Suitability of BlackBox Dataset for Style Analysis in Detection of Source Code Plagiarism

Olfat M. Mirza and Mike Joy
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
{O.M.Mirza,M.S.Joy}@warwick.ac.uk

Georgina Cosma
School of Science and Technology
Nottingham Trent University
Nottingham, NG1 4FQ, UK
Georgina.Cosma@ntu.ac.uk

Abstract—**Plagiarism is one of the most common problem that has been increasing in the field of higher education. Many research papers have highlighted the issue of plagiarism in context to its detection and source that is often obtained from the text books and online sources, there is a variety of easy ways for students to copy others' work. Coding style can be used to detect source code plagiarism because it relates to programmer personality but does not affect the logic of a program, thus offering a way to differentiate between different code authors. The immediate objective of this paper is to identify whether a data set consisting of student programming assignments is rich enough to apply coding style metrics on in order to detect similarities between code sequences, and we use the BlackBox data set as a case study.**

Keywords-component; Source Code Plagairism Detection; Style Analysis; Coding Style

I. INTRODUCTION

In the field of higher education, the issue of plagiarism is becoming a crucial concern, and many researchers have also highlighted this in their study [1]. The main reason for this is the way technology has transformed the lifestyle and the way of gathering information; people now rely more on the computer, internet sources, and use web engines to find a solution to every question and get a detailed overview. This has indeed increased the dependency of people on these elements. In regards to education perspective, traditional system has been collaborated with online resources, web equipped classrooms, and has eased the access to online references that are major incentives that give rise to plagiarism.

Plagiarism is reusing, copying or paraphrasing somebody else's work without making appropriate references 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 [2] 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 [3] clarifies plagiarism from an ethical point of view and identifies six plagiarism forms:

i. Copying word-to-word;
ii. Paraphrasing the original text;
iii. Plagiarism through secondary source;
iv. Plagiarism in the form of any source;
v. Plagiarism of thoughts;
vi. Plagiarism of authorship.

Parker and Hamblen have given a very apt definition for source code plagiarism, which is "A program that has been produced from another program with small number of routine transformations". This modification can be related to simple transformation to very complex ones that may belong to any of the six categories of program modifications, which have been identified by Faidhi and Robinson [5].

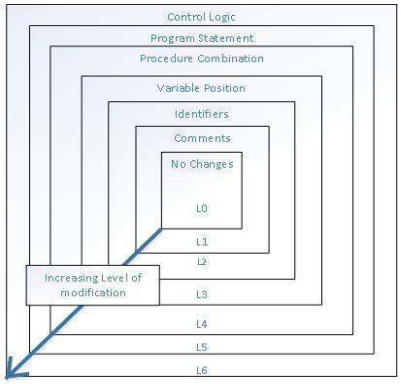

Figure 1. Program Plagiarism Modification (Faidhi and Robinson (1987))

Detection of source code plagiarism has been analysed in various contexts [24], but research on the analysis of coding style for large datasets is limited. We investigate a dataset consisting of genuine student programming assignment submissions to determine if it is sufficiently rich to form a basis for coding style analysis [23], and in order to achieve this, we used the BlackBox source code dataset. In this paper, content analysis approach is used to answer our two research questions (detailed in section III), and the analysis is

978-1-5090-3989-0/17/$31.00 ©2017 IEEE

based on finding out how suitable random samples taken from the dataset are.

The paper is organized as follows. In section II we present the background to the use of coding style as a technique to identify source code plagiarism. We then identify in section III the proposed methods used and the source code file sampling strategy. Section IV details the results of the data analysis. The paper concludes with a summary and suggestions for future research on this topic.

II. BACKGROUND AND RELATED WORK

This section summarises the background of the work related to identifying source code plagiarism, and details several different methods and sampling strategies for using different coding style techniques to identify source code plagiarism.

