Plagiarism detection using software tools: a study in a Computer Science degree

A. Bugarín, M. Carreira, M. Lama, X.M. Pardo

Department of Electronics and Computer Science, School of Engineering, University of Santiago de Compostela, {alberto, mjose, lama, pardo}@dec.usc.es

Keywords
Plagiarism prevention and detection, e-learning, e-evaluation.

1. EXECUTIVE SUMMARY

1.1. Background

Plagiarism of projects submitted by students for evaluation in courses at the undergraduate level is a problem that produces increasing concerns in instructors in universities, since detection of plagiarized documents is becoming harder in continuously assessed courses, where the number of projects delivered by students is huge. Plagiarism presents particular features in Computer Science and related degrees, such as: (i) students must learn distinction between plagiarism and software reutilization as a basis of professional honesty, (ii) project-based assessment is widely extended and cannot be fully based on individual interviews with students, (iii) implementation-related skills are assessed by reviewing the source code of projects and (iv) sharing of knowledge has to be promoted among students.

1.2. Alternatives

In this paper we describe a study on plagiarism detection in programming projects of 8 courses of a BSc in Computer Science. 865 projects of different size (from 20 to 2000 source code lines) written in C and Modula-2 programming languages were screened using two plagiarism detection software tools that produce originality reports for each project including a global similarity index (SI). The reports were individually analysed in detail by the instructor of each course showing that even projects with very high SI values are not actually plagiarized. Quantitatively, 26 projects among the 100 ones that were evaluated by the tools as having SI >75% exhibited plagiarism evidences to some extent (3% of total). Usual reasons for high SI in non-plagiarized projects were legitimate reuse of code, the repetitive syntax of programming languages, or use of common modules for basic tasks usually solved in the same way. Due to this, it became clear that a manual in-depth individualized post-analysis of the reports needs to be done in order to avoid false positives. Having high quality and usability review facilities (such as highlighting similar fragments among documents, quick navigation between fragments, and easy access to external sources of potential plagiarism) are very valuable additions to these tools, which help to reduce time devoted to the necessary manual inspection of documents. Such features are very welcome by users.

1.3. Conclusions

It became clear after the study that inclusion of knowledge to plagiarism detection tools is a need when applied to programming projects. This knowledge is related to (i) a description of the resources in the courses and the minimum SI threshold that is acceptable (stating the reusable code, ...), (ii) the implicit information that instructors provide when a given document is labelled as plagiarized or not and (iii) including automated learning mechanisms for refinement of the plagiarized fragments detection. Addition of these features to a plagiarism detection software tool together with a good integration in the assessment workflow are key issues for constructing a valuable support system to e-learning based continuous assessment in programming courses.

2. INTRODUCTION

Plagiarism in the work submitted by students is a problem that generates growing concern among instructors, as it is significantly increasing (Hart, 2004; JISC, 2007) widespread at all educational levels, at least in two ways:

• Increasing facilities that students have for accessing to on-line resources (search engines, on-line information repositories, encyclopedias, books and magazines …)

• Work, project or report-based assessment of courses involving a huge number of documents to be evaluated by instructors.

It becomes necessary, therefore, that instructors be assisted on the detection and prevention of plagiarism in university teaching, in order to ensure the authorship of works, essential in a system of continuous assessment based on projects, but also to promote the idea of professional honesty in works authorship and references to the work of others and to recognize the efforts of students working autonomously.

In the area of computer science and related degrees this issue presents some particular features such as:

• Students must be specifically educated in the distinction between plagiarism and software reuse as a basis of professional honesty

• Programming courses assessment is often made based on software projects and can not always be addressed through personal interviews (especially in areas with high numbers of students) that, in addition tend to focus on aspects design, functionality and usability

• Assessing the skills related to the technical quality of implementations is usually based on a direct revision of the source code,

• Students should be promoted, even in individual projects, to share knowledge and information
• Sometimes, due to the number of projects or the nature of the proposed activities, reviewing work is done in a distributed manner by several instructors. In other cases, the same activity is proposed during several courses.

Although plagiarism is usually approached from a control and suspicion point of view, we understand that the main objective of plagiarism detection and prevention is to ensure confidence in the assessment system based on projects (central in adapting computer science studies to the European Higher Education Area) for students and teachers, an issue essential to ensure its viability. Only through being confident on the originality of the projects a reliable assessment of the work can be done and therefore the competences and credits awarded to students who successfully pass each subject will be reliable.

3. DESIGN OF THE TEST AND RESULTS

Within this context strict policies on plagiarism, which are increasingly supported by the use of computer resources, particularly software tools for automatic detection of plagiarism (Bull, 2001; VAIL, 2008), have been established by universities. These tools usually make comparisons among documents and deliver a report that usually includes a numerical assessment of the degree of originality of the work (similarity index, SI) and also visually highlights those fragments that are suspicious of being plagiarized, allowing quick access to the document/s which allegedly came from. Software tools for automatic detection of plagiarism are very diverse:

• From a functional point of view there are tools designed to detect plagiarism in general discursive documents (in different document formats such as .PDF, .RTF, .TXT) or in specific programming source code (Java, C, C + +, C #).

