STYLE ANALYSIS FOR SOURCE CODE PLAGIARISM DETECTION

Olfat Mirza, Mike Joy

Abstract:

Plagiarism has become an increasing problem in higher education in recent years. A number of research papers have discussed the problem of plagiarism in terms of text and source code and the techniques to detect it in various contexts. There is a variety of easy ways of copying others' work because the source code can be obtained from online source code banks and textbooks, which makes plagiarism easy for students. Source code plagiarism has a very specific definition, and Parker and Hamblen define plagiarism on software as "A program that has been produced from another program with a small number of routine transformations". The transformations can range from very simple changes to very difficult ones, which can be one of the six levels of program modifications that are given by Faidhi and Robinson. Coding style is a way to detect source code plagiarism because it relates to programmer personality without affecting the logic of a program, and can be used to differentiate between different code authors.

This paper reviews a number of publications which report style comparison to detect source code plagiarism in order to determine research gaps and explore areas where this approach can be improved. A summary of the plagiarism techniques in which style analysis can help identify plagiarism is presented.

Key words: Source Code Plagiarism Detection, Style Analysis, Coding Style

1 Introduction

Plagiarism has become an increasing problem in higher education in recent years and researchers have shown that plagiarism is increasing (Hammond, 2004). One of the reasons is that technological advances have changed our lifestyle and the way we seek information, and we have become more reliant on computers, the Internet and web search engines to find answers and seek more information about almost anything. This in turn has made us more dependent on these facilities. In the context of education, traditional education today is complemented with online resources, web classrooms and easy online access to references, which provide various incentives for plagiarism.

Plagiarism is reusing, copying or paraphrasing somebody else's work without making proper reference to the original author, or by intentionally attempting to make the plagiarized work appear to be original (as in the case of student plagiarism).Hannabuss (2001) defined plagiarism as "the unauthorized use or close imitation of the ideas and language/expression of someone else". There are various forms of (text) plagiarism and Martin (1994) clarifies plagiarism from an ethical point of view and identifies six plagiarism forms: (i) word-by-word copying; (ii) paraphrasing; (iii) plagiarism of a secondary source; (iv) plagiarism of the form of a source; (v) plagiarism of ideas; (vi) authorship plagiarism.

Source code plagiarism has a very significant definition and Parker and Hamblen (1989) defined software plagiarism as "A program that has been produced from another

program with small number of routine transformations". The transformation can take place from very simple changes to very difficult ones, which can be one of the six levels of program modifications that are given by Faidhi and Robinson (1987). The range can be listed as follows, where each level includes the modifications included in the previous levels: level 1 – changes in comments; level 2 – changes in identifiers; level 3 – changes in declarations; level 4 – changes in or additions to adds redundant statement or variables; level 5 – changes in the structure of selection statements; level 6 – changes in decision logic.

There has been significant research on how to identify source code plagiarism. For example, packages have been developed using both syntactic and structural language-dependent plagiarism (Bowyer and Hall, 1999; Prechelt et al., 2000) and other techniques such as latent semantic analysis (Cosma and Joy, 2012). The student perspective on source code plagiarism is also an important factor when prevention strategies are being considered. For example, Joy et al., (2010) studied source code plagiarism from a student perspective by conducting a survey on a sample of computer science students in 18 UK universities.

One way of source code plagiarism detection is to detect the authorship of the code from the way the code is written, the "coding style", which may be derived from coding conventions (sets of guidelines for a particular programming language, perhaps defined for use by a particular institution or company). These conventions usually cover such aspects as file organization, indentation, comments, declarations, use of white space, naming of variables, programming practices, programming principles, programming rules of thumb, and architectural best practices.

According to Kernighan and Plauger (1978) the coding style of a computer programs should not only satisfy the personal programmer style, but also promote readability by humans. Different programming languages have different coding styles, for example a C language style may not be appropriate for the BASIC programing language, but generally the types of rules are common between programming languages.

According to MacDonell et al. (2002), source code style analysis can be used for the purposes of: (i) author identification; (ii) author characterisation, to determine some programmer characteristics for a piece of code; (iii) plagiarism detection, to find similarities between sets of code without referring to the original source; (iv) author discrimination, to determine whether code is written by one programmer or many; and (v) author intent determination, whether characteristics of a fragment of code are deliberate or accidental.

The paper is organized as follows. In section 2 we present a review of work on the use of coding style as a technique to identify source code plagiarism. In section 3 we compare the contributions to the field and suggest possible gaps in our understanding of the effectiveness of the technique, and in section 4 we summarise our findings.

