ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 8, October 2012

Implementation of Enhanced CloSpan Algorithm for CP-Miner

Amita Kiran. B
M.Tech (CSE),
Department of CSE
CMRCET, Hyderabad, AP.

Soujanya.K.L.S
Associate professor
Department of CSE
CMRCET, Hyderabad, AP.

Raju. G
Assistant Professor
Department of CSE
CMRCET, Hyderabad, AP.

***Abstract*- Copy-pasted code is very common in large software and product line software because programmers prefer reusing code via copy-paste in order to reduce programming effort. Copy pasted code is prone to introducing errors. Unfortunately, it is challenging to efficiently identify copy-pasted code in large software. Existing copy-paste detection tools are either not scalable to large software, or cannot handle small modifications in copy-pasted code. In this paper we propose an enhanced CloSpan algorithm for CP-Miner tool that uses data mining techniques to efficiently identify copy-pasted code in large software including operating systems. Specifically, it takes less than 20 minutes for CP-Miner with enhanced CloSpan Algorithm to identify 190,000 copy pasted segments in Linux and 150,000 in FreeBSD.**

***Index Terms:* software product lines, code reuse, code duplication, data mining.**

I. INTRODUCTION

Pioneered by Software Engineering Institute (SEI), software product line practices have become a viable software design and development paradigm to improve productivity and quality through large-scale reuse. Software product line development, on the other hand, also faces difficult challenges [1] [2]. Although traceability mechanisms between the phases of the software development life cycle (SDLC) are constructed to support the reuse of software artifacts and maintain their consistencies, it is rather difficult and resource intensive to maintain them over time. The traditional engineering practices of maintaining traceability from modification requests to the change of source code is enhanced by source code mining to help recover reuse modules and identify code defects. New modification requests are generated from identified defects and are fed into the maintenance process by integrating with the software configuration management systems and defect tracking systems. To enhance the software maintenance process with integrated source code mining to help recover and improve the traceability of reuse modules. After modification requests are processed and defects are fixed through the traditional maintenance process, reuse modules in related product lines are identified through software clone detection, upon which new modification requests are generated and fed back into the software maintenance process for review and correction. Data mining and software clone detection techniques have been extensively studied and used to identify redundant / reuse code in a program or a set of programs [3].

Different software clone detection techniques have been developed and a number of tools are available for detecting reuse patterns and software plagiarism [4]. We used the CP-Miner tool [5] to perform source code mining for reuse patterns. CP-Miner uses data mining techniques to identify copy-pasted code in large software and detects copy-paste bugs. "Copy-pasted code" here refers to duplicated code present in the software. The duplicated code may also contain some modifications and the extent of modifications can be set as thresholds for the tool to go beyond identical code modules. CP-Miner because its scanning is token-based, flexible to search textual patterns without depending on the complex build environment of the software product lines. It is also efficient in extracting copy-paste patterns from large code base. [6]

We present CP-Miner, a tool that uses data mining techniques to efficiently identify copy-pasted code in large software suites including operating system code and also detects copy-paste related bugs. It requires no modification or annotation to the source code of the software being analyzed.

CP-Miner is token-based. This approach has advantages over the other two. First, a string-based approach does not exploit any lexical information, so it cannot deal with simple modifications such as identifier renaming. Second, using parse trees can introduce false positives because two segments with

identical syntax trees may not be copy-pasted. Therefore, it also has to compare the sub trees by matching tokens. This method is not scalable because the complexity of comparing sequences of trees is $O(N^4)$. Although the scalability issue of parse tree-based method can be tackled by hashing sub trees it cannot perform robustly when the duplicated code contains modifications. CP-Miner also leverages some source code level and syntax information to make the analysis more accurate. This will be described in detail in. Most of them consume too much time or memory to be scalable to large applications or do not tolerate modifications made in copy-pasted code.

II. METHODOLOGY

Copy- pasted code segments are usually similar to the original ones, detection of copy-pasted code involves detecting code segments that are identical or similar Previous techniques for copy-paste detection can be roughly classified into three categories: (1) *string-based*, in which the program is divided into strings (typically lines), and these strings are compared against each other to find sequences of duplicated strings [6]; (2) *parse-tree based*, in which pattern matching is performed on the parse-tree of the code to search for similar sub trees (3) *token-based*, in which the program is divided into a stream of tokens and duplicate token sequences are identified

Identifying Copy-Pasted Code

To detect copy-pasted code, CP-Miner first converts the problem into a frequent subsequence mining problem. It then uses an enhanced algorithm of CloSpan [7] to find basic copy-pasted segments.

