Source Code Plagiarism in Computer Engineering Courses

Wolfgang Granzer
Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group
Treitlstraße 1-3, 1040 Vienna, Austria
w@auto.tuwien.ac.at
Friedrich Praus, Peter Balog
University of Applied Sciences Technikum Wien, Department of Embedded Systems
Höchstädtplatz 5, 1200 Vienna, Austria
{praus,balog}@technikum-wien.at

***Abstract*— In today's university life, teachers are often confronted with plagiarism. A special form of plagiarism is source code plagiarism typically found in programming courses at universities and schools. Detecting or even preventing source code plagiarism is by no means a trivial task. Therefore, this paper explains and discusses different methods that can be used to prevent and detect source code plagiarism. The second part of this paper is focused on automatic tools that assist in detecting plagiarism. Finally, an approach is presented which can be used to detect source code plagiarism in PLC (programmable logic controller) programs.**

***Index Terms*— Didactics, Plagiarism, Computer Engineering, PLC.**

I. Introduction

The motivation for students to engage in cheating is different as the cases arise. In some, students may have too little knowledge, motivation, or time to solve a particular challenge. However, often the motives are simple convenience and ignorance (i.e., being unaware about the ethical and legal facets of cheating). Since the term "cheating" has many different meanings, it can be interpreted differently. It can mean the illegal use of text books, cheat sheets, or the neighbor's help during exams, paying others to solve or create work for assignments, and also *plagiarism*.

There are many different definitions of plagiarism. In addition to a detailed survey about common interpretations of the term plagiarism in [1], a good definition can be found in [2]: "Plagiarism is the practice of claiming or implying original authorship of (or incorporating material from) someone else's written or creative work, in whole or in part, into one's own without adequate acknowledgment."

Plagiarism violates other people's or companies' intellectual property rights (IPR). According to [3], IPR regulate the rights the creator of a work grants to the general public with the main parts: *Copyright, Patents, Trade Marks, Design Rights, Passing off, and the law of confidential information*. Regarding plagiarism, the copyright is the most important part that is violated. This copyright exists for any original work from the time of creation on and regulates the legal options someone has regarding another ones work.

The work presented in this paper was funded by the City of Vienna, department MA27, under grant number MA27-Projekt 10-04.

Plagiarism can also happen unintentionally. This must not be neglected. For example, especially inexperienced students may copy from others without properly referencing their work. Nevertheless, also unintentional plagiarism comes along with the same drawbacks since it is still a violation of IPR. Thus, informing and educating students about cheating, plagiarism, IPR, and the resulting consequences is an important objective of higher education institutions.

For companies, plagiarism mainly has a financial aspect – money is lost if their work is stolen instead of being bought or licensed. From an educational point of view, the main consequence of plagiarism is that students are not getting familiar with the content of a certain lecture. For example, a simple "copy-and-paste" of the solution of an exercise does not require any knowledge about the content of teaching. Last but not least, education has not only the objective to teach new knowledge, but also to impart ethical principles. Furthermore, plagiarism lessens the work of others who have solved exercises on their own.

It is important to define which kind of intellectual creations (according to IPR laws) can be plagiarized and where plagiarizing starts. In education, traditionally books, theses, presentations, and papers have long been the only sources that are prone to plagiarism. Today, with the spreading of the Internet, source code, scientific publications and books and any information available on the Internet (e.g., forums, wikipedia) are equally subjected to plagiarism.

Due to the increasing number of participants in programming courses at universities, special attention has to be paid on *source code plagiarism*. The main resources of source code are the Internet (e.g., sourceforge, forums) and probably other students. Different types of source code plagiarism can be identified. First, source code can be taken from the openly accessible libraries. Since open-source allows per definition the (re-)use of existing source code (under limitations), this kind of plagiarism is not illegal. However, from the lecturers point of view, reusing source code from the Internet may not be tolerated. Second, the source code may also be obtained from (last year's) example solutions given out by the lecturers themselves or by other students.

Finally, source code can also be copied from other students. This can happen with or without the consent of the creator of the work. A problem may arise if source code is copied with the consent of the producer. In this case, the legal aspects are not fully clear. Still, in a university, it is considered plagiarism.

Connected to this problem is the case, where students are working in teams to solve an exercise. Clearly, all of them are the creators of the work and so (ideally) all have profound knowledge about the way how the exercise has been solved. Therefore, it is not clear whether such a situation should be identified as plagiarism or not. A possible solution is to either explicitly prohibit or allow teamwork by the lecturers (cf. Section II).

