# Educating computer programming students about plagiarism through use of a code similarity detection tool

Tri Le, Angela Carbone
Office of the Pro-vice chancellor Learning and Teaching
Monash University
Melbourne, Australia
tri.lenguyen@monash.edu
angela.carbone@monash.edu

Judy Sheard, Margot Schuhmacher
Faculty of Information Technology
Monash University
Melbourne, Australia
judy.sheard@monash.edu
margot.schuhmacher@monash.edu

Michael de Raadt
Development Manager
Moodle Pty Ltd
Perth, Australia
michaeld@moodle.com

Chris Johnson
Faculty of Information Technology
Australian National University
Canberra, Australia
chris.johnson@anu.edu.au

*Abstract*— **Technology empowers students but can also entice them to plagiarise. To tackle this problem, plagiarism detection tools are especially useful, not only in popular thinking as a deterrent for students, but also as an educational tool to raise students' awareness of the offence and to improve their academic skills. Commercial text matching tools (e.g. Turnitin) are at a high level of maturity. These tools offer the ability to interact with students, making them suitable for an educational objective. Additionally, they can be readily integrated into learning environments enabling uniform application at an institutional level. On the other hand, computer source code matching tools, despite their successful detection performance, are mostly used as standalone tools that are difficult to adopt at an institutional level. The research presented in this paper describes the trial and evaluation of a tool that is seamlessly integrated into the Moodle virtual learning environment. The tool provides code similarity scanning capability within Moodle so that institutions using this learning environment could apply this tool easily at an enterprise level. Additionally, the educational aspects available in text matching tools have been added into the tool capability. The tool relies on two popular code matching services, MOSS and JPlag, as underlying engines to provide good code similarity scanning performance. The evaluation of the tool from both academics' and students' perspectives indicates a considerable level of interest in using the tool, and supports the suitability of this tool for wider institutional adoption in the computing education community.**

***Keywords—academic integrity; plagiarism detection tool; computer source code***

## I. INTRODUCTION

The possibility to easily access an abundance of information has provided unparalleled learning opportunities for students but also presents them with a temptation to plagiarise that is sometimes difficult to resist. At least, when exposed to such an environment, students are very susceptible to the pitfall of unintentional plagiarism, if they are not equipped with proper academic skills. This problem is compounded by the increasingly diverse background of tertiary students, who have different understanding and attitude toward this form of academic violation. In fact, studies have found that a worrisome number of students admitted to plagiarising or have a relaxed attitude toward this offence [1, 2]. The problem of plagiarism does not exclude IT students, who are generally much more proficient at making use of technology in their study than students in other fields. Numerous reports have affirmed the problem of cheating among IT students [3-5], and underlined the need for better education on academic integrity and more effective plagiarism detection methods to deter IT students from committing such offences.

Unfortunately, lessons on plagiarism and proper citation often have limited effect since they are typically given at the start of courses and not often deemed to be relevant by students until they begin their assignments [6]. In addition, it might not be feasible for academics teaching large classes to manually check for students' improper use of materials and provide formative feedback thus ensuring that only deliberate offences are penalised. In this situation, plagiarism detection tools prove advantageous, not only by offering an efficient deterrent to students, but also by playing the role of an educational tool to provide an "experimental learning environment" [7]. In this environment, students can receive formative feedback to improve their paraphrasing and citation skills. Numerous case studies on the benefits of text matching tools, with prevalent attention given to Turnitin, have proven the value of their support in maintaining academic integrity [8-10].

However, even with an abundance of code matching tools available, there is a dearth of studies on the adoption and the benefits of these code matching tools in improving academic integrity in the programming field. The reasons for this shortage might be due to their lack of convenience, not to mention their poor integration into popular learning environments, which is especially important for institutional adoption. With these weaknesses, the current code matching tools are just detection engines. They can only be used as standalone tools on individual machines on an ad hoc basis, but cannot be conveniently applied institution-wide to provide students with an environment to learn about academic integrity, as well as a deterrent against this form of cheating.

This research attempts to facilitate the adoption of code matching tools in academic institutions by building a tool which integrates two popular source code similarity scanning engines, MOSS and JPlag, into Moodle [11] – a widely used virtual learning environment. Drawing on results from our previous research that investigated academic practices of assessing programming assignments, experiences with code matching tools and their opinions on a prototype of the tool, this research developed a fully featured version of the tool with a special focus on institutional adoption and educational intent. Evaluation of the tool in a programming unit with both academics and students indicated a considerable level of interest from academics, yet some divergence in students' opinion. Our analysis supports the potential of the tool for plagiarism education in computer source code and the use of the tool for wider institutional adoption in the computing education community.

The next section of the paper provides some background to the adoption of source code plagiarism detectors, perceived benefits of detection and punishment, as well as education and prevention, along with the results of our past research. The third section describes the tool design with the view of making it appropriate for educational purposes and institution-wide adoption. The fourth section presents our approach to evaluate the tool suitability for academic use. The fifth section presents the result of this research and the final one concludes the findings, with planned further work.