2 Literature Review

Seven main approaches to the use of coding style have been applied

(1) Oman and Cook (1989) used typographic or layout style – that is, the formatting and commenting of source code which does not affect the execution of the program

(Oman and Cook, 1988). Pascal source code was used to test the approach on three algorithms presented in each of six computer science textbooks, and a style checker has been designed based on a protocol mechanism which identified, for example, whether comments are lined, blocked or occur after keyword, and the use of upper case, lower case and underscores in attribute names. On the other hand, style analysis checks, for example, whether inline comment on the same line as source code, blocked comments (two or more occurring together) and lower or upper case characters only (all source code). For each condition they apply a Boolean value which is true to denote the presence of that characteristic in the code, otherwise it is false.

(2) Spafford and Weeber (1993) explained source code features which might identify the author of the code and refer to their work as software forensics. They divided the analysis of the code into two different parts: analysis of the executable code and analysis of source files. The executable code analysis targets: (i) data structures and algorithms; (ii) compiler and system information; (iii) programming skills and system knowledge; (iv) choice of system calls; and (v) error handling. The source files analysis contains: (i) the programming languages used; (ii) the formatting style chosen; (iii) special features, such as special environments required by some compilers; (iv) comment style, which varies from writer to writer, (some coders tend to not write anything); (v) variable naming style, including length and capitalization (etc.); (vi) mistakes in spelling and grammar in variable names and comments. Use of language features, scoping, execution paths, bugs and metrics are also highlighted as features to be considered in source file analysis. However, no evaluation is reported in this paper.

(3) Krsul and Spafford (1997) reviewed the literature on identifying the author of a program, and noted that there are three communities which benefit from authorship identification techniques, namely the legal community, the academic community and industry.

A taxonomy of sixty metrics was created, including metrics, style rules and best practice, derived from several sources. These sources include: 236 style rules identified by Oman and Cook (1991), the complexity metrics listed by Conte, Dunsmore and Shen (1986), the 70 programming rules noted by Kernighan and Plauger (1978), van Tassel's (1978) book chapter on readability and programming style, Ranade and Nash's (1993) style rules for C, and Ledgard's and Tauer's (1987) list of C "programming proverbs" that contribute to programming excellence.

The large number of rules was further distilled into three main categories: (i) layout metrics, such as white space use and placement of brackets; (ii) style metrics, such as comment lengths and average variable lengths; and (iii) structure metrics, such as average function length and usage of common data structures.

The extracted features were used by a software analyser program, which was tested on a total of 88 programs authored by 29 students. Using the discriminant analysis statistical approach a subset of metrics was chosen to classify the programs by author, with a reported accuracy of 73%.

(4) Kilgour et al., (1998) used a fuzzy logic approach to capture more elements of authorship (fuzzy logic is a form of many valued logic). They identified two kinds of metrics. The first is quantitative, which presents the numerical variables, and includes proportion of blank lines, proportion of lines that are or include comments, and

average length of identifiers. The second is qualitative, which measures the fuzzy logic variables, and includes braces on separate lines, the degree of indentation used and meaningfulness of identifiers. The fuzzy values are presented as never/almost never; occasionally; sometimes; most of the time; always/almost always. An experiment was performed using 8 C++ programs for the purpose of illustrating how fuzzy logic metrics could be defined.

(5) MacDonell et al. (1999) focused on the area of developing models to discriminate between authors. Feed-forward neural networks, multiple discriminant analysis and case based reasoning are 3 techniques for authorship discrimination which the authors apply to a collection of 26 metrics which has been captured by the IDENTIFIED tool (Gray et al., 1998; Sallis et al., 1998) from a collection of 351 C++ programs by 7 authors. The experiment results archived 88% classification accuracy in case based reasoning and 81.1% accuracy with the other techniques.

(6) Ding and Samadzadeh (2004), used the Krsul and Spafford (1997) categorisation of coding rule: layout, style and structure to organise candidate metrics. These metrics are used from 3 different resources (Krsul and Spafford, 1997; MacDonell *et al.*, 1999; Gray *et al.*, 1998). The authors tested 255 Java programs from 46 authors using discriminant analysis for classification and achieved 62.7% accuracy

(7) Burrows and Tahaghoghi (2007) described a system that uses an information retrieval approach for source code attribution, based on source code tokenization. The source code is tokenized and instead of software metrics, *n*-grams (contiguous sequences of *n* items) are indexed in a search engine. An experiment using 1640 student programs written in C could identify the true author with 78.78% accuracy.

