CoMoTo - the Collaboration Modeling Toolkit

Charlie Meyer, Cinda Heeren, Eric Shaffer, and Jon Tedesco
cemeyer2@illinois.edu, c-heeren@illinois.edu, shaffer1@illinois.edu, tedesco1@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

We are excited to introduce CoMoTo – the Collaboration Modeling Toolkit – a new, web-based application that expands and enhances well-known software similarity detection systems. CoMoTo is an end-to-end data management, analysis, and visualization system whose purpose is to assist instructors of courses requiring programming exercises to monitor and investigate the extent of student collaboration, both allowed and illicit. We describe CoMoTo's interface, which was designed to facilitate scrutiny of collaboration data projected along student, course, assignment, etc. attributes, and to allow for interactive visualization of pairwise similarity measures via a dynamic graph. We also elaborate on the details of CoMoTo's implementation. Finally, we briefly discuss two use cases that foreshadow CoMoTo's broad utility in student code analysis, not only for plagiarism detection, but also for investigating early student coding styles, and for evaluating software similarity detection systems, themselves.

Categories and Subject Descriptors: K.3.2 [Computer and Information Science Education]: Computer Science Education

General Terms: Reliability, Human Factors.

Keywords: Program Similarity, Pedagogy

1. INTRODUCTION

As educators, we spend countless hours adapting our interactions with students so as to maximize learning, and yet we still have very little insight into the way students work and the way they think when we are not there. In this paper we describe CoMoTo – the Collaboration Modeling Toolkit – a web-based software similarity visualization and analysis tool we have developed with the goal of adding value to the output from well-known source code similarity detection systems. Such systems have become an important tool in Computer Science education. While they are most commonly used in programming courses to detect plagiarism, a deeper analysis of code similarity reveals common programming styles among students, and even the sociology of how collaboration networks develop and function within a class. It is this richer understanding of student work that we achieve via CoMoTo.

The genesis of CoMoTo corresponds to our need to organize and understand software similarity system results when we applied it to student solutions to programming assignments. Course policy allowed constrained and cited collaboration, so we had an immediate need to assure that the course policy was being followed, and we wanted to avoid manual scrutiny in favor of a more complete and reliable evaluation. The output from the system gave us pairwise similarity measures for each pair of students in the class. We represented this output as a graph where each node was a student, and edges between pairs of students existed if their code similarity measure exceeded some reasonable threshold. This first, primitive implementation of a graphical visualization of software similarity results revealed that we had an opportunity to discover much more about student learning. The size of our course and many semesters of archived student code submissions provided us with an excellent and challenging testbed within which to develop our analytical tools. Since then, the software has been packaged into a end-to-end system that allows for simple student source code entry, flexible choice of software similarity detection system, many different projections of the similarity data, and interactive graphical visualizations.

This paper is organized as follows: Section 2 provides the background necessary to understand CoMoTo and describes related work, Section 3 is a tour of CoMoTo's interface and data flow, Section 4 is a technical description of CoMoTo's implementation, Section 5 briefly describes two use cases for CoMoTo's analysis, and Section 6 suggests future work.

2. BACKGROUND

Numerous similarity detection systems have been described in research literature. A good survey describing software similarity detection has been written by Roy and Cordy [11]. Most modern similarity detection systems begin by converting the source code into a token sequence. Similarity detection is then accomplished by measuring the similarity of sequences. How this measurement is performed is the main differentiator among these systems. YAP [15], JPlag [5] and the Measure of Similarity System (MOSS) [12] from Stanford all work in this manner. MOSS is perhaps the most widely used such system and is the tool we began using in our large introductory data structures course approximately 2 years ago.

There have been several studies published regarding the use of similarity detection to discover software plagiarism. Prechelt et al [4] report on their experience using JPlag. In this study, JPlag correctly identified more than 90 percent of the known plagiarism instances while demonstrating acceptable run-time efficiency. The authors also detail a taxonomy of plagiarism techniques they encountered and the relative success of each. In another recent study, Wise [14] compares the relative merits of Plague and YAP. While slower, YAP was seen as easier to use and arguably more accurate in detecting plagiarised code. To our knowledge, there has been no prior published study regarding the efficacy of MOSS. Moreover, our study differs from the others mentioned here because we are interested in explaining code similarity, whatever its cause, rather than focussing strictly on plagiarism.