## II. Background

### A. *Plagiarism detectors and their adoption*

Almost any type of work can be plagiarised, from essays (the most prevalent type of assignments) to computer programs or more creative artefacts such as artworks. However, the types of assessable artefacts that have been comprehensively researched and supported by a wide range of plagiarism detection tools are natural language (i.e. essays) and computer languages (i.e. source code). While most plagiarism detectors in natural languages often search for copied chunks of text, source code plagiarism detectors need to look beyond the common code refactoring that students can make to disguise their offence. Some of these disguising techniques are easy to achieve with the assistance of modern programming environments. Nevertheless, due to the strict syntax of programming languages, source code plagiarism detectors are considered by some as having better detection than their natural language counterparts [12]. However, current code matching tools can only compare pair-wise within a cohort and not over large repositories or the Internet, as popular text matching tools do. This drawback is defended by the argument that different pieces of computer source code could not be blindly put together as free text can in essays. Moreover, because of rigid syntaxes and the abundance of code available on the Internet for common problems, it is very probable that any specific, small piece of code could be found somewhere on the Internet with just minor differences in indentation and variable names. Therefore, a student assignment would only be suspected as being plagiarised when it is similar in large portions to another program.

As plagiarism incidents increase, institutions, alerted to this phenomenon, have progressively implemented organisation-wide procedures backed by the adoption of new technology, in attempt to ensure academic integrity. Since essays are the prevalent type of assignments across disciplines, it is not surprising that text matching tools, especially Turnitin, receive particular interest. In fact, data from the Turnitin site point out that the system has been used by 90% of institutions in UK, and more broadly, by around 10,000 institutions over the world. Contrary to the widespread adoption of Turnitin, some reports suggested that the use of code matching tools was disproportionate to the extent of plagiarism in programming assignments. A survey among institutions in the UK showed that only one quarter of them adopted code matching tools [13]. Separate reports also indicated that several Australian institutions such as RMIT applied code matching tools within their institution process [3], but there has been no overall statistics found on the adoption of such tools. The disparity between the adoption of text and code matching tools is further reflected by a profusion of research and case studies around Turnitin at institutions, compared to only a small number of reports on the use of code matching tools.

### B. *Plagiarism detectors as a deterrent for plagiarism*

Plagiarism detectors are often used solely for punitive purposes to combat the explosion of materials on the Internet and the proliferation of "essay mills". Their effectiveness as a deterrent to students varied across different studies, for both essays and source code. Most reports showed that the adoption of Turnitin has significantly reduced the rate of plagiarism. For example, Martin [14] reported a study over five consecutive semesters using Turnitin to scan students' assignments and imposed penalty marks for different degrees of actual plagiarism found. The average plagiarism rate detected dropped dramatically from 6.31% in the first semester to 1.5% in the last one, as students developed a stronger belief that their offence would be caught. Another similar study reported by Bilic-Zulle et al. [15] found that a group of students who were informed that their work will be examined by plagiarism detection software plagiarised significantly less than two other groups of students who were just given an explanation about academic integrity or asked to write an essay on the topic (2% compared to 17% and 21% respectively). In computer source code, the deterrent effect of plagiarism detection has also been confirmed. Bowyer and Hall [16] reported that plagiarism reduced considerably over two semesters using MOSS. In the first semester, 10 out of 80 students plagiarised, compared to only 9 out of 140 students in the second. However, not all studies supported these findings. A study by Youmans [17], found that there were almost no differences in plagiarism behaviour observed between a group of students who were warned about the use of Turnitin and a control group who were not. Later in their study, Bowyer and Hall [16] found that plagiarism in their course shifted to another form undetectable by software, such as hiring an outsider to do the assignment. However, for text matching tools, the number of studies that reported improvements on academic integrity outnumbered those that questioned their effectiveness, while studies in source code matching tools are still very limited in number.

### *C. Plagiarism detectors as an educational tool to promote academic integrity*

Dick et al. [18] argued that formative approaches to combating the plagiarism problem are always preferable to punitive approaches, from the learning and teaching perspective. Plagiarism detection tools are gradually being used for a more student-friendly educational purpose of raising students' awareness of academic integrity. However, no research was found on the educational effects of source code matching tools, perhaps due to their limitation as standalone tools. However, a lot of research has been devoted to the impact that formative feedback, given by text matching tools, has had on enhancing academic integrity awareness and on the citation skills among students. In a study by Barrett and Malcolm [6], Turnitin was used to give individual feedback to a large group of students in different courses on the originality of their work. As a result of this feedback, the number of students having an actual plagiarism rate higher than 15% fell from 26% in the first assignment to 3% in the second one. Stappenbelt and Rowles [7] emphasised the necessity of providing students with an "experimental learning environment" provided by Turnitin where students can pre-check their work before submission. An impressive reduction of 79% of average similarity rate in students' first draft was observed after only one assignment. Besides the improvement in the raw plagiarism rate, Rees and Emerson [8] found that, feedback provided by Turnitin helped students enhance other important academic writing skills, which include improvement in citation errors, better paraphrasing, and synthesis from more diverse referencing sources.