(8) Ohno and Murao (2008) used simple tokenised coding style rules for Java source code categorised in three token groups: (i) basing point tokens, such as opening and closing braces; (ii) identification tokens, such as single and double spaces; and (iii) other tokens, such as reserved words and identifiers. The authors proposed a new method called Coding Model (CM) which based on the Hidden Markov Model (HMM) that quantifies the features based on student's coding style (Ohno and Murao, 2009). They conducted an experiment using Java code, which confirmed that the coding models can distinguish between source code produced by different students. Also, they proposed a combined method that measures the similarity among programs by SIM (Similarity measurement), which is a structural method that measures the similarity between two computer program by reducing the parse trees of the code to strings, then applying a string matching algorithm to find common token sequences (Gitchell and Tran, 1999). The authors expect the combined method to reduce the number of false positives detected (Ohno and Murao, 2011).

(9) Shevertalov et al. (2009) described a novel method for author attribution based on source code discretization, which is the process of transferring the continuous metrics and equations into discrete counterparts. For example some developers tend to use verbose language to write a comment in the source code, so instead of counting how many characters, words or lines they have used, the lengths of comments are categorised as short, medium and long. An optimum set of discretized metrics is identified with the help of genetic algorithms, which are adaptive heuristic algorithms

informed by the process of natural selection (Shevertalov et al., 2007), and the system has been evaluated with a dataset of 75 000 Java source code files from 20 authors.

(10) Arabyarmohamady et al. (2012) proposed a coding style plagiarism detection framework, which performs the detection in two phases. In the first phase a compact representation is produced of the code, and in the second phase the extracted attributes are input into three different modules to detect the plagiarised code and to determine the authorship. The system was evaluated on 120 student assignments in C/C++. There are three main findings in this paper: first, the system is fast and can work on large datasets since the two phase approach creates a feature file for each document to reduce the time. Second, the framework provides a method to detect the original author and the user of the code. Last, the framework is capable of detecting plagiarised documents which have been copied from internet or implanted by third party.

(11) Bandara and Wijayarathna (2013) presented a new source code author identification system based on an unsupervised feature learning techniques. The system uses nine source code metrics, each of which is then tokenised, and an unsupervised neural network technique called Sparse Auto-encoder (Bengio, 2009) is used to extract features which finally train the Logistic Regression supervised learning algorithm (Bishop, 2007).

They used in their experiment 5 large datasets, with java programming language. The result of their evaluation failed when there are more than one author, but succeeds to identify the single authors.

(12) Caliskan-Islam et al, (2015) investigate a new method to classify author's source code, using machine learning. First they start with parsing the source code then secondly, define some different features to represent syntax and structure program code. Thirdly, a random forest classifier (Breiman, 2001) trained for classification. Google Code Jam is an international programming competition and they used their code for the evaluation which achieved the results of 95.33% using 2250 C++ programs.

3 Results and Discussion

The contributions discussed in this paper represent work which has taken place over a period of 20 years. Although the approaches taken superficially appear similar (the act of measuring coding style is closely related to attribute counting), the approaches taken are distinct. Although (1) and (3) simply employ attribute counting, (1) focuses on the existence of particular attributes whereas (3) counts instances of features. Approaches involving the application of algorithmic techniques taken from elsewhere form a major theme, such as the fuzzy logic approach in (4), discriminant analysis in (5) and (6), *n*-grams in (7), Hidden Markov Model in (8), discretization in (9), neural networks in (11) and random forest in (12). The analysis of executable code is possible in addition to source code, as demonstrated in (2), and in (10) an optimisation approach allows the analysis of large data sets.

However, with the exception of (9), the evaluations reported in these studies are modest. Whilst all the authors who have implemented and tested their systems report some degree of success, there are insufficient data to compare the different approaches with any degree of precision.

Table 1

	Language	Evaluation dataset size	Principle Method	Year
1	Pascal	18	Existence of style attributes	1989
2	C	0	Analysis of both source code and executable code	1992
3	C	88	Categories of style metrics	1995
4	C++	8	Fuzzy Logic	1998
5	C	351	Discriminant analysis and case-based reasoning	1999
6	Java	255	Discriminant analysis	2004
7	C	1640	*N*-grams	2007
8	Java	20	Tokenization and Hidden Markov Model	2008–2011
9	Java	75 000	Discretisation	2009
10	C/C++	120	Two phases of analysis (and optimisation)	2012
11	Java	5 datasets	Neural network and logical regression	2013
12	C++	2250	Random Forest	2015

The literature review of coding style for the purposes of plagiarism detection has revealed a wide variety of algorithms which have been used in conjunction with the raw attribute counting normally used to measure coding style. Most of these approaches have been evaluated with relatively small datasets, sufficient to evidence that they have some degree of effectiveness.

