Integrating Source Code Plagiarism into a Virtual Learning Environment: Benefits for Students and Staff

Thanh Tri Le Nguyen1, Angela Carbone2, Judy Sheard1, Margot Schuhmacher1

1 Faculty of Information Technology, Monash University
PO Box 197, Caulfield East, Victoria 3145
tri.le@monash.edu
judy.sheard@monash.edu
margot.schuhmacher@monash.edu

2 Office Pro Vice-Chancellor (Learning and Teaching), Monash University
PO Box 197, Caulfield East, Victoria 3145
angela.carbone@monash.edu

Abstract
Source code plagiarism is a growing concern in computing related courses. There are a variety of tools to help academics detect suspicious similarity in computer programs. These are purpose-built and necessarily different from the more widely used text-matching tools for plagiarism detection in essays. However, not only is the adoption of these code plagiarism detection tools very modest, the lack of integration of these tools into learning environments means that they are, if used, just intended to identify offending students, rather than as an educational tool to raise their awareness of this sensitive problem. This paper describes the development of a plugin to integrate the two well-known code plagiarism detectors, JPlag and MOSS, into an open source virtual learning environment, Moodle, to address the needs of academics teaching computer programming at an Australian University. A study was carried out to evaluate the benefits offered by such integration for academics and students.

Keywords: source code matching tools, academic integrity, plagiarism.

1 Introduction
Ongoing reports of serious plagiarism incidents, as well as a relaxed attitude amongst students towards this offence, have motivated a lot of research to combat this problem from both education-prevention and detection-punishment perspectives (Sheard and Dick, 2011, Dick et al., 2008). A lot of tools have been developed to assist academics with educating their students about plagiarism and detecting this offence should it occur. A very popular example of such a tool is the widely used text matching tool Turnitin, which is not solely intended as a detection tool for instructors, but is also a feedback tool to educate students on academic integrity, with features such as allowing draft submissions and providing students with access to a similarity report.

Plagiarism involving text (as found, for example, in essays) is the most widely reported form of plagiarism. Plagiarism of computer source code, although less widely reported, has higher incident rates (Sheard et al., 2002, Wagner, 2000). This can be explained by competition amongst students, pressure to create error-free programs, an abundance of readily accessible solutions and a tradition of reusing past assignments (Roberts, 2002). Furthermore, source code is much more constrained than natural languages, which leads to a high degree of similarity between programming assignments as opposed to essays. With the help of modern programming environments, it is remarkably easy to refactor source code by just a few mouse clicks and make it look very different in appearance. This constitutes a substantial challenge for teachers to detect plagiarised code since they typically do not have enough resources to manually check for similarities, unless some students commit the same distinctive mistakes or implement similar unusual approaches.

Some plagiarism detectors intended specifically for source code such as MOSS (Aiken, 1995) and JPlag (Prechelt et al., 2002) can help a great deal in detecting suspiciously similar code, since their algorithms are specifically designed to overcome common disguising techniques. However, these tools are much less known and adopted than their essay plagiarism detection counterparts (Lancaster and Culwin, 2010). As standalone tools, the source code plagiarism detection tools are neither convenient to use, nor able to provide feedback to students, as can be done with an educational tool like Turnitin.

This paper reports on a project to bring code plagiarism detectors closer to their target users with a plugin integrating the two popular plagiarism detection services, JPlag and MOSS, into Moodle, a widely used virtual learning environment. The aim of the project was to promote the adoption of source code plagiarism detectors, not only as detection tools, but also as educational tools, in order to raise students’ awareness of academic integrity. This would be achieved by providing students with feedback on the similarity of their work via a report. The pre-evaluation phase of the project investigated needs and issues faced by academics in their current practices. After integrating the two selected tools, the source code plagiarism detection tools are much more convenient to use, allowing both teachers and students to detect suspicious similarity and providing feedback to students about their work.
detectors, JPlag and MOSS, into Moodle in a way that addressed academics’ needs, the post-evaluation phase investigated the academics’ and students’ perceptions of the usability, benefits and impacts of the integration via the similarity report generated on a real-life assignment.

The next section of the paper gives some background, on currently available source code plagiarism detectors coupled with their adoption and impact. The following section presents the research method leading to the integration of the plagiarism detectors into Moodle and the investigation of the benefits of this integration. The fourth section presents the result of the research, and the final section concludes the findings of this study.

2 Background

2.1 Definition of source code plagiarism

Plagiarism is one of the most common forms of academic offence, yet the concept of plagiarism is arguably obscure and confused. One of the most cited definitions of plagiarism is from Britannica Encyclopedia, which regards plagiarism as “the act of taking the writings of another person and passing them off as one’s own” (Britannica, 2012). When applied to computer source code, an early paper defined plagiarism as “a program which has been produced from another program with a number of routine transformations” (Parker and Hamblen, 1989). However, these definitions are not particularly clear and there seems to be no consensus amongst computing academics on one single universal definition which can apply unambiguously to every case (Cosma and Joy, 2008). The reason for this divergence is due to the nature of programming languages which have less freedom and flexibility than natural languages, the ease of code modification, and the encouragement for code reuse. These characteristics make it easy to disguise plagiarised code, or to unintentionally commit the offence, not to mention the fact that offenders can rely on the ambiguity of the rules to claim innocence (Wagner, 2000).

