Plagiarism Detection in Code-Based Assignments

M. Ouellet, D. Guay, J. Watso, P. Morneau-Gagnon, C. Martel, E. Merlo
Department of Computer and Software Engineering
École Polytechnique de Montréal
{maxime-3.ouellet, david-shaun.guay, jeremy.watso, philippe.morneau-gagnon, colin.martel, ettore.merlo}@polymtl.ca

Abstract

Many engineering and computer science programs include computer based courses in which students are evaluated on code that they write individually or in groups. Plagiarism of code is typically difficult to detect, because courses have several markers, who cannot readily compare current submissions to those of students in other classes or in previous years.

Our team is currently developing a plagiarism detection Web service that allows instructors to submit code and detect similarities between assignments and between projects. The interface is Web-based and is designed to smoothly integrate into instructors’ marking activities. The presented Web service compares all uploaded assignments, and identifies structural similarities. Since our service is based on structural analysis, superficial changes to variable names or control statements are not enough to conceal plagiarism. The results of similarity analysis are summarized in a simple format that indicates which assignments are the most similar and reports differences between assignments at the fine grain level of lexical tokens.

The strength of this automated approach is that it removes the burden of plagiarism detection from instructors. In addition, it is fully repeatable and also provides a high level of accuracy with no false negatives.

1 Introduction

Within academia and especially in the context of assignments and projects, plagiarism is a problem that universities have to address through policies and tools. With the development of Internet, plagiarism has become even more accessible, because resources can easily be accessed through the simple click of a button and artifacts can be copied. Automated tools that detect plagiarism in text documents are freely and commercially available [1][2][3][4]. These tools are not easily applicable to code-based assignments, which are often required in engineering, computer science, and other university programs. Plagiarism of code fragments is typically difficult to detect, because assignments are often graded by several markers, who cannot readily compare current submissions to those of students in other sections or in previous years. This is especially true in the case of courses given to a lot of students, because of the size and the number of classes.

Moreover, applying text-based plagiarism detection tools to code fragments may not work that well, because source code textual and literary properties may be less rich and relevant than in text. In source code, formal and structural properties contain more relevant information than lexical ones. For example, systematic differences between identifiers in code may be irrelevant from a plagiarism detection point of view, while differences in the algorithmic structure of operations may indicate more relevant differences in the meaning and the execution of a program.

Unfortunately, while the technology to detect software plagiarism exists, few tools implement it in an accessible and easy-to-use way at a university-wide level or across universities. Existing tools are either only available as stand-alone programs [5][6] or require user registration and computer science knowledge [7][8] to be successfully used.

To address this problem, our team is currently developing a plagiarism detection Web service that allows instructors to submit code and detect similarities between current and/or past submissions. The user interface is Web-based and is designed to smoothly integrate into the instructors’ marking activities.

This paper presents the description and the design of a tool that could be used by any instructor. The presented architecture is portable and extensible, so that it can eventually be used and customized by multiple universities. User interface is programmable through Web services and is meant to be easily usable. A brief overview of source code similarity technology is also presented.
Section 2 gives an overview of the plagiarism detection technology we used. Section 3 presents the Service Oriented Architecture and section 4 describes future improvements to our tool. Finally, conclusions can be found in Section 5.

2 Plagiarism Detection Technology

In order to successfully identify similarities between multiple code fragments, we built our application on top of a tool called CLAN (CLone ANalyzer) [9, 10].

CLAN computes code fragment similarities by considering software metrics that are derived from a syntactic analysis of the code and that account for size, control flow, method communication and coupling. Fragments extracted from an Abstract Syntax Tree (AST) are compared on the basis of software metrics as described in [9, 11, 12]. Using these metrics, similar code fragments called clones can be identified in the source code, thus making it possible to massively screen assignments for mutual similarities.

Figure 1 outlines the process used by CLAN for detecting software clones and ranking similarities in a C/C++ program.