### *D. Our preliminary research*

The lack of usability and integration of source code matching tools into a learning environment constitutes a major barrier to their wide adoption at the institutional level. To promote their adoption, not only do these tools have to be made more straightforward to use, the process of using them should be streamlined to fit the assessment practices of academics within their virtual learning environment. This problem was tackled by systematically investigating the benefits that these source-code matching tools could offer to academics and how they could best be utilised to ensure academic integrity. Our preliminary study in [19] undertook interviews with six academics teaching programming to acquire an insight into their assessment practices and determine the value of source code matching tools in raising students' awareness of plagiarism and ensuring academic integrity. A prototype of the tool was built, which acted as a bridge between Moodle and the selected plagiarism detection engines, MOSS and JPlag, with additional educational enhancements. Although the native report interface of JPlag and MOSS was retained, the tool allows academics to publish a de-identified version of the similarity report to students as formative feedback. This prototype was evaluated by both academics' and students' perspectives by trialling the tool on an assignment in a foundation programming unit. The generated similarity report was given to academics and students in a second interview to elicit their comments and opinions on the tool. The tool received considerable interest from participating academics, who appreciated its ease of use and efficiency, especially in large classes with multiple markers, allowing easy cross checking between different groups of students. However, the academics considered that the use of the tool may generate a significant level of anxiety in the students. Rather than preventing students from excessive collaboration, it may also hinder constructive exchanges of ideas amongst them. An important feature of the prototype is the ability to publish a de-identified similarity report to students to educate them about academic integrity in programming.

## III. Design of the tool

Preliminary evaluation of a prototype tool showed promising benefits for student assessment. The prototype minimised the overhead of using code matching services, and streamlined the use of code matching services within the assessment process. These improvements are important to promote the adoption of code matching tools within institutions. Further development of the prototype resulted in a complete and fully-featured tool, with an educational purpose suitable for institutional use. This section outlines the development considerations of this tool and its features.

### *A. Special development consideration*

As mentioned above the tool was built upon the previous prototype, but with a focus on its educational purpose and suitability for institutional-wide adoption. Issues and comments raised by academics from the initial prototype were carefully considered to reduce the potential barriers to institutional adoption. The issues and resolution include the following:

- **Institutional policy**: organisations have different requirements in terms of sending students' work to external services. Factors to consider include students' confidentiality, copyright of students' work and the storage of these works on external servers. Addressing this issue involved consultation with the university's academic integrity services.
- **Usability**: usability is a major factor in the wide adoption of the tool. It is important that the interface is sufficiently intuitive and the adoption overhead is minimal, so that the tool does not discourage users at first time of use. Previous evaluation results from

academics and consultation with a university usability specialist were used to enhance the tool's ease of use.

- **Robustness**: stability is a very important criterion for institutional adoption. Communication with external services could introduce a point of failure, requiring detailed testing. Along with in-house testing, the tool was released to the Moodle community to obtain feedback and bug reports.
- **Seamless integration into the virtual learning environment**: Moodle facilitates the integration of plagiarism detection services via a plagiarism plugin API. Thus, code matching services can be integrated seamlessly within the Moodle assignment module. Feedback from our preliminary work suggested additional features for closer integration with the assessment process including the ability to mark suspicious works and to inform students on the unusual similarities of their work.
- **Educational features**: one of the main purposes of the tool is to play a role in raising students' awareness of academic integrity. A draft submission feature and an improved publication of the students' report were implemented.

## B. *Overall operation*

The tool acts as an intermediary between the learning environment and plagiarism scanning services to provide a transparent and effortless scanning of programming assignments in Moodle (Fig. 1). After some simple configuration to enable similarity scanning for an assignment, the tool automatically extracts, organises and de-identifies the data. The data are then packed into the required format before being sent to the external system. The tool ensures students' work is anonymised by masking occurrences of students' name and ID. This protects their privacy, which is strictly required by many institutions. This scanning process can be manually triggered at anytime or scheduled in advance by the academics. Once the scanning has finished, the report is available to all staff involved in the Moodle course (e.g. subject). In addition, the academic can control whether a de-identified version of the report is made available to students as a form of feedback.

