Evaluation of Source Code Copy Detection Methods on FreeBSD

Hung-Fu Chang
University of Southern California
University Park Campus,
University of Southern California
Los Angeles, CA 90089, USA
+1 213 740-9621
hungfuch@usc.edu

Audris Mockus
Avaya Labs Research
233 Mt. Airy Rd.
Basking Ridge, NJ, USA 07920
+1 908 696-5608
audris@avaya.com

ABSTRACT

Studies have shown that substantial code reuse is common in open source and in commercial projects. However, the precise extent of reuse and its impact on productivity and quality are not well investigated in the open source context. Previously, we have introduced a simple-to-use method that needs only a set of file pathnames to identify directories that share filenames and partially validated its performance on a set of closed-source projects. To evaluate this method and to improve reuse detection at the file level, we apply it and four additional file copy detection methods that utilize the underlying content of multiple versions of the source code on the FreeBSD project. The evaluation quantified unique advantages of each method and showed that the filename method detected roughly half of all reuse cases. We are still faced with a challenge to scale the content based methods to large repositories containing all versions of open source files.

Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance and Enhancement – Restructuring, reverse engineering, and reengineering; Version control

General Terms

Algorithms, Measurement

Keywords

Cloning, Version Control, Clone Detection, Code copying, Open Source

1. Introduction

Code reuse is a technique that reduces redundant work by copying existing code to another program during software development. Previous studies [2, 3] suggested that highly reused code provides more reliable code and requires less maintenance efforts. Besides, research on large-scale reuse detection indicated more implicit advantages of understanding code reuse relations among different projects and version control systems. For example, the code reuse relation may help trace bugs among all reused copies if we find one in anyone of them. Because source codes often embed the knowledge or expertise, knowing code transfer also means that we can discover the knowledge transfer between projects. Furthermore, we can identify the original authors of the program. Our particular objective is to join multiple version control systems via detected instances of file copying to analyze the complete history of each file.

Software repositories store the whole path of files, for example, "/directory1/directory2/file" and the content of each version of the file. Most code reuse detection methods focus on determining the copied files, reused components or reused functions based on the content of the underlying source code. Our previously proposed method [5], Filename Comparison (FC), suggested that it may be sufficient to have only the file paths to identify reuse at the file and directory level, without the need to extract and process massive volumes of multiple versions of the underlying source code. Although the accuracy of FC method has been validated with experiments on Avaya's projects with known instances of reuse, we could not establish how many files that were copied without our knowledge were missed by FC detection approach. This issue is particularly salient in open source projects, where no instances of file copy are known a priori. Therefore, constructing a validation process suitable for open source code is essential to establish the performance of FC and other methods. We designed and implemented four additional easy-to-implement file-level copy detection methods in this process. As other traditional methods, these methods rely on the underlying source code. However, the comparisons are based on the entire sequence of versions of a file instead of being based on a single (often final) version. Furthermore, many open source projects have not been systematically investigated by any current copy detection technique. We propose the validation process using methods as a systematic way to quantify the file reuse relations in open source projects and other situations where there is no golden reference to validate a method.

Our ultimate objective is to create a more promising solution to detect the code copy patterns in the large-scale data such as the set of all open source projects. To achieve that, we start by applying copy detection on FreeBSD project (of nontrivial size, yet manageable) and compare clone detection methods against each other to validate the detection. After the FreeBSD experiment, we plan to apply a similar procedure on dataset including all open source projects.

MSR'08, May 10–11, 2008, Leipzig, Germany.
Copyright 2008 ACM 978-1-60558-024-1/08/05...$5.00.

The rest of paper is arranged as follows. Section 2 starts with the related work. Section 3 reviews the Filename Comparison and describes other four file copy detection methods used in this study. Section 4 presents experiment results. We conclude our study and propose future work in Section 5.

