TAPS: A MOSS Extension for Detecting Software Plagiarism at Scale

Dana Sheahen
Georgia Institute of Technology
Atlanta, GA, USA
danasheahen@gatech.edu

David Joyner
Georgia Institute of Technology
Atlanta, GA, USA
david.joyner@gatech.edu

L@S 2016, April 25-26, 2016, Edinburgh, Scotland UK.
ACM 978-1-4503-3726-7/16/04.
http://dx.doi.org/10.1145/2876034.2893435

Abstract

Cheating in computer science classes can damage the reputation of institutions and their students. It is therefore essential to routinely authenticate student submissions with available software plagiarism detection algorithms such as Measure of Software Similarity (MOSS). Scaling this task for large classes where assignments are repeated each semester adds complexity and increases the instructor workload. The MOSS Tool for Addressing Plagiarism at Scale (MOSS-TAPS), organizes the MOSS submission task in courses that repeat coding assignments. In a recent use-case in the Online Master of Science in Computer Science (OMSCS) program at the Georgia Institute of Technology, the instructor time spent was reduced from 50 hours to only 10 minutes using the managed submission tool design presented here. MOSS-TAPS provides persistent configuration, supports a mixture of software languages and file organizations, and is implemented in pure Java for cross-platform compatibility.

Author Keywords

Software plagiarism; MOSS; cheating; academic integrity.

Introduction

Plagiarism in computer science classes can damage the reputation of institutions and their students. It is a problem that must be addressed by instructors to preserve the integrity of awarded class credit, particularly for an accredited degree program such as the Online Master of Science in Computer Science (OMSCS) program at Georgia Tech. However, as the number of students increases and as classes and assignments are repeated from semester to semester, more assignments must be cross-checked with an ever-increasing pool of previous submissions using available software plagiarism detection algorithms such as Measure of Software Similarity (MOSS) [1]. Even if the rate of plagiarism remains constant, detection at a larger scale becomes unwieldy without persistent workflow plans and straightforward management tools.

The MOSS Tool for Addressing Plagiarism at Scale (MOSS-TAPS) repackages and organizes submissions for plagiarism detection for courses that repeat a coding design assignment from semester to semester.

Need

Why check for software plagiarism?

Cheating in computer science classes in the form of software plagiarism exists at an estimated incidence rate of 10% or more [2] per assignment. It is therefore essential to routinely authenticate software submissions from students to expose dishonesty, deter future problems, and to assure students that their honest efforts are not undermined by students that copy previous solutions.

Classes in the OMSCS at Georgia Tech have the same concerns in this regard as any on-campus program, with the added complication of a larger scale, i.e. more students, more assignments to check, more combinations to cross-check.

Challenge

Current tools

MOSS-TAPS specifically submits assignments to MOSS, one of the most effective software plagiarism detection algorithms available [4] [5], and currently offered as a free (for non-commercial use) web service executed on Stanford University servers. To test for software plagiarism using MOSS, an instructor submits a batch of individual student code directories, also called the corpus, to MOSS servers using a PERL script provided by MOSS, and receives, as a result, pairwise similarity percentages and counts via a web page. The results must then be analyzed and "suspicious pairs" reviewed individually. MOSS provides ordering of the results by highest similarity (most suspicious), with additional links for each pair to color-coded webpages for review.

Use-case example

A recent example of the challenges faced in this process occurred in the CS7637 Knowledge Based Artificial Intelligence (KBAI) course at Georgia Tech [3]. The class includes three or four open-ended coding projects that students must complete. The online version of the class averages 200-400 students; 1400 students have taken the course through spring 2016, with an 80% completion rate.

As the pool of past participants increases, so does the pool of correct solutions. In order to fully check for plagiarism in a current class, it is necessary to compare not only the current submissions against each other, but

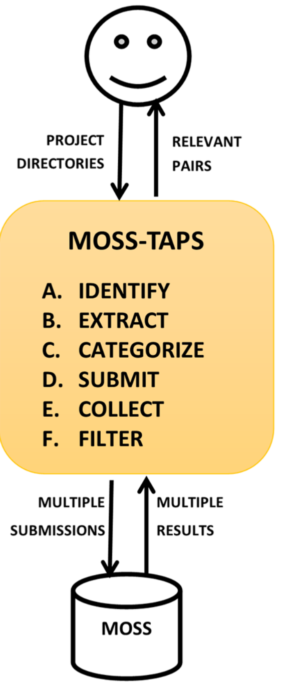

Figure 1: MOSS-TAPS provides management of multiple language submissions, arbitrary directory depths, multiple project iteration submissions, and results filtering.

all previous assignment submissions against the current submissions.