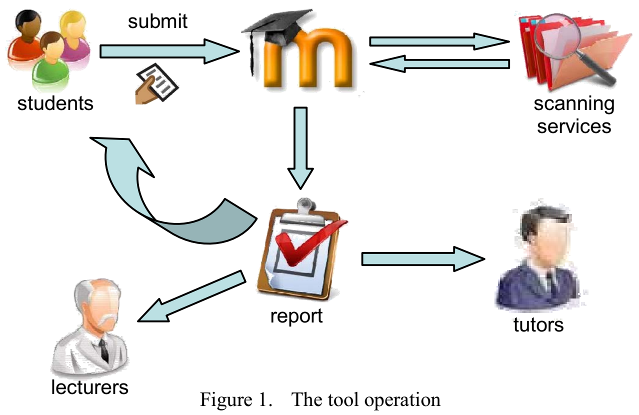


Figure 1. The tool operation

## C. *Features of the tool*

### 1) *Standard installation and setting*

The tool complies with the standard Moodle plugin specification ensuring a familiar deployment process for Moodle administrators. Minimal configuration effort is required to provide a MOSS and JPlag account. Furthermore, the administrator has the ability to enable the plugin at the course level, so as to not confuse academics with a feature that is irrelevant to them in non-programming courses.

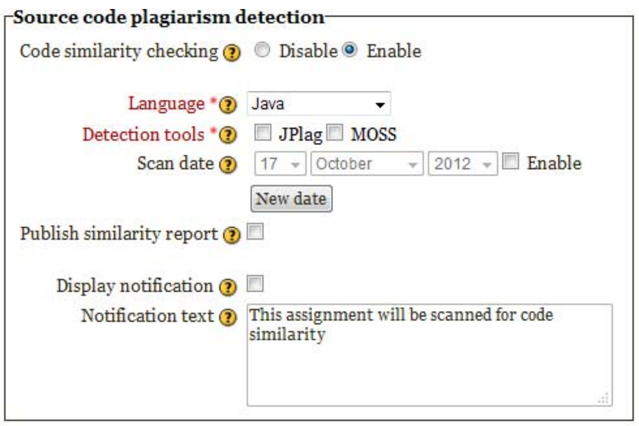
### 2) *Simple assignment configuration*

Academics familiar with Moodle are easily able to configure an assignment for code similarity checking as the assignment creation process remains the same. Additional configuration settings are depicted in Fig. 2. These include the ability to select the programming language, the service (JPlag and MOSS) to use and set scanning dates. Academics can also display a notification for students and grant them access to a de-identified version of the similarity report.

### 3) *Similarity report*

Knowing that the main overhead of using the tool for academics is the time spent examining the similarity report, many features were devoted to making the navigation within the report, and the filtering of suspicious similarity, simple and convenient.

The report was integrated within the interface of Moodle to provide a consistent user experience (Fig. 3). This new interface combines the advantages of both services identified in the evaluation of our previous prototype. In addition, it provides an interactive filtering feature. The report comprises three elements: the display and filter option box, the statistics histogram and the similarity table. The options box allows teachers to filter students according to their similarity rates and view the report with different styles of presentation. A histogram showing the distribution of similarity rates across the whole class helps academics to determine the suspicious similarity threshold for the assignment, and view specific pairs of assignments having a similarity rate within that specified range. The similarity table compares students' work pair-wise. As academic participants in our last study expressed diverse preferences with regard to the tabular form of JPlag and the simple list of MOSS, these reporting styles can be switched in


Source code plagiarism detection

| | |
|---|---|
| Code similarity checking | Disable [✓ Enable] |
| Language * | Java |
| Detection tools * | ☐ JPlag ☐ MOSS |
| Scan date | 17 October 2012 ☐ Enable |
| | New date |
| Publish similarity report | ☐ |
| Display notification | ☐ |
| Notification text | This assignment will be scanned for code similarity |

Figure 2. Similarity scanning configuration

the new report interface. The tabular report of JPlag groups all the students similar to one student in a row, as opposed to the simple list from MOSS, which displays the pairs in order of similarity.

The comparison interface is similar to that of JPlag and MOSS (Fig. 4), which places a pair of submission side-by-side and marks the portions of similar code with the same colour. In addition, this page allows academics to mark a student who is identified as having suspicious similarity. Students who are marked are highlighted on the similarity report and the grading page. In addition, if the settings permit, these students will be informed of the similarities when they see their assignment result.