2. Related Work

Many kinds of code copy detection approaches have been proposed in the past. For instance, Ducasse [9] suggested a pattern matching technique to compare strings divided from programs to find out copies. Kamiya [10] developed a tool - CCFinder to identify duplicated codes according to the matched syntax trees from the lexical analysis of source code (our AST based approach is similar the CCFinder). However, until now, existing methods have not been used to quantify reuse in large open source repositories because most of these studies emphasize developing better algorithms or tools to detect clones. They also tend to be applied on relatively smaller datasets in comparison to the collection of all the open source codes.

Bellon et al. [14] proposed an evaluation technique that defined three types of clones to compare 6 existing clone detection tools. Their experiment was based on eight large C and JAVA programs with a total of almost 850 KLOC. The results indicated that one tool cannot analyze programs across language (only for C) and some tool's performance in worse cases was disappointing even on a single relative large program of their dataset. However, their study emphasized the more significant evaluation methods and the quality of current existing clone tools rather than applying them to investigate all open source codes.

Because the version history of a software system represents the evolution process of software itself, Godfrey and Zou [15] suggested an origin analysis to detect function or file merges and splits that have happened between versions. Although their approach can detect the relations between versions, it may be limited within a single system and also requires human efforts on deciding the real match of two entities (function or file). It might not be suitable to be applied across different software projects (systems) such as all OSS projects.

German [12] investigated binary reuse by examining dependencies in the Debian distribution. Here we are only looking at the instances where the source code was copied, not reused without change in binary form.

A service offered by Google allows users to search source code. It is not clear how large the dataset or the methods employed by this service.

Stefan et al. [4] believe knowledge reuse has been particularly salient in code reuse but there are few systematic investigations of code reuse in open source software projects. Their approach invited open source developers to join the survey in order to quantify code reuse among some open sources projects. Although they observed the knowledge reuse behaviors, their method lacks copy detection methods that can automatically identify the code reuse in open source projects. The data they collected may not be as large as the FreeBSD repository and much smaller than all OSS code.

3. Method

Our focus is to measure reuse at the granularity of individual file in open source repositories - not at a finer granularity of a method or a function. We refer to this as a large-scale reuse as opposed to small-scale reuse that investigates reuse at the function, method, or even a code block level. Our interest in large scale reuse is motivated by our overreaching objective to reconstruct a complete history of each source code file fen if it spans multiple version control systems or other types of repositories. To simplify further discussion we define two terms - Files and Reused Files. The term “File” represents the whole pathname of a file in a repository. If at least one nonempty (>60 characters) version of the first File is identical (is represented by the same string) to at least one version of the second File, the files are called Reused Files. We refer to this way of identifying reuse as Identical Content method (IC).

We first apply FC method and then compare it to IC method on the entire FreeBSD repository. We then further validate the copy instances that are detected only by the FC method but are not identified by IC method. To accomplish that, we apply Nilsimsa, Abstract Syntax Tree, and Vector-Space methods on this subset of files and manually inspect a small sample of mismatches. Our fundamental assumption underlying the validation process is that different methods are likely to detect somewhat different instances of copying and, therefore, instances obtained by at least one of the five methods would provide an approximation to the full extent of reuse. We start by describing each method, present a comparison among them, and discuss the results of the validation.

3.1 Algorithms

3.1.1 Filename Comparison

In Filename Comparison method, Reused Files are detected by finding directories that have a large fraction of identical filenames. It contains two steps: (1) finding directory pairs with a large fraction of identical filenames; (2) considering files with the same names in an identical directory pair to be Reused Files. More details are presented in [5].

3.1.2 Identical Content

Identical Content method considers entire content in the version of a source code file as a string. File A and file B are determined to be Reused Files if there is at least one nonempty version of file A matching at least one version in file B (the two strings representing these versions are identical).

This method presents a way to organize our sample data. We extract the content of all versions of all files in our target open source site and then place them into an associative array indexed by the content (See Table 1). The array is implemented using Berkeley db using hash functions.

The IC method has the largest storage requirements, and, as other content-based methods requires retrieval of all versions of the code, but is computationally the fastest among content-based methods used in validation.

