An Anti-Plagiarism Editor for Software Development Courses

Peter Vamplew and Julian Dermoudy

School of Computing
University of Tasmania
Private Bag 100, Hobart 7001, Tasmania

Peter.Vamplew@utas.edu.au

Julian.Dermoudy@utas.edu.au

Abstract

Plagiarism is a serious issue in undergraduate computer science courses involving assessment of programming assignments. The electronic nature of these assignments means copying others' work is very easy, and the lack of variation between legitimately independent solutions makes the detection of plagiarised solutions difficult. The primary tool in combating plagiarism should be education of students about the issue. The need still exists however, for means to detect plagiarism when it does occur, and automated tools can provide valuable assistance in this task. Most such tools developed so far have focused on analysing the content of the final work submitted by students.

In contrast this paper describes an anti-plagiarism approach based on consideration of the entire process of producing the submitted source-code, rather than just the source-code itself. It describes a text editor and related software which have been implemented based on the Eclipse development environment. These tools aim to discourage plagiarism by making the copying process more labour-intensive, and to aid in detection by storing data about document creation along with the document content.

Keywords: Plagiarism, computer science education, programming.

1 Introduction

Plagiarism by students is a serious and widespread educational issue. Stokes and Newstead (1995) found that 64% of students reported having knowingly copied work at least once during their University studies, whilst Sheard, Dick, Markham, Macdonald and Walsh (2002) state that several recent studies have reported admissions of academic misconduct from around 90% of students. Within individual units and pieces of assessment, several authors report plagiarism rates on the order of 10% to 30% (Zobel and Hamilton (2002), Culwin, MacLeod and Lancaster (2001), Wagner (2000)).

The causes of plagiarism have been widely investigated (Sheard et al. (2002), Sheard, Carbone and Dick (2003), Wagner (2000), Zobel and Hamilton (2002)). A detailed discussion of these causes is beyond the scope of this paper, but some of the most commonly reported causes are:

- a lack of time to undertake the assessment task;
- fear of failure, or the costs of failure;
- assessment tasks which are too difficult for the student's current skill level;
- inadequate resources (hardware, software, library, access to teaching staff, etc.);
- perceived irrelevance or lack of educational merit of the assessment task.

If plagiarised submissions for assessment tasks are not detected and penalised, the efforts of those students who complete these tasks legitimately are devalued. In addition assessment exercises are usually also intended to act as formative learning opportunities for students — clearly students who plagiarise all or parts of their assignments learn less than they would from undertaking the work themselves. Zobel (2004) suggest that students who cheat in the hope of catching up later may become caught in a "cycle of plagiarism", as they never develop the skills required to carry out the assessment exercises independently.

The statistics available on detection of plagiarism however are in stark contrast to those on its incidence - Culwin et al. (2001) reported that an average of 5% of students were caught plagiarising in a year, implying that a significant amount of plagiarism is currently undetected and therefore unrectified.

There are two main forms of plagiarism — either students re-use material from external sources without suitable acknowledgement, or they copy all or part of another student's work, with or without the permission of that student. Whilst the first form is likely to be a significant issue in essay-based assignments, the latter is more common within programming units. A third possibility also exists — that of a student paying a third party to undertake the assessment task for them. Zobel (2004) reports a case of this form, and suggests it may be more common than previously thought.

Two aspects of programming units pose particular problems from the point of view of discouraging and detecting plagiarism.

First, the electronic nature of the material being produced by students makes plagiarism particularly easy to undertake. Material can be copied with practically no effort either by directly copying a file, or by cutting-and-pasting source code from one file to another. Subsequent minor editing such as changing identifier names, rephrasing comments, altering the use of whitespace and indentation, or modifying the structure by re-arranging sections of the code, is often used to attempt to disguise the origins of the code.

The second difficulty lies in the detection of plagiarised assignments. The constraints of programming languages compared to natural languages mean that solutions to programming tasks are likely to display a high degree of similarity, even if they have been developed independently. Anecdotally this means that if the detection of copied code is based on manual inspection, plagiarism is more likely to be detected when this code is a poor solution to the task, as weaker solutions are more likely to contain distinctive features. Wagner (2000) notes that such unusual features are often seen as "a smoking gun" — conclusive evidence of the occurrence of copying. Therefore those students who are benefiting the most by obtaining higher marks from copying good solutions are also more likely to escape punishment.