A. Literary Stylistic

Many researches have been undertaken over identifying source code plagiarism. For instance, packages containing both structural dependent [6], [7] and syntactic plagiarism makes the use of latent semantic analysis technique [8]. The perspective of a student about source code plagiarism should also be considered when prevention techniques are developed and used. In the study conducted by Joy et al. [9], the perspective of students was studied through a survey conducted in 18 universities on computer science students.

One method to identify source code plagiarism in order to proclaim the authorship of the text is by identifying the way in which the code has been written, often referred as "coding style," which can be decoded from the set guidelines of a programming language that defines its usage in context to any organisation or institute. These conventions include a wide range of aspects such as declaration, programming practices, programming rules of thumb, white space, comments, indentation, architectural best practices file organisation, and naming of variables.

As per Kernighan and Plauger [11], the style of coding a particular computer program should fulfil the requirements of personal programmer style and also enhance the readability aspect of humans. Every coding style follows different programming languages that are distinct in their styles, for instance, a program written in the C language may not be apt for the usage in BASIC programing language. Most of the rules of programming languages are commonly followed in all programming languages.

B. Computational

Coding style is a powerful tool that can identify different sources of plagiarism, as it refers to the personality of the programmer and does not influence the logic for which the program runs and thereby, it can be easily put into the use of finding the difference between varied code fragments. managements that are similar in function. A number of research projects have investigated authorship analysis for source code, and four particular methodological approaches have been documented: manual inspection, statistical analysis, machine learning and similarity measurement [10]. Prechelt et al. [7] have classified it into two main groups of automated plagiarism identification for the program code that are, feature comparison and structure comparison [12].

The detection of plagiarism involves a number of techniques that are classified on the basis of the approaches mentioned below.

1) ***String/token-based approach:*** Under this approach, the process of tokenizing the text is followed, and a sequence is created by normalizing a simple string, which involves (for example) omitting the white spaces from the document.

2) ***Structure-based approach:*** This method involves identifying similar words keeping in mind the structure of the document. This task is performed by creating tree like structures for program source code and later comparing them.

3) ***Metrics-based approach****:* The approach has used quantified static features that were extracted as metrics from source codes to calculate similarity.

This section has identified the characteristics of coding style that can be used to detect source code plagiarism without affecting a programming language's logic. The techniques for detecting plagiarism which are based on different approaches are also mentioned.

III. PROPOSED METHOD

In this paper, we perform a content analysis on random samples of source code files taken from a large data set of student coursework submissions in order to identify whether the dataset contains sufficient files on which such a technique is likely to work. The questions that are often asked are the following.

RQ1: What is an appropriate dataset for testing the accuracy of plagiarism detection?

RQ2: Does the testing dataset contain rich enough data to be used?

The next subsection describes and discusses the BlackBox dataset and the sample size used to extract data. Furthermore, it summaries the efficiency of BlackBox as a tool that is used to collect data.

A. BlackBox Dataset

The study sample of the exploratory study was a random sample of Java code files from a repository called BlackBox [20]. BlackBox is a project that collects data from users of the BlueJ online educational software tool. BlueJ is a Java integrated development environment (IDE) designed for beginners [22] and BlueJ has become free and open source software [14]. The primary focus for development of BlueJ was to address the issues related to teaching programming

languages that are oriented towards objects: higher level abstraction and more complex program structure [15], [16], and [17].

There have been previous experiments which have used BlueJ to identify the user's behavior while they write Java code. The studies considered (for example) error types such as missing semicolons, bracket expected, illegal start of expression and unknown class [18]. These data included source code edits, compilation results, and the use of various tools within BlueJ (such as the debugger).

BlackBox has been running for over two years and contains the results of over 100 million compilation events from over one million programs run with BlueJ. BlackBox contains code written by a wide variety of programmers, ranging from complete beginners to professional software developers [19], [20]. Furthermore, the BlackBox dataset is totally anonymous for the purpose of supporting any research experiment.