Table 1. A schematic table of the structure of sample data

Content 1	filename1/version3;filename1/version5; filename2/version4;…; filename20/version4
Content 2	filename1/version1;filename4/version9
...	…
Content M	filename 8/version3;filenameN-1/version2; filenameN/version4;…filenameN/version5

3.1.3 Nilsimsa

This algorithm accumulates trigrams from the file content and then hashes the summation into a 64 digit hex code. The different bits between two Nilsimsa codes are on a scale of -128 to +128. For example, if we get 92 after comparing two Nilisimsa codes, we know 36 bits are different and 220 bits are the same. In this method, we setup around 24 bits as our thresholds (around 10 %) and apply it between two file versions; that is, we identify two files are Reused Files if the different bits between any one version of one file and any one version the other file are smaller 24 bits.

3.1.4 Vector-Space

Like Identical Content and Nilsimsa, Vector-Space method is also applied on two file versions to define Reused Files. We extract programming language keywords (ex: include and main) to build term-by-document matrices between two file version contents and then compute the similarity (cosines of two matrices). We setup the similarity value 0.9 as the threshold.

Table 2. Method comparison

Method	Pros/Cons
Filename Comparison (FC)	Does not require retrieval and processing of the code. Simple to apply and fast on large-scale data. Cannot determine which version of a file matches. Misses cases where individual files were copied or renamed
Identical Content (IC)	Simple to apply and fast (once data has been retrieved and stored in the array). Miss cases where copies involved a slightest edit in the content. Is less likely to detect reuse in repositories without version history. Requires a large network bandwidth to retrieve and disk space to store the data (this drawback applies to all content-based methods).
Nilsimsa	Compare files (versions) without removing any text. Programming language independent. Requires some computation to compare 64 digit hex codes. May suffer from many false positives.
Vector-Space	Programming language independent. Requires time to extract language related tokens from files (versions). May suffer from many false positives.
Abstract Syntax Tree	Can detect control flow reuse. Needs to know about programming language syntax.

3.1.5 Abstract Syntax Tree (AST)

We approximate the Abstract Syntax Tree by extracting control flow keywords and block delimiters from two different versions; then each AST becomes a string. By using string similarity comparison method on two strings, if their similarity value is over the threshold 0.8, we think these two versions are duplicated. We can also identify Reused Files by its definition. We use code to extract AST provided by Prof. A. Hassan.

3.1.6 Discussion of Reused File detection methods

Table 2 summarizes the advantages and disadvantages of each method. By understanding the pros and cons, we can understand the possible false-positive cases in each method. In addition, a better Reused File detection can be created by integrating different methods though this is beyond the scope of this paper.

3.2 Filename Comparison Validation

Above four content-based methods can be used to validate the FC method. Because IC method indicates reuse only when two versions share identical content (our definition of reuse), it is used as the first choice in the validation process. Figure 1 shows four possible validation situations.

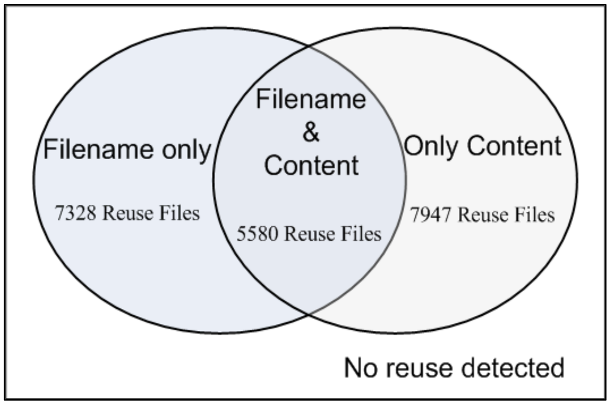

Figure 1. Validation groups

(1) Filename & Content:

Reused Files are found by both FC and IC.

(2) Filename only:

Reused Files are found by FC but not by IC.

(3) Only Content:

Reused Files are not found by FC but found by IC.

(4) No reuse detected:

Neither FC nor IC can detect any Reused Files.