The detection of plagiarism is also affected by non-technical factors such as the changing structure of tertiary education. Developments such as increased class sizes and cross-campus teaching of units may make the use of multiple markers more common. This reduces the likelihood of plagiarism being detected by manual means as each marker only sees a subset of the submissions.

2 Plagiarism Prevention and Detection

The approaches to minimising the incidence and impact of plagiarism fall into two complementary categories — preventing plagiarism from occurring, and detecting cases of plagiarism when the preventative measures fail.

2.1 Educational approaches to plagiarism

Several authors have suggested guidelines for units which are aimed at reducing both the need and the opportunity for students to engage in plagiarism. This section summarises some of the most common recommendations.

Ethics education, warnings and penalties

Clearly plagiarism and similar unethical actions are undesirable in the broader world outside of university study, as well as within academia. Therefore the primary action in combating plagiarism should be educating students about the nature of plagiarism, and the reasons why it is unacceptable. This process should also help to address differing attitudes to plagiarism which may exist between students from different cultures.

There also need to be clearly specified policies for the treatment of plagiarism once detected, including the penalties which will be incurred. Zobel (2004) argues that deterrent policies are only likely to be effective if students perceive that plagiarism is likely to be detected. This is supported by the results of Braumoeller and Gaines' (2001) comparative study of two groups of political science students. One group was provided with a strong written and verbal warning about plagiarism prior to writing a paper, whilst the second group was informed that their papers would be checked by plagiarism detection software The study found that the formal warning had "no discernable deterrent effect" whilst the announcement about the use of plagiarism detection software had a significant impact on the levels of plagiarism amongst the second group. Similarly, Weinstein and Dobkin (2002) reported a significant — although not total — reduction in the incidence of plagiarism in a class where the use of plagiarism-detection software was publicised compared to a control class where the use of this software was not announced to the students.

Coursework design

Sheard et al. (2003) report that in many cases students report that plagiarism arises from lack of motivation due to a perceived lack of educational value in the work being carried out, or due to frustration with poorly designed or under-resourced tasks. Poorly designed assessment items can increase both the motivation of a student to cheat, and the ease with which this can be achieved. To avoid this coursework should be interesting, constructively aligned with the objectives of the course, of a suitable level of difficulty, and timed to avoid imposing excessive workloads at the end of semester. Assignment exercises should be changed from year-to-year to avoid the re-use of solutions from previous years, and should contain unique elements to impede the inclusion of generic code from online sources.

Additional assessment

One means of verifying the results of programming assignments is to additionally assess the programming expertise of students via some other means. Options include supervised programming exercises in laboratories (Zobel and Hamilton (2002)), interviewing students about the workings of their code (Culwin et al. (2001), Sheard et al. (2002)), and including programming tasks within examinations (Wagner 2000).

Incremental submission

Many computer science departments now require students to regularly submit draft submissions, or automatically generate an 'audit trail' based on students submission of files to the compiler (Wagner 2000). This material may provide useful insight in cases where plagiarism is suspected to have occurred. This approach to reducing plagiarism by monitoring the development process is closely aligned with the techniques proposed in this paper.

2.2 Technological approaches to plagiarism

An alternative to relying on the observational abilities of markers as the sole means of identifying cases of copying is to make use of automated or semi-automated software tools. These tools perform some analysis of the files submitted by students, and identify to the marker which of the files are most likely to have been generated via

plagiarism. They may also provide other information such a numerical measurement of the likelihood that plagiarism was involved in a particular submission, and the possible source of plagiarised material.

2.2.1 Content-based file comparison

These software tools operate by performing comparisons of the content of all the submitted files, either against each other or against alternative sources of material such as online resources. Some systems may apply linguistic techniques, looking for inconsistencies in the style of language used within a single submission as possible evidence of the material being written by multiple authors.

A variety of software tools based on these concepts are available, either for free or as commercial services. These tools also differ in whether they run as standalone applications on a local machine, or whether they require files to be uploaded to a remote host via a web-site.

The strength of the latter approach (as exemplified by websites such as turnitin.com (Turnitin (2004)) is the large and continually expanding database of papers they retain for future comparisons. One of the reasons these sites require the uploading of all files to be tested is so those files can themselves be added to the site's database. These tools are particularly well suited to identifying papers which have been plagiarised in part or in whole from online 'paper-mills': a growing number of web-sites which offer papers on a wide range of academic topics — again either for free or at a cost.

This content-based approach to plagiarism detection is best suited for essays, as the likelihood of two independently derived submissions containing large sections of identical text is low. Cases of plagiarism would thus stand out from the other assignments.

