Teaching Cheating — To let the Assignment fit the Crime

Rick Duley
North Perth, Western Australia 6006
rickduley@gmail.com

ABSTRACT

Plagiarism persists throughout academia. It is like a lunatic asylum — everyone knows it exists, no-one wants to be associated with it. Students (and, regrettably, faculty) are regularly exposed as cheats and frequently see their careers ruined as a consequence. Much of the problem appears to be that few really understand what constitutes plagiarism, which frequently gives rise to pleas that the offence was committed inadvertently.

Proactivity in this field appears to be scarce. Many commentators refer to detection and punishment but research has revealed little work in the area of stopping the issue arising. Given that some training in computer programming has become 'de rigueur' in most undergraduate programs, this paper presents a programming approach to addressing the general problem of plagiarism. In short, it advocates the policy of setting a thief to catch a thief, of using a study of plagiarism, and an assignment to create cheat-catching software, as a salient warning to eschew the practice.

Categories and Subject Descriptions

K.3.2 Computer Science Education — Information Systems Education
K.4.1 Abuse and Crime Involving Computers — Ethics — Intellectual Property Rights
K.5.1 Copyrights — Patents — Proprietary Rights
K.7.4 Professional Ethics

General Terms

Documentation, Legal Aspects

Keywords

Plagiarism, programming

1 INTRODUCTION

> *"Talk to members of staff in any university and — particularly if you close the door behind you — deep concerns about student cheating and plagiarism will often emerge."* [2]

Plagiarism has certainly been known for centuries and its incidence shows no sign of diminishing. It strikes down people in all phases of a career, knows no *'statute of limitations'* and tends to be treated with some severity. In 2002 a Vice-Chancellor of Monash University in Melbourne (VIC) resigned on the exposure of the commission of plagiarism some twenty years beforehand. Courts are rigorous in their treatment of the matter as Standler noted in 2000:

> *"In every plagiarism case I have found involving a student or professor, the court upheld the punishment imposed by the college. Further, the court often make gratuitous, pejorative comments about the bad character of the plagiarist, which show that it is unwise for a plagiarist to complain how he/she was treated."* [3, p.14]

Few universities are likely to trumpet their plagiarism incidence from the bell tower but neither would many boldly assert that, *"That doesn't happen here!"*. Recent communications with national and international colleagues revealed no-one who denied the existence of the problem — even when the specific subject was plagiarism amongst faculty! As it is, in general, the subject seems to be one of those legally discussed between consenting adults in private (though not, one assumes, by Ladies).

While some reported cases of plagiarism are undeniably deliberate [4], commentators often mention inadvertent plagiarism as a major cause — either the perpetrator does not know how to properly credit material from an external source ([5, p.2], [6]) or they simply don't know what plagiarism is ([7], [8, p.17]) or they are not aware of the benefit which accrues from citation:

> *"Citing a source, whether paraphrased or quoted, reveals that they have performed research work and synthesised the findings into their own argument. Using sources shows that the student engaged in 'the great conversation', the world of ideas, and that the student is aware of other thinkers' positions on the topic."* [9, p.3]

This paper takes the view that much of the blame for the plagiarism lies squarely with the educators and that the exhortation, *"The best way to prevent cheating, plagiarism and collusion is the threat of detection."* [2] (and, presumably, consequent punishment) strains the boundaries of natural justice — that ignorance of the law may be no excuse but educators are supposed to be dispellers of ignorance, not judges. It is a proactive view based on demonstrated reasons why students (and, presumably, graduates) don't cheat as shown in Table 1 (scale tops at 1.000).

2 PROACTION

Firstly, one must solve problem of who is to dispel the ignorance and in which context, of who will cheerfully allot a portion of the limited time at their disposal to a topic of importance to all but which *'should have been dealt with ages ago'* and, of course, *'by someone else'*? The only answer is, *"I will!"* We may not be able to do anything about students who would rather copy than take the time to do the work [10] but we cannot ignore the larger issue. Taking a proactive stance on the matter, ensuring that students think about the issue and have the skills and determination to avoid it, permitting the positive factors of pride and sense of self-worth to take over, is an attractive alternative.

2.1 Teaching about Plagiarism

Examples of plagiarism and its possible consequences are not hard to find, for example:

2.1.1 Sic gloria transit!

Saturday, July 6, 2002: readers of Melbourne's *'The Age'* scan the

lead article headline *'Plagiarism: Fresh claims against Monash uni head"*. One of the Monash vice-chancellors was accused by a junior colleague of copying another person's work into his 1976 book *'Drinking to Alcoholism: a Sociological Commentary"* — a charge supported by highlighted copies of both texts revealing several passages copied. This accusation followed the revelation by *'The Times'* of London two weeks previously that the accused had admitted plagiarism in books published in 1979 and 1983. Despite these claims the Monash University Council had, the previous week, passed a unanimous motion of confidence in their vice-chancellor. *'The Age'* requested a research fellow at the University of Melbourne's Centre for Applied Philosophy and Public Ethics to examine the texts at the heart of the new accusation.

> *" 'A lot of academic work borrows in some way from other work and it's very difficult to draw a sharp line, but on the face of it, it looks as though it's on the wrong side of the line.' he said. '...Plagiarism is important because it's dishonest. There's a particular issue with academic plagiarism; you're holding yourself out to have expertise you don't have. To what extent is* [the accused vice-chancellor] *an authority on the sociology of alcoholism?' "* [11]

One week later, *'The Age'*, under the headline *'Quiet rejoicing in Monash corridors'*, reported the accused's *'precipitate departure'* apparently when he was asked by the University Council to resign [12].

2.2 Programming against Plagiarism

Undergraduate programming assignments are often thought of as having no relevance to anything else in undergraduate studies, yet nothing *should* be further from the truth. Undergraduate programming assignments are often thought of as being trivial, yet nothing *need* be further from the truth. Undergraduate programming assignments are often thought of as having no practical use, yet (if the assignment is to produce plagiarism detection software and use it) nothing *is* further from the truth.

2.2.1 Code Plagiarism

Detection of plagiarised source code is different from detection of free-text