Figure 1 shows total numbers of files in each area. If the source code is changed after a copy, the Identical Content method would be unable to detect those files as reused. Most of these cases may be still identified by the other three content-based methods. Therefore, we apply Nilsimsa, Vector-Space and AST methods on Filename only Reused Files to validate reuse that was not identified by Identical Content method.

We apply the remaining three methods on this subset to further validate FC method and to compare Nilsimsa, Vector-Space, and AST methods. The following steps describe this process:

Step 1: Apply the three methods on FC-only subset.

Step 2: Extract and categorize Files detected as reused by a single method (in addition to the filename method). For example, reuse detected only by the AST method but not by Nilsimsa or Vector-Space method.

Step 3: Randomly sample several files from these sets detected by one method. This way we manually check only a sample where a method is most likely to have produced a false positive. Otherwise we are likely to spend most of manual comparison effort on files that are not false positives and we would need a much bigger sample to see a meaningful number of false positives. The size of the sample should be large enough to make inference about method's error rate on that set.

Step 4: Assign two experts to investigate the reuses and record the results and reasons.

Step 5: Compare two result sets.

To get a more complete understanding of performance of all methods, we plan to apply the last three methods to the entire dataset to estimate the reuse cases missed by IC and FC methods. More extensive manual validation may allow to test our underlying assumption that different methods are likely to detect different instances of copying.

4. Results

The sample data were extracted from the FreeBSD project. The project had a total of 57128 Files and 492583 versions of which 360877 are distinct and nonempty. All the versions of all Files contain 8.16e9 characters. For comparison, the File list takes only 2.6e6 characters – a difference of 3.5 orders of magnitude.

Because the current AST tool we have works only on C or C-like programming language (ex: JAVA), we apply our methods only to C or C-like sample data here. Consequently, 47559 Files were extracted and 12908 Reused Files were found by Filename Comparison method and 13077 Reused Files were found by Identical Content method. Figure 1 shows the distribution of those Reused Files. According to the total number Reused Files detected by both methods, we can say that at about 43 % C-language related Files ((7328+5580+7947) / 47559 = 43%) are reused in FreeBSD.

Upon inspection of Reused Files detected by both methods, we noticed that many clones were detected among different platforms; for example, file "gen/_set_tp.c" was identical in subdirectories "amd64", "sparc64", "powerpc", "ia64", and "i386" of the "/freebsd/src/lib/libc/" directory. Other clones appear to relate to directory restructuring, fore example, "bit_fix.h" is reused in "/freebsd/src/gnu/usr.bin/as/" and in "/freebsd/src/contrib/binutils /gas/".

Table 3 shows the results of validation using the other three content-based methods on the Filename only subset. Both Nilsimsa and AST methods detected around 3000 Reused Files but Vector-Space method detected only 1120 files. Fewer reuses detected imply that Vector-Space method might be influenced by the language relevant keyword frequency. For some small size files, Vector-Space is unable to detect copying. Furthermore, 812 files were not detected as Reused Files by the three methods suggesting that these files are false positives (incorrectly identified as reused) by the Filename comparison method. The false positive rate would then be 4% (818/(20855-818)).

Table 3. Nilsimsa, Vector-Space and AST results in Filename only zone

Method	Number of Reused Files Detected
Nilsimsa	3027
AST	3143
Vector-Space	1120

Table 4 presents expert (represented by the two authors) evaluation of the 60 samples identified as copied by only one of the three methods. It shows that both experts agreed that AST method correctly identified 12 Reused Files in this set. We found that most of those Non-Reused Files were not C-language code, for example, "c.t" file. For the Nilsimsa-only samples, only one sample caused disagreement between two examiners. We also found that Nilsimsa appears to match primarily on the copyright notice. This is not particularly surprising, given the small size of the files. The most controversial method is Vector-Space, where both examiners have disagreement on 12 files suggesting that even manual comparison may need firmer guidelines (the disagreements were largely caused by different interpretations of what constitutes copying). Based on agreed cases Vector-Space had the highest false positive rate and, in conjunction with the fact that it identified the smallest number of copied files, it implies that Vector-Space may not be particularly suited for copy detection. We heard a similar opinion from private communications with other researchers in this area.