Some research has tried to determine the boundary between the accepted practice and plagiarism in computer programs through surveys of academics and students. However, opinions recorded are diverse, especially in subtle cases where the student’s contribution is substantial (Cosma and Joy, 2008). This variation can be explained by the differences in teaching and assessment objectives. Therefore, judgement about plagiarism cannot be made independently of context (Dick et al., 2008).

2.2 Source code plagiarism detectors

While essay plagiarism detection tools search for consecutive words to identify copying, source code plagiarism detectors have to take into account the modifications that students can make to disguise their code. Faidhi and Robinson (1987) characterised six levels of modification from simple to complicated. From the original program (level 0), plagiarists could change comments and indentations (level 1), identifier names (level 2), declaration of constants, variables and procedures (level 3), program modules (level 4), program statements (level 5) and finally logic expressions (level 6). With the help of modern programming environments, modifications at level 3 and below only require very primitive knowledge of a programming language, while it is controversial that level 6 could even be considered as plagiarism (Cosma and Joy, 2008).

Different kinds of source code plagiarism detectors are able to find copied code at different levels, depending on the type of algorithms they use. There are basically three categories of detection tools in this regard: software-metric based, token-based and semantic-based. Software-metric based detectors, the most primitive ones, use metrics in software engineering such as program size, complexity, as well as the number of keywords, operators and operands to compare program code (Parker and Hamblen, 1989). The token-based method generally finds common consecutive tokens between two programs after eliminating possible variations such as identifier names (Prechelt et al., 2002). Semantic-based detectors, the most sophisticated ones, construct parse trees or program graphs for each piece of code before matching them together (Liu et al., 2006). While the last category of detectors may be able to detect plagiarised code at the highest level, they do not scale well for large sets of programs, and no publicly available tools in this category were found. For student assignments, token-based tools, which encompass most of the popular detectors, are considered to be the most appropriate for educational use.

2.3 The adoption and effects of plagiarism detection tools

Reports suggest that the use of source code plagiarism detectors is quite limited compared to the scale of the problem and the adoption of these tools is much less than their natural language counterparts (e.g. Turnitin). In fact, only one quarter of the institutions in UK reported having adopted code plagiarism tools (Lancaster and Culwin, 2010). As a result of this lack of use, when students were suddenly checked for plagiarism in their programming assignments, the offences were found to be unexpectedly high (Wagner, 2000, Daly and Horgan, 2005, Bowyer and Hall, 2001).

Whilst some papers have reported studies of the effects of using a text matching tool on students’ text-based plagiarism practices (Biggam and McCann, 2010, Rolfe, 2011), not as much research has reported on the same issue with source code plagiarism. Bowyer and Hall (2001) observed that the incident of plagiarism dropped considerably after a period of time using MOSS, but later discovered that plagiarism was shifted to another source undetectable by software such as acquiring a program from students from a previous running of the course or even hiring outsiders. The problem was only detected when two students in the same class acquired the same program from one source, which the authors called the “ghost author phenomenon”. Similar incidents were also reported at RMIT University where JPlag is used officially for every programming assignment (Zobel, 2004, D’Souza et al., 2007).

Besides detection, a suggested educational benefit of the detectors is the raising of students’ awareness of the offence by providing them with feedback on the similarity of their code to others. A search in the literature found no works that have been done to evaluate the
educational effects of this kind of feedback from source code matching tools. However, similar research on Turnitin by Biggam and McCann (2010) found that this feedback could facilitate a smoother transition between school and university for first year students by helping them to improve their referencing skills and awareness of academic integrity.

3 Research approach
The widespread problem of plagiarism in programming assignments, as opposed to the low adoption of source-code plagiarism detectors, has motivated this research to seek a better understanding of the benefits and limitations of these tools and promote their adoption. The project proceeded in three phases:

- Phase 1: Investigation of the current plagiarism detection and handling practices of academics and their satisfaction with these practices.
- Phase 2: Development of a plugin to integrate the two well-known third party plagiarism detection tools MOSS and JPlag into Moodle.
- Phase 3: Evaluation of the benefits of the plugin in making MOSS and JPlag code plagiarism scanning services an inherent part of the assessment process within Moodle. This will be conducted with a real assignment and from the perspectives of both academics and students.

The remainder of this section describes each phase, explaining the purpose of the phase, a justification of the approach, the method involved and its potential limitations.