Some commercial applications for software similarity analysis are related to intellectual property issues, especially in the case of incorporating license constrained open-source code in an proprietary application. The approaches reported in [13, 14, 15, 16] offer open-source license infringement detection capabilities.

The clone relation used in this paper is based on our previous work [10, 11, 12, 17, 18]. Other approaches for clone analysis have been presented in [19, 20, 21, 22, 23, 24]. Empirical studies and evaluation of clone detection approaches can be found in [9, 25, 26, 27].

Very interesting and comprehensive surveys on the clone detection literature can be found in [28, 29].

3 Service Oriented Architecture

Source code similarity analysis and plagiarism detection are documented in literature [5, 7, 8]. To promote their adoption, the main objective behind the development of our plagiarism detection application was to simplify the use of CLAN technology, in order to make it accessible and available to those instructors who may need it.

We developed our application in three separate modules that are virtually independent from one another: a Similarity Analysis (SA) Web services, a Task Scheduler (TS), and a Web User Interface (UI). These three modules will be detailed in the following sub-sections.

The deployment diagram in Figure 2 illustrates how our three modules are interconnected. As can be seen, our Web UI, which is shown as a Web application, communicates with our SA Web service. Communication between SA Web service and TS is done through the use of a common database. Our Web UI has its own separate database, allowing it to handle data that may not necessarily be significant for the SA Web service such as logging information, supplementary task information and users’ contact information.

3.1 Similarity Analysis Web Service

In order to have a modular and extensible application, we built a SA Web service module to define a Web Application Programming Interface (API). The advantage of using Web services is that it allows us to make basic functionalities, in this case code plagiarism detection, independent of the user interface. Using Web services makes it possible to make specific technologies available to a wide audience without forcing them to use our Web UI. This means that a university could build its own user interface that uses our services, giving their staff a complete plagiarism detection tool without having to support the technology itself with the desired user interaction.

The SA Web service is built around basic concept of "tasks". Users can submit a new code analysis task to
the SA Web service by providing the necessary information: programming language, archive of files to analyze, and so on. Once this task is received by the SA Web service, it is assigned a unique identifier which is returned to the user. From there, a user can request the status or the results of any task he might have by sending the appropriate request to the SA Web service.

Using the basic principles of Representational State Transfer (REST), we were able to build a complete and simple RESTful Web service that makes plagiarism and clone detection functionalities available to any potential external Web application and external users.

RESTful Web services allow us to make specific resources, in this case tasks, available to users. Using standard HTTP requests, a user can easily access every functionality the SA Web service offer.

To ensure secure data transmission, our service supports HTTPS and user authentication. This means that in order to use the SA Web service, users must first register themselves through the service, which will give them a unique identification key. Subsequent uses of the Web service will then require that this unique identification key be sent by users so that he can identify themselves.

### 3.2 Task Scheduler

Because of hardware-related constraints and physical limitations such as available memory, disk space and processor speed, only a finite number of tasks can be executed at the same time. However, considering that a variable and possibly high number of tasks could potentially be requested in a short amount of time, we found it necessary to implement a task scheduler that has full control over how many tasks are currently being executed and which task is next in line.

In order for the system to be flexible, we created a hierarchy of priorities divided amongst three levels: providers, users and tasks. Table 1 defines those different levels. The scheduler first start by determining which provider is next in line, then determines the next user in line for this provider, and finally executes the task with the highest priority for this user.

The three levels of priorities allow to have control over which provider should have the highest priority in the system. Each provider has control over the priorities of its users and each user controls the priorities of his tasks. In order to prevent an high-priority element from monopolizing the application for multiple tasks in a row, we also implemented a dynamic priority system that takes into account the time since when an element has been waiting its turn in the scheduler.

### 3.3 Web User Interface

The Web UI implemented for our university has been designed to give access to the basic functionalities of plagiarism detection: creating a user, logging in, creating tasks and consulting task results.