Table 4. Summary of random sampling results

Method	Both True	Both False	Disagreement
AST	12	8	0
Nilisimsa	12	7	1
Vector-Space	3	5	12

5. Conclusion and Future Work

Our primary contribution is to propose a large-scale copy detection and validation process for repositories where the information about the copy patterns is not easily obtainable, as, for example, in open source projects. We also extend the concept of copy detection to the comparison files having multiple versions and exemplify the methods and the validation process on FreeBSD CVS version repository.

In particular, we validated previously introduced Filename Comparison method that uses only file paths without the need to retrieve and process file content. We found FC to detect a significant fraction of Reused Files in FreeBSD. Despite its severe limitation of not using file content, it detected around 60% of Reused Files that were identified using content based methods and it has produced a 4% false-positive rate. We also plan to validate the basic assumption of our validation procedure that different methods detect somewhat different instances of reuse and to estimate file copy patterns in a much larger database of all open source projects.

Based on expert investigation on Reused Files detected by a single content-based method, we conclude that Vector-Space method may not be suitable for copy detection in the source code. We may be able modify it in the future to get better results, but we would not recommend using it based on our current experience. We still look for a more promising method to detect Reused Files in large-scale data. Fortunately, expert evaluations combined with validation with multiple automatic methods help us evaluate the performance of various content-based methods and to approach our ultimate objective. Evaluation on FreeBSD showed us that some content-based methods have to overcome the computational challenge to be scaled to much larger scale data.

6. Acknowledgements

We thank Prof. D. German, Prof. A. Hassan, and Dr. D. M. Weiss for their helpful suggestions, and, especially, Prof. A. Hassan for providing us AST approximation code.

7. References

[1] Brenda Baker. On finding duplication and near duplication in large software system, IEEE Working Conference on Reverse Engineering 1995.

[2] B. Lague, D. Proulx, E. Merlo, J. Maryland, J. Hudepohl, Assessing the benefits of incorporating function clone detection in a development process, IEEE International Conference on Software Maintenance 1997.

[3] Akito Monden, Daikai Nakae, Toshihiro Kamiya, Shin-ichi Sato and Ken-ichi Matsumoto. Software quality analysis by code clones in industrial legacy software, Proceedings of the 8th International Symposium on Software Metrics 2002.

[4] Stefan Haefliger, Georg von Krogh and Sebastian Spaeth. Code reuse in open source software. Management Science, Articles in Advance, pp. 1-14.

[5] Hung-Fu Chang and Audris Mockus. Constructing universal version history. ICSE’06 Workshop on Mining Software Repositories, pp. 76–79, Shanghai, China, May 22-23 2006.

[6] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati. An Open Digest-based Technique for Spam Detection. ACM, vol. 41, no. 8, pp. 74-83. The 2004 International Workshop on Security in Parallel and Distributed Systems.

[7] Michael W. Barry and Murray Browne. Understanding search engines: mathematical modeling and text retrieval. SIAM 1999.

[8] Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo SantAnna and Lorraine Bier. Clone detection using abstract syntax trees. In Proceedings of the 8th International Symposium on Software Metrics 1998.

[9] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. International Conference on Software Maintenance 1999.

[10] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Software Engineering, Vol. 28, No.7, 2002.

[11] Audris Mockus. Large-scale code reuse in open source software. International Workshop on Emerging Trends in FLOSS Research and Development, May 20-26 2007.

[12] Daniel M. German. Using Software Distributions to Understand the Relationship among Free and Open Source Software Projects.ICSE’07 Workshop on Mining Software Repositories, pp.24, 2007.

[13] Cory Kapser and Michael W. Godfrey. Improved tool support for the investigation of duplication in software. International Conference on Software Maintenance 2005.

[14] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. IEEE Transactions on Software Engineering, vol. 33, no. 9, pp.577-591, Sep., 2007.

[15] Michael W. Godfrey, Lijie 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, Feb., 2005