However, with perhaps a single exception, no substantial evaluation of any individual approach has been performed, and no detailed comparative study has been published.

4 Conclusion and Future Work

This paper reviews a number of publications which report style comparison to detect source code plagiarism, and identifies research gaps and explore areas where this approach can be improved.

Style analysis to detect source code plagiarism has been discussed through a literature review where language, evaluation dataset, methods and year of publication has been the main points considered.

These results are not conclusive and further research needs to be done with more evaluation datasets and different techniques. Further research will focus on how big datasets can impact on the final results of using different methods to establish authorship, and how Integrated Development Environment (IDE) code formats and the use of automated code generators can affect authorship detection.

5 References

Arabyarmohamady, S., Moradi, H. and Asadpour, M., 2012. A Coding Style-based Plagiarism Detection. *In International Conference on Interactive Mobile and Computing Aided Learning (IMCL)*. Amman, Jordan. IEEE, pp. 180–186.

Bandara, U., and Wijayarathna, G., 2013. Source code author identification with unsupervised feature learning. *Pattern Recognition Letters,* 34(3), pp. 330–334.

Bengio, Y., 2009. Learning Deep Architectures for AI. *Foundations and trends in Machine Learning,* 2(1), pp. 1–127.

Bishop, C. M., 2007. *Pattern recognition and machine learning*. New York: Springer.

Bowyer, K. W. and Hall, L. O., 1999. Experience using "MOSS" to Detect Cheating on Programming Assignments. In FIE'99 Frontiers in Education.*29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings.* San Juan, Puerto Rico, pp. 18–22.

Breiman, L., 2001. Random Forests. *Machine Learning*, 45(1).

Burrows, S., and Tahaghoghi, S. M. M., 2007. Source code authorship attribution using *n*-grams. *Proceedings of the Twelth Australasian Document Computing Symposium*, Melbourne, Australia, RMIT University, pp. 32–39.

Calishkan-Islam, A., Harang, R., Liu, A., Narayanan, A., Voss, C., Yamaguchi, F., & Greenstadt, R., 2014. *De-anonymizing Programmers via Code Stylometry.* Available from: https://www.cs.drexel.edu/ac993/. [Accessed: 30 January 2015].

Conte, S. D., Dunsmore, H. E. and Shen, V. Y., 1986. *Software Engineering Metrics and Models*. Benjamin-Cummings.

Cosma, G. and Joy, M. S., 2012. An Approach to Source-Code Plagiarism Detection and Investigation using Latent Semantic Analysis.*IEEE Transactions on Computers,* 61(3), pp. 379–394

Ding, H. and Samadzadeh, M. H., 2004. Extraction of Java program fingerprints for software authorship identification.*Journal of Systems and Software,* 72(1), pp. 49–57.

Faidhi, J. and Robinson, S., 1987. An Empirical Approach For Detecting Program Similarity and Plagiarism within a Programming Environment. *Computers and Education*, 11(1), pp. 11–19.

Gitchell, D. and Tran, N., 1999. SIM: a utility for detecting similarity in computer programs. *ACM SIGCSE Bulletin*, 31(1), pp. 266–270.

Gray, A., Sallis, P. and MacDonell, S., 1998. IDENTIFIED (Integrated Dictionary-based Extraction of Non-language-dependent Token Information for Forensic Identification, Examination, and Discrimination): A dictionary-based system for extracting source code metrics for software forensics. *Proceedings of the 1998 International Conference on Software Engineering: Education and Practice*. pp. 252–259.

Hammond M., 2004. Cyber plagiarism: Are FE Students Getting Away with Words. *Plagiarism: Prevention, Practice and Policies Conference*; St. James

Hannabuss, S., 2001. Contested texts: issues of plagiarism. *Library Management*, 22, pp. 311–318.

Joy, M. S., Cosma, G, Yau, J. Y-K. and Sinclair, J. E., 2010. Source Code Plagiarism – A Student Perspective. *IEEE Transactions on Education*, 54(1), pp. 125–132.