3.1 Phase 1 – Investigating the current practice
In order to integrate the detectors in a manner that meets academics’ needs, it was important to obtain insights into the current practices of plagiarism detection and the difficulties encountered. The aims of this phase were twofold:

- **Gain understanding of the current practices in identifying and dealing with source-code plagiarism:** this was the main focus of this phase since it helped determine potential enhancements of the integration of the detectors into the virtual learning environment the academics used.
- **Determine the level of satisfaction with the current practices and the difficulties encountered:** this would highlight areas of improvement to academics’ assessment practices that the tools may offer.

Given the exploratory nature of this phase, a semi-structured interview approach was adopted. The interview method was considered over the other qualitative methods, such as surveys and focus groups, due to its flexibility in exploring the academics’ views. While a survey could reach a larger group of participants in a limited time, it does not offer the opportunity to delve into the ideas raised by academics, which were of particular interest to the research. Moreover, interviews make it easier for respondents to provide their opinions and personal experiences, which were actively sought in this phase.

The study was conducted with Monash University academics, including both lecturers and tutors, who were involved in teaching programming units at any level. Twenty-two academics were approached for interview. This was done by sending an email invitation to lecturers and tutors of programming units. Every academic who agreed to take part in the research was interviewed.

A limitation of interviewing only Monash academics is the lack of representativeness for the whole IT academic community in general. Since academics from the same university often adhere to the same standards and processes, they might be likely to share similar views and practices. This potentially results in the findings of the research and the implementation of the plugin having less applicability to other institutions.

3.2 Phase 2 – Developing a Moodle plugin
Two popular source code plagiarism detectors, MOSS (Aiken, 1995) and JPlag (Prechelt et al., 2002), were identified as having good detection performance and good reputations in the academic world. Taking into account the needs and difficulties raised in the previous phase, the aim of this phase was to develop a Moodle 2 plugin for the integration of these two detectors with the following characteristics:

- **Seamless assignment creation and submission:** the use of these plagiarism scanning services should be effortless and transparent to the users, without imposing any additional constraints to the normal submission process.
- **Publishing a limited version of the similarity report to students:** seeing plagiarism detectors also as an educational tool and a means of feedback to students, the plugin would allow lecturers to publish a limited version of the similarity report on the students’ code, whilst ensuring confidentiality by masking identities.

3.3 Phase 3 – Evaluating the plugin
The aim of Phase 3 was to evaluate the plugin based on academics’ and students’ opinions of the similarity report generated from a live assignment. It was considered that using a live assignment rather than artificial data would make it easier to elicit feedback from lecturers and, in particular, students, since a live assignment is more natural and provides a specific context. Academics and students would be given the report to view in order to provide feedback in an interview. The following were the issues that the interview focused on:

- The academics’ comments on the usability of the similarity report and any improvements it offers to their assessment process.
- The students’ opinions about the value of the feedback on their work given by the generated report and its impact on their ethical conduct.

A medium size programming assignment was deemed appropriate for trialling the tool. The first assignment of a Java unit\(^1\) was selected because of the suitability of its

\(^{1}\) A unit of study at Monash is equivalent to a subject in other university
schedule to the research timeframe. This unit is a foundation programming unit for post-graduate students. More than 20 students in the unit were introduced to the research project in a lecture and invited to submit their assignment work to a Moodle sandbox. The sandbox was separated from the University virtual learning environment so that the confidentiality of students’ work could be ensured. As plagiarism is a sensitive topic for some students, at the time of participation, students were assured that their marks would not change and the generated report would not be shown to their lecturer and tutor. The incentive for students to participate in the research was that they would have the opportunity to see the report and give comments about it.

Academic participants in this phase were recruited from the participants in the first phase. In the first interview, academics were asked whether they were willing to evaluate a similarity report in the latter phase of the research. Everyone who gave consent was recontacted in this phase.

Interviews were conducted individually and consisted of open-ended questions. Participants were free to express their opinions and relate their experiences. Interesting issues raised by participants were explored more deeply. All interviews with academics were recorded and transcribed. However, only hand written notes were taken during the student interviews to remove any possible discomfort with the recording.

4 Results

Following the research approach described above, this section details the findings collected through the interviews in phase 1 and phase 3. Recurrent themes were extracted from the transcript and classified into emergent categories.

4.1 Phase 1- Investigating the current practices

Six academics agreed to participate in this research. It was found that the academics participating in this study followed very similar practices of dealing with plagiarism, although their satisfaction towards the effectiveness of these practices varied.

4.1.1 Academics’ practice of dealing with plagiarism

The study found that most academics deal with plagiarism more on the prevention side than the detection side. Most participants at the time of interviews did not have a formal process to detect plagiarism, while they mentioned many methods on the prevention and education side.

4.1.1.1 Plagiarism prevention

Prevention strategies were raised the most during the interviews. The following are the approaches mentioned by academics.

Assignment design

Most academics interviewed affirmed that they spent considerable effort to make sure that their assignments are not easy to find on the Internet, thus making them more difficult for the students to plagiarise:

“I’m happy that my assignment is not adapted from somewhere else where [students] can get the code” … “My responsibility as a lecturer is to create an assignment that is difficult to plagiarise. I can’t always blame the students.”