MOSS employs a local document fingerprinting algorithm called *winnowing*, the details of which can be found in [12]. Offered as service on the Web, the input to MOSS is multiple sets of source code with each set corresponding to a distinct program (submitted by a student in our case). MOSS proceeds to measure the similarity between each program, generating a similarity score between 0 and 100 for each pair. In our usage, the top 500 most similar pairs are reported along with highlighting of the similar portions of code. As one would expect, MOSS is sophisticated and ignores whitespace and naming differences. It also allows for the blocking of certain parts of the code from comparison. This is useful if the submitted source code contains boilerplate or instructor-provided code.

We describe the primary course for which CoMoTo was designed because the use cases of Section 4 are motivated in part by the course structure. At our university, CS2 is composed of a mini-course in C++, followed by an intensive and classical treatment of data structures. Enrollment varies between 200 and 300 students per semester. Gender balance in the course is typically 15% female, and nearly all students are traditional aged. The course is required for Computer Science and Computer Engineering students, and as such, the enrollment is typically $\frac{1}{3}$ CS majors, $\frac{1}{3}$ Computer Engineering majors and $\frac{1}{3}$ other (though almost all are mathematical sciences or engineering).

In a typical semester, 30% of a student's grade is determined by his/her completion of programming assignments, of which 7 are assigned. In general, the assignments are considered challenging. Students are encouraged to work in small groups (policy varies across semesters), and they are required to cite their collaborators.

3. INTERFACE AND DATA FLOW

CoMoTo enhances the basic source code comparison features of Moss in several ways. A diagram summarizing the data flow of the system is found in Figure 1.

Through its web interface, CoMoTo facilitates simple data import from the Subversion [13] version control system, which we use to manage student programming assignment archiving and submission. This integration is a marked improvement from the original interface to MOSS, and it can easily be reconfigured to handle a variety of code collection mechanisms.

In CoMoTo, all data associated with MOSS analysis is accumulated, stored, and maintained in the tool. Code submissions are saved in CoMoTo until they are explicitly deleted, eliminating the need for repeated source code import over many analyses. This simplifies comparison of current student submissions for a given assignment with student submissions for a same or similar assignment from past semesters of a given course. Likewise, the pairwise similarity measures and highlighted code provided by MOSS are also saved in CoMoTo until deleted by the user, allowing for accurate record keeping across many offerings of a course.

CoMoTo projects and filters the MOSS analysis in many different ways. Initially, the tool parses the output of the MOSS engine partitioning the pairwise results into a report that indicates whether matching programs were submitted during the same semester, between different semesters, or optionally, whether a program matches an instructor supplied solution. These partitioned results can be filtered even farther. For example, users can limit the display to only show source code matches where at least one of the submissions comes from the current semester, limit the display to only show matches which meet a minimum threshold for similarity, or limit the display to only show a specific number of matches for any given student. In addition, CoMoTo can optionally parse student names from specific meta-data files which instruct the tool to regard certain groups of students as partners when running the analysis. This allows instructors to effectively monitor collaboration when limited group work is acceptable. Figure 2 shows the analysis page corresponding to a single student. Of particular interest is the list at the bottom of the screen of all significant similarity measures associated with that student. At a glance, an instructor can tell whether the student consistently works with the same partner(s) across multiple MPs, or whether he/she violates course policy, for example.

Finally, all CoMoTo's analyses are sent to an interactive graphical visualization that allows the instructor to navigate the data from many different perspectives and configurations. A rudimentary screenshot is given in Figure 3.

We require that student work be submitted in one of two possible formats, either via file archive or from a Subversion repository. In both cases, we prescribe that the student code must be arranged so that each student has a separate directory at the root of the archive or repository, and that within that student directory there is a directory for each programming assignment. Instructor-provided code should be formatted similarly. Beyond that, there are no restrictions on the structure of the student code.

