ABHIYANTRIKI
An International Journal of Engineering & Technology
(A Peer Reviewed & Indexed Journal)
Vol. 3, No. 5 (May, 2016) http://www.aijet.in/ eISSN: 2394-627X

Source Code Plagiarism Detector for Java Code

Krishna Kumar Singh*
UG Student
University of Pune, Pune, MS, India

Pallavi Dhormal
UG Student
University of Pune, Pune, MS, India

Payal Yadav
UG Student
University of Pune, Pune, MS, India

Arti Waghmare
Assistant Professor
University of Pune, Pune, MS, India

Abstract
Plagiarism defined as the act of taking or attempting to take or to use (whole or parts) of another person's works, without referencing or citation him as the owner of this work. A particular case of plagiarism is software plagiarism. In this paper different ways of doing source code plagiarism are discussed. A tool for detecting source code plagiarism has been introduced.

***Keywords**: Plagiarism, Repackaging, Obfuscation, Detection, Impact, Intentional plagiarism.*

****Author for correspondence*** singhkrishna1994@gmail.com

1. Introduction

Plagiarism is an act of illegally taking other's work or idea and manipulating it and showing it to be their own. More specific way to describe it would be – literature theft, piracy, or cyber-cheating. Usually it is intentional but even an improper citation of real work in a copied work could fall under plagiarism [1]. Source code plagiarism in programming languages is different from other kinds of plagiarism. It is easy to do, but difficult to detect. Possibility of similar programs is very large when people work on same problems. It is also common that people make few changes to the original source code without manipulating the program's output. It isn't necessary that plagiarism only occur due to copying the original source code but if coder includes comments, source code input data and user interface screen that can also be included as a parameter for plagiarizing. It can also include adding up of several blocks of code copied from different source codes, or a combination of genuine work and plagiarist works. Such ways of manipulating code makes it difficult to track plagiarism. Source code modification can be grouped into either lexical re-organisation or structural re-organisation [2]. Lexical change is simple change that can be done using a text editor without having any coding knowledge, such as addition/removal of comments and changing any identifiers. Lexical modification can easily be traced or caught whereas; plagiarist requires good level of coding knowledge to do structural manipulation. Such kind of changes makes tracing of copied code even much more complicated. Examples of structural changes are modifying iterations, modifying conditional statements, rearranging the order of statements, using different procedure to invoke a function and vice versa, calling procedure call from the body of the procedure itself and vice versa, including statements that will not change the output of the source code, changing operators to its equivalent form, etc. [3].

Plagiarists can make many transformations place can vary from the easy (like modifying comments from the code, or replacing the names of variables) to the more professional (changing control structures with similar e.g. replacing a “for” loop with a “while” statement) [4]. Potential methods or disguising programs include:

- Changing comments
- Changing data types
- Using different identifiers
- Combining redundant variables or statements
- Changing the structure of selection statements
- Mixing copied and original program statements

2. Pre-Knowledge

There are various approaches for detecting plagiarism in source code. Each of these different methods works on some parameter of source code plagiarism. For example, there are methods which are designed mainly to compare programs written in several different programming languages. There are also approaches which are designed to handle professional source code plagiarism but require too much detection time as compared to other approaches. Structure-based method is one of the approaches that are considered suitable. It uses tokenization and string matching algorithm to detect similarity. Some of already available source code plagiarisms detection tools that work on structure-based technique are Plague, MOSS and JPlag.

MOSS: MOSS is available free to use in academics. Moss provides support for Ada programs, Java, C, C++, plain text and Pascal along with this MOSS also supports UNIX and windows OS. First of all, MOSS convert source code into tokens. It then uses an algorithm called robust winnowing algorithm. Robust Winnowing Algorithm was developed by Schleimer et al. but the internal functioning of this method works is not disclosed. This algorithm selects a set of token hashes. In the comparing process of set of files, an inverted index is developed to map the fingerprints of document to documents and their positions within each and every document. Next, each program file is used as a query against this index, which returns a list of documents in the collection containing the fingerprints in common with the query. The number of matching fingerprints of pairs of document is the result of MOSS. MOSS sorts these results and show highest-score matches to user [5].

Plague: Plague is one of the structure-oriented systems. Plague support programs which are only developed using C. Plague works in many different steps. In the first step, source code is changed into structure domains. After this, Heckel algorithm is used to compare newly generated structure domains of first step. This algorithm is designed basically for normal text files and it is developed by Paul Heckel. Plague returns results in list and then use a convertor to generate this list to give results in a way, so that amateur user can easily understand it [6].

