A Design and Implementation of the Source Code Plagiarism Detection System

Byung-Ryul Ahn *, Bae-young Choi **, Moon-Hyun Kim **
* anbr0305@skku.edu
** bychoi@imtl.skku.ac.kr
** mhkim@ece.skku.ac.kr

Abstract. As the software industry develops at a rate speed, anyone can copy or plagiarize without difficulty contents that are becoming digitalized. To make it worse, the development of various contents that be illegally copied and plagiarized are resulting in the increasing infringement on and the plagiarism of the intellectual property. This dissertation tries to put forth the method and the theory to effectively detect any plagiarism of the source code of programs realized in various languages. This dissertation analyzes the advantage and disadvantage of the plagiarism test software, and especially, presents a method to detect possible plagiarism by using the Pattern Matching to overcome its disadvantage. And it also intends to introduce more developed automatic detection system by overcoming the problems with the method of Pattern Matching.

Keywords: Multimedia Contents Protection, software piracy, Plagiarism Detection System, Pattern Matching

1 Introduction

With the computer technology and the information getting more and more important, the infringement upon the intellectual property and the plagiarism is increasing. Though the plagiarism and illegal copying are in rage, there has not been much research both in Korea and abroad to come to grips with that problem. It takes a lot of human labor, time and resources to judge and assess the plagiarism. Therefore, a more efficient methodology and objective and systematic approaches are called for. A lot of research and development are expected to be carried out in that kind of field in the future and it will be easier to detect any case of plagiarism and illegal copying and to keep the damage at the minimum while bring resolution to the conflicts over the intellectual property[1-4].

Documents and programs are the targets most vulnerable to the plagiarism. The plagiarism of documents constitutes when the source of professional books, magazines, internet medium or documents of any specific organization is not made clear. Plagiarism is not confined to documents but includes cases when the source code of programs realized in various languages is copied from the material posted on the internet or any organization for use without any change.

Works to detect the plagiarism should be implemented in a quick and systematic way. And the result from it should be objective and accurate. But it takes a tremendous amount of time for human to detect the illegal copying of software that are developing day in and day out, and the there might be a lack of objectivity considering that much subjective view of the person who detects such illegal copying is likely to be reflected. Therefore, many plagiarism-detecting programs have been developed to detect those illegal copying. However, most of current software does not bring out effective result as the plagiarism gets more complicated, precise and diverse.

This dissertation analyzes the program copying software and makes remark on both the advantage and disadvantage of it. Moreover it will introduce the pattern patching method to detect the plagiarism of software, and also put forth a system to detect the plagiarism of software with more precise and varied structures applying the method of current pattern matching, as well as its design.

2 Detection of plagiarism using the pattern language

The purpose of pattern language is to break the source of current program into patterns to match with a systematic and abstract language for the purpose of detecting the plagiarism of programs. Moreover, it has the advantage of overcoming the current plagiarism detecting software by using the pattern.

2.1 Current research using Pattern Language

Table 1 points to the plagiarism detecting system that applies the most representative pattern language.

Table 1. Method that uses Pattern Language

Current research using Pattern Language	Overview
Scuple: A Reengineer's	Searching for the pattern of program language expressed automatically in the source code, and making the system's

Tool for Source Code Search[5]	prototype to a simple, brief and general pattern language concept by using pattern symbol (It goes through the process of using the pattern symbol and re-create as a structured language)
Fast Partial Evaluation of Pattern Matching in Strings[6]	It compares the matched source statically, and presents way to segment the static function efficiently. (It carries out the matching for the character of text to define the pattern for the matching character, forming a language with structure).
Organizing the table similar among the keywords for the purpose of plagiarism test[7-9]	It carries out the test based on the structure It compares the keyword among various programs that are compared. The calculation of similarity applies different weighted value among program keywords.
Automatic analysis of functional program style [10]	It defines the pattern under 7 cases to measure the rate of occurrence. 1) REMOVE if <cond> then true else false 2) REMOVE COMPARSION WITH BOOLEAN 3) INTRODUCE PARTIAL APPLICATION 4) REMOVE UNUSED VARIABLE IN PATTERN 5) CONVERT if <var> TO PATTERN 6) CONVERT if <var>=<const> TO PATTERN 7) CONVERT hd/tl USE TO PATTERN