Hughes, Brown, Jakobson, Philpot, Dwight-Moore, Jarret, Grainger and Short (2002) assessed the CopyCatch software package for possible use within the University of East London. CopyCatch uses linguistic techniques to analyse a file's use of 'lexical words' (those words whose usage would be expected to vary between different authors). The study found that CopyCatch's performance was less effective in a problem domain which was technical in nature and in which similarities between student papers would be expected to occur; papers which were reported as having high similarity measures were later found not to be copied when examined by a human marker.

It is to be expected that these problems would be increasingly manifest when comparing student programs, due to the highly restricted nature of programming language vocabulary and structure compared to that of natural language. In this case it would be expected that most assignments would display quite high similarity measures. Hence distinguishing cases of plagiarism from this background level of expected similarity would require both increased effort and superior judgement on the part of the marker. Our experience with the use of the turnitin software for analysis of student programs bears out this expectation.

2.2.2 Content-based comparison of source code

There is a long history of development of software tools for determining the similarity of pieces of source code, dating back at least to Ottenstein (1976). Most early tools were based on various software metrics such as counts of the number of occurrences of particular operations. These metrics are only loosely related to the actual functionality of the code being examined, and as such had only limited success in the detection of plagiarised code.

Superior systems have more recently been developed which are based on analysis of the underlying structure of the code. Perhaps the two best-known and most widely used systems are JPlag (Prechelt, Malpohl and Philippsen (2002)) and Measure of Software Similarity (MOSS) (Aiken (2004), Schleimer, Wilkerson and Aiken (2003)). These parse the source code, tokenising it, and then apply comparison algorithms to the tokenised form of the code. They exhibit high levels of robustness to attempts to disguise plagiarism by discarding aspects of the code which are frequently edited post-copying such as white-space, comments and identifier names.

Chen, Francia, Li, Mckinnon and Seker (2004) however demonstrate that both JPlag and MOSS are susceptible to attempts to disguise code by adding extraneous lines which do not affect the function of the program. They propose an alternative comparison technique, inspired by those used in bioinformatics, which is less sensitive to this form of deception.

3 An Anti-Plagiarism Editor: Key Features

We endorse the view of Zobel and Hamilton (2002) that there is no 'magic bullet' in combating plagiarism. The most effective processes will be those which attack plagiarism from multiple perspectives using a number of complementary techniques. Thus the anti-plagiarism software described in this paper takes a significantly different approach from the systems described in the previous section.

We have developed a two-tiered approach: one which monitors the student's actions during the development of their submitted work, and one which audits the work after its submission. This is in contrast to the majority of existing plagiarism-detection systems which examine only the content of the submitted work, ignoring the process involved in its creation.

This approach of monitoring the file-creation process has some similarities to PowerResearcher - a commercial software package developed by Uniting Networks (Beasley 2004). PowerResearcher provides a research development environment which integrates many of the features required for research or coursework. It acts to reduce the incidence of plagiarism by providing tools to increase student productivity (thereby reducing the demands on student time which are often cited as a reason for plagiarism), and to aid in the automatic tracking and citation of sources of information, thereby reducing the

risk of accidental plagiarism. In addition, it aids in the detection of plagiarism by monitoring student research activity and maintaining time-based logs of the student's actions. These logs can be reviewed by academic staff members to ensure that they are consistent with the work actually submitted by the student.

The concept of monitoring the actual creation of the submitted work rather than simply examining the end-product is common to both PowerResearcher and our system. The primary difference is that PowerResearcher has a more general focus on supporting research activity (particularly the production of written work such as essays and reports), whilst our system is specifically designed for tasks involving the creation of source-code.

The key component of our system is a specialised text editor (the Anti-Plagiarism Editor or APE) whose use is mandatory for students in developing their programs. This editor incorporates features which firstly discourage plagiarism by preventing the lowest-effort forms of copying, and secondly aid in the detection of copied work through the inclusion of additional information about the code creation process along with the source code in the file to be submitted.

A second software component (which has been named Gorilla) is used by the marker to compare the students' submissions on the basis of this additional information. Gorilla is applied to a batch containing all student submissions, and performs pair-wise comparisons of these files. It has been shown through initial trials to conclusively detect files which have been produced via copying without any need for subjective judgement on the part of the marker, or any need for collaboration in the case of multiple markers.

The main aspects of the these two software components will now be presented.