Due to these new challenges in the field of plagiarism, this paper is focused on source code plagiarism. In Section II, a brief introduction to source code plagiarism is given. Additionally, basic countermeasures are described. Section III is focused on the detection of source code plagiarism using automatic tools. Finally, Section IV describes how automatic tools can be used to detect plagiarism of PLC (programmable logic controller) programs. Since available tools are only partly usable for PLC programs, it is shown how a generic plagiarism detection algorithm can be adapted to support the detection of plagiarism of PLC programs.

II. Challenges in Source Code Plagiarism

According to [4], methods that counter source code plagiarism, can be divided into two categories. *Plagiarism prevention*, on the one hand, has the aim to avoid the opportunity of plagiarism a priori. In the case of source code plagiarism at education institutes, this means that students shall not have the opportunity to submit a solution to a programming exercise that he/she did not solve by him/herself. To avoid such a source code plagiarism, lecturers have to create exercises that are unique for each student. The best way would be to create specifications that are unrelated to each other. Obviously, this approach is impracticable for courses with a large number of attendees. The situation is further aggravated, if the same course is offered at regular intervals (e.g., each term). To avoid that students submit solutions that have been already previously submitted, the lecturers cannot reuse existing exercises. An alternative approach would be to define a common exercise and vary it in a way that the solution is different for each student. Consider, for example, a programming exercise where the birth date of the student is used as parameter that changes the way to solve the problem (and therefore the solution) slightly. If such a parameter that changes the way to solve the problem is used appropriately, it is possible to prevent plagiarism while the effort of exercise creation remains manageable.

On the other hand, *plagiarism detection* methods are used to identify plagiarism after the violation of IPR. In the case of source code plagiarism at education institutes, these methods have the aim to detect plagiarism after the students have submitted the solution of their exercises. While prevention methods set precautions that avoid plagiarism, detecting methods try to disclose already existing violations. Obviously, prevention methods should be preferred whenever possible since the positive effect is long-term. But as mentioned above, there are situations where prevention methods cannot be used due to possibly high administration effort. Detection mechanisms, on the other hand, are less time-consuming from the lecturers' point of view. Therefore, in situations where prevention methods are inapplicable, plagiarism shall be at least detected by using detection mechanisms. The rest of this paper is focused on detecting plagiarism.

Detecting source code plagiarism in programming courses can be achieved in two different ways. First, it is possible to invite the students to a personal interrogation after they have submitted their solutions (*manual plagiarism detection*). In such a conversation with the lecturers, the lecturers have the opportunity to verify whether the student has profound knowledge about the content of teaching or that the student has solved the challenge on his/her own. Obviously, these personal interrogations are very time-consuming and therefore, this form of plagiarism detection can only be used in small courses.

Second, special software tools can be used to detect source code plagiarism automatically (*automatic plagiarism detection*). Compared to manual methods like the interrogations mentioned above, the use of automatic tools results in less effort for the lecturers, if the tool has been adapted to satisfy the needs of the course environment. However, due to the nature of these automatic tools, there is always the chance that a legal solution is tagged as a false-positive plagiarism. Another problem with automatic tools is that students may work in teams. Sometimes, this situation is further aggravated since there may be courses where students have to submit their own solutions of the programming exercise but team work is not explicitly forbidden. Obviously, the submitted solutions (or at least parts of it) that are the result of (partial) team work will be similar. In such a case, automatic tools cannot be used since they are not able to figure out whether the student was involved in solving the challenge or not. Therefore, in such situations, a differentiation between plagiarism and team work is not easy to achieve especially if automatic tools are used.

To combine the advantages of manual and automatic detection methods, a hybrid approach can be used. An automatic tool can be used to identify solutions that are suspicious of being copied. Afterwards, the authors of these solutions are invited to a personal interrogation. Here, the lecturers can verify whether the solution was a real plagiarism by making a code review. If the student is familiar with his/her own code, and the student has enough profound knowledge about the content of teaching, the output of the automatic tool is a false-positive or the student was the originator and another student has copied the solution.

III. Automatic Tools for Source Code Plagiarism Detection

Detecting source code plagiarism using automatic tools is by no means a trivial task. The two most important objectives of a plagiarism detection algorithm are the *accuracy* (i.e., the detection-rate and the error-rate of the algorithm) and the *efficiency* (i.e., the complexity of the algorithm as well as the amount of time the algorithm needs to produce the result) [5]. On the one hand, the algorithms shall be trustworthy. On the other hand, the error-rate i.e., the probability of a false-positive decision shall be as low as possible. However, to achieve a high accuracy, a deeper analysis of the source code is required. Obviously, a detailed analysis is more time-consuming and therefore efficient algorithms are necessary.