• There are stand-alone applications, whilst others can be integrated with some of the more known e-learning platforms (Moodle, WebCT, …), sometimes including other functionalities such as correction or collaborative management of various activities and/or users.

• From the standpoint of its architecture there are tools that analyze the documents stored on a local directory tree and tools that request documents be uploaded to a central storage server. Some of them compare documents from external sources and / or an internal documents base.

• Finally, there are free software and commercial tools.

Therefore, the choice of a particular tool requires a thorough analysis and assessment of the requirements desired and the specific case of use. Since our interest was to do a test for analyzing the projects in a number of programming courses in a Computer Science degree the first natural choice was to select a source code oriented tool, since these tools make use of the particular characteristics of source code programs (such as their syntax, the existence of declarative part) for producing reports that a priori should be more accurate. After conducting a review of this type of tools JPlag (Prechelt, 2002) free access Web Service was selected. Furthermore, for the sake of making comparison on the results obtained, Turnitin commercial general-purpose tool was selected, since it could be integrated with our university corporate e-learning platform, it allowed comparison to external sources and also produced high quality and usability reports. In this case a second aim of the test was to assess the performance this general-purpose tool in a specific technical domain such as computer programming source code documents. The most relevant features of the study conducted were:

• 865 projects delivered by students in 8 programming courses written in C and Modula-2 languages were used for the test.

• Six courses involved small and major programming projects (from 150 code lines up to 2000 lines). Two courses included text (non source code) documents or code written in other languages

• Documents (source code comments, additional documentation or the full work) were written in Spanish or Galician language.

A small number of control documents (repeated documents, texts copied and pasted from Internet resources) were also included in the test. All these control documents were detected by both tools.

All of the originality reports (OR) delivered by JPlag and Turnitin were individually analyzed in detail by the instructors. The OR included an overall similarity index (SI) for each project with regard to the others. The SI of a document is a measure (between 0% and 100%) assessing the degree of similarity of that document with other documents in the document database. Therefore, the higher the SI of a document, the lower its originality and therefore the higher the suspicion it is wholly or partly plagiarized. In spite that the SIs are calculated differently in each tool, the results of our tests indicate that similar qualitative considerations can be done.

Our results may suggest incorrectly that there has been a very high level of plagiarism among the documents analyzed as, for example, about half of documents were labelled with SI>50%. However, manual analysis conducted by instructors showed that even projects with very high SI values were not actually plagiarized. Quantitatively, only a total of 26 among the 113 projects with SI>75% were plagiarized in part (a 3% of the total number of projects). The causes of this apparent non correlation between SI estimations and the actual plagiarism are different and dependent on particular characteristics of each activity, but can be generalized in the following terms:
• In some of the activities students were asked to reuse their own code written for previous projects or to incorporate mandatory source library functions. Both tools correctly identified all these situations, which are not plagiarism but an own goal of the activity.

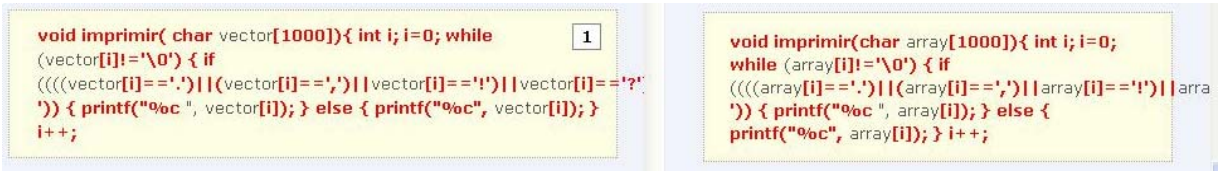
• Almost all documents with a high SI documents were not actual plagiarism. A great weight of the value of the index was due to a legitimate code reuse, by the repetitive syntax of programming languages (primarily in the declarative part, in particular data types, variable names and libraries importation) or modules commonly used in programs for elementary tasks that are usually done in the same way.

• There were also cases in the Turnitin reports where high SI in large documents was due to an accumulative effect (a high number of non-significant coincidences). In fact, the mere incorporation of works with legitimately similar contents increases SI values for new documents.

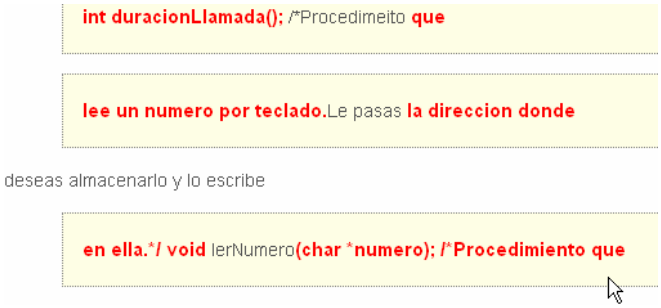
• Documents with a low SI were original. Although this seem to be obvious, it has the direct benefit that instructors can directly focus their attention on documents with a value exceeding a certain threshold that can be experimentally estimated for each activity.

In our test there were no plagiarism cases from external sources, so this feature of the Turnitin tool was not useful in practice for our case.