3.1 File identification data

The key to the plagiarism detection capability of this software is the additional data stored whenever a file is saved. A file consists of the verbatim text created by the student within the editor (the source code of their program), together with header information related to the creation of the file. This header facilitates the identification of files which have been electronically copied from a common origin even if they have been the subject of later editing.

Various possibilities could be employed for this header data. The current implementation of APE maintains a record of the system clock time whenever the file is saved and appends a complete list of this information to the source code when saving. If this file is copied and subsequently edited, the lists of save times of the two files will be identical up to the point where the copy took place, enabling their common origins to be detected by Gorilla.

Obviously the inclusion of any additional data in the saved file will interfere with the processing of that file as source code by a compiler or interpreter. Two possibilities exist to address this issue.

One approach is to simultaneously save two different versions of the file — one containing just the source code for compilation purposes, and a second with both the source code and the header data. This approach introduces several complications — the existence of, and the consequential need to submit, two files is likely to confuse students (particularly at first-year level), and would impose additional work on Gorilla to perform initial checking to ensure that the source code in the two files was identical before carrying out any further analysis.

A simpler alternative, and the approach used in the current version of APE, is to store the identification data in the form of comments at the start or end of the code. This simplifies the submission and testing processes, but requires APE to be cognisant of the programming language used for the source code so as to be able to use the correct commenting syntax.

Note that in some cases a file may be distributed to students as a starting point for their solution — to support this Gorilla provides a facility whereby the marker can specify that the header information inherited from a specified file is not to be considered when comparing student files.

3.2 File authentication

In order to be compatible with compiler/translator systems APE produces plain text files. As the file identification data is clearly visible and editable within these files, the possibility exists for students to alter this data within another editor after copying the original APE file and hence bypass Gorilla's file comparisons.

The solution to this issue is two-fold: to encrypt this data and to include authentication codes in the text file. These authentication codes are based on both the source code and the file identification data. Whenever a file is loaded into either APE or Gorilla, the file's contents are used to generate an authentication code which is compared to that stored in the file. Any mismatch in these codes indicates that the file has been modified using an editor other than APE — if this occurs within APE an error message will be reported and the file will not be loaded into the editor. In Gorilla, in such a case the file will be brought to the immediate attention of the marker.

3.3 Restricted copy-and-paste facilities

The combination of file identification and authentication features effectively eliminates the easiest form of plagiarism — namely, direct copying of a file. Another simple plagiarism technique is, however, still available — the transferring of text from one file to another via a copy-and-paste operation. Therefore, it is necessary to prevent the importing of text into APE from another program (such as a web browser, email program or another text-editor), or the transfer of text from one file to another via copying-and-pasting between different editor windows within the APE application.

This could be achieved by not providing any copy and paste options within APE, but that would be undesirable

since APE should still support these operations within its own text-editing area for flexibility and efficiency of use. The solution chosen is to equip each window within APE with its own internal clipboard, and to source all paste operations from this local clipboard rather than on the system clipboard used by other applications.

It is then both safe and desirable to allow APE to export text to other applications. This would allow students to copy and paste part of their program into a text message when querying a lecturer via email rather than having to transmit their entire code file as an attachment. This feature is achieved by defining the copy operation within APE to copy the currently selected text to both the internal and system clipboards.

3.4 Summary

With these three key features, the APE/Gorilla software ensures that any attempt to plagiarise code via the forms of least-effort (file copying, and copy-and-paste) will be either impossible to perform or guaranteed to be detected.

A third alternative method of plagiarism is still available to students — that of manually re-typing the code to be copied. Restricting students to this option significantly increases the amount of work required to copy code and therefore should make plagiarism a less attractive option for many students.

The continued existence of this means of plagiarism means that the basic form of Gorilla cannot be used as the sole means of plagiarism detection. Some form of content-based analysis will still be required — either manual, via one of the software systems described in Section 2, or implemented in a future revision of Gorilla.

4 Implementation issues

The implementation of the APE and Gorilla tools required many decisions to be made, relating to both technical issues and the manner in which these tools will be deployed.

4.1 Integrating APE into current teaching

It is vital that the adoption of anti-plagiarism tools should not be at the expense of student's educational experience. Therefore APE should provide at least the same level of functionality from a student perspective as the alternative editors which are currently being used.

For this reason APE is based on the Eclipse development environment developed by the Eclipse Foundation, a consortium of computer industry bodies, including IBM (Eclipse Foundation, 2004). Eclipse is an open-source project released under the Common Public License, aimed at creating an industry quality integrated software development environment. Basing APE on Eclipse provides immediate support for advanced development environment functionality such as syntax highlighting, automatic indentation, search-and-replace operations and debugging support. Eclipse is also based on a plug-in architecture, which aided in integrating APE's required functionality into the existing Eclipse structure.