Kernighan, B. and Plauger, P., 1978. *The Elements of Programming Style*. Second Edition, New York: McGraw Hill.

Kilgour, R. I., Gray, A. R., Sallis, P. J., Macdonell, S. G., 1998. A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis. *In Fourth International Conference on Natural Processing.* Dunedin, New Zealand: Springer-Verlag, pp. 865–868.

Krsul, I. and Spafford, E. H., 1997. Authorship analysis: Identifying the author of a program. *Computers & Security*, 16(3), pp. 233–257.

Ledgard, H. F. and Tauer, J. C., 1987. *With Excellence: Programming Proverbs*. Hayden Books.

MacDonell, S. G., Gray, A. R., MacLennan, G. and Sallis, P. J., 1999. Software forensics for discriminating between program authors using case-based reasoning, feedforward neural networks and multiple discriminant analysis. *Proceedings of 6th International Conference on Neural Information Processing*, (ICONIP'99), vol. 1. pp. 66–71.

MacDonell, S. G., Buckingham, D., Gray, A. R., & Sallis, P. J., 2002. Software Forensics: Extending Authorship AnalysisTechniques to Computer Programs. . *Journal of Law and Information Science*, 13(2), pp. 34–69.

Martin, B., 1994. Plagiarism: A Misplaced Emphasis. *Journal of Information Ethics*, 3(2), pp. 36–47.

Ohno, A. and Murao, H., 2009. A New Similarity Measure For In-Class Source Code plagiarism Detection. *Innovative Computing, Information and Control*, 5(11), pp. 4237–4247.

Ohno, A. and Murao, H., 2008. A quantification of students' coding style utilizing HMM-based coding models for in-class source code plagiarism detection. *3rd International Conference on Innovative Computing Information and Control,* ICICIC'08, pp. 553.

Ohno, A. and Murao, H., 2011. A two-step in-class source code plagiarism detection method utilizing improved CM algorithm and SIM. *International Journal of Innovative Computing, Information and Control*, 7(8), pp. 4729–4739.

Oman, P. W. and Cook, C. R., 1989. Programming Style Authorship Analysis. *Proceedings of the 17th ACM Computer Science Conference*, pp. 320–326.

Oman, P. W. and Cook, C. R., 1988. A paradigm for Programming Style Research. *ACM SIGPLAN Notices,* 23(12), pp. 69–78.

Oman, P. W. and Cook, C. R., 1991. A Programming Style Taxonomy. *International Journal of Computer Mathematics*, 4, pp. 309–325.

Parker, A. and Hamblen, J. O., 1989. Computer Algorithms for Plagiarism Detection.*IEEE Transactions on Education*, 32(2), pp. 94–99.

Prechelt, L., Malpohl, G. and Philippsen, M., 2000. JPlag: Finding Plagiarisms among a Set of Programs. *Journal of Universal Computer Science*, 8, pp. 1016–1038.

Ranade, J. and Nash, A. 1993. *The Elements of C Programming Style*. McGraw-Hill.

Sallis, P. J., MacDonell, S. G., MacLennan, G., Gray, A. R., and Kilgour, R. I., 1998. Identified: Software Authorship Analysis with Case-Based Reasoning. *Proceedings of the Addendum Session of the Fourth International Conference on Neural Information Processing (ICONIP'97)*, Dunedin, New Zealand, pp. 53–56

Shevertalov, M., Kothari, J., Stehle, E. and Mancoridis, S., 2009. On the Use of Discretized Source Code Metrics for Author Identification. *1st International Symposium on Search Based Software Engineering, SSBSE 2009*, pp. 69–78.

Shevertalov, M., Stehle, E. and Mancoridis, S., 2007. A Genetic Algorithm for Solving the Binning Problem in Networked Applications Detection. *IEEE Congress on Evolutionary Computation*, *CEC* 2007, pp. 713–720.

Spafford, E. H. and Weeber, S. A., 1993. Software forensics: Can we track code to its authors?. *Computers and Security.* 12(6), pp. 585–595

Van Tassel, D., 1978. *Program Style, Design, Efficiency, Debugging and Testing*. Prentice-Hall.

Copyright statement

Authors

Olfat Mirza (O.M.Mirza@warwick.ac.uk) Computer Science Department, University of Warwick, Coventry, CV4 7AL, UK.

Mike Joy (M.S.Joy@warwick.ac.uk) Computer Science Department, University of Warwick, Coventry, CV4 7AL, UK.