Programming Similarity Checking System

Ahmad Shukri Mohd Noor, Farizah Yunus, Hoo Jian Liang and Nur F. Mat Zin
School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu,
21030 Kuala Nerus, Terengganu, Malaysia.
ashukri@umt.edu.my

***Abstract*—Nowadays, with the rapid use of Internet, the student becomes easy to copy information with just click over the website. The opportunity to make a copy of someone else's ideas or code without any citation of the original owner is known as plagiarism. Phenomena of plagiarism has become a serious issue among students where students are commit to copy information in class, whether it is plain text or source code. However, the plagiarism can be accidentally, especially for the source code. In a programming class, students study similar material of textbook and attended to the same class. Thus, it is hard to detect and determine the plagiarisms that occur among students. Therefore, plagiarism detection play an important roles in detecting any copy of information including source code. In this paper, the Programming Similarity Checking System has been proposed which is a source code plagiarism detection system in helping Information Technology's (IT) lecturer for identifying plagiarism between student's programming. Students are allowed to upload file online and lecturers are able to check the plagiarism results among students. As a result, plagiarism among student can be minimized by using proposed Programming Similarity Checking System.**

***Index Terms*—Copy Information; Plagiarism Detection; Similarity Checking System; Source Code.**

I. INTRODUCTION

Plagiarism is a global problem that occurs in many different area of our life including at universities. The widespread use of computers and the rapid development of technologies such as Internet have made it easier to plagiarize the work of others. Most cases of plagiarism are found in academia, where documents are typically essays or reports. However, plagiarism can be found in virtually any field, including scientific papers, art designs, and source code. Plagiarism can be classified into several categories such as documents, source code, algorithm and others. But, this article focuses on the problem of determining the similarity of the source codes.

Source code is any human-readable computer language. A source code programming can be written in different programming languages, such as Java, C, C++, PHP and etc. Software and its accompanying source code, is typically falls within one of two licensing paradigms: open source and close source. If the software is open source, the source code is free to use, distribute, modify and study. But, if the software is close source, which mean that source code is kept secret, or is privately owned and restricted.

Plagiarism in source code can be defined as to take or copying the whole or the parts of source code written by other people and this plagiarism is difficult to detect [1]. Involvement of students in source code plagiarism often happened in programming class that contribute with various reasons such as assignment submission, programming phobia, inadequate access to computer programming and time constraint due to time management failure. As a consequence, it has become a common practice among student to reuse the source code because it is difficult and impossible to detect plagiarism manually.

Therefore, this paper proposed developing a Programming Similarity Checking System based on web-based by applying JSP application architecture. So that, students are allowed to upload the file via online system and lecturers are able to check the similarity results of source code among students. It is designed to detect and thus discourage the students to copy exercise programs in programming education. On the other words, this system is an automatic system in helping Information Technology's (IT) lecturer determining similarity between students' source codes.

The rest of this paper is organized as follows: section II discusses the related works of plagiarism detection system, section III and IV discusses the system design and system implementation respectively and section V discuss on conclusion and future works.

II. RELATED WORKS

Detection of plagiarism can be either manual or software-assisted. Manual detection requires substantial effort and excellent memory, and is impractical in cases where too many documents must be compared, or original documents are not available for comparison. Software-assisted detection allows vast collections of documents to be compared to each other, making successful detection much more likely. Meanwhile, source code plagiarism detection only focuses on the plagiarism of source codes. A distinctive aspect of source-code plagiarism is that there are no essay mills. Since most programming assignments expect students to write programs with very specific requirements, it is very difficult to find existing programs that already meet them.