The functionality provided by Eclipse is comparable or superior to that of the existing editors used in many units where APE is planned to be introduced. Eclipse is also expected to be widely adopted in industry, and therefore students gain the additional benefit of being exposed to a tool which they may well be required to work with post-graduation.

4.2 File identification and authentication data

As discussed later in Section 6.2, the possibility exists for the extension of the functionality of APE and Gorilla in the future by including additional items in the file identification data. To support this future extension the format used to represent this data within APE files is based on an XML schema (Bray, Paoli, Sperberg-McQueen, Maler and Yergeau (2004)), rather than a simpler, unstructured representation. In this way any future additions to the data being gathered can be added as optional tags within this schema, thereby ensuring backwards compatibility between future versions of Gorilla and earlier releases of APE.

The file identification data is used as the input to an algorithm which both encrypts the data and generates an authentication code, which is appended to the encrypted data. The details of the encryption and authentication algorithm are deliberately left obscure both here and in the code release of APE, so as to maximise the difficulty for any dedicated plagiarist attempting to subvert this aspect of APE. Suffice it to say that APE uses third-party encryption code which has been rigorously tested, and which provides a level of encryption based on a key-size large enough to make APE secure from anything short of a concerted attack.

The encrypted data and authentication code are binary in nature and therefore cannot be directly written to the text files produced by APE. Therefore this data is piped through an ASCII converter prior to being written to file.

Figures 1 to 3 illustrate various views of a file produced by APE. Figure 1 shows the actual content of the file, as it would be displayed if it was opened in a generic text editor. The series of comments preceding the source code contains the encrypted, ASCII-converted header data.

```
; ----------START-OF-AUTO-GENERATED-----DO-NOT-ALTER----------
; cVaBc8c0c0c0c0c0c0c0c1dGc1alb_dEaTducldudbcGa0bta3bpcEcUbtbwdZdDdEa4dxdRb_dHdSbw
; bUcjd9d2a9dxaWd_c0bLciapb8cWdBdSbCasbpbadtbJatbXd0bYc0aXbuc0d3dHbCczdxcpdgbJcJaY
; dBbydXc1bncpcJdxcEaXbTc8dSaKdyb0bhbjd8aeasaaaqbKaWaCdvbCcUbKbdcNcLducDaDbRdDdfd0
; blbidEbHaza3caavaTbldKcAd9bUbCcldAaAbQaxbDaBaFaDbGc5alb0bicEc7czc7dlbndsbCa6cmc8
; azcfcPbQa4cyaqbFb3draKa0avbqdPbiaja0bSdDcSb7cVchblcid0d4agdaaRbHc5cwambNdsbFcQdx
; dLclaHb8bicbcMahbwc9aJbLckcTbsdhdPbnc_dpdtbCbPd3aNbrd5bsaybEcBbqcDaecscwa0cuaqdL
; ddcicPcjdVcPdUc4c4d6b6dVaic7b9audjaJcVd6b_cMbvc1duanbVcKada6byb7aMayb2cqc6dSbzc8
; aHciakdkdUbSdectbVcUcCc4ducMbMagb1awczc0dcbubicnc8cqcac0cvdTb0czaJbbcdaMaWc5a_cs
; dvbyaqa0cXcLdsdDaRdFdBdlc5dCcjdRcNd4cvczdXcTduaCbgclccb2a8d9cAdka0crb8dGc1c0c0
; ------------------------------END-OF-AUTO-GENERATED-------------------------------
(defrule Test
   (initial-fact)
=>
   (printout t "Hello World" crlf)
)
```

Figure 1: A sample text-file produced using APE, showing the encrypted header information embedded in comments at the start of the file

Figure 2 is a screenshot from APE, showing that the header information is not displayed within this view.

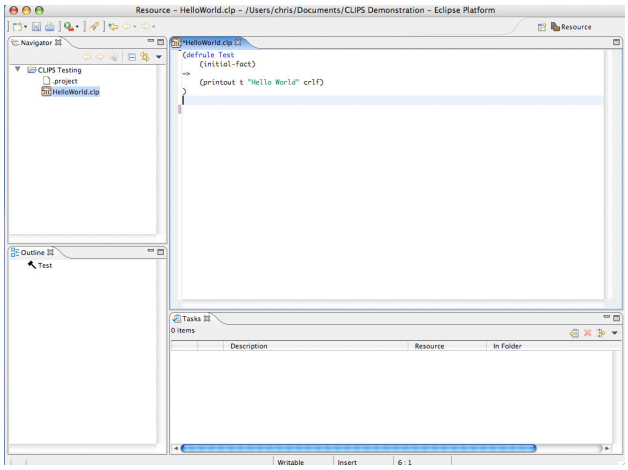

Figure 2: The same file as in Figure 1, as it would be displayed when loaded in APE.