4. IMPLEMENTATION

CoMoTo is a web-based tool, which makes it available to anyone with a web browser (in contrast to Moss which is

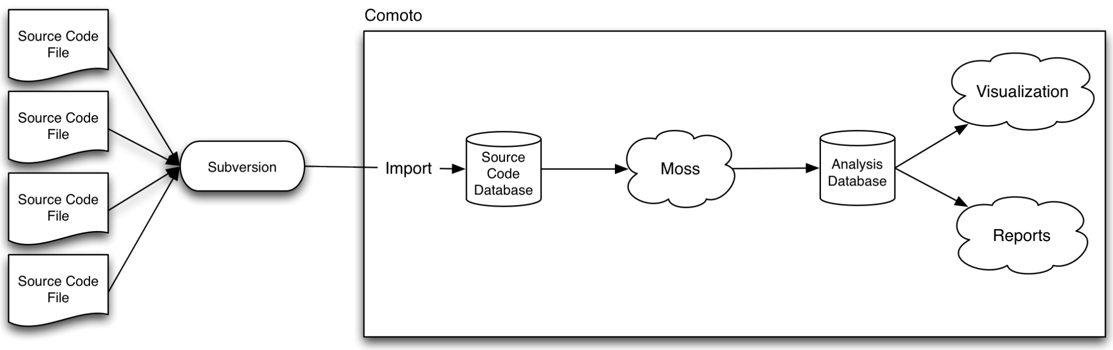


Figure 1: *CoMoTo data flow diagram.*

designed for terminal use). We built CoMoTo using the Pylons [8] MVC framework which allowed us to incorporate many other Python libraries into the tool to enhance its feature set. We utilize Pygments [6] for source code syntax highlighting, pygraphviz [7] for generating graphical visualizations of our analysis data, pysvn [9] for integration with Subversion , matplotlib [2] for generating statistical charts of our analysis data and python-ldap [10] for integration with the university directory. We utilize the MySQL [3] database engine to store and query data. For privacy reasons, we have chosen to license our own copy of Moss for use with CoMoTo so we can run all analysis locally and ameliorate concerns about confidential student information leaving university resources. Lastly, CoMoTo provides a XML-RPC based API allowing others to write their own tools or front-ends that utilize the CoMoTo functionality and/or data. We are currently developing an interactive Java-based visualization of the CoMoTo data which utilizes this API to interact with the tool.

In order to deploy a copy of CoMoTo, a dedicated server is recommended due to the high resource usage of the tool when performing analysis or generating visualizations of the data. We currently have our production copy of CoMoTo deployed on a virtual machine, configured with a four core Intel Xeon processor and four gigabytes of memory. During normal usage, the database back-end uses about a quarter of our system's main memory, and one of the four cores is at full utilization. Memory requirements are pushed when generating visualizations of our data using Graphviz. Our machine is running CentOS 5.5 and CoMoTo runs on top of the stock Python 2.4 distribution that comes bundled with the operating system. We installed several libraries as mentioned above to support the tool. We serve CoMoTo using a Paste HTTP server proxied behind a public facing Apache HTTP server. The Apache server is used to serve all static content and provide university-integrated authentication, while the Paste server serves up all the dynamic content and runs the core of the tool.

5. USE CASES

We anticipate that CoMoTo will be used to answer research questions about the form and extent of student collaboration in programming courses, and to investigate the *differences* between student code, but we would be remiss if we avoided mentioning its utility in detecting and processing infractions of academic integrity. This section briefly discusses two case studies that begin to tell the story of the current and future value of the tool. These examples should be viewed as a reflective proof-of-concept, rather than rigorous research.