In regard to the design of assignments, the academics felt it is preferable that an assignment is not too small and has a variety of possible approaches and designs, so that students who work independently would produce significantly different code. This would also reduce the temptation to plagiarise and make it easy to recognise striking similarities later:

“In my assignment, most of the stuff is very unique and most of the time students come up with a unique solution”

Exam weighting

Another method to reduce the desire to plagiarise is to give the exam a much higher weight than the assignments, since it is very difficult to cheat during an exam. The intention is to encourage students to work on the assignment to acquire the skills needed to succeed in the exam. This type of mark distribution is used in introductory programming units, where assignments are quite simple and students’ code is likely to be similar to each other:

“It’s hard to [pass] because the exam has the majority of the mark. If they do cheat in the assignment, […] they are not able to pass anyway”

Avoiding assignment reuse

Most participants emphasised the need to avoid reusing the assignment over many semesters:

“If I use the same assignment over and over, then I am inviting people to do plagiarism because I know that some of the students know people who did the subject last year or the year before. I always make changes to the assignments every year, so they can’t copy and paste from last year’s students without understanding”.

Nevertheless, the extent of variation differed between academics. One participant claimed that he changes his assignments completely each semester, whereas another reported that sometimes he reused old assignments. The majority stated that they make some variations but the main ideas and concepts remain unchanged. Depending on the extent of variation, this strategy may only prevent blind copying. Since the assessed concepts were the same, students could adapt a past year’s program with much less effort than doing it from scratch:

“If they have access to [last year’s] student work, it would be helpful to them because the concept is the same. But they cannot copy and paste”

Considering the effectiveness of these efforts, academics also maintained that what they do would just reduce the problem of students copying and pasting from each other. However, there are always other sources for students to plagiarise from:

“Students can post questions on forums and, sometimes, get the whole program from others”… “Using the Internet, students can hire others to do an assignment for them… Sometimes, they can outsource it to India…”
**Tutorial assistance**

A couple of participants mentioned following up with students during tutorials in order to provide adequate feedback. As one explained:

“In every tutorial, the students must complete their work and submit a lab sheet each week. Every week, I check the lab sheet and give them feedback. So I get a fairly good idea of what each individual is capable of.”

Besides identifying students who need help and assisting them early so that they can do the assignment by themselves, knowing the students’ levels of performance also helps academics know who to pay attention to later when investigating suspicious plagiarism cases.

4.1.1.2 Plagiarism education

Raising students’ awareness of plagiarism is important, especially in the area of source code where the boundary between plagiarism and proper practice is vague. Every academic in the study claimed that they often remind their students about plagiarism. Some stated that they explained their expectations before an assignment:

“I made it quite clear to all the classes that we are happy for them to discuss the assignments, but the implementation – that has to be their own. Be prepared to justify your design in the interview... I also posted something more on the discussion board.”

Others, however, were less pro-active and just mentioned it as a standard procedure in a very general manner:

“I don’t know that I made it explicit. I always say to them that the work that you submit must be your own work, not someone else’s. They might find it’s silly since I am saying the obvious.”

4.1.1.3 Plagiarism detection

When it comes to plagiarism detection in programming assignments, the methods that every participant used were either program demonstration in the lab and/or in-depth interview with the students:

“I detect plagiarism through a demonstration. Whatever they submitted to Moodle, I ask them to download and run it. Then I go to a particular class, and point to a complicated piece of code and ask them to explain it. I also ask more conceptual question, such as how did you do it or why do you use this approach.”

However, interviewing is not possible in every situation. One academic said that he could not rely on interviewing since many of his students were not able to meet face to face for interviews:

“Many students of mine are distance learning. So many times I don’t interview.”

Other than interviewing or lab demonstration, most participants stated that they also read the code for marking and giving feedback, and during this process plagiarism cases are sometimes uncovered when students do weirdly similar things or commit the same mistakes:

“I also read the code of the students fairly thoroughly because I have to give feedback. When I read the code, sometimes I pick up something very unusual and another student did exactly the same thing or made the same error. Then I compare the code to see if it has too much similarity.”

If the academic monitors students closely in tutorials, they can target some students who they observe working together in groups:

“Because we know the students quite well, we would tend to know who are working together and we could start to identify this during the process of marking. We do a little bit of cross checking to see how they are similar.”

or, they can target students whose performance in the assignment differs too much from what they have demonstrated in labs:

“If they produced exceptionally good work and I know that this student had trouble with programming in the tutorials, then I know usually something is going on”.

With regard to the use of technology, it was found that only one participant had used a technology to detect plagiarism:

“I sometimes use the diff command in Linux to compare two codes where I found some similarities.”