Finally Figure 3 shows the decrypted header information as it would be displayed to a marker using Gorilla. It can be seen from the left-hand pane that this particular file has been saved twice. The details of the first of these save operations is displayed in the right-hand pane.

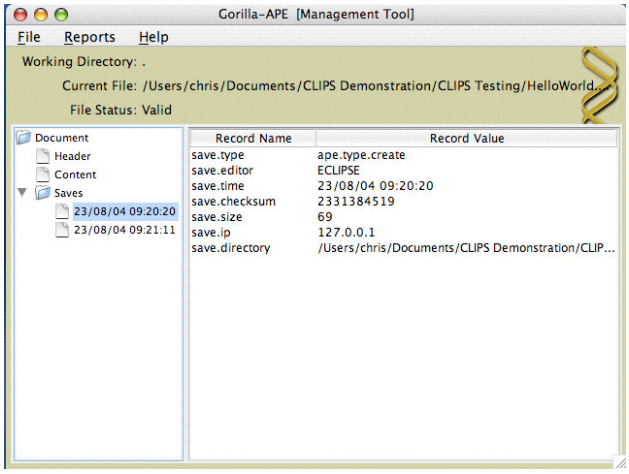

Figure 3: The same file as in Figures 1 and 2, as it would be displayed when loaded in Gorilla.

5 The role of APE and Gorilla

APE and Gorilla are not intended to be the sole approach to minimising plagiarism, but instead merely an additional component within the School of Computing's existing structure.

The School's current approach can be characterised as consisting of three stages:

- educating students about the nature and unacceptability of plagiarism;
- making the cost/benefit ratio of plagiarism less attractive to students; and
- detecting and punishing cases of plagiarism when they do occur, according to the University's Ordinances and Rules.

5.1 Student Education

Education of students forms the first stage in combating plagiarism, with the aim of deterring plagiarism from occurring rather than catching it after it has taken place. As part of a University-wide policy, all students enrolling in each unit are provided with a Unit Handout at the start of the semester. As well as detailing course content, assessment details and assignment deadlines, this handout includes a discussion of plagiarism. It explains what actions are regarded as constituting plagiarism, why these actions are unacceptable, and the possible penalties for students who indulge in these activities.

In addition, it is required that every piece of work submitted for assessment is accompanied by a signed cover sheet attesting that the submission is entirely the work of the individual or group making the submission. Assignments submitted without a signed cover sheet will not be marked. This acts to remind students of the seriousness with which plagiarism is treated.

As mentioned earlier, Braumoeller and Gaines (2001) found that warnings had little impact on student behaviour unless backed up by a reasonable expectation that plagiarism would be detected. Therefore the simple act of notifying students of the adoption of APE and outlining its basic functioning may serve as a more significant deterrent to plagiarism than the current unit handouts and cover sheets.

5.2 Reducing benefits and increasing costs of plagiarism

Regardless of the underlying factors leading to plagiarism, the final decision to plagiarise is in order to achieve a higher mark than would be obtained by applying the same amount of effort legitimately. Therefore, the incidence of plagiarism should be reduced if either the benefits to be obtained from it are reduced, or if the amount of effort required is increased.

The primary means of reducing the potential benefits of plagiarism is to reduce the contribution of assignments to the final grade for the unit. It would be expected that students who cheat on the assignment will learn less as a result, and therefore will be more likely to fail the exam. This line of reasoning is emphasised to students at the start of courses as a disincentive to engage in plagiarism.

As discussed in Section 3.4, APE complements this approach by substantially increasing the amount of effort required to carry out plagiarism. With no means of electronically copying code without being detected, plagiarists will be forced to re-type any code which they wish to copy.

5.3 Detection and punishment

The School's current approach to detecting plagiarism relies mainly on manual detection by markers, although the increased use of multiple markers has lead some lecturers to experiment with content-based plagiarism detection tools such as turnitin.