According to Roy and Cordy [2], the algorithms can be classified as based on either:

i. Strings – look for exact textual matches of segments, for instance five-word runs.
ii. Tokens – as with strings, but using a lexer to convert the program into tokens first. This discards whitespace, comments, and identifier names, making the system more robust to simple text replacements.
iii. Parse Trees – build and compare parse trees. For instance, tree comparison can normalize conditional statements, and detect equivalent constructs as similar to each other.
iv. Program Dependency Graphs (PDGs) – captures the actual flow of control in a program, and allows much higher-level equivalences to be located, at a greater expense in complexity and calculation time.
v. Metrics – metrics capture 'scores' of code segments according to certain criteria; for instance, "the number

of loops and conditionals", or "the number of different variables used". Metrics are simple to calculate and can be compared quickly, but can also lead to false positives: two fragments with the same scores on a set of metrics may do entirely different things.

vi. Hybrid approaches – for instance, parse trees + suffix trees can combine the detection capability of parse trees with the speed afforded by suffix trees, a type of string-matching data structure.

Most of the researchers have been proposed various plagiarism approaches for detecting source code written in C, C++ or JAVA language [3]. Each approaches focuses on certain characteristics of code plagiarism. One of the approaches that suitable for detecting plagiarism in programming course is the structure-based method [4], which mostly use tokenization and string matching algorithm to measure similarity [5]. Besides, the type of token formation reduces the dependency on a particular language [6]. Some of existing plagiarism detectors that employ structured-based are MOSS [7], YAP3 [8], JPLAG [9], PDE4Java [10] and MOSS-TAP [11].

Measure of Software Similarity (Moss) is an automatic system for determining the similarity of programs [7]. The algorithm behind moss is a significant improvement over other cheating detection algorithms. But Moss is not a system for completely automatically detecting plagiarism. It is still up to a human to go and look at the parts of the code that Moss highlights and make a decision about whether there is plagiarism or not. Currently, MOSS can analyze code written in eight different programming languages, including C, C++, Java, etc and two platforms which are UNIX and Windows Moss is being provided in the hope that it will benefit the educational community. Moss is fast, easy to use, and free.

The author in [8] proposed a plagiarism detector of Yet Another Plague (YAP) series. YAP is a tool that currently has 3 implementations, where each implementation using a fingerprinting methodology with different algorithms. The implementations have a tokenizing and a similarity checking phase. The last version of YAP is YAP3 that uses the Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) algorithm.

P. Lutz introduced JPlag [9] which is a system that finds similarities among multiple sets of source code files. JPlag works by converting each program into a stream of canonical tokens and then trying to cover one such token string by substrings taken from the other (string tiling). It currently support Java, C#, C, C++, Scheme, and natural language text. But, it does not compare to the internet. It is designed to find similarities among the student solutions, which is usually sufficient for computer programs. JPlag is free but users are required to create an account. Besides, JPlag has a powerful graphical interface for presenting its results.

A. Jadalla proposed the Plagiarism Detection Engine for Java (PDE4Java) [10] which detects code-plagiarism by applying data mining techniques. The engine consists of three main phases; Java tokenization, similarity measurement and clustering. PDE4Java has one extra feature, which is adaptive reporting of the clusters of suspicious plagiarized programs.

The MOSS Tool for Addressing Plagiarism at Scale (MOSS-TAPS) [11] which is introduced by D. Sheahen proposed repackages and organizes submissions for plagiarism detection for courses that repeat a coding design assignment from semester to semester. The basic MOSS script is guaranteed to work in UNIX, but not necessarily on other platforms. MOSS-TAPS provides persistent configuration, supports a mixture of software languages and file organizations, and is implemented in pure Java for cross-platform compatibility.

Even though all the above works proposed plagiarism detectors, but there are still have a drawbacks in order to meet the requirement of users. Thus, this paper proposed the Programming Similarity Checking System based on web-based by applying JSP application architecture to allow students to upload the file via online system and lecturers are able to check the similarity results of source code among students easily.

III. System Design

Programming Similarity System implements prototyping as a software development process to help anticipate changes that may be required in the development process. Prototyping is chosen because prototyping will faster to provide a system interaction to user. Therefore, it will help software developer to quickly refine real requirements needed by the user. In addition, confusing or difficult functions can be identified and errors can be detected much earlier.