Another participant had previously used a tool to detect similarities in Java code, but considered its overhead not to be worth the value that it brought:

“From my previous experience, I am not very convinced that software can do a good job. Software can just pick up blatant similarity. Then, if it is so obvious that software can pick it up and I can pick it up myself, then going through the hassle of setting the software, and feeding all the assignments to it, is not worth the trouble.”

The remaining participants explained that they did not use tools because they were not aware of any that were suitable:

“I didn’t use any formal plagiarism detection tools... I haven’t found out about them yet.”

or because they thought these tools were unnecessary because interviewing alone could ensure that the students actually understand what they submit:

“I can be pretty sure that students don’t get the mark for what they don’t learn… It’s probably [too] much trouble to examine what comes back from a plagiarism detector.”

Another reason was that the purpose of plagiarism detectors seems to be orthogonal to the purpose of teaching and learning:

“Using plagiarism detection is something that is essentially a punishment. That is probably not a core point I think. It’s far more important to use that opportunity to learn these things and improve themselves.”

In brief, when it comes to plagiarism detection, academics just follow the normal process of assessment: using demonstration and interviews to check the students’ understanding and reading code to give feedback or a mark. There are no separate stages that are devoted solely to detecting plagiarism. Plagiarism was discovered in an
ad-hoc manner when some unusual similarities or striking progress captured the marker’s attention. At the time of the interviews, none of the participants were using code plagiarism detectors, although one participant had tried one in the past but found that it posed too much overhead on the assessment process.

4.1.2 Difficulties and satisfaction with the current methods of detecting plagiarism

By understanding academics’ satisfaction with current plagiarism detection practices and difficulties they encounter, possible improvements could be identified. It was observed that their level of satisfaction and the difficulties reported depended largely on their degree of tolerance towards plagiarism, despite their similar approaches to plagiarism detection.

4.1.2.1 Level of satisfaction with the current plagiarism detection method

Academics who believed that assessing the students’ learning and understanding is sufficient claimed that their current approach is satisfactory:

“If the students know the stuff as much as I know, there is little point in searching for plagiarism… Students in the interview, they can’t get away with it because they don’t know it… [With what I currently do], I can be pretty sure that the students don’t get the mark for what they haven’t learned.”

On the other hand, academics who took copying seriously expressed their discontent with their current practice, since they felt certain that some plagiarism cases escaped detection:

“It’s possible that the students can get somebody else to do their work and really study well and understand it so that during the interview they can explain it… I’m not satisfied in the sense that I’m sure there are some cases that go unnoticed.”

However, these academics also affirm that what they have done is adequate within the limited time they have

“But given all the constraints on time, I don’t feel that I should put a whole lot more effort to detect plagiarism. Overall, I think it’s reasonable”

4.1.2.2 Difficulties encountered

Class size

Large class sizes were one of the major difficulties raised by academics. With a class of hundreds of students, it is merely by chance that cases of plagiarism are detected:

“It depends on how many students I have. A few years ago, I had hundreds of students. It was lucky to find some plagiarism cases since there were so many students. But if there are only a few, it is very easy”

and also, there are many markers who grade the assignments independently, which further reduces the chance of detection:

“I also read the assignments and sometimes pick up some similarity. But it doesn’t ensure anything because we have many classes and markers”

Time constraints

Another difficulty encountered by many academics was the limited time they could spend on assessing each student, for both the interview and the marking:

“We have a two hour lab. I don’t want to spend more than 15 minutes per student just trying to detect plagiarism”

Therefore, some academics admitted that they sometimes did not want to spend more time finding similarities, even when in doubt:

“Even if I remember all this, for me to go back to find it takes too much of my time. If I know who their friends are, I just look through those ones, but if I couldn’t find [anything], I probably wouldn’t go through too many assignments, just to find something I thought I remember that is similar.”

Low variation between assignments

There are some kinds of assignments for which students’ work is inevitably similar. For example, when students were given skeleton code or only required to adapt code given in their lab. In these cases, it is not possible to tell if the students plagiarised, as one academic pointed out:

“The point is that the size of the program is often not big enough to be very distinctly different, and in more advanced units, we actually give them code so they will be very much the same.”

However, all other participants claimed that the way of implementation should be different even when the same approach was taken:

“In my assignment, there are a few ways that are quite clear that they are the best ways to approach it… I still expect some deviation in the way classes are implemented or the logic on how it works”

Overall, the academics in the study thought that their approaches were effective in reducing plagiarism, although they admitted that some cases of plagiarism can slip through the process undetected. For academics that were happy with assessing students based only on the students’ understanding, this process was satisfactory. However, those who wanted to make sure that the students actually did the work themselves thought that what they currently did was not enough due to time limitations for marking and interviewing, as well as the difficulty of manual checking in large classes where there are many markers.