What characterizes this pattern language is the use of pattern symbol for the conversion into a structured language. In addition to that, there is a method to give the unit of pattern specific meaning according to each level of declaration, type, variable, function, expression, statement. There is also a method to compare and analyze the key word that has the weighted value on the bases of the structure, and a method to measure the rate regarding the occurrence of the 7 patterns.
Particularly, the method used by SCRUPLE : A Reengineers Tool for Source Code Search[5] is as in the below ;

Table 2. SCRUPLE : Pattern Symbol

	Syntactic Entity	Pattern Symbol
1	declaration	$d
2	type	$t
3	variable	$v
4	function	$f
5	expression	#
6	statement	@

Table 3. Example of pattern matching

A match	Pattern

while($v_1<25){
$v_3[$v_1] = $f_5($v_1);
$v_1++;
}
	do{
	squares[x] =
	squareof(x);
	x++;
	}while(x<25)

If the pattern symbol is divided into a big bundle as is shown in the table 2, it is easy for changing the name of simple variable, but ineffective for the change in the location of the variable, data type and substitution. And it is difficult to reflect the structure of program by giving the keyword the weighted value by extracting the keyword sequence. We suggest the solution by using Similarity directory comparison stage, Static Analysis stage and Speculative Analysis in order to overcome such problems.

3 Design of system to detect the software plagiarism

Fig 1 shows the function of the system in each stage to detect the software plagiarism that this dissertation presents.

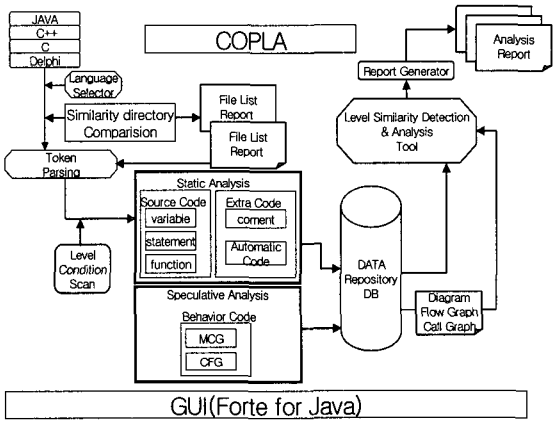

Fig. 1. Design of system to detect the software plagiarism

It goes through the processing of parsing by the Token through the language source (program) and the parsed token information is used to detect the variable copying which is the static analysis, and the function copying which is the speculative analysis.

3.1 The stage of Similarity directory comparison

Identifying the similarity between the original copy and illegal copy is very difficult to maintain the objectivity because even the assessment of illegal copy is based on the professional knowledge and personal experience in detecting the plagiarism of current source code. Especially, the similarity between the original copy and illegal copy is very helpful for the assessment, which goes through the following 5 stages for the comparison of similar directory.

Table 4. Comparison of similar directory by the stage

stage	Comparison
1 stage	Directory name, Sub-Directory name comparison
2 stage	File name, File count, File size comparison
3 stage	Header information, Super class name comparison
4 stage	Method name, Method count comparison
5 stage	Method structure comparison

The stage 1 is often used when the original copy and the illegal copy has exactly identical directory name. The stage 2 is used when the original copy and the illegal copy has only different directory name. The stage 3 is used when the package of original copy was reduced or added as well as the directory name of illegal copy. The stage 4 is used when unnecessary header (import) was added and the class name was altered to copy the original one. Finally, the stage 5 is used when the method name and method count was altered as well as for the stage 4.

Dividing the stage 5 as in the below, an index becomes available to detect the possible illegal copying by comparing the directories in each stage. And then the irrelevant directories are removed and only similar directories are extracted, considerable reducing the overhead that is compared for the assessment.

3.2 Static Analysis stage

Static analysis stage is a stage that converts the program source code to a pattern language to detect the plagiarism. This stage applies scruple and various current methods mentioned in the previous chapter 3, and defines new pattern symbol that tries to make up for the problems.

Variable Pattern Base Symbol

Table 5. Pattern Label 1 step: making LINE BY LINE Pattern & changing the name of simple variable