Figure 1 shows the process model for prototype development. Prototyping methodology has 4 main stages. The first stage is establishing prototype objectives. In this stage, elicitation and gathering technique has been used for gathering information or requirement. Besides, the project team needs to meet stakeholder for fully understanding about this project. Second stage is defining prototype functionality. During this stage, it involves requirement analysis and system design. System analyst needs to identify functional and non-functional system requirement before started the project. Moreover, system analyst also needs to identify the relationship that exists between the entity types, and transform it from conceptual database design to logical database design and physical database design. Next is to develop a prototype. At this stage, a prototype will be developed by deciding what to put into and what to leave out of the prototype system for reducing prototyping costs and accelerate the delivery schedule. The stage of development will be repeated if the development of a prototype is not successful. The last stage is to evaluate prototype. In this stage system developer needs to test a complete system by using the module and subsystem testing, and perform unit testing. At the end, system developer will deliver a complete system that fulfills user and system requirements.

A. System Architecture

In order to achieve the system robustness, flexibility and resistance to potential change, this system applied three-tier architecture. This architecture consists of three layers, which are the user interface layer, the application logic layer and the database layer. The three-tier architecture aims to solve a problem of repeated design and development. Besides, this architecture also aim to make the application development work more easily and efficiently. The first tier which is interface layer is run on the end-user's computer; the Graphical User Interface (GUI) of browser is using HTML/HTML 5, CSS, JavaScript, Ajax, and JSP. This tier offers the user a friendly and convenient entry to communicate with the system. The second tier is the application logic layer which performs the controlling functionalities and manipulating the underlying logic

connection of information flows. Finally, the data modeling job is conducted by the database layer, which can store, index, manage and model information needed for this application. Database layer is run on the database server; the communication with the database is through Java Database Connectivity (JDBC), whereas Database Management System (DBMS) which stores the data required by the middle tier.

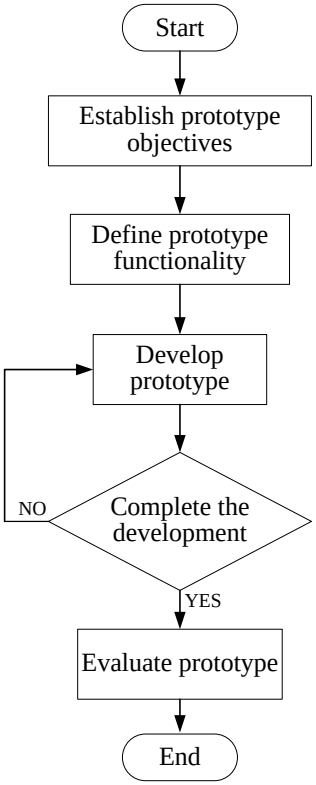


Figure 1: Process of prototype development

B. System Analysis and Design

The purpose of the system analysis and design is to show how the system will be implemented during implementation.

System analysis is the process of gathering and interpreting facts, diagnosing problems, and using the information to recommend improvements to the system. System design is the process of defining the architecture, components, modules, interface and data for system to satisfy specified requirements.

Figure 2 shows the class diagram for Programming Similarity Checking System. Class diagram consists of classes and represent the relationship between class entities.

Figure 3 represents the process flow of similarity calculation which is a main part of the programming similarity checking system. This process consist of six steps which are upload file, lexical analysis, 4-grams representation, comparator, calculator and similarity.