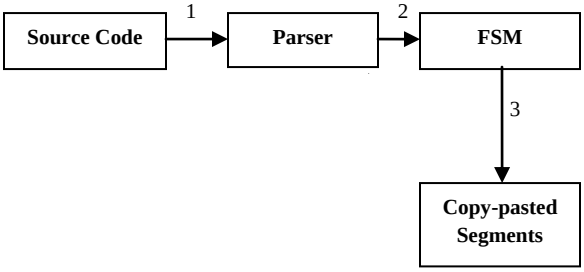

Figure: Block diagram for Identifying Copy-Pasted Segments

FSM-Frequent Subsequence Mining

1. Code
2. Sequence database
3. Code Patterns

i) **Parsing Source Code**

The main purpose of parsing source code is to build a sequence database (a collection of sequences) in order to convert the copy-paste detection problem to a frequent subsequence mining problem. A statement is mapped to a number by first tokenizing its components, such as variables, operators, constants, functions, keywords, etc. To tolerate identifier renaming in copy-pasted segments, identifiers of the same type (such as variable, function, type, etc., not data type) are mapped into the same token. Similarly, all function names are mapped into the same token. All the components of a statement are tokenized, a hash value digest is computed using the "hashpjw" hash function proposed by Weinberger (see [8]), chosen for its low collision rate.

Besides a collection of sequences, the parser also passes to the mining algorithm the source code information of each sequence. Such information includes 1) the nesting level of each basic block, which is later used to guide the composition of larger copy-pasted segments from smaller ones and 2) the file name and line number, which is used to locate the copy-pasted code corresponding to a frequent subsequence identified by the mining algorithm.

ii) Mining for Basic Copy-Pasted Segments

After CP-Miner parses the source code of a given program, it generates a sequence database with each sequence representing a basic block. In the next step, it applies the frequent subsequence mining algorithm, CloSpan, on this database to find frequent subsequences with support value of at least 2, which corresponds to code segments that have appeared in the program at least twice. The main reason is that the original CloSpan algorithm was not designed exactly for our purpose nor was other frequent subsequence mining algorithms. Most existing algorithms, including CloSpan, have the following two limitations that we had to overcome in CloSpan to make it applicable for copy-paste detection:

1. Adding gap constraints in frequent subsequences.
2. Matching frequent subsequences to copy-pasted segments.

III. ALGORITHMS

Enhanced CloSpan (Closed Sequential pattern mining) to mine these patterns. It first mines a closed sequence candidate set which contains all frequent closed sequences. The candidate set may contain

some non-closed sequences. Thus, CloSpan needs a post-pruning step to filter out non-closed sequences. In order to efficiently mine the candidate set, we introduce a search space pruning condition: Whenever we find two exactly same prefix-based project databases, we can stop growing one prefix.
CloSpan can be outlined as two steps
1. It generates the LS set, a superset of closed frequent sequences, and stores it in a prefix sequence lattice and
2. It does post-pruning to eliminate non-closed sequences
CloSpan can be applied to both small and large databases.

Algorithm 1 Closed Mining (D, min_sup, L)

Input: A database Ds, and min_sup.
Output: The complete closed sequence set L

1: remove infrequent items and empty sequences, and sort each item set of a sequence in Ds;
2: S^1 ← all frequent 1-item sequence;
3: S ← S^1;
4: for each sequence s∈ S^1 do
5: CloSpan(s, Ds, min_sup, L);
6: eliminate non-closed sequences from L;

The above algorithm Closed Mining, illustrates the framework which includes the necessary preprocessing step. It first sorts every item set and removes infrequent items and empty sequences.
Then it calls CloSpan recursively by doing depth-first search on the prefix search tree and building the corresponding prefix sequence lattice. Finally it eliminates non-closed sequences.

Algorithm 2 CloSpan (s, Ds, min_sup, L)

Input: A sequence s, a projected DB Ds, and min_sup.
Output: The prefix search lattice L.