5.1 Plagiarism Detection

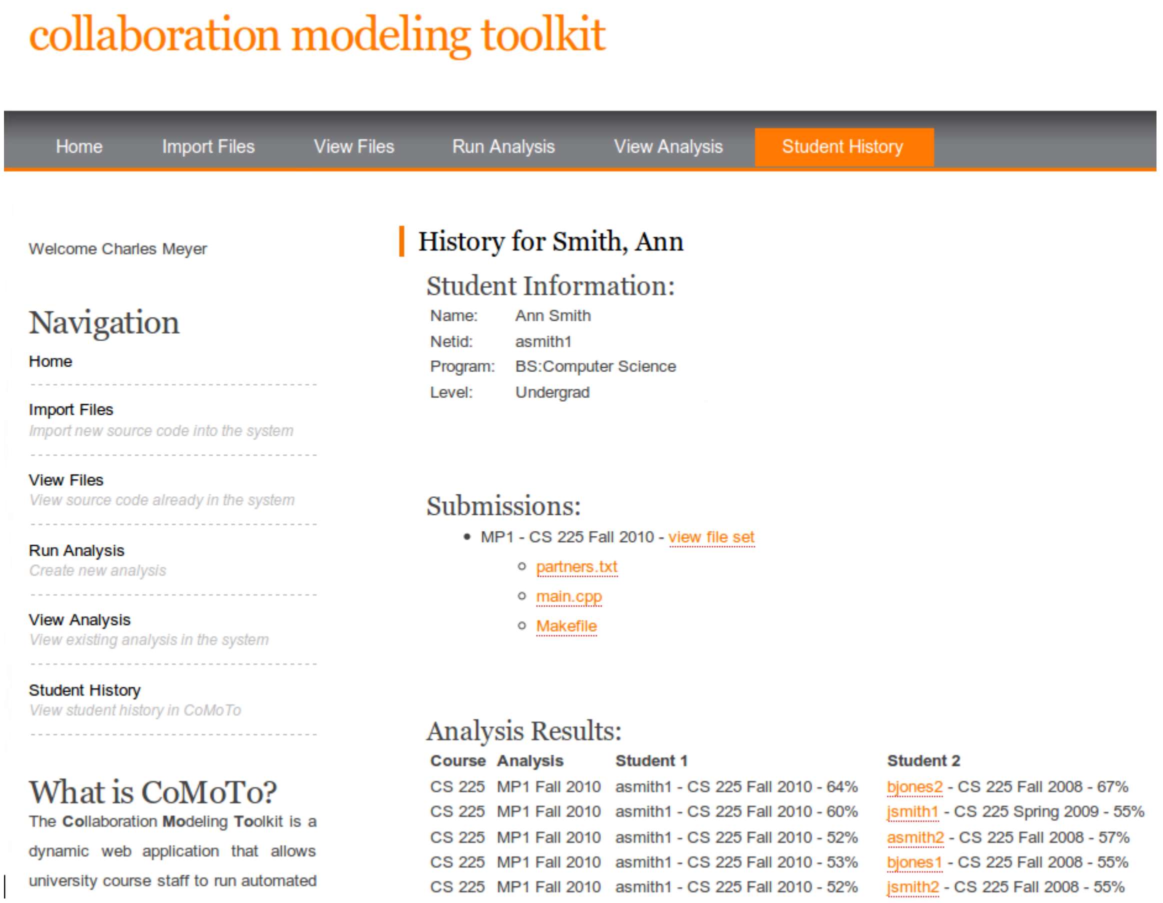
It is a sad truth that too many of our students do not follow our course and departmental policies on appropriate collaboration. In any given semester, approximately 25% of CS2 students are flagged by CoMoTo (via MOSS) for infractions of academic integrity due to plagiarism. Every flagged code submission is further scrutinized for some piece of convincing evidence, or a "tell", which invariably exists. (In the very rare case that a tell cannot be found, an accusation is not made–that is, we do not solely rely on CoMoTo to determine plagiarism.) In a class of 100s of students, this translates to dozens of accusations, which in turn translates into a similar number of 20 minute instructor-student conversations. It is in these conversations where CoMoTo has a profoundly positive effect on our course.