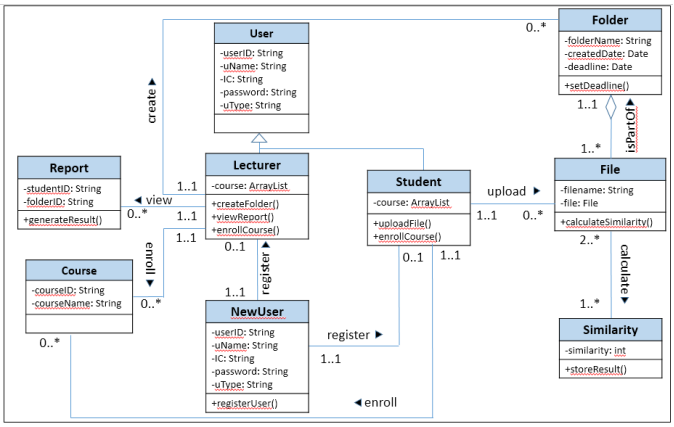


Figure 2: RECs for Programming Similarity Checking System

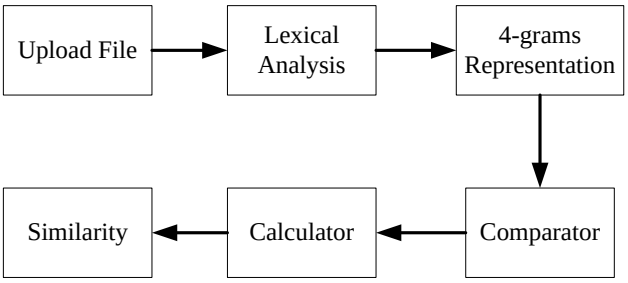


Figure 3: Process flow of similarity calculation

The first step for similarity calculation is upload a file that consists of source codes which is to compare with the existing source codes. The second process is lexical analysis, which is the process of converting a sequence of characters into a sequence of tokens as shown in Table 1 for the example of source code "int a=1". The token is a group of characters having a collective meaning, meanwhile lexeme is a particular instant of token. The next process is 4-gram representation which is a process of break that sequence of token into smaller blocks as shown in Table 2. From Table 2, each new token consists of 4 tokens from the original for 4-gram tokens.

Table 1
Converting Process of Lexical Analysis

Token Type	Lexeme
Data_type	int
Identifier	a
Operator	=
Numeric	1
Separator	;

Table 2
Process of 4-grams Representation

IDs	Token Type	Lexeme	4-gram Representation
2	Data_type	int	2374
3	Identifier	a	3748
7	Operator	=	7483
4	Numeric	1	4838
8	Separator	;	8389

Then, the comparator component will compare the source code to check the similarity as shown in Table 3. From both of the source codes, the comparator detects the similarity of [2374| int, 7483| =, 8237| ;]. After that, the calculator will calculate the percentage of similarity based on Figure 4 by using equation (1).

Table 3
Process of Comparator for Two Source Code

Programming 1 int a = 1; System.out.printIn("Hello");		Programming 2 int b = 1; String a = "Hello";	
Lexeme	4-gram Representation	Lexeme	4-gram Representation
int	2374	int	2374
a	3748	b	3748
=	7483	=	7483
1	4838	1	4833
;	8389	;	8337
System. out. printIn	3898	String	3379
(	8988	a	3798
"hello"	9882	=	7982
)	8823	"hello"	9823
;	8237	;	8237

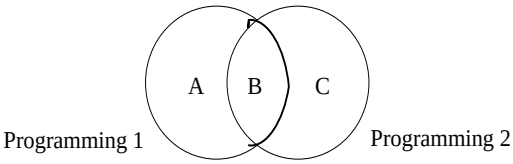


Figure 4: Calculation of similarity source code for two different programming

$$Simlilarity = \frac{Programming\ 1\ \cap Programming\ 2}{Programming\ 1\ \cup Programming\ 2} \times 100\% = \frac{B}{A+B+C} \times 100\% \quad (1)$$

where: A = total number of blocks appeared in programming 1 but not in programming 2
B = total number of blocks appeared in programming 1 and programming 2
C = total number of blocks appeared in programming 2 but not in programming 1

IV. System Implementation

The implementation is the most important phase in the development process. The implementation phase is carried by referring to the design phase to produce an organized user-interface in various aspects.

Figure 5 shows the system menu hierarchy of the Programming Similarity Checking System. The top level items are the most general which is login menu and lower levels are increasingly specific such as report menu.