Using our application, a user can create a plagiarism detection task by going through a simple task creation wizard, which asks the users for some information about the task to accomplish: programming language, archive file containing the assignments to analyze, and so on. This task is then sent to the SA Web service and will eventually be picked up by the TS. Once the task is completed, the user will be able to access to its status screen to see the results. Figure 3 shows an interaction using our user interface with the results for a test plagiarism detection task.

As can be seen in the example in Figure 3, results of plagiarism detection task are composed of pairs of assignments accompanied by a score. Typically, the higher the score between two assignments, the more similar those assignments are.

To decide about a similarity threshold to select pairs of assignment for further administrative processing, the histogram of score distribution over all pairs of processed assignments is computed and visualized, as shown in Figure 3.

Figure 4 shows real results obtained after running our tool on multiple C++ projects, as reported in [10]. Looking at this histogram, we may want to better examine all pairs of projects that scored above 0.95, because they are unusually similar, since there is a significant gap between score 0.90 and 0.95. Also pairs that scored around 0.90 could be further examined, since their similarity score is clearly above average.

In order to facilitate the examination of highly similar assignments, our UI allows users to visualize the differences between pairs of similar fragments using Dynamic Programming (DP) alignment of the syntactic tokens of fragments and showing differences in color.
The UI visualizes the code of both fragments side-by-side displaying the differences between them in color, as shown in Figure 5. It becomes easier for users to interpret the similarity results provided by the tool.

4 Future Improvements

4.1 Multi-language support

The current version of our application supports plagiarism detection for C++ code-based assignments.

More programming languages should be addressed in the future. Similarity is already available for Java and PHP in our laboratory and it could be integrated in the presented Web architecture. Other imperative languages such as Matlab and C#, scripting languages such as Perl and Python, and functional languages such as Lisp or ML could also be considered for similarity analysis if desired.

4.2 Improved Capacity

As the number of users of our application increases, it is likely that one computer to handle all code analysis tasks won’t be sufficient. Our application was designed so that multiple schedulers can run at the same time, on different computers. The only requirement is that the files to analyze received by the SA Web service must be accessible on the computers running the schedulers. This could be accomplished by using a shared network drive or a software package to share resources such as Samba.

Running schedulers on multiple computers would allow for a distributed handling of tasks, which means that modifying the capacity of our application is obtained by adding or removing a computer running our TS. Each computer can also run an arbitrary number of tasks at the same time: this value can easily be changed in a configuration file, which allows users to optimize the application based on the available hardware resources.

5 Conclusion

Plagiarism detection in code-based assignments is an area that can benefit from automated tools.

A metrics based plagiarism detection approach in an academic environment has been presented.
The Web UI we have built is an interactive layer built on top of existing technologies, with the goal of making these technologies widely accessible to instructors. We created our own Task Scheduler, a Similarity Analysis Web service to create and manage code analysis tasks and an Web User Interface using these services.

We believe that the presented tool can help enforcing plagiarism detection policies, especially in domains such as computer and software engineering, where source code assignments are often required. Indeed, the presented approach has been successfully used to discourage plagiarism in course projects.

One of the strengths of the automated approach to plagiarism detection is that it can shift the burden of plagiarism detection from instructors to an automated tool. In addition, our approach is repeatable, objective, and auditable.

Measuring the impact of the presented tool on plagiarism reduction is also an area that we would like to explore. Once educators start using our application, it will be possible for us to use metrics and do an in-depth analysis of the effects of automated code plagiarism detection in courses. Ethical issues will have to be addressed for research on impact of the tool.

Further research can include more programming languages, empirical studies and tool evaluation, refinement of similarity criteria, and investigation and improvement of ergonomics of human interactions.

Related research topics can include investigation of UI testing, Service Oriented Architecture (SOA) testing, addressing security issues such as SQL-injection problems (SQLI), Cross Site Scripting (XSS), Role Based Access Control (RBAC), and denial of service attacks.

The development of the presented tool itself has already involved students at Bachelor level in a comprehensive project at Ecole Polytechnique and visiting summer students as engineering trainees.

Acknowledgment

The research on CLAN has been funded in part by the National Sciences and Engineering Research Council of Canada.

References