Early in a typical conversation, after explaining the charges of plagiarism, our CS2 instructor shows CoMoTo's graphical visualization of the similarity measures. Invariably, when a student identifies himself/herself with a node in the graph that is connected to others in a manner that is disallowed in course policy, a confession and explanation are forthcoming. Once that step is out of the way, the meeting moves past the infraction itself and the focus changes to advising and assisting the student toward a successful completion of the course. In the past four semesters, only 5 students (of over 100) have persisted in denying that they cheated, and of those 5, *none* have formally protested the penalty assessed against them (typically the loss of a letter grade in the course).

While it is unfortunate to speak with students in the context of cheating, anecdotally, the effect of the instructor-student interaction is very positive.

5.2 Coding Style

The very first time we applied CoMoTo to a student programming assignment in CS2, we were shocked at how *dis*similar student code really is. The assignment in question was a



The **Co**llaboration **Mo**deling **To**olkit is a dynamic web application that allows university course staff to run automated

History for Smith, Ann

Student Information:

Name:	Ann Smith
Netid:	asmith1
Program:	BS:Computer Science
Level:	Undergrad

Submissions:

- MP1 - CS 225 Fall 2010 - view file set
  - partners.txt
  - main.cpp
  - Makefile

Analysis Results:

Course	Analysis	Student 1	Student 2
CS 225	MP1 Fall 2010	asmith1 - CS 225 Fall 2010 - 64%	bjones2 - CS 225 Fall 2008 - 67%
CS 225	MP1 Fall 2010	asmith1 - CS 225 Fall 2010 - 60%	jsmith1 - CS 225 Spring 2009 - 55%
CS 225	MP1 Fall 2010	asmith1 - CS 225 Fall 2010 - 52%	asmith2 - CS 225 Fall 2008 - 57%
CS 225	MP1 Fall 2010	asmith1 - CS 225 Fall 2010 - 53%	bjones1 - CS 225 Fall 2008 - 55%
CS 225	MP1 Fall 2010	asmith1 - CS 225 Fall 2010 - 52%	jsmith2 - CS 225 Fall 2008 - 55%

Figure 2: *Screen shot of student data.*

very simple image manipulation task, wherein the students were asked to rotate an image by 180 degrees. They used an image library to read the bitmap data into a two dimensional array of pixels. Beyond that, the solution required a pair of nested `for` loops, some assignment statements to swap pixels, and a call to a library function to write the data into a new bitmap file. We expected the CoMoTo analysis to cluster all the students into a complete graph, because we were certain they would all write the same 20 lines of code. In fact that was not at all the case. Perhaps due to a fairly loose collaboration policy, there were a few small cliques in the graph, but most notably, there were very few edges in the graph at all, for a reasonable similarity measure (30%).

From a pedagogic standpoint, the lack of similarity among solutions indicates that even after taking CS1 the students did not produce idiomatic code in response to a common programming problem. It has been argued, by Kernighan and Ritchie [1] among others, that writing idiomatic code is important from a software engineering standpoint. Employing shared idioms makes source code more easily understood and maintained, increasing programmer efficiency and reducing the number of bugs introduced later in the software life-cycle. Even though our problem was simple and the code was short, very few students had significantly matched solutions. This result suggests an area ripe for improvement in our early courses, and in this case, CoMoTo would be a useful tool for assessing the effectiveness of new teaching strategies designed to give students a more common approach to writing fundamental constructs.