Figure 1 shows two examples of plagiarized fragments in C source code programs. It can be seen that a simple change in the variable names or small changes in the comments is not a valid strategy to avoid detection.



a)



b)

Figure 2. Plagiarized sections (bold) in one of the C language projects in this study. Non-bold names correspond to: a) variable names that were changed in the two projects and b) to slight changes in the text or function names.

In general, it should be noted that for each activity a SI threshold can be estimated without too much difficulty for defining the border between non-plagiarized and possibly plagiarized documents. Since the latter documents have to be manually screened by instructors, quality and clarity of the reports and facilities for accessing to documents are critical, since they contribute to reduce inspection time. In this context, we emphasize the importance of the colour code that is usually used in reports to highlight the fragments that the tool considers to be non-original, the utility of hyperlinks to the allegedly copied fragments came and the navigation tools for accessing parts of the documents. These functionalities allow rapidly confirming or discarding an immediate suspicion of plagiarism between any two documents. In our test, the functionalities that offered Turnitin were much richer than the ones by JPlag.

It should be noted that most of these analysis tasks could be carried out by support staff that would play the role of the main reviewer of the originality of work. Within this context instructors can focus on the genuinely academic task of assessing the work, without going into any consideration about suspicions about its originality.

3. CONCLUSION

The study allowed assessing the use two tools (JPlag and Turnitin) for automatic detection of plagiarism in programming projects in a computer science degree. Very similar qualitative conclusions can be obtained with both tools, which can be summarized as: a low similarity index is an evidence of originality, while a very high similarity index is usually an evidence of plagiarism. Intermediate cases need manual screening and comparison among documents. For this task, clarity and richness in the reports provided by Turnitin proved to be decisive, since they significantly reduce the time devoted to analysis of individual documents. On the contrary, accumulative effects in Turnitin reports make information provided by its similarity index less informative in general than JPlag's.

The local comparison among the projects submitted by students resulted more important in this area than the detection of plagiarism from external sources, since no cases of external copy occurred in our tests. Nevertheless, mechanisms for making incremental analysis are of interest, mainly when a

given activity is proposed to different groups of students (to be assessed by a group of instructors) in different years. From this point of view these tools are helpful for maintaining a kind of collaborative intelligence which is helpful for improving detection of plagiarism.
Our study is a first attempt to quantitatively estimate the extension of plagiarism (which almost all the teachers have the perception that occurs, but have no evidence of its extension) and providing some empiric evidences to support the decision on which automatic detection tool/s to implement for the case of programming projects in computer science degrees. From our data, a 3% of the projects were plagiarized to different extents, ranging from small code fragments to almost a complete medium size projects. The study keeps currently going on including more projects and new programming languages.
A crucial point, from the university organizational point of view, is the integration of these tools with the corporative e-learning platforms. Providing a transparent integration between these tools and the work submission procedures would be a very valuable service for e-learning platforms and will help to face the necessary task of integrating the detection of plagiarism within the workflow of students' assessment.
Lastly, entering rather in the field of innovation for plagiarism detection tools, it seems clear the need to incorporating knowledge in this type of tools, mainly in terms of:
1. The description of the subjects from the standpoint of the resources used and the similarity tolerable in documents handled by students (indicating which are reusable, to what extent ...),
2. The information implicitly or explicitly provided by the instructors when considering or discarding as plagiarized the fragments classified by the tool as "very similar"
3. Incorporating learning mechanisms for automatic refinement and improvement of the discovery results.
Incorporating these features to the software tools to detect plagiarism, along with a successful integration into the workflow of the students assessment, are key elements in building support systems to continuous assessment based e-learning of programming courses in computer science and related degrees.

4. REFERENCES

Bull, J., Collins, C., Coughlin, E. and Sharpe, D. (2001) *Technical Review of Plagiarism Detection Software Report*. Retrieved May 15, 2008 from http://www.jiscpas.ac.uk/documents/resources/Luton_TechnicalReviewofPDS.pdf

Gaither, R (2007). *Resources for instructors. Plagiarism detection services.* University of Michigan. Retrieved May 15, 2008 from http://www.lib.umich.edu/acadintegrity/instructors/violations/detection.htm

Hart, M., Friesner, T. (2004), Plagiarism and Poor Academic Practice – A Threat to the Extension of e-Learning in Higher Education? *Electronic Journal of E-learning, 2, 2*. Retrieved May 15, 2008 from http://www.ejel.org/volume-2/vol2-issue1/issue1-art25.htm

JISC-Joint Information Systems Committee (2007) *Institutional issues in deterring, detecting and dealing with student plagiarism.* Retrieved May, 15, 2008 from http://www.jisc.ac.uk/media/documents/publications/plagiarismfullreport.pdf

Prechelt, L., Malpohl, G., Philippsen, M. Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science, 8, 11, 1016-1038, 2002. Retrieved May, 15, 2008 from http://www.jucs.org/jucs

VAIL-Virtual Academic Integrity Laboratory (University of Maryland University College). Retrieved May, 15, 2008 from http://www.umuc.edu/distance/odell/cip/vail/home.html