*4) Educational features*

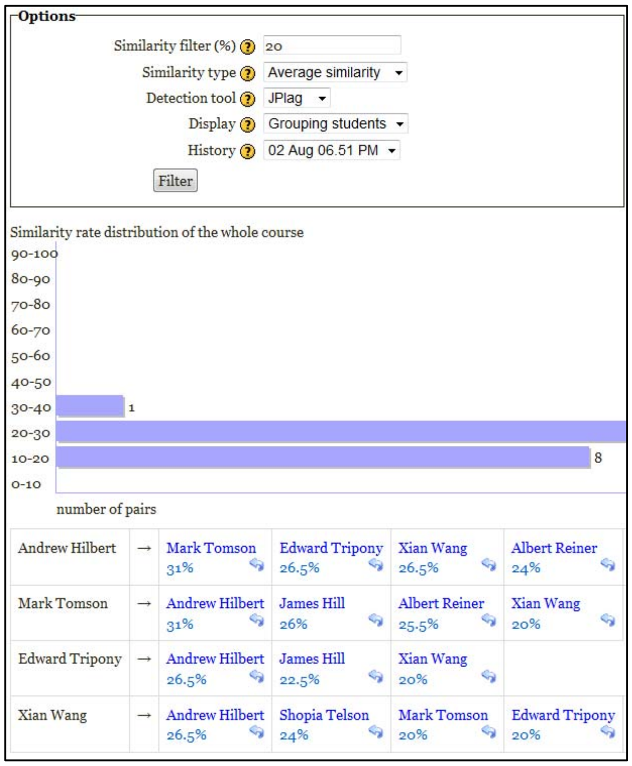
Giving feedback to students and providing them with an "experimental learning environment" are the innovative features of Turnitin that current code matching services cannot provide when used as a standalone product. Since the tool is designed for an educational rather than a punitive purpose, this new version of the tool also supports draft submissions as well as the ability to publish the similarity report to students as feedback. The fundamental difference between code matching tools and their text counterparts is that code matching tools only make comparisons between submissions in the same cohort, whereas text matching tools can typically make comparisons with materials on the Internet. Therefore, the way a draft submission works for programming assignments is also different. The tool developed allows academics to schedule multiple scanning times or to manually trigger the scanning multiple times before the final submission date. In this way, students can see the reported similarities and make the necessary modification before the due date.

As the history across different times of scanning could provide interesting information to academics, all students' code and scanning results are retained for each scan performed. The history is displayed in the form of a chart, where teachers can click on the bars to display students' code at a past scanning time. This history can only be viewed by academics, and is not accessible to students.

## IV. Research Approach

The tool was evaluated by academics and students to determine its suitability for academic assessment practices and as an educational tool. An online survey was used to gather feedback from the participants once they had an opportunity to experiment with the tool. Each participant was granted access to a "sandbox" so they could trial the tool at their convenience, before responding to the questionnaire. The process of trialling the tool with academics was completely separated from that with students.

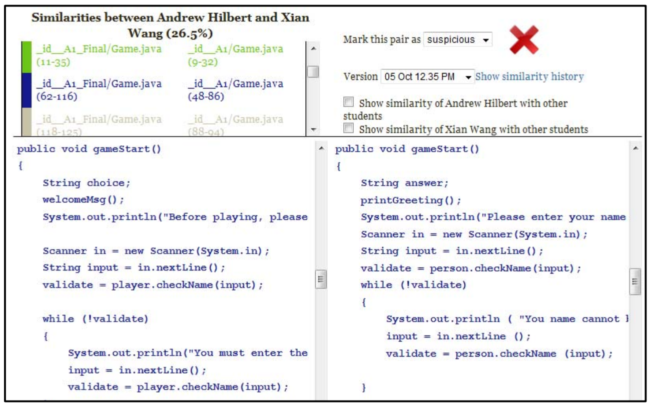
For academics, a simulation of a real assignment submission was carried out with each participant. This process required them to set up an assignment submission with code similarity scanning enabled. Then the researcher submitted real de-identified code from a previous year's assignment. After the submission finished, the academic was notified to trigger the scanning, examine the similarity report and answer the questionnaire. More than thirty academics teaching programming across an Australian multi-campus University were invited to participate in the research. This was done by consulting the programming unit list and emailing an invitation to the lecturers. A separate course on our Moodle sandbox server was created for each academic who agreed to participate, with detailed instructions on how to login and setup an assignment submission. However, no further help regarding the use of the tool was provided, other than the context sensitive help available on the site and references to the official documentation. This was intended to give participants an authentic first-time experience of using the tool and to acquire untainted information on the difficulty of using it through their feedback.



Similarity filter (%): 20
Similarity type: Average similarity
Detection tool: JPlag
Display: Grouping students
History: 02 Aug 06.51 PM

| | | | | | |
|---|---|---|---|---|---|
| Andrew Hilbert | → | Mark Tomson 31% | Edward Tripony 26.5% | Xian Wang 26.5% | Albert Reiner 24% |
| Mark Tomson | → | Andrew Hilbert 31% | James Hill 26% | Albert Reiner 25.5% | Xian Wang 20% |
| Edward Tripony | → | Andrew Hilbert 26.5% | James Hill 22.5% | Xian Wang 20% | |
| Xian Wang | → | Andrew Hilbert 26.5% | Shopia Telson 24% | Mark Tomson 20% | Edward Tripony 20% |

Figure 3. The report view


Similarities between Andrew Hilbert and Xian Wang (26.5%)

| | |
|---|---|
| _id__A1_Final/Game.java (11-35) | _id__A1/Game.java (9-32) |
| _id__A1_Final/Game.java (62-116) | _id__A1/Game.java (48-86) |
| _id__A1_Final/Game.java (118-125) | _id__A1/Game.java (88-94) |