B. Source Code Dataset and Sample Size

In this exploratory study, one of the focal issues was to determine the intended sample size suitable to be used in this and in future studies. The question that is usually posed is, what number is reflective of the actual population? This question cannot be answered or determined by number only, and common factors include the aim of the study, the population size and the sampling error [13].

The sample size of a dataset like BlackBox is measured as Random Sample Size. According to Cohen et al. [21] a random sample is 250 for any population more than a million with a 90 percent confidence level and ±3 confidence intervals. The population here is formed of Java source code files. For qualitative experiments, which gathered initial ideas and pointers for the research, smaller scale populations were used in order to keep the process manageable. Therefore, four random samples were downloaded from BlackBox, with each of the random samples containing 250 Java files. These sample datasets were downloaded from the BlackBox dataset to justify the study of coding style analysis and to determine the percentage in the sample belonging to each group.

C. Preprocessing the Source Code Files

Source code file preprocessing was applied in this study and although the file name and the real ID of the author are hidden in the code using hashes, each file has its own author. The task of preprocessing the files was performed using the following metrics:

1) *Removing any white space;*
2) *Removing the file header.*

After initial preprocessing, the content analysis of the files identified five sub groups, as explained in section IV-A, to which some significant proportion of the files from BlackBox dataset could be assigned based on their properties.

IV. EXPERIMENT

In order to perform the experiment, we designed a small program based on the Java programming language which initially performs a random sample fetch from the BlackBox dataset. BlackBox contains some duplicated files (identified by having the same ID), and thus the fetcher was designed to choose one file if there are more than one files with the same ID to avoid duplication. After preprocessing the source code files, the second stage is to count the number of lines and the size of each source code file. This is followed by measuring the complexity of each of the files by counting the number of loops by searching in the file and finding common loop words such as: for, if, if-else and while. Documentation in the files was also searched. When the system identified the features, the next stage was to group them according to the pre-defined categories.

A. Grouping the Source Code Files

It is considered important to have a case study of what types of source code files BlackBox contains. The first random group of source code files was downloaded from the BlackBox dataset and thereafter subgroups were created based on: features of the number of lines per source code file; and the complexity of the code in each file in the group. Each file in the first random group was analyzed according to its content. The content of these files relates to the structure of the source code. Then to validate the subgrouping categories, three more random groups were downloaded in order to examine what the majority of file types in the dataset were and whether the initial file subgroupings were valid for the remaining three random groups. Five main subgroups, based on the number of code lines and the complexity of code, were identified for the each random group.

1. The first subgroup contains files which are called "template" or "common ground" files (62 out of 250 files in the first random sample). This means that the files are the same in terms of layout, style and structure, and that this is provided by the BlueJ IDE. Files such as these are usually not going to provide help in identifying similarity or detecting plagiarism, and we propose such files can be ignored by plagiarism detection algorithms.
2. The second subgroup consists of short and simple code files. The length of the code was less than 40 lines and the level of loops was less than 3. The majority of the files were assigned to this sub group. The simplicity, and consequent similarity, of such files may cause them to be difficult to distinguish for the purpose of plgiarism detection.
3. The third subgroup consists of simple code files. The length of the code was on average more than 40 lines and less than 100 lines. The level of loops was more than 3 loops and included some nested loops.

4. The fourth supgroup consisted of code files which were long and complex. The length of the code was more than 100 lines and the level of the loops and the nesting of the loops was more complex than for group 3. Such files contain rich data and coding style based detection algorithms are likely to be successful when applied to them.
5. The fifth supgroup in this study contained files which were incomplete or empty, and can thus be safely excluded by detection algorithms.

B. Experimental Results

The main objective of the statistical analysis is to validate the grouping methods and to find out how rich the dataset can be in order to identify the coding style for the purpose of detecting plagiarism in source code. Since the first and the fifth groups (see section IV-A) contain files which a detection algorithm can safely ignore and they have not been used in the analysis discussed in this section. The names of the second, third and fourth subgroup have been changed to first, second and third. The statistics of each random sample are presented in Tables 1-4. Each random sample comprises source-code files belonging to the three subgroups.