4.2 Phase 2 – Building a Moodle plugin

4.2.1 The need for integrating the detection tools into Moodle

It was found in the previous phase that academics were concerned about plagiarism and paid attention to minimising the motive and temptation to plagiarise amongst their students. Furthermore, they tried to detect plagiarism in their marking process. However, due to the constraints of time and resources, most academics acknowledged that discovering plagiarism was quite ad-hoc and accidental, relying on unusual similarities, which captured their attention. The adoption of a good plagiarism detector could make this process less dependent on chance.
The two major difficulties raised by academics in detecting plagiarism, large class sizes and time limitations, could be mitigated by an automated process using the tools. Moreover, the detectors could make cross checking between markers much easier by generating a single report across the whole class.

However, as one participant remarked, code similarity detectors, when used as stand-alone tools, impose considerable overhead. The repetitive tasks of downloading and extracting all students’ submissions and then organising them into the required directory structure are time consuming and error-prone. This is compounded by the non-intuitive interface making the adoption curve steeper. Moreover, these plagiarism detectors, when used as stand-alone, are “essentially a punishment”, as one academic thought. It cannot serve as a means of providing feedback for students and raising their awareness of academic integrity.

Considering all the reasons above, it seems reasonable to propose that integrating the detectors into a virtual learning environment like Moodle could significantly promote their adoption. It would enhance the usability of the tools by providing an intuitive interface in the learning environment familiar to academics. In addition, it would make the similarity scanning transparent and effortless, at the same time allowing academics to provide students with restricted access to the similarity report.

4.2.2 Features of the plugin

Considering the needs and difficulties mentioned above, the plugin was developed with features that make the detection tools fit seamlessly into the assessment process on Moodle, namely:

- **Automatic filtering of code files**: students can submit an archive file containing a bunch of project files and documentation apart from code. The plugin will extract only code files to send them to the scanning service. Therefore, no additional constraints are imposed on the system other than the normal submission and marking process.

- **Automatic scanning and reporting**: the assignments are extracted and submitted automatically to the scanning services at the date specified by the lecturer when configuring the assignment.

- **Publishing similarity report to students**: the plugin enables teachers to allow students to view the report on the similarity of their code with others’. If permitted, students could only see a restricted version of the report, with names masked.

Currently, our system keeps the native reporting interfaces of MOSS and JPlag. The MOSS interface presents a simple list of pairs in decreasing similarity order, whereas JPlag groups all the students having high similarity with one student in a row. In addition, JPlag also computes the similarity distribution of all the pairs.

4.3 Phase 3 – Evaluation of the plugin

The live assignment selected to evaluate the plugin involved the development of a small “number guessing game” in which the user and the computer took turns to guess a random number within a specified range. Fifteen students agreed to take part in the study and submitted their code to our sandbox. Most of them were international students, and many were in their first semester of study.

4.3.1 Overview of the generated report

All of the 15 submissions were of similar structures. The students followed the design they had been recommended in their lab, with a Player and a Game class, which were used for storing the information of the player and controlling the game logic respectively. Depending on how carefully the students handled input errors and how much boilerplate code (e.g. set and get methods) they wrote, the length of their submissions varied. Excluding comments and blank lines, the average number of lines of code of all submissions was 193. The generated similarity report showed the pairwise similarity rate among 15 submissions ranged from 0% to 35%. Overall, MOSS gave considerably lower percentages than JPlag. The similarity distribution given by JPlag was symmetrical with half of all the pairs having similarity rates in the middle range (10%-20%), a quarter of the pairs in the lower range (0-10%) and the other quarter in the higher range (20-30%), plus three “exceptional” pairs having similarities distinctly higher than the others. MOSS produced a very different distribution with most of the pairs (70%) having similarity rates below 10%, the other 30% of the pairs having rates ranging from 10%-20% and one “outlier” pair had an outstanding rate of 20.5%. For a small and strictly specified assignment such as this, the similarity rates were considered quite low.

4.3.2 Evaluation: academic perspective

Four academics who participated in the first phase were reinterviewed in this phase. In addition, we extended our invitation to the Moodle administrator of the faculty, who is also experienced in teaching programming, to get a more varied perspective on the plugin. In the Phase 1 interview, academics expressed different opinions with regard to using detection tools, from a high level of interest to doubts about their benefits. This phase reinvestigated their views after introducing them to the plugin and presenting them with the similarity reports. Overall, academics expressed favourable opinions on the plugin and their interest in using it in a further trial.

**Improvement of the plugin to the assessment process**

Academics all had a positive first impression of the plugin. A variety of enhancements to the marking process mentioned by academics included time efficiency, reduced effort, more stable detection and better student awareness of academic integrity.

Although every academic agreed that the plugin cannot automate the detection process entirely, most academics affirmed that the readily available detectors could save them a lot of time and effort, not to mention better detection effectiveness:

“What I used to do when I mark… if I think it’s plagiarism, I put a mark at the top […]. At the end, I tried to figure out which one is similar to which one. It’s very time consuming, especially when I have 50-60 [students]. I have so many things to do. Most of
the time, I just try to go to as much as possible [until] I give up. A tool like this makes life much easier.”