Mark this pair as: suspicious
Version: 05 Oct 12.35 PM
Show similarity history
Show similarity of Andrew Hilbert with other students
Show similarity of Xian Wang with other students

```
public void gameStart()
{
    String choice;
    welcomeMsg();
    System.out.println("Before playing, please

    Scanner in = new Scanner(System.in);
    String input = in.nextLine();
    validate = player.checkName(input);

    while (!validate)
    {
        System.out.println("You must enter the
        input = in.nextLine();
        validate = player.checkName(input);
```

```
public void gameStart()
{
    String answer;
    printGreeting();
    System.out.println("Please enter your name
    Scanner in = new Scanner(System.in);
    String input = in.nextLine();
    validate = person.checkName(input);
    while (!validate)
    {
        System.out.println ( "You name cannot
        input = in.nextLine ();
        validate = person.checkName (input);

    }
```

Figure 4. The comparison view

Students went through a completely different process. A computer programming unit for 2nd year students was selected because of its suitability to the research timeframe. This unit covers object-oriented design and development, and is generally taken by student with previous programming experience. Nearly 20 students in the unit were introduced to the research in a lecture and invited to submit their current programming assignment to our sandbox. Students were assured that their confidentiality would be respected and their lecturer and tutors would not have access to the generated report.

Although the two surveys had different sets of questions for academics and students, both focused mainly on the usability and the educational benefits of the tool.

- The academic survey asked academics about the intuitiveness of the interface, the performance of the tool and the relevance of the similarity report. Other questions were related to the educational aspect of the tool, including the tool's new features which are the multiple draft submissions and the publication of the report to students.
- The student survey was designed to investigate the impacts of using the tool on students' behaviour and their concerns if the tool was adopted.

Most questions were designed using a 5 point Likert scale, but the survey also had some open-ended questions for the participants to give additional comments and opinions.

## V. Research Results

This section reports on the evaluation of our tool from both the academics' and students' perspectives. An online survey was used after participants had some hands-on experience with using the tool.

### A. *Evaluation: Academics' perspective*

#### 1) *Participants' background*

Seven academics accepted the invitation to participate in the research. As all academic participants were from the same institution, their background was quite homogenous. Most participants had never used a code matching tool before, except one who had tried a tool of this kind once previously. All academics were highly aware of the plagiarism problems in their units, with many stating that they "very often" or "quite often" encountered suspicious similarities in students' assignments. However, the time spent on detecting plagiarism varied among the participants, with only two academics stating that they spent no time or little time, and all others claiming to have spent *"some time"* to *"a moderate amount of time"*.

#### 2) *Usability of the tool*

Since usability is an important promoter for the adoption of the tool on a large scale, a significant portion of the questionnaire was dedicated to evaluate this aspect. Overall the responses indicated an appreciation for the tool's usability. Some suggestions were also provided to improve the usability of the tool.

##### a) *Tool configuration*

All participants encountered no problems in setting up the tool. They found the process straightforward and therefore needed little, if any assistance. Most participants used only the context sensitive help along with the available video guide. There was one person who only needed to read the quick context sensitive help while configuring the tool, and one other who only referred to the user guide, which was perhaps the most time-consuming resource to go through.

##### b) *The similarity report*

User experience with the reporting interface was positive. All academics appreciated the scanning performance and the content and layout of the report. Participants claimed that it did not take them much time to examine the report. Several academics placed a lot of importance on the convenience of navigation between students' code files and switching the view between students. This was noted as an extremely helpful feature especially in cases where there might be a lot of students to examine in one class, because it minimised the time taken to examine the report. An improvement suggested by academics was the need for help to be made available on the similarity report page, for those that were using the tool the first time. For example one participant stated:

> *"I had to watch the video to really understand how to interact with the report and findings, but once I did that it was easy to follow"* (A-1)

#### 3) *Improving academics' efficiency - assessing students*

All academics agreed that the tool would improve their efficiency to assess students' work. Three out of seven stated explicitly that it would improve their efficiency by a considerable extent, and the remainder said that it would help to some extent.

#### 4) *Education around academic integrity*

Academics were asked to express their beliefs about the impact and effect of the tool's features on student behaviour. Specifically, what influence would the draft submission and reporting features have on the students and their assessment? Overall, academics tended to favour the use of both of these educational features, while being cautious about the potential drawbacks they would have.

##### a) *Publication of a similarity report*

Overall academics indicated that they supported the publication of the report to students. They claimed that it would raise students' awareness on academic integrity and provide a good learning experience for the students. More specifically, four out of seven participants agreed that publishing the report to students would be beneficial (with 1 neutral and the other 2 did not answer). However, their opinion on the possible anxiety that such feature could provoke to the students diverged from each other.

##### b) *Draft submissions*

Academics also favoured the draft submissions feature, but to a lesser extent than the report publication. Three out of four academics believed that the opportunity for students to resubmit drafts for plagiarism checking was beneficial to educate students about academic integrity. However, several academics were unsure as to whether this feature would entice students to make continually more sophisticated disguises in their code after each resubmission iteration.