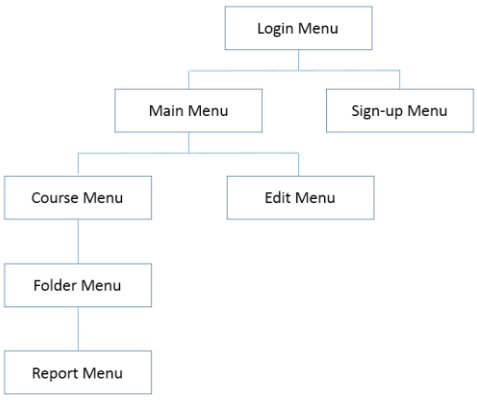


Figure 5: System Menu Hierarchy of Programming Similarity Checking System

In the implementation part of the system, description of module will be explained with the aid of figure in interface design. Firstly, Programming Similarity Checking System allow user go to Sign-up Page by clicking sign-up button on Login Page as shown in Figure 6(a). After that, user can key in his data in Sign-up Page as presented in Figure 6(b) to create his own account. While the user had an account, user can log in to the Programming Similarity Checking System by clicking login button on Login Page. Once the user successfully login, user can manage his account. By clicking on the link of user name allow user go to Edit Page to update his account and link of logout allow user to logout from the system as shown in Figure 6(c).

(a)

(b)

(c)

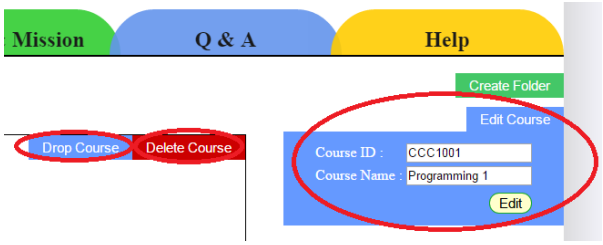
Figure 6: Interface of (a) Login, (b) Sign up and (c) Main page

Programming Similarity Checking System has two types of users which are lecturer and student. For lecturer, lecturer are able to create course and add course in Main Page as shown in Figure 6(c). The different between create course and add course are create course allow lecturer to manage and create new course for users to add. Add course are only be able to add the course which had been created. After click into Course Page as presented in Figure 7(a), lecturer are able to manage the course and create folder. Course management as

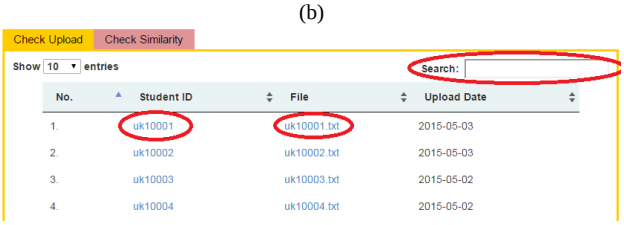
illustrated in Figure 7(b) allow lecturer drop course, delete course and edit course. The difference between drop and delete course is dropped course can be add back, but deleted course will delete all the relate data which can't be add back. Therefore, delete course can only be done by the course's creator. While, Create folder button allow lecturer to create new folder by setting the deadline.

Figure 7(c) shows check upload window, where lecturer can click on selected student's id to view the student record, download selected file and filter search. While, in check similarity window in Figure 7(d), lecturer can set the plagiarism range. For example, plagiarism range set to above 75.00%, then all the similarity results which are above 75% will be generate in red fonts.

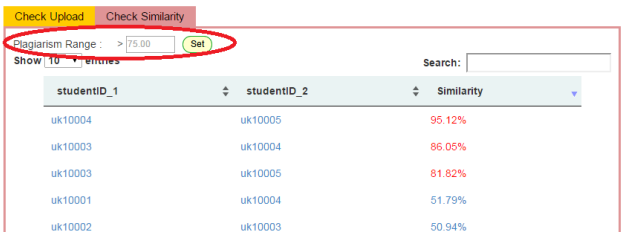
(a)



(b)



(c)