The University statutes dictate the disciplinary action to be taken once an act of plagiarism is detected. The most common penalty is the loss of some or all of the marks which would otherwise have been achieved for the assignment. Penalties of greater severity are available for more serious offences or for repeat offenders.

In theory Gorilla's role would be to detect any cases of plagiarism which have resulted from direct copying of files. In practice however, given that the outline of APE and Gorilla's operation will have been explained to the class prior to the assignment, it is anticipated that very few students will be foolish or desperate enough to still carry out such actions. Therefore Gorilla's impact will be largely via its influence on student behaviour rather than its actual detection capabilities (this raises the interesting question as to whether plagiarism detection software actually has to fulfil its specifications, or indeed exist at all, in order to be an effective deterrent). This deterrent effect is of course the preferred option — to prevent plagiarism from being carried out in the first place rather than to detect it only after it has occurred.

In terms of disciplinary action, Gorilla's only impact would be if the file identification data allowed the original author of the submitted material to be identified, as discussed below in Section 6.2. This may then have some influence on the penalties assigned to the parties involved in the case. Such information is readily available under many operating systems and may easily be incorporated in future revisions of APE.

6 Future work

6.1 Assessing APE and Gorilla

Now that initial, ex situ trials have been completed successfully, APE can be deployed for classroom evaluation. The software will be trialled in the third-year unit Knowledge-Based Systems in the first semester of 2005. The trial will examine APE's comparative ability to detect plagiarism in competition with other techniques such as manual inspection of assignments and tools such as JPlag and MOSS. It will also assess APE's effect on student behaviour and its level of acceptance by students.

The first aspect of the trial will be achieved using a methodology similar to that of Prechelt et al. (2002). The actual submissions made by students will be augmented by the creation of additional submissions produced by copying some randomly selected genuine submissions, and applying the code disguising techniques commonly observed in plagiarised assignments, as well as techniques designed to circumvent Gorilla's testing process. All submissions will then be distributed to the markers, and the usual assessment practice will be followed. In addition the files will also be submitted to the JPlag and MOSS tools, and processed via Gorilla. A comparison will be made between the ability of these various techniques to detect plagiarised files, both those which were seeded and any (unknown) genuine cases.

The impact on student behaviour will be assessed via a pair of anonymous surveys of students, conducted at the commencement and the end of the unit. The first survey will address student attitudes to plagiarism, in a manner similar to the surveys performed by Sheard et al. (2002, 2003). It will also measure students' perception of the likelihood that various forms of plagiarism would be detected. This survey will be conducted prior to making students aware of the existence and functionality of APE and Gorilla. The follow-up survey will be conducted at the end of the unit, after students have experienced using APE. It will measure any changes in students' attitude to plagiarism since the first survey, and will also assess their opinion of the APE tool itself.

6.2 Extending functionality

The most obvious extension to Gorilla would be to implement content-based analysis of the submission files, so as to complete its plagiarism detection capabilities. This could be achieved by linking Gorilla to the existing JPlag code or integrating it with a Web-based detection service rather than re-implementing this functionality.

Another advanced capability would be to incorporate into the file further details about the process involved in creating the file. Gorilla could then analyse this information to identify files which are more likely to have been produced via plagiarism as indicated by the behaviour of the user in creating them. For example unusually frequent use of the search-and-replace facilities (particularly late in the creation of the file) may indicate changing of variable names to reduce the similarity to the file from which this code was copied. Similarly an unusually sequential creation of the file may indicate a file being re-typed from a printout of another student's code. These behavioural signatures alone would not be sufficient evidence to prove plagiarism, but they could be used to flag some files as being worthy of closer attention by the marker.

One means of supporting this form of analysis would be to more closely capture the relationship between the source code and the save times, by storing the source code in terms of the modifications made since the previous save. In this way the contents of the code at each save point could be reconstructed, which would allow a comparison to be made of the two files at their point of divergence, to establish exactly how much of the code content was plagiarised. The major issue to be resolved is how this additional information can be captured without significantly increasing the size of the resulting files.

Another item of information which would be useful in dealing with cases of plagiarism is the identity of the original author of the code, as mentioned above. An alternative might be to save the MAC address of the computer on which APE is executing at the time of each save operation. This additional data could prove extremely useful in situations such as the 'mytutor' case (Zobel 2004), where an 'external tutor' sold assignment solutions to a large number of students in the same class — these submissions could have been flagged for further attention as they would most likely all have been prepared on the same non-university machine.