Generally, the automatic process of detecting plagiarism[1] can be divided into three steps [5]. In the *abstraction phase*, both the used data (e.g., global and local variables and data structures) as well as the functionality of the program (i.e., the program logic) is analyzed. Then, the analyzed data and instructions are stored in an abstract way. The aim of this phase is to provide a generic representation of the program which can be further processed. The variables and data structures are often stored

[1]For the rest of this paper, the term plagiarism is used as a synonym for source code plagiarism

in tables where the identifiers as well as the data types are stored. The program logic, on the other hand, can be represented, for example, as flow charts. In the second phase, the stored information is filtered and irrelevant information is discarded (*filtration phase*). Finally, the filtered information is compared to identify program or data fragments that are similar and may indicate plagiarism (*comparison phase*).

The accuracy and efficiency of automatic tools is mainly influenced by the level of linguistic analysis of the abstraction and filtering phase. On the one hand, tools that are based on lexical analysis are simple and fast. A typical example would be a simple tool that counts tokens of a specific type and compares the amount of tokens found inside the programs. Obviously, since these tools do not consider the semantic of the program (i.e., the program logic), they are inaccurate and can be fooled very easily. On the other hand, to increase the accuracy of plagiarism detection, the functionality of the program has to be analyzed. This can be achieved by a detailed semantic analysis which is, however, time-consuming especially if complex programs have to be analyzed.

Since the accuracy of source code analysis algorithms on the one hand, and their efficiency on the other hand conflict, modern plagiarism detection tools use an iterative process to analyze source code. In a first step, the program is analyzed at a macro level. This means that the program is divided into a small portion of relatively large program fragments. These fragments are analyzed and, if they are suspected of plagiarism, these fragments are analyzed more in detail (micro level analysis).

IV. Plagiarism Detection in PLC Programs

Today, many different software tools that are dedicated to detect source code plagiarism exist. While earlier tools only support simple analysis mechanisms that are based on token counting [6], [7], modern plagiarism detection tools consider the program structure and semantic. Some of the most important ones are Plague [8], SIM [9], YAP3 [10], Moss [11] and JPlag [12]. A brief introduction to these tools is given in [13], [14]. Furthermore, an evaluation and comparison of them can be found in [15].

Since the syntax of programming languages varies, automatic tools have to provide support for those languages. Obviously, it is impossible to support all different programming languages and so, most automatic tools such as the ones mentioned above only support the most common programming languages (e.g., C/C++, Java).

However, PLC (programmable logic controller) programs are written in completely different languages. These languages are defined in part 3 of the International Standard for Programmable Logic Controllers (IEC 61131-3 [16]) which has been published by the International Electrotechnical Commission (IEC). In this standard, 5 different programming languages (three graphical and two textual) are specified. These are:

- Instruction List (IL): textual
- Structured Text (ST): textual
- Function block diagram (FBD): graphical
- Sequential function chart (SFC): graphical
- Ladder diagram (LD): graphical

While writing PLC programs using graphical languages is easier for beginners (and therefore for students), textual PLC programming languages are more suitable for automatic machine processing (e.g., a parser). However, it is possible to transform each PLC program from one IEC 61131-3 language into another (under some limitations). Therefore, the textual languages IL and ST are more suitable for automatic plagiarism detection. Figure 1 shows an example of a PLC network that has been designed using the PLC programming tool Siemens STEP 7-Micro/WIN. This simple network consists of a functional block `CTU` (up-counter). In the upper part of this figure, the network is represented as a ladder diagram. In the lower left corner, the corresponding instruction list is shown.