JPlag: JPlag is available publically as free accessible service. JPlag can be used to check plagiarism of source code written in C, C++, Scheme and Java. User gave different programs as input in JPlag. First, source code in the program is parsed and then it is transformed into token strings. After conversion JPlag matches these sequences of strings by using Running Karp-Rabin Greedy String Tiling algorithm. Then the comparison result is generated in HTML file, which can be visited by using any browser. The HTML files of results page keeps the pairs of source code that are doubted to be plagiarized. User can see results of different source code pairs separately. Different styles of fonts in generated file of HTML shows different results, like the source code pairs with similar code will have different style of font from other pairs. In this way, user can distinguish results very easily [7]. JPlag has been used extensively by various academic institutions, both at the undergraduate and the graduate level.

3. Implementation

A. ***Normalization***: Normalization is the process of rewriting all Java files in a certain way that will simplify the comparison later on. Normalization can be applied on two formats i.e. Source file or token stream. It includes:
   - Removing comments
   - Uniform renaming of identifiers
   - Sorting of all class members according to their size

B. ***Attribute selection***: First phase is to separate out the attributes to be used in further phases for similarity detection [9]. For example variables, lines, and data types.

C. ***Token generation***: Each different attribute selected in the first phase is given a unique token. After this the attributes with similar tokens are grouped together [10].

D. ***Token count***: In this phase the number of tokens generated for each attribute is counted. The count will be used to detect the amount of similarity between two programs. This is done by comparing the number of similar tokens between the programs. A threshold is set based on which it is decided whether two programs are considered similar or not.

4. Conclusion

In this project we are developing a source code plagiarism detection tool which is based on attribute counting. The advantage of the framework are that it is fast and can work on large volume data since it creates a token list for each source code eliminating the need to process the whole code every time. This framework will be obfuscation proof. The future scope of this application will be making tool compatible with more different types of programming languages like C, C++, Python etc. In future this tool could focus upon blocks of code to know if scanned block of code is plagiarized or not.

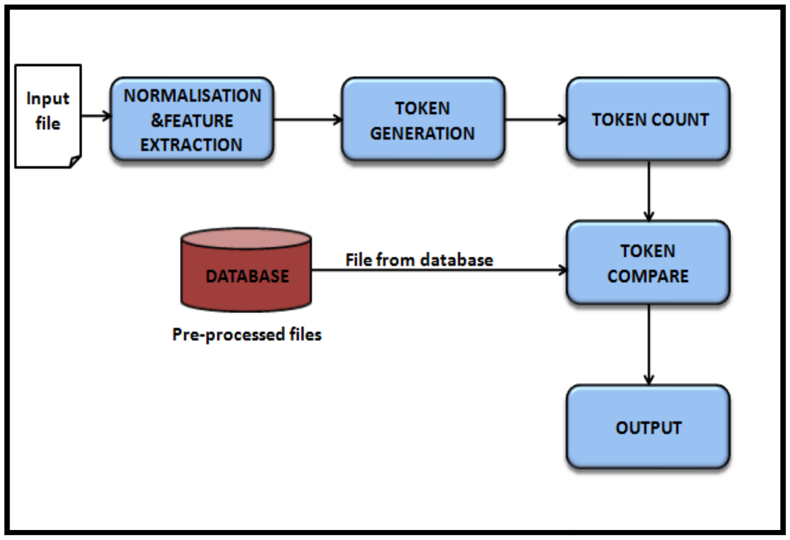

Fig. 1: System Architecture

References

[1] Tripath Tiwari, & P Nithyanandam. (2015). Avoiding plagiarism in research through free online plagiarism tools. 4th IEEE International symposium on emerging trends and technologies in information services held at Canada.

[2] Fangfang Zhang, & Dinghao Wu. (2014). Program logic based software plagiarism detection. 25th IEEE International Symposium on Software Reliability Engineering.

[3] K Žáková, J Pištej, & P Bisták. (2013). Online tool for student's source code plagiarism detection. 11th IEEE International Conference on Emerging eLearning Technologies and Applications, USA.

[4] Arabyarmohamady, H Moradi, & M Asadpour. (2012). A Coding Style-based Plagiarism Detection. IEEE International Conference on Interactive Mobile and Computer Aided Learning.

[5] D Chuda, P Navrat, B Kovacova, & P Humay. (2012). Issue of (Software) Plagiarism: A Student View. *IEEE Transactions on Education*. Vol. 55, No. 1.

[6] S Mann, & Z Frew. (2006). Similarity and originality in code: plagiarism and normal variation in student assignments. Proceedings of the 8th Australian conference on computing education. Vol. 52.

[7] L Prechelt, G Malpohl, & M Philippsen. (2006). Finding plagiarism among set of programs with JPlag. J. Univ. Computer.

[8] C Daly, & JM Horgan. (2009). Automatic Plagiarism Detection. Proceedings of the IASTED International Conference on Applied Informatics. pp. 255-259.

[9] J Brassil, S Low, & N Maxem. (1994). Marking and Identification T Copying. Proceedings of 3th IEEE. Vol. 3.

[10] HT Jankowitz. (1988). Detecting plagiarism in student Pascal programs. *Computer Journal*. Vol. 31, No. 1, pp. 1-8.