Entity	Pattern Symbol	Means
declaration	$*d="{name}" Indicating when class declaration is made	indicating the class name declaration the name of class can be indicated ex)$*d{class name}
declaration	$*d_="{name}"#import	indicates only the class name that is imported ex)$*d_"{IO}"#import=> name that is imported
Type	$t_{variable type}	data type of variable $t_d=>double type $t_i=>int type $t_by=>byte type $t_f=>float type $t_l=>long type $t_sh=>short type
		$t_b	Boolean type
		$t_S	String type
		$t_C	Char type
Variable	t_typev_ sequence of declaration	t_iv_1=a:-> First declaring variable of integer type variable
Variable expression	$v_{Variable}#def	indication about the declaration of variable
	$v_{Variable}#used	indication of declaration for the use of variable
	$v_{Variable}#redef	indication of re-declaration of variable
function	$f_{sequence}	$f_1=> Follows the sequence of declaring the function
	Indicating the sequence of declaring the function on the program	
	$f_{name}	$f_1="*{name}*"
	$f_1="*{main}*"-> can be seen as the main of function's name	
{ }	{.....}#name	Make entry of function's name at the start and end of function
[ ]	[" "]	defines the form of output or expression ex)System.out.println("aaa"); ;
{* *}	{*for|while|do while*}	Indicating the start and end of controlling text

Table 6. Pattern Label 2 step: variable Type alteration

Variable	Type	Means
Figure type variable	t_figv_1#[define|used][re-]	Figure type variable ->$t_fig
Character type variable	t_chrv_1#[define|used][re-]	Character type variable ->$t_chr

Table 7. Pattern Label 3 step: changing the location of variable

Variable	Type	Means
Location of variable	$v_1_1	Expressing sub for the variable with same function

Table 8. Sub set: Variable that is used for the function like Pattern Label 3 step

Variable	Sub Set for the variable	Means
Variable group	$v_1_1(sequence for the variable)	Explains the sub variable by referring to the variable with the same function

Table 9. Variable Table (Sub set)

Variable	expression	Means
definition	#def#re-	Declaration of variable(re-declaration)
	#used#re-	Use of variable(re-use)
TYPE	$t_i	integer type
	$t_d	double type
	$t_by	byte type
	$t_f	float type
	$t_l	Long type
	$t_sh	Short type
	$t_b	byte type

	$t_S	String type
	$t_C	Char type
Sequence of declaration	$v_{Sequence of declaration }	Sequence of declaration

As in the above tables, a basis for the material to analyze the type of illegal copying can be obtained to change the location of the variable, change the data type and substitute the variable, In addition to changing the name of variable through the diversity of pattern symbol by the entity.
The systemic symbol by the pattern allows a consistent expression of more concrete language, namely, the keyword and statement, which are the basic elements of language, and the overall generalized picture of variables, enabling the judgment on whether it is illegally copied.

3.3 Speculative Analysis stage

The method call should be analyzed according to the order that the method is called by generating the event after the program is executed because some of it is dynamically called through the event occurred by the user. However, it should be analyzed through inferential method because of the time limit and the required personnel. Under the inferential method call, another method call is sought within the called method by searching for the method that is called within the main of program.

The following is the basic algorithm:

- ① Remove other code except the code related to the method call,
- ② Search for the main method,
- ③ Search for the method called within the main method,
- ④ Search for another method within the called method,
- ⑤ Repeat the process of ③~④ in the above,
- ⑥ Repeat the process of ② if there is no more method call.

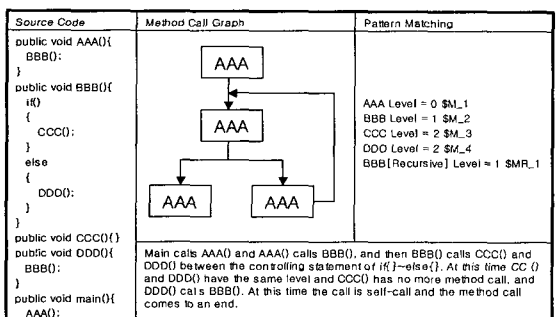

Fig. 2. Function (Method) Call Graph

Fig. 2 shows a simple example of figuring out the sequence of the method call.

It could be found more easily and more accurately whether it is an illegal copy, by matching the call sequence of method and function.

Table 10. Method (Function) Pattern Symbol

Entity	Pattern Symbol	Means
Method	$M_i	method called at ith
	$MR_i	ith method that was self-called
	$E_i	ith method called by the external file or class
	$ER_i	ith method self-called by external file or class
Function	$F_L	Loop function such as for, while, etc
	$F_C	Control function such as if, switch, etc

In the event of referring to the external class, the sequence of method call can be figured out in the same way as the algorithm in the above by testing the input file folder and import text. Once the sequence of method call is figured out, the function call inside the method has to be figured out. The function call means the controlling text or branching statement such as if, for, while, switch in the method. It is not easy to change the sequence of the function call when copying programs because it is subject to a strict rule of grammar. In this regard, the comparison of the function call in the wake of the method call can lead to the judgment on whether it is a illegal copy.