However, a couple of other academics also mentioned the extra time they would need to spend examining the reports:

“It actually takes more time to examine the report. But it helps better ensure academics integrity. And while checking the report, I also read the students’ code. So, it takes less time to review the code”

Some academics emphasised the usefulness of the plugin for large classes with many tutors who mark separate groups of students:

“When I have one class with a hundred students, there are often four to five tutors. It’s not easy for the tutors to sit together to do the [cross checking]. A tool like this one would be handy”

A feature of the plugin that attracted opposite opinions is publishing the report to students. Some academics commented that it could serve as an effective plagiarism deterrent:

“It’s OK for the system to say we ran the scanning through and we don’t find any similarity. It may alert them and stop them from plagiarising.”

Nevertheless, many concerns were raised. A major issue is the potential anxiety of students when their code contains a high degree of coincidental similarity:

“If I tell the students who may not plagiarise that there is 30% similarity between me and you, it may make the students anxious.”

Moreover, this may result in a negative impression among the students, making them think their lecturers are primarily interested in catching them:

“We really don’t want students to think that we’re doing a switch hand, that we are having the system to drag them into trouble”

These concerns were further investigated with students’ perspectives on the plugin. Some academics preferred to have a fine-grained control over the publication of the scanning results to students. For example, they wanted to filter what reports were sent out to students, particularly those that were above a plagiarism threshold:

“It’s better if the system lets the lecturers to decide to post the student one by one”

Usability of the similarity report
All five academics agreed that the report was very easy to read. They commented that the interface and design of the report helped them to quickly identify suspicious cases:

“I am really happy with the interface. I could navigate between the similarities very quickly”

There were somewhat different comments on the reporting interface of MOSS and JPlag. More academics preferred the tabular style of JPlag listing all students similar to each student on a row than a simple list of pairs in MOSS:

“[The interface is] fine with MOSS. But JPlag is better. With this style of presentation, I can see all the students who have a high similarity with one student… It makes the examination easier”

Nevertheless, one academic expressed a different opinion:

“JPlag is a bit more confusing… Not all the students having a high similarity rate appear on the right hand side. It’s more confusing to follow than the simple view of MOSS”

A major benefit of JPlag over MOSS expressed by the academics was the statistics of similarity distribution between every pair:

“The statistics help me to see immediately a few suspicious cases, those who have much higher similarity than the average”

Generally, the additional features provided by JPlag over MOSS were appreciated by all the academics.

Satisfaction with the tool
Overall, participants were positive about the plugin and expressed interest in trying it in their units, though each to a different extent. Some participants expressed their satisfaction with the report:

“I am impressed by this. The tools clearly told me what the similarities are… all the things according to the highest to lowest”

Others expressed a more reserved opinion on the tool’s usefulness, and expressed their desire to experiment with the plugin in their units:

“The answer now is yes. I think it is useful, but until I try it I can tell how useful it is… Yes, definitely. I am curious about it and want to try”

Overall, academics expressed satisfaction with the plugin and the detectors after experimenting with its features and examining the report. They felt the plugin offers many benefits, including reducing effort in plagiarism detection, more effective use of time, and improved detection effectiveness especially in large classes. Publishing the report to students was generally supported, although there were some concerns about students’ anxiety and confidentiality.

4.3.3 Evaluation: student perspective
This section presents the opinions of students on the report and its influence on their ethical behaviour. Among the 15 students who had initially submitted their assignments for the research, three were interviewed in this study. One of them had a little programming background before joining the unit, and the other two were learning programming for the very first time. These students all had the detected similarity rate a bit higher than the average (ranging from 25% to 31%). However, a look through the similarity report showed that they were unlikely to have copied code from each other.

Students’ opinion on the similarity report
All the students stated that the similarity rate of their work was higher than they expect, although they agreed that the tools were quite accurate in highlighting the similar portions in their code. For example, one student expressed that he did not expect such a high similarity since he did the assignment alone. However, he thought that the tools picked up the interesting similarities.

Another student stated that they should be assessed based on their understanding of what had been done, not
based on the similarity rate with others, since it is very easy for assignments to be similar.

A common concern raised was that students thought that coincidental similarity is inevitable and that such a report on their work may be misleading.

It was observed that a higher than expected similarity caused certain anxiety amongst participants. After the report was released, one student sent an email justifying his incorrect similarity rate (which was just 24.3% and lower than the interviewed students), by pointing out the false positives in the report. This student did not take part in the interview.

“I [have] just seen the result of your system. My code was [reported] as having a 24.3% similarity with another student and I think it’s not right... For example, the group of println statements in lines 41-48 is very different, but marked as similar... The similarity score is higher than reality... I worked independently in this assignment”

As a contrast, another participant stated that he thought his similarity rate was normal, though he had the highest similarity rate amongst the three students interviewed (30.9%). He was confident that the examiner would see that there is no sign of plagiarism in his assignment. This student said he had had experience with an essay plagiarism detector in his undergraduate study.