7 Conclusion

APE and Gorilla form a two-stage system for reducing the incidence of plagiarism in software development courses, in conjunction with existing anti-plagiarism strategies. They can be applied to any units where a stand-alone text editor or development environment is currently used for creating program files. They are based on the inclusion of file identification and authentication data along with the source code when files are saved. This additional data facilitates the detection of files which have a common ancestry, regardless of any subsequent editing of those files, and thereby significantly increases the amount of effort required for one student to plagiarise another's code.

An implementation of APE and Gorilla based on the Eclipse open-source code-base has been completed, incorporating all of the key features as described in Section 3. This software will be trialled in the teaching of the Knowledge-Based Systems unit in the first semester of 2005.

8 Acknowledgements

The development of APE and Gorilla was funded by a University of Tasmania Teaching Development Grant, and carried out by Chris Dalton of ALife Consulting (www.alife.com.au). We wish to acknowledge the suggestions made by Chris Dalton and Professor Arthur Sale.

9 References

Aiken, A. (2004): MOSS: A System for Detecting Software Plagiarism, http://www.cs.berkeley.edu/~aiken/moss.html, Accessed 10th August 2004.

Beasley, J. B. (2004): The Impact of Technology on Plagiarism Prevention and Detection: Research Process Automation, A New Approach For Prevention, *Proc. "Plagiarism: Prevention, Practice and Policies 2004": Joint Information Systems Committee Plagiarism Advisory Service Conference*, June 28-30, Newcastle, UK.

Braumoeller, B. and Gaines, B. (2001): Actions Do Speak Louder than Words: Deterring Plagiarism with the Use of Plagiarism-Detection Software. *PS: Political Science and Politics* **34**(4):835-839.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E. and Yergeau, F. (eds.) (2004): Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/REC-xml/, Accessed February 2004.

Chen, X., Francia, B., Li, M., Mckinnon, B. and Seker, A. (2004): Shared Information and Program Plagiarism Detection, *IEEE Transactions on Information Theory*, **50** (7): 1545 - 1551

Culwin, F., MacLeod, A. and Lancaster, T., (2001): Source Code Plagiarism in UK HE Schools - Issues, Attitudes and Tools, Technical Report SBU-CISM-01-01, South Bank University, 2001.

Eclipse Foundation, http://www.eclipse.org, Accessed 20 August 2004.

Hughes, G., Brown, S., Jakobson, M., Philpot, C., Dwight-Moore, P., Jarrett, N., Grainger, T. and Short, B. (2002): Report on the Viability of CopyCatch Plagiarism Detection Software, internal report, University of East London.

Ottenstein, K., (1976): An algorithmic approach to the detection and prevention of plagiarism, *SIGCSE Bulletin*, **8**:4: 30-41.

Prechelt, L., Malpohl, G. and Philippsen, M. (2002): Finding Plagiarisms Amongst a Set of Programs with JPlag, *Journal of Universal Computer Science*, **8**(11): 1016-1038.

Schleimer, S., Wilkerson, D. and Aiken, A. (2003): Winnowing: Local Algorithms for Document Fingerprinting, *Proc. SIGMOD 2003*, June 9-12 2003, San Diego.

Sheard, J., Dick, M., Markham, S., Macdonald, I. and Walsh, M. (2002): Cheating and plagiarism: perceptions and practices of first year IT students, *Proc. 7th Annual Joint Conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark: 183-187.

Sheard, J., Carbone, A. and Dick, M. (2003): Determination of Factors which Impact on IT Students' Propensity to Cheat, *Proc. Fifth Australasian Computing Education Conference*, Adelaide. Australia :119-126.

Stokes, F. and Newstead, S. (1995): Undergraduate cheating: who does what and why? *Studies in Higher Education* **20**(2):159-172.

Turnitin (2004): http:///www.turnitin.com. Accessed 26 October 2004.

Wagner, N., (2000): Plagiarism by Student Programmers, http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html Accessed 10th August 2004.

Weinstein, J. and Dobkin, C. (2002): Plagiarism in U.S. Higher Education: Estimating Internet Plagiarism Rates and Testing a Means of Deterrence, http://webdisk.berkeley.edu/~Weinstein/Weinstein-JobMarketPaper.PDF. Accessed 15 August 2004.

Zobel, J. and Hamilton, M. (2002): Managing Student Plagiarism in Large Academic Departments, *Australian Universities Review* **45**(2):119-126.

Zobel, J. (2004): "Uni Cheats Racket": A Case Study in Plagiarism Investigation, *Proc. Sixth Australasian Computing Education Conference*, Dunedin, New Zealand.