4. System Implementation and Experiment

In this experiment, the result of CodeMatch[11] that is development by Zeidman Consulting, Windiff[12] included in the tools provided by Microsoft Visual Studio and automatic detection system of software copy using pattern suggested in this thesis.

The following is comparative list to decide whether to copy or not.

- Change of variable's position, change of variable's type, and replacement of variable,
- Change several sentences to a sentence,
- Input of dummy code, change of sentence's position.

CodeMatch compares two source codes by the 5 items below and output the result.

- Matching Source Code Lines(output the line number that is accurately accord between two source codes),
- Matching Comment Lines(output the line number that comments between two source codes are accord),
- Matching Semantic Sequences(output the line number that functions same with the two source codes),

- Matching Words(output the word that is accord between two source codes),
- Matching Partial Words(output the word that is partially accord between two source codes).

However, the matching Semantic Sequences function in CodeMatch is so weak, even though it is so important, it is decided not to detect properly in case of changing several sentences among several sentences in two source codes into a sentence, inputting unrequired dummy code in the middle of source code, changing the position of sentence, changing the position of variable, changing data type of variable, and replacing variable etc.
Windiff extracts same name and similar name by analyzing pile structure using directory comparison in each model, and decides whether to copy or not based on it. However, in case of inputting useless dummy code, it cannot sort out if it is copied or not in the case of correcting several sentences into a sentence.

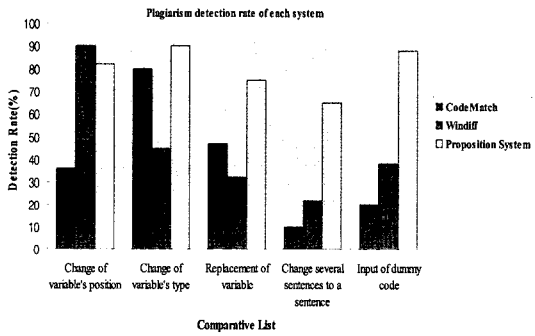

Fig. 3. Result of each comparative list

Fig. 3 is the result of each comparative list. Rather than CodeMatch and Windiff, system suggested in this thesis has better function of sorting out whether to copy or not. I think that the weak points that is not properly detected in CodeMatch and Windiff can be complemented and whether to copy or not can be decided more accurately in more parts by systemizing the symbol according to several patterns suggested in this thesis.

5. Conclusion and future research

The plagiarism of software is taking various forms in the rapidly-developing software field, and the number of cases involving the review of plagiarism is also increasing. Under these circumstances, a lot of tools have been developed to automatically detect such plagiarized programs. However, devices to detect such plagiarism using more diverse, detailed system, is required to detect a lot of plagiarized program language and more detailed and sophisticated plagiarism.
This dissertation analyzes a variety of problems with current detecting tools and also analyzed a system that applied the pattern matching system to develop a precise tool designed to detect the plagiarism. In addition to that, this dissertation presented a system that has a concept more advanced that the current pattern matching system. The automatic detecting tool, presented here, will play an important role for an objective and precise detecting by providing the testing person with the result from the detection of plagiarism at various levels and in various fields.
In the future, various methods such as graphs and charts should be defined and designed for more objective and precise reporting of the result as to the plagiarism detected through various methods and stages presented here, and in addition to that, more precise interface should be designed to allow the testing person to conduct various tests without problems.

References

[1] http://www.plagiarism.org.
[2] professor club, http://www.gyosuclub.com
[3] http://www.jplag.de
[4] http://www.mccabe.com
[5] Snatanu Paul "SCUPLE : A Reengineer's Tool for Source Code Search", University of Aarhus.
[6] Made Sign Ager, Oliver Dancy, and Henning Korsholm Rohde "Fast Partial Evaluation of Pattern Matching in Strings" Department of Computer Science University of Aarhus.
[7] Jun Myung-Jae "Constitution of mutual similar table between key words for copy detection" Pusan National University.
[8] J. t. oh's homepage. http://user.chollian.net'jtoh/bioinfo/bio_aln.htm.
[9] Michael J. Wise. YAP3:Improved Detection of similarities in computer program and other texts. University of Sydney, Australia.
[10] Greg Michaelson, "Automatic analysis of functional program style", ASWE'96 IEEE.
[11] http://www.zeidmanconsulting.com.
[12] http://msdn.microsoft.com/library/en-us/tools/windiff.asp.