In the summer 2015 KBAI class, its third iteration, the submission management problem became quite cumbersome as 400 current projects were checked against each other as well as 500 previous ones. In particular, the instructor identified the following challenges to the process:

- Mixed Languages: This particular class allows student choice of Python or Java implementations, so all of these assignments required separation by the instructor before submission to MOSS.
- File Management: The student online submission scheme included zip files and multiple depths of directories that had to be expanded and normalized by the instructor before submission to MOSS.
- Filtering: The projects were checked against previous semester projects, but the matches found among previous projects needed to be manually ignored. Additionally, the projects were checked against previous assignments that the projects built upon within the semester, but matches to the same student needed to be manually ignored.
- Cross-platform: The PERL script provided for MOSS did not work on a Windows machine and had to be submitted with Linux. An available Windows version was unable to handle the number of files required.

Design Solution

The design approach to solving these challenges is to require the instructor to store each past project collection in one directory ("Corpus") and each current semester project in another ("Current"). The tool is responsible for finding the actual source code files under each student's submission, organizing and submitting the files to MOSS, and filtering the results so that only relevant comparisons are viewed.

Configuration and Mixed Language Support

The basic MOSS submission requires the user to specify a software language, MOSS user ID, and some optional parameters. MOSS-TAPS addresses these configuration setups persistently through default settings and a configuration file. In addition, MOSS-TAPS allows a mixture of software language submissions for a single project. This supports the case where a software coding assignment allows students to choose which language he/she wishes to use for implementation.

File Management of Submissions

MOSS-TAPS supports a file structure that includes unique student directories of any depth located immediately under a project directories. MOSS-TAPS searches each student directory for relevant source files, unzips any compressed files or directories, and packs the files for submission. This removes the necessity of manually putting all the nested, same-language, packaged files from a complex project in a single directory or requiring a particular directory structure arrangement from students.

Filtering

The result provided by MOSS is an ordered list of comparison pair links presented on a webpage. MOSS-TAPS filters the list on the webpage to exclude comparisons within previous projects and between a student and his or her own previous assignments. The results are provided as comma delimited files for each of the software languages identified.

Cross-Platform Support

The basic MOSS script is guaranteed to work in UNIX, but not necessarily on other platforms. The MOSS site provides links to a number of unofficial open source 3rd party alternatives for socket management of the MOSS server protocol. One such is MOJI [6], an unofficial Java client for MOSS. MOSS-TAPS incorporates MOJI to interface with the MOSS server. MOJI and MOSS-TAPS are pure Java, and provide cross-platform support.

Results

In the summer 2015 KBAI class example provided above, the instructor spent 50 hours organizing, submitting, and filtering the class assignments for the purpose of software plagiarism detection. For the fall 2015 iteration of the class, the instructor spent only 10 minutes on the submission process with this tool design, even though an additional semester's previous results were added to the corpus. The time savings and ease of submission gained ensure that this necessary step in the instructor's grading workflow can be incorporated smoothly even with large numbers of assignments.

Open Source and Future Enhancements

MOSS-TAPS was inspired and designed to solve particular use-cases for classes at the Georgia Institute of Technology. There are a number of enhancements and additional use-cases that can be built on and expanded from MOSS-TAPS going forward. MOSS-TAPS is a true Work-in-Progress and is provided as an open source solution hosted publicly on GitHub at https://github.com/danainschool/moss-taps .

Acknowledgements

This work was completed in a class offered in the Georgia Tech OMSCS program, supported by the Georgia Tech College of Computing, Udacity, Inc., Georgia Tech Professional Education, and AT&T. We are grateful to all members of these organizations that played critical roles in developing this program and allowing this research to take place. We would like to thank Education Technology mentor Ken Brooks for his continued support and encouragement in this endeavor as well.

References

1. Alex Aiken. 2014. Moss - A System for Detecting Software Plagiarism. Retrieved January 8, 2016 from https://theory.stanford.edu/~aiken/moss/
2. Charlie Daly and Jane Horgan. 2005. Patterns of plagiarism. *SIGCSE Bull.* 37, 1 (February 2005), 383-387.
3. Ashok Goel and David Joyner. 2014. CS7637: Knowledge-Based AI:Cognitive Systems [Online Course]. Retrieved January 8, 2016 from https://www.udacity.com/course/knowledge-based-ai-cognitive-systems--ud409
4. T. Lancaster and F. Culwin. 2004. A comparison of source code plagiarism detection engines. *Computer Science Education*, 14, 2, 101-112.
5. Divya Luke, Divya P.S., Sony Johnson, and Elizabeth Varghese. 2014. Software Plagiarism Detection Techniques: A Comparative Study. International Journal of Computer Science and Information Technologies. *International Journal of Computer Science and Information Technologies.* 5020-5024.
6. Bjoern Zielke. 2014. MOJI unofficial Java client for Moss [gitHub]. Retrieved January 8, 2016 from https://github.com/nordicway/moji