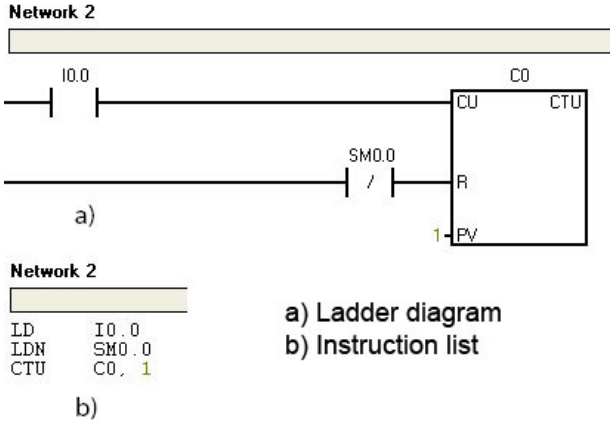

Fig. 1. PLC programming languages according to IEC 61131-3

However, a first look at the available detection tools shows that these tools are only partly usable to detect plagiarism of PLC programs. Therefore, the remainder of this section is focused on plagiarism detection of PLC programs. While in Section IV-A the problem of fooling detection tools with respect to PLC programs is discussed, Section IV-B described a possible detection method that can be used for PLC programs.

A. Fooling PLC Plagiarism Detection

A problem that arises with the use of automatic detection tools is that creators of source code may try to fool the detection tool. In [17], a list of possibilities to fool detection tools is given. With respect to plagiarism detection of PLC programs, these are[2]:

- *Text replacement*: A simple method to obscure plagiarism is to rename identifiers. For example, in a PLC program, the address of internal memory bits can be changed (e.g., use `M2.0` instead of `M1.0`).
- *Code reordering*: Code reordering means that parts of the program code are reordered in a way that the behavior of the program remains. In a PLC program, this can be done within a single network (e.g., reorder an ”and” operation like

```
LD M1.0
A M2.0
= Q0.1
```

to

```
LD M2.0
A M1.0
= Q0.1
```

[2]All examples given here are conform to the IL language as specified in IEC 61131-3.

), within a subroutine (e.g., reorder networks that are unrelated to each other) or even within the program (e.g., move code fragments from one subroutine to another subroutine or reorder subroutines).

- *Code rewriting*: Here, a single instruction or even a whole function is rewritten. For example, in a PLC program, the program logic of a PLC network can be rewritten in a way, that a `CTD` (down-counter) can be replaced by a `CTU` (up-counter).
- *Spurious code insertion/deletion*: To fool the detection tool, code that does not do anything meaningful can be inserted. Consider, for example, a useless network (e.g., any operation on a variable that is not used anywhere else) that does not influence the behavior of the program is inserted into a PLC program.
- *Source code mixing*: This means that plagiarized code (e.g., open-source libraries or code from other students) is mixed with own code. In a PLC program, someone can use foreign subroutines or libraries and include them into his/her own code. It is even possible that plagiarized code from multiple sources is used.

B. A Possible Approach to Detect Plagiarism of PLC Programs

In [5], a generic approach to detect source code plagiarism is presented. While this approach is, in principle, suitable for any programming language, the authors use the algorithm to detect plagiarism of C programs. Therefore, in the remainder of this section, it is shown how this basic algorithm can be applied on PLC programs.

The algorithm itself consists of 5 steps. In the first step (called *design generation*), the program is transformed into a structure chart. This structure chart consists of two parts: a symbol table called data dictionary and a tree representing the program logic. In the case of PLC programming languages, the data dictionary stores the used memory addresses including the chosen alias name, the memory type (e.g., input register, output register, variable memory, memory bits, special memory bits, counter memory, timer memory) and the used addressing mode(s) (bit-wise, byte-wise, word-wise or double word-wise). The program logic of the PLC program is stored in a tree. While the PLC main program itself acts as the root node for the tree, the different PLC networks and PLC subroutines of the main program are represented as subnodes of this root node. The operations and functional blocks are leaves of these subnodes.

In the second step, the structure chart is partitioned into strongly-coupled regions (*region delineation*). These regions are assigned a given region type. However, since the region types presented in [5] are dedicated to the C programming language, new ones have to be defined for PLC programs. A possible solution would be to classify the PLC networks according to the operation of the functional block. For example, the network

```
LD I0.0
LDN SM0.0
CTU C0,1
```

can be classified as a network of type `Counter` since the functional block `CTU` represents an up-counter.

In the next step (*abstract comparison*), the before classified regions are compared pair-wise with the regions of the other PLC programs that are under test. The main aim of this stage is to find regions that are basically similar. If there is a significant correlation between regions of different PLC programs, the algorithms continues with this region. Otherwise, the region is discarded and it is not further processed. Details on the comparison algorithm can be found in [5].

If a region is tagged as a suspicious one, a detailed comparison of this region is performed (step 4: *micro comparison*). Here, the nodes i.e., the statements of the regions, are compared more in detail.

Finally, the data dictionaries generated before are also compared for similarity.