*5) Suitability of the tool*

One argument against the use of a source code matching tool is that, with the strict constraints on program syntax and well established solutions to many problems, source code could be similar even when written individually and not plagiarised. This is especially true with assignments that require a small program to be written. Participants were asked whether the tool would be suitable in this context. Academics generally disagreed with this argument, stating that it is likely that source code would be similar to some extent, but it should not be unusually similar.

> *"Code can be similar even if written individually but the code comparison would then show that a large proportion of the class had [the similarity less than 20%]. It is the few individuals with >50% similarity that [I] would focus on anyway."* (A-2)

However, there was one academic who agreed with this argument, saying that:

> *"We should keep in mind that the markers still have to make the judgement call… And I think it is also okay to have some code "similar" but not others… The tool should only be used as a guide for markers."* (A-3)

*6) Improvements suggested*

Academics suggested several improvements to the tool. These included:

- sending an email when the scanning has completed;
- improving the navigation within the reports; and
- generating a report synthesising students' submission history and their similarity rate across different courses. Such report would help academics to give more attention to suspicious cases.

## *B. Evaluation: Students' perspectives*

*1) Participants' background*

Thirteen students agreed to submit their current assignment to the sandbox; however, only 9 answered our survey. Since this was a second year IT unit, all participants, except one, have already had some experience in programming. Apart from one student who stated that they had been exposed to a text matching tool before, others did not have any experiences with plagiarism detectors at the time of the study.

*2) Usability of the tool*

Most students (eight out of nine) agreed that the use of the tool did not interfere with their normal submission process. Most students deemed the report to be easy to understand. They stated that the similarity produced was accurate, but that the reported similarity rate was higher than expected. Many of the students felt that the report included parts of the code that were evidently similar across the whole class, thus should not have been considered.

> *"[The tool] is not helpful for certain aspects, such as getters and setters where most people if not all have named them the same and have coded them exactly the same."* (S-1)

*3) Impacts of the tool on students' behaviour*

*a) Impacts on ethical conduct*

Unsurprisingly, most students stated that they were less likely to share part of their code if the tool was adopted. Specifically, six out of nine students would stop sharing code (and others were unsure), five would stop sharing ideas and three would be reluctant to share a solution. This result raises the concern that the use of the tool could hinder productive forms of sharing that should be actively encouraged amongst the students. Overall, most students stated that they would not change the way they worked on their assignment, except one who suggested the following.

> "*I would take greater effort to modify my source code to reduce similarity.*" (S-2)

*b) Anxiety levels*

There was some anxiety observed among students, especially the ones that had unexpectedly high similarity. One student said:

> *"I feel a little bit scared that this application will think I have cheated, as on one of the comparisons, i have a 40% similarity to another student, in which case I did not speak or share code [to] anyone, even though the code is very similar."* (S-3)

*4) Suitability for use*

Contrary to the academics' opinion, nearly half of the students considered that the tool is not suitable for use since source code may have a lot of coincidental similarity (four out of nine and the rest were neutral). A lot of arguments were given by students on the unsuitability of the tool:

> *"A tutor doesn't compare all parts of code, only select parts where plagiarism is possible [while the tool does compare even the most trivial parts]."* (S-3)

> *"Given that the scope of assignments is very narrow, there will always be large portions of the source code that is similar even between students who have never discussed their submissions… Similarities do not always concretely correlate to cheating"* (S-4)

Nevertheless, students' opinion diverged on whether the assessment should be based on their understanding alone, ignoring the levels of code similarity reported. Three out of nine agreed, four disagreed and the remaining participants were unsure.

*5) Draft submissions and access to the similarity report*

Students generally wanted to view the report if their assignments were being scanned by the tool. However, not all of them wanted to be able to a submit draft. It is possible that, with the current implementation of the tool, draft submission could allow a student to see others' anonymised code before the assignment is due, and this is not desirable to some.

## VI. Conclusion and future work

This paper describes the construction and evaluation of a tool, in the form of a Moodle plugin. The two main features of the tool include a code similarity scanning service and the reporting of code similarity. The tool uses the MOSS and JPlag detection engines, and was redesigned from a prototype developed as part of previous research.

The fully featured tool attempts to tackle barriers to the adoption of source code matching tools at the institutional level based on common academics' practices identified from our past research. The revised tool was developed with a particular focus on usability, suitability to academics' practices and student education on academic integrity. By integrating the tool into Moodle, the overhead involved in using code matching services is minimised, and students were provided with a learning environment that could make them more aware of academic integrity.

An evaluation of the tool from academics' and students' perspective was performed within an Australian university. Generally, academics appreciated the usability of the tool, and expressed a high level of interest in making use of the tool in their teaching and assessment. While students expressed some concerns over its adoption, the concerns raised by the students (including that they would be less likely to share ideas, and raised level of anxiety) were recognised by academics. However, the appreciation of almost every academic participant on the tool's usability, and their affirmed interest to use the tool, proved the potential of its widespread adoption.