Random Sample 1

		Group1	Group2	Group3
N	Valid	84	50	34
	Missing	0	34	50
Mean		24.64	67.52	195.79
Median		24.00	63.00	158.00
Mode		18	57[a]	143[a]
Std. Deviation		10.361	16.180	123.396
Variance		107.341	261.806	15226.532
Skewness		.227	.336	2.799
Std. Error of Skewness		.263	.337	.403
Minimum		8	45	100
Maximum		44	99	673

a. Multiple modes exist. The smallest value is shown

Table 1

Random Sample 2

		Group1	Group2	Group3
N	Valid	123	60	38
	Missing	0	63	85
Mean		22.37	66.15	227.92
Median		22.00	62.00	164.50
Mode		13	51[a]	260
Std. Deviation		8.968	19.741	147.035
Variance		80.431	389.689	21619.156
Skewness		.332	.065	1.514
Std. Error of Skewness		.218	.309	.383
Minimum		6	22	102
Maximum		44	99	587

a. Multiple modes exist. The smallest value is shown

Table 2

Random Sample 3

		Group1	Group2	Group3
N	Valid	140	41	38
	Missing	0	99	102
Mean		22.89	65.54	249.45
Median		21.00	61.00	167.50
Mode		15	47	180
Std. Deviation		10.296	14.517	260.824
Variance		106.001	210.755	68029.119
Skewness		.525	.531	3.091
Std. Error of Skewness		.205	.369	.383
Minimum		5	46	100
Maximum		45	99	1323

Table 3

Random Sample 4

		Group1	Group2	Group3
N	Valid	124	61	40
	Missing	0	63	84
Mean		22.00	63.90	192.65
Median		19.00	61.00	170.50
Mode		19	55	142
Std. Deviation		9.608	14.830	102.966
Variance		92.309	219.923	10601.977
Skewness		.471	.741	2.670
Std. Error of Skewness		.217	.306	.374
Minimum		7	41	106
Maximum		43	100	649

Table 4

In this analysis of the BlackBox source code there were some clear indications that some portions of the BlackBox source code would be usable for this coding style analysis study. First, the number of valid files in each random sample are similar. For example, the number of valid files in random sample 1 in group two is 50, in random sample 2 it is 60, in random sample 3 it is 41 and in random sample 4 it is 61. Also, the number of files found in group three across the random samples is similar. The maximum and the minimum values corresponding to the number of source-code lines in each group for each random sample is similar. It is clear that group three got the highest number of lines, and it gives a rich style analysis of this group. This suggests that the random samples are representative of the files contained in the BlackBox source code dataset. According to the feature analysis based on physical attributes that were applied to extract coding style, the number of lines was one of the main features considered when categorising the random sample into five subgroups. In addition, the mean, the median and the mode in random samples 1- 4 are similar to each other. The analysis of the results shows that the three subgroups (subgroups 2-4 described in section IV-A) offer a rich source to which coding style analysis can be applied for the purpose of detecting plagiarism.

V. Conclusion and Future Work

This paper explores the suitability of methods based on coding style analysis which unite a content based analysis with random samples. The random samples were taken from the BlackBox source code dataset which contains student coursework, to justify using such a dataset to detect plagiarism. Preprocessing was an important first step to categorising groups of files based on coding style, and the groups have been divided into subgroups according to specific features identified through the content analysis of the random samples. The results suggest that the BlackBox source code dataset of student coursework is suitable for applying coding style based plagiarism detection techniques, since such a dataset contains sufficient files which are rich enough for such an analysis to be meaningful.

Future work will include applying machine learning algorithms to the random samples to study the frequency of the keywords and identifiers which are found in the files, and to determine how these approaches can improve the plagiarism detection approach.