In this section, only the basic principles of this generic algorithm are presented. However, a prototype implementation which can be used to detect source-code plagiarism of PLC programs is still missing. Using this implementation, the suitability i.e., the accuracy and efficiency has to be evaluated. Especially, it has to be investigated whether the algorithm can be fooled using the possibilities listed in Section IV-A.

V. Conclusion and Future Work

In this paper, a survey on source code plagiarism has been given. After a brief introduction, different methods to counter source code plagiarism are described. These methods can be classified into methods that prevent plagiarism a priori and methods that detect plagiarism. While, in general, a plagiarism prevention should be preferred, plagiarism detection methods are less time-consuming due to assistance of automatic software tools.

The second part of the paper is focused on these automatic tools that assist in detecting source code plagiarism. A description how these tools generally work is given. As shown in this paper, many different tools exist with support for different programming languages. Since the detection of plagiarism of PLC programs is not natively supported, an outlook on how this can be achieved using a generic detection algorithm is presented. However, since a prototype implementation is still missing, there is room for future work.

References

[1] H. Maurer, F. Kappe, and B. Zaka, "Plagiarism - A Survey," *Journal of Universal Computer Science*, vol. 12, no. 8, pp. 1050–1084, Aug. 2006.

[2] Wikipedia, "Plagiarism," Online[09.04.2008]: http://en.wikipedia.org/wiki/Plagiarism, Apr. 2008.

[3] M. Madhavan, "Intellectual Property Rights: An Overview," Online[09.04.2008]: http://www.jisclegal.ac.uk/pdfs/IPROverview.pdf, Mar. 2006.

[4] R. Lukashenko, V. Graudina, and J. Grundspenkis, "Computer-based Plagiarism Detection Methods and Tools: An Overview," in *Proceedings of the International Conference on Computer systems and technologies*, 2007, pp. 1–6.

[5] B. Belkhouche, A. Nix, and J. Hassel, "Plagiarism Detection in Software Designs," in *Proceedings of the ACM Southeast Conference*, Apr. 2004, pp. 207–211.

[6] K. J. Ottenstein, "An Algorithmic Approach to the Detection and Prevention of Plagiarism," *SIGCSE Bull.*, vol. 8, no. 4, pp. 30–41, 1976.

[7] J. L. Donaldson, A.-M. Lancaster, and P. H. Sposato, "A Plagiarism Detection System," in *Proceedings of the twelfth SIGCSE technical symposium on Computer science education*, 1981, pp. 21–25.

[8] G. Whale, "Identification of Program Similarity in Large Populations," *The Computer Journal*, vol. 33, no. 2, pp. 140–146, 1990.

[9] D. Gitchell and N. Tran, "Sim: a utility for detecting similarity in computer programs," *SIGCSE Bull.*, vol. 31, no. 1, pp. 266–270, 1999.

[10] M. J. Wise, "Yap3: improved detection of similarities in computer program and other texts," *SIGCSE Bull.*, vol. 28, no. 1, pp. 130–134, 1996.

[11] A. Aiken, "Moss: A System for Detectiong Software Plagiarism," Online[09.04.2008]: http://theory.stanford.edu/ aiken/moss/.

[12] L. Prechelt, M. Malpohl, and M. Phlippsen, "JPlag: Finding plagiarism among a set of porgrams with JPlag," *Universal Computer Science*, vol. 2, pp. 1016–1038, 2002.
[13] K. W. Bowyer and L. O. Hall, "Experience using 'MOSS' to detect cheating onprogramming assignments," in *Proceedings of the 29th Annual Frontiers in Education Conference*, vol. 3, Nov. 1999, pp. 13B3/18–13B3/22.
[14] C. Arwin and S. M. M. Tahaghoghi, "Plagiarism detection across programming languages," in *Proceedings of the 29th Australasian Computer Science Conference*, 2006, pp. 277–286.
[15] T. Lancaster and F. Culwin, "A Comparison of Source Code Plagiarism Detection Engines," *Computer Science Education*, vol. 14, no. 2, pp. 101–112, Jun. 2004.
[16] Internalion Electrotechnical Commision (IEC), "Internation Standard IEC 61131-3: Programming Languages," 2003, Second Edition.
[17] M. Freire, M. Cebrian, and E. del Rosal, "AC: An Integrated Source Code Plagiarism Detection Environment," Online[09.04.2008]: http://arxiv.org/abs/cs/0703136v1, Mar. 2007.