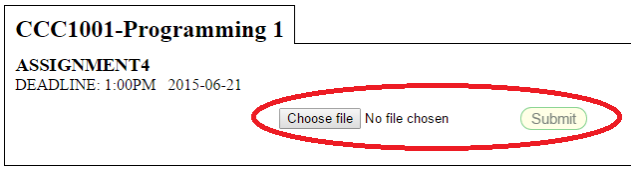


(d)

Figure 7: Interface for lecturer; (a) Course interface (b) Manage course interface (c) Check upload and (d) Check similarity

Meanwhile, for student as a user, student are allow to add course at main page which is created by lecturer as shown in Figure 6(c). Then, after click into Course Page in Figure 8(a), student are able to drop course and click into the folder. Next, go to the Folder Page in Figure 8(b). Student are allow to upload file to the selected folder for similarity checking.

(a)



(b)

Figure 8: Interface for student; (a) Course Page and (b) Upload file

V. Conclusion and Future Works

Plagiarism will give a negative impact on the learning process, especially among students if this issue is not taken as a serious issue. Source code plagiarism detection focuses on the problem of determining similarity among the source codes. Due to the involvement of students in source code plagiarism often happened in programming class, this paper proposed Programming Similarity Checking System to provide a platform for student to upload programming files and enabling lecturers to check the plagiarisms between students. As a result, plagiarisms among students can be minimized and control.

This system has the potential to be expanded and enhance for the future works. For example, file upload can be support more format likes pdf format and system can be supported more programming language such as C.

Acknowledgment

The research was supported by Ministry of Higher Education of Malaysia (MOHE) for the grant of Fundamental Research Grant Scheme (FRGS). (Ref: FRGS/2/2014/ICT07/UMT/02/1).

References

[1] D. Luke, D. P. S, S. L Johnson, S. Sreeprabha and E. B. Varghese, "Software plagiarism detection techniques: A comparative study," *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, no. 4, pp. 5020-5024, 2014.

[2] C. Roy, J. Cordy, "A survey on software clone detection research," *Technical Report No. 2007-541*, School of Computing Queen's University at Kingston Ontario, Canada, pp. 43-56, 2007.

[3] A. Christian and S. M. M. Tahaghoghi, "Plagiarism detection across programming languages," in *ACSC '06 Proceedings of the 29th Australasian Computer Science Conference*, 2006, pp. 277-286.

[4] V. T. Martins, D. Fonte, P. R. Henriques, and D. da Cruz, "Plagiarism detection: A tool survey and comparison." *OASIcs-Open Access Series in Informatics*. vol. 38. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 143-158, 2014.

[5] A. S. Bin-Habtoor and M. A. Zaher, "A survey on plagiarism detection systems," *International Journal of Computer Theory and Engineering*, vol. 4, no. 2, pp. 185-188, Apr. 2012.

[6] S. Chauhan, A. Arora, and Y. Singhal, "Plagiarism detection of C program using assembly language," *International Journal of Computer Applications*, vol. 158, no. 3, pp. 17-22, Jan. 2017.

[7] A. A. Moss, *A System for Detecting Software Plagiarism*. University of Berkeley, CA, 2005.

[8] M. J. Wise. "YAP3: Improved detection of similarities in computer program and other texts," *ACM SIGCSE Bulletin*, vol. 28, no. 1, pp. 130–134, 1996.

[9] P. Lutz, M. Guido, and M. Phlippsen, "JPlag: Finding plagiarisms among a set of programs," *Fakultätfür Informatik Technical Report 2000-1*, Universität Kalrsruhe, Karlsruhe, Germany, 2000.

[10] A. Jadalla, and A. Elnagar, "PDE4Java: Plagiarism detection engine for java source code: a clustering approach," *International Journal of Business Intelligence and Data Mining*, vol. 3, no. 2, pp. 121-135, Jan. 2008.

[11] D. Sheahen, and D. Joyner, "TAPS: A MOSS extension for detecting software plagiarism at scale," in *L@S '16 Proceedings of the Third (2016) ACM Conference on Learning @ Scale*, 2016, pp. 285-288.