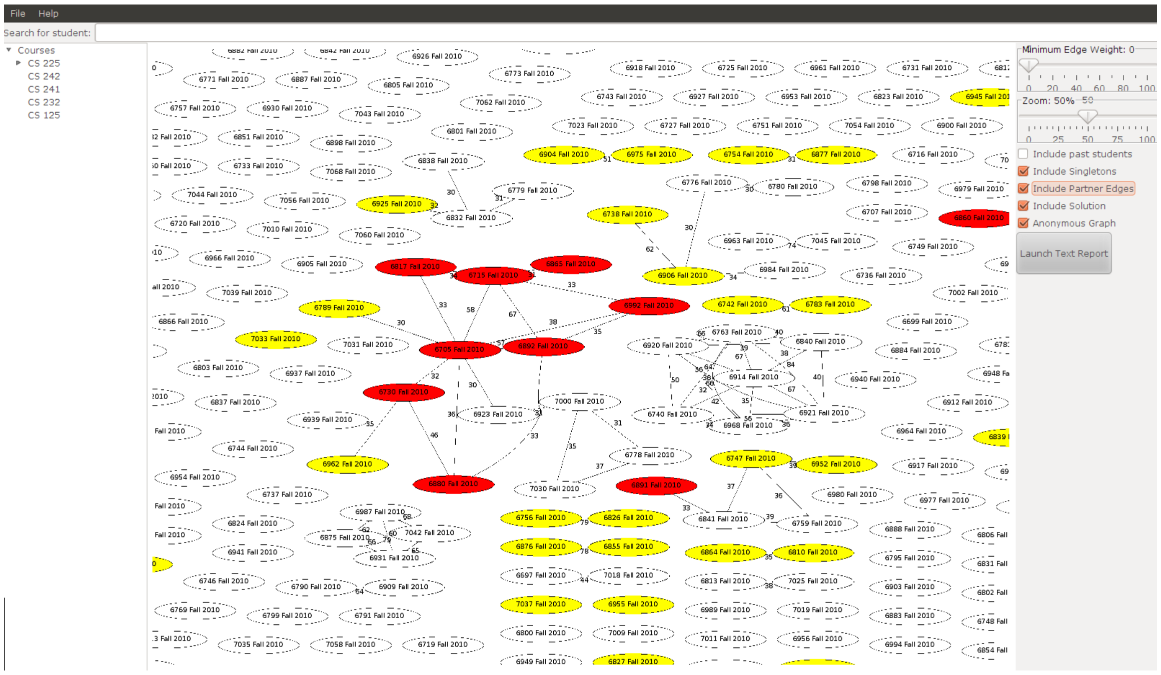
6. FUTURE WORK

CoMoTo is now tried and tested enough to be deployed broadly, and to be made available to others who may find it useful. We are ready to discuss the logistics of adapting the tool for a general audience and are excited about its potential.

Our next research objective is to prescribe an experimental methodology that employs CoMoTo to answer questions about student collaboration patterns with statistical reliability and validity. To do this we intend 1) to consult with statisticians so we understand the inferences we can reasonably make, and 2) to incorporate measures of student performance into the tool so we can measure the effectiveness of different collaboration models. Once the methodology is established, we will be in a position to pose and answer a host of questions about student code development and collaboration practices.

7. REFERENCES

[1] B. W. Kernighan and R. Pike. *The Practice of Programming.* Addison-Wesley Inc, 1999.
[2] matplotlib. http://matplotlib.sourceforge.net/.
[3] MySQL. http://www.mysql.com/.



Minimum Edge Weight: 0

Zoom: 50%

- [ ] Include past students
- [✓] Include Singletons
- [✓] Include Partner Edges
- [✓] Include Solution
- [✓] Anonymous Graph

Figure 3: *Screen shot of interactive visualization page.*

[4] L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with jplag. 8(11):1016–1038, 2002.

[5] L. Prechelt, G. Malpohl, and M. Phlippsen. Jplag: Finding plagiarisms among a set of programs. Technical report, 2000.

[6] Pygments. http://pygments.org/.

[7] pygraphviz. http://networkx.lanl.gov/pygraphviz/.

[8] Pylons. http://pylonshq.com/.

[9] pysvn. http://pysvn.tigris.org/.

[10] python ldap. http://www.python-ldap.org/.

[11] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEENÕS UNIVERSITY*, 115, 2007.

[12] S. Schleimer. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data 2003*, pages 76–85. ACM Press, 2003.

[13] Subversion. http://subversion.apache.org/.

[14] M. J. Wise. Detection of similarities in student programs: Yap'ing may be preferable to plague'ing. In *SIGCSE '92: Proceedings of the twenty-third SIGCSE technical symposium on Computer science education*, pages 268–271, New York, NY, USA, 1992. ACM.

[15] M. J. Wise. Yap3: Improved detection of similarities in computer program and other texts. In *SIGCSEB: SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education*, pages 130–134. ACM Press, 1996.