with abstract clone information named clone region descriptor. Davies et al. propose Software Bertillonage for determining the origin of code entities with anchored signature matching method [8]. These studies are closely related to our work. However, their objectives are different from ours in the sense that they analyze various characteristics of code fragment in their local repositories. In our case, we analyze the query code in the Internet repositories.

6 Conclusion

In this paper, we have proposed a novel concept for open code clone search, and then presented its search model and detailed processes for the prototype system *OpenCCFinder*. We have conducted two case studies, which show the applicability of our approach.

Acknowledgements This work was supported by KAKENHI (No.21240002, No.23650015). Early version of *OpenCCFinder* is developed by our earlier student Yusuke Sasaki.

References

[1] Arne, P.H.: Jacobsen v. Katzer - Open Source License Validation: How Far Does It Go?, *The Computer & Internet Lawyer*, Vol. 25 No. 11 (2008), pp. 27–31.

[2] Apache ObJectRelationalBridge OJB, http://db.apache.org/ojb/

[3] Base64: Public Domain Base64 Encoder/Decoder, http://iharder.sourceforge.net/current/java/base64/

[4] Ebert, C. (ed.): Open Source Software in Industry, *IEEE Software*, Vol. 25, No. 3(2008), pp. 52–53.

[5] Roy, C. K., Cordy, J. R. and Koschke, R: Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, *Science of Computer Programming*, Vol. 74, No. 7 (2009), pp. 470–495.

[6] Duala-Ekoko, E. and Robillard, M. P.: Clone Region Descriptors: Representing and Tracking Duplication in Source Code, *ACM Tran. on Software Engineering*, Vol. 20, No. 1 (2010), pp. 3.1–3.31.

[7] Cordy, J., Inoue, K., Koschke, R. and Jarzabek, S. (eds.): *4th International Workshop on Software Clones* (*IWSC 2010*), Cape Town, South Africa, 2010.

[8] Davies, J., German, D. M. and Godfrey, M. W.: Software Bertillonage: Finding the Provenance of an Entity, in *Proc. of Working Conference on Mining Software Repositories* (*MSR 2011*), Honolulu, Hawaii, 2011, pp. 183–192.

[9] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Trans. Softw. Eng.*, Vol. 31, No. 3 (2005), pp. 213–225.

[10] Godfrey, M. and Zou, L.: Using Origin Analysis to Detect Merging and Splitting of Source Code Entities, *IEEE Trans. Softw. Eng.*, Vol. 31, No. 2(2005), pp. 166–181.

[11] Kim, M., Sazawal, V., Notkin, D. and Murphy, G.: An empirical study of code clone genealogies, in *Proc. of the 2005 European Software Engineering Conference and 2005 Foundations of Software Engineering* (*ESEC/FSE 2005*), Lisbon, Portugal, 2005, pp. 187–196.

[12] Xia, P., Manabe, Y., Yoshida, N. and Inoue, K.: Development of a Code Clone Search Tool for Open Source Repositories, *IPSJ SIG Technical Reports*, Vol.2011-SE-174, No.2 (2011), pp. 1–8.

[13] Livieri, S., Higo, Y., Matsushita, M. and Inoue, K.: Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder, in *Proc. of 29th International Conference on Software Engineering* (*ICSE 2007*), Minneapolis, MN, 2007, pp. 106–115.

[14] SSHTools Project, http://sourceforge.net/projects/sshtools/.

[15] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K.: MUDABlue: An Automatic Categorization System for Open Source Repositories, *Journal of Systems and Software*, Vol. 79, No. 7 (2006), pp. 939–953.

[16] Thummalapenta, S., Cerulo, L., Aversano, L. and Di Penta, M.: An empirical study on the maintenance of source code clones, *Empirical Software Engineering*, Vol. 15, No. 1 (2009), pp. 1–34.

[17] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7(2002), pp. 654–670.

夏　　沛

2010 年上海交通大学ソフトウェア工学部卒業．現在，大阪大学大学院情報科学研究科博士前期課程 2 年．コードクローン分析やビジュアライゼーションに関する研究に従事．

眞 鍋 雄 貴

2006 年大阪大学基礎工学部情報科学科退学．2011 年同大学大学院情報科学研究科博士課程修了．同年同大学院情報科学研究科特任助教．博士 (情報科学)．ソフトウェア工学，特にソフトウェア再利用，ソフトウェアライセンスの研究に従事．情報処理学会，ACM 各会員．

吉 田 則 裕

2004 年九州工業大学情報工学部知能情報工学科卒業．2009 年大阪大学大学院情報科学研究科博士後期課程修了．2010 年奈良先端科学技術大学院大学情報科学研究科助教．博士 (情報科学)．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．ソフトウェア科学会，情報処理学会，電子情報通信学会，人工知能学会，IEEE，ACM 各会員．

井 上 克 郎

1984 年大阪大学大学院基礎工学研究科博士後期課程修了 (工学博士)．同年，大阪大学基礎工学部情報工学科助手．1984〜1986 年，ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 年大阪大学基礎工学部助教授．1995 年同学部教授．2002 年大阪大学大学院情報科学研究科教授．2011 年 8 月より同研究科研究科長．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．