VI. References

[1] M. Hammond, "Cyber-plagiarism: are FE students getting away with words?," in *Plagiarism: Prevention, Practice and Policies Conference*, 2004.

[2] S. Hannabuss, "Contested texts: issues of plagiarism," *Library management,* vol. 22, no. 6/7, pp. 311-318, 2001.

[3] B. Martin, "Plagiarism: a misplaced emphasis," *Journal of Information Ethics,* vol. 3, no. 2, p. 36, 1994.

[4] A. Parker and J. Hamblen, "Computer algorithms for plagiarism detection," *IEEE Transactions on Education,* vol. 32, no. 2, pp. 94-99, 1989.

[5] .J. Faidhi and S. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Computers & Education,* vol. 11, no. 1, pp. 11-19, 1987.

[6] K. Bowyer and L. Hall, "Experience using" MOSS" to detect cheating on programming assignments," in *Frontiers in Education Conference, 1999. FIE'99. 29th Annual*, 1999.

[7] L. Prechelt, G. Malpohl and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *J. UCS,* vol. 8, no. 11, 2002.

[8] M. Joy and G. Cosma, "An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis," *IEEE Transactions on Computers,* vol. 61, no. 3, pp. 379-394, 2012.

[9] M. Joy, G. Cosma, J. Y.-K. Yau and J. Sinclair,, "Source code plagiarism—a student perspective," *IEEE Transactions on Education,* vol. 54, no. 1, pp. 125-132, 2011.

[10] S. Burrows, A. Uitdenbogerd and A. Turpin, "Application of information retrieval techniques for source code authorship attribution," *nternational Conference on Database Systems for Advanced Applications,* pp. 699-713, 2009.

[11] B. Kernighan and P. Plauger, "The elements of programming style," *The elements of programming style, by Kernighan, Brian W.; Plauger, PJ New York: McGraw-Hill,* 1978.

[12] E. Jones, "Metrics based plagarism monitoring," *Journal of Computing Sciences in Colleges,* vol. 16, no. 4, pp. 253-261, 2001.

[13] G. Israel, Determining sample size, University of Florida Cooperative Extension Service, Institute of Food and Agriculture Sciences, EDIS, 1992.

[14] M. Kolling, "Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot," *International Journal of People-Oriented Programming (IJPOP),* vol. 4, no. 1, pp. 5-32, 2015.

[15] . M. Kolling and D. Barnes, "Objects first with Java: A practical introduction using BlueJ," in *Prentice Hall*, 2005.

[16] . M. Kolling, "Using BlueJ to introduce programming," in *Reflections on the Teaching of Programming*, Springer, 2008, pp. 98-115.

[17] R. Barrett and J. Malcolm, "Embedding plagiarism education in the assessment process," *International Journal for Educational Integrity,* vol. 2, no. 1, 2006.

[18] M. Jadud, "A first look at novice compilation behaviour using BlueJ," *Computer Science Education,* vol. 15, no. 1, pp. 15-40, 2005.

[19] A. Altadmri and N. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, New York, 2016.

[20] N. Brown, M. Kolling, D. McCall and I. Utting, "Blackbox: A large scale repository of novice programmers' activity," in *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014.

[21] L. Cohen, L. Manion and K. Morrison, Research methods in education, 2013: Morrison, Keith.

[22] M. Kolling and J. Rosenberg, "Guidelines for teaching object orientation with Java," in *ACM SIGCSE Bulletin*, 2001.

[23] O. Mirza, M. Joy and G. Cosma, Style Analysis for Source Code Plagiarism Detection – an Analysis of a Dataset of Student Coursework, *The 17th IEEE International Conference on Advanced Learning Technologies (ICALT)*. Timisoara, Romania, 3-7 Jul 2017.

[24] O. Mirza, M. Joy, "Style analysis for source code plagiarism detection." *Plagiarism Across Europe and Beyond 2015: Conference Proceedings*. pp. 53-61, 2015.