Although participants were told before joining the research that the report was confidential and would not affect their result, and that there were many reasons for code to be similar, anxiety still occurred. It seems that anxiety does not only depend on the similarity rates found but also on the students’ prior experiences with such a tool. If students have been exposed to a similar tool before, they are likely to feel more comfortable in how the results would be interpreted, even if the similarity is quite high.

**Impact of using the plugin on students’ behaviours**

The participants indicated that the strongest impact of using the plugin would be in discouraging them from sharing code with classmates. All of the students said that they “will not dare to share code with anyone” since there is little chance to escape the detectors.

As for sharing on a conceptual level, opinions of students diverged. Some students said that they would be even afraid of sharing ideas and approaches since the chance of coming up with similar code would be much higher. However, the student who had had some programming knowledge prior to joining the unit, said that he would not feel anxious about sharing his algorithm or method in a general way, since it is not likely that there would be a similar implementation based on just the ideas.

All students interviewed said that they would be afraid of copying code from another student since there is a good chance for them to be detected. One student stated that if he copied the code from someone else, he would have to modify the code substantially to avoid being detected. This job requires time and understanding of someone else’s code; therefore, he concluded that it is better and safer to write his own program.

### 4.3.4 Overall summary

The evaluation of teachers’ and students’ perspectives on the Moodle plugin showed that it offers appropriate features to assist academics in detecting plagiarism. The plagiarism detectors are quite efficient and potentially effective in discouraging students from plagiarising. The academics stated that the plugin was easy to use and the report interface allows for faster plagiarism detection than manual scanning. Although, more time must be spent in examining the report, this time could be offset by less code reading time and better detection. The plugin is especially useful for a large class with many markers, where manual cross checking is often not possible. The students stated that plagiarism detectors would deter them from copying and discourage them from collaborating. However, using the tool also produced some undesirable effects. When students know that such tool is being used, it may also cause anxiety amongst students and hinder some forms of exchanging of ideas that should be encouraged. The extent of anxiety would depend on their exposure to other plagiarism detectors and their programming experience.

Publishing the report to the students was generally supported by the academics, provided that confidentiality is maintained and the results are explained to the students. However, there were different opinions from the students. One student considered that the report is part of a student’s result and they should be allowed to see it, whilst another student did not want their code to be revealed to others.

### 5 Conclusion

This paper described a study to investigate the benefits of source code plagiarism detectors for current assessment practices and to promote their adoption by integrating two well-known detectors into Moodle via a plugin. The plugin was built to respond to the difficulties encountered by academics in their current practices and its usefulness and impact were evaluated from both lecturers’ and students’ perspectives.

Whilst academics devote considerable attention to the issue of plagiarism, with a range of measures for prevention and detection, many of them were unsatisfied with their current practice since they were quite certain that some cases of plagiarism went unnoticed when their class size was large and their time to spend on marking was limited. All participating academics did not currently use any plagiarism detection tools for code, as they had not investigated any tools or they found that tools posed considerable overhead.

Taking into consideration all of the difficulties expressed by the academics in this study, a plugin to integrate JPlag and MOSS into Moodle was implemented. The plugin makes these detectors effortless to use and enables the scanning results to be made available to every marker. In addition, it offers the possibility of giving feedback to students and therefore has value as an educational tool.

Evaluation from the lecturers’ and students’ perspectives showed that the tools could assist academics effectively in detecting plagiarism, and deter the students from sharing code together. Making the similarity report
public to the students was considered a good idea by most of the academics as they saw it raising students’ awareness of plagiarism although there were some concerns about confidentiality and the students’ levels of anxiety. In fact, interviews with students showed that their first exposure to a plagiarism detector might arouse significant anxiety, since non-plagiarising students expected a negligible similarity, which often was not the case. This anxiety may also vary according to the students’ programming experience and their understanding of the process.

The evaluation of our current plugin has also revealed a lot of room for improvement. Our current system uses the native interfaces of JPlag and MOSS, with just a few modifications to hide identities in the student view of the report. The feedback from this study indicates that it is better to incorporate the result of different engines into a single interface which combines the advantages of each, helping academics browse the results and filter suspicious cases faster. The draft submission is also a very interesting feature that is available in some text matching systems. Our current plugin permits students to see the final report, but does not offer the option to resubmit their work if they find their similarity rate is high.

A major limitation in the research method is that academic participants are just given a demonstration of the plugin before evaluating it instead of actually experimenting with it themselves. This, perhaps, limits the richness of the information they could give.

With these identified shortcomings in the research method and the plugin, future directions of our research are to improve the plugin and to re-evaluate it by giving participants hand-on experience of configuring and using the tool.


Zobel, J. (2004): "Uni cheats racket": a case study in plagiarism investigation. ACE '04 Proceedings of the Sixth Australasian Conference on Computing Education, Darlinghurst, Australia, 30, ACM.