1: Check whether a discovered sequence s□ exists s.t. either s⊆ s□ or s□⊆s, and I(Ds)=I(D s□);
2: if such super-pattern or sub-pattern exists then
3: modify the link in L, return;
4: else insert s into L;
5: Scan Ds once, find every frequent item α such that
a) s can be extended to (s<>i α),or
b) s can be extended to ((s<>s α);
6: if no valid α available then
7: return;
8: for each valid α do
9: Call CloSpan(s<>i α, Ds<> i α, min_sup, L);
10: for each valid α do
11: Call CloSpan(s<>s α, Ds<>s α, min_sup, L);
12: return;

By using above algorithms we are able to mine the copy-pasted segments. When CP-Miner generates a frequent subsequence, it maintains a list of IDs of supporting sequences. Based on these IDs in which location these copy-pasted segments are identified.

IV EXPERIMENTAL RESULTS AND CONCLUSION

Our results from Table1 shows that CP-Miner has found a significant number of copy-pasted segments in the evaluated software. The total amount of copy-paste accounts for 17.7-22.3 percent of the source code in these software suites, with Apache being the lowest and Linux being the highest

Software	Segments	CP%
Linux	122,282	15.3%
FreeBSD	101,699	14.9%
Apache	4,155	13.1%
PostgreSQL	12,105	16.5%

Fig. 9 shows that the amount of copy-pasted code increases as the operating system code evolves. For example, Fig. 9a shows the amount of copy-pasted code in Linux from version 1.0 to 2.6.6 over time. As Linux's code size increases from 141,000 to 4.4 million lines, copy-pasted code also keeps increasing from 23,000 to 975,000 lines. In terms of CP%, the percentage of copy-pasted code also steadily increases along software evolution. For example, Fig. 9a shows that CP% in Linux increases from 16.2 percent to 22.3 percent in Linux from version 1.0 to

2.6.6 and Fig. 9c shows that CP% in FreeBSD increases from 17.5 percent to 21.7 percent from version 2.0 to 4.10. However, the CP% remains relatively stable over the several recent versions for both Linux and FreeBSD. For example, the CP% for FreeBSD has been staying at around 21-22 percent since version 4.0.Most of the growth of CP% comes from a few modules, including "drivers" and "arch" in Linux and "sys" in FreeBSD. Fig. 9b shows copy-pasted code in the module "drivers" individually through multiple versions. The percentage of copy-pasted code increases more rapidly in this module than in the entire software suite. For example, in version 1.0, the CP% is only 11.9 percent in this module, but it increases to 20.4 percent in version 2.2.0. This is probably because Linux supports more and more similar device drivers during this period.

With Enhanced CloSpan Algorithm for CP-Miner we are able to find out the Copy-Pasted Segments in large Software and also in product line software for detecting code reuse patterns.

REFERENCES

[1]C. Riva and C. Del Rosso, Experiences with Software Product Family Evolution, *Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSE'03)*, 2003.
[2] G. Deng, G. Lenz, and D. C. Schmidt, Addressing Domain Evolution Challenges in Software Product Lines, *Lecture Notes in Computer Science*, Vol. 3844, Springer Berlin / Heidelberg, 2006.
[3] Michael Jiang and Jing Zhang, Hong Zhao, Yuanyuan Zhou," Maintaining Software Product Lines – an Industrial Practice.2009
[4] T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: a Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code. *IEEE Transaction on Software Engineering*, 28(7):654–670, July 2002.
[5] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: A Tool for Finding Copy-Paste and Related Bugs in Operating System Code," Proc. Symp Operating System Design and Implementation, pp. 289-302, 2004.
[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, CP-Miner: finding copy-paste and related bugs in large-scale software code, *IEEE Transactions on Software Engineering*, 2006.
[7] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets, 2003.
[8]A.V. Aho, R. Sethi, and J. Ullman, Compilers: Principles, Techniques and Tools. Addison-Wesley, 1986.

First Author

Amita Kiran. B received her B.Tech Degree in Information Technology from Kakatiya University. She is pursuing her M.Tech in Computer science at CMRCET from JNTU at Hyderabad. Her research interests include software engineering and data mining .

Second Author

Soujanya K.L.S. received her M.Tech and pursuing her Phd in Computer science. She is presently working as associate professor in CMRCET Hyderabad. Her research interests include software engineering and data mining .

Third Author

Raju . G received her M.Tech in Computer science. He is presently working as assistant professor in Computer science department at CMRCET Hyderabad. His research interests include software engineering, cloud computing ,data mining.