A limitation of the research was the homogenous background of the participants, which also limits the variety of feedback received. More interesting insights could potentially be yielded from participants who have past experience using a similar tool as a standalone service, since they would have a more accurate view on the value added by the tool.

Having students volunteer to submit their assignments could be considered as a limitation, since offended students would be reluctant to take part in such sensitive research. Therefore, the responses obtained on the impact of the tools from a limited to a group of students, may not be representative of the wider student community.

Future work includes prioritising and implementing the suggestions made by academics that participated in this study and to undertake a similar study on a larger and more diverse population of academics and students in the computing field. The educational effectiveness of source code matching tools could then be evaluated extensively and more thoroughly in live settings.

## ACKNOWLEDGMENT

The authors of this paper wish to thank Netspot for sponsoring the development of the Moodle plugin (the tool), the Moodle community for feedback on the tool, and all the academics and students who participated in this research.

## References

[1] J. Sheard, M. Dick, S. Markham, I. Macdonald, and M. Walsh, "Cheating and Plagiarism: Perceptions and Practices of First Year IT Students," in Proc. 7th Innovation and technology in computer science education, 2002, pp. 183-187.

[2] J. Sheard, and M. Dick, "Computing student practices of cheating and plagiarism: a decade of change," in Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, Darmstadt, Germany, 2011, pp. 233-237.

[3] D. D'Souza, M. Hamilton, and M. C. Harris, "Software development marketplaces: implications for plagiarism," in Proc of the ninth Australasian conference on Computing education, 2007, pp. 27-33.

[4] N. R. Wagner. "Plagiarism by Student Programmers," http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html.

[5] J. Zobel, ""Uni cheats racket": a case study in plagiarism investigation," in ACE '04 Proceedings of the Sixth Australasian Conference on Computing Education, 2004.

[6] R. Barrett, and J. Malcolm, "Embedding plagiarism education in the assessment process," in *International Journal for Educational Integrity,* vol. 2, no. 1, 2006, pp. 38-45.

[7] B. Stappenbelt, and C. Rowles, "The effectiveness of plagiarism detection software as a learning tool in academic writing education," in 4th Asia Pacific Conference on Educational Integrity (4APCEI), 2010, pp. 29.

[8] M. Rees, and L. Emerson, "The impact that Turnitin has had on text-based assessment practice," in *International Journal for Educational Integrity,* vol. 5, no. 1, 2009, pp. 20-29.

[9] M. Davis, and J. Carroll, "Formative feedback within plagiarism education: Is there a role for text-matching software," in *International Journal for Educational Integrity,* vol. 5, no. 2, 2009, pp. 58-70.

[10] V. Rolfe, "Can Turnitin be used to provide instant formative feedback?," in *British Journal of Educational Technology,* vol. 42, no. 4, 2011, pp. 701-710.

[11] J. Cole, and H. Foster, *Using Moodle : Teaching with the Popular Open Source Course Management System 2nd Edition*: O'Reilly Media, 2009.

[12] E. Roberts, "Strategies for promoting academic integrity in CS courses," in 32nd Annual Frontier in Education, 2002, pp. F3G-14.

[13] T. Lancaster, and F. Culwin, "A Comparison of Source Code Plagiarism Detection Engines," in *Computer Science Education,* vol. 14, no. 2, 2010, pp. 101-112.

[14] D. F. Martin, "Plagiarism and technology: A tool for coping with plagiarism," in *The Journal of Education for Business,* vol. 80, no. 3, 2005, pp. 149-152.

[15] L. Bilic-Zulle, J. Azman, V. Frkovic, and M. Petrovecki, "Is there an effective approach to deterring students from plagiarizing?," in *Science and engineering ethics,* vol. 14, no. 1, 2008, pp. 139-147.

[16] K. W. Bowyer, and L. O. Hall, "Reducing Effects of Plagiarism in Programming Classes," in *Journal of Information System Education,* vol. 12, no. 3, 2001, pp. 141-148.

[17] R. J. Youmans, "Does the adoption of plagiarism-detection software in higher education reduce plagiarism?," in *Studies in Higher Education,* vol. 36, no. 7, 2011, pp. 749-761.

[18] M. Dick, J. Sheard, and M. Hasen, "Prevention is Better than Cure: Addressing Cheating and Plagiarism Based on the IT Student Perspective," *Student Plagiarism in an Online World: Problems and Solutions*, T. S. Roberts, ed., pp. 160-182: Hershey, PA : Information Science Reference, 2008.

[19] T. Le, A. Carbone, J. Sheard, and M. Schuhmacher, "Integrating Source Code Plagiarism into a Virtual Learning Environment: Benefits for Students and Staff," in Australasian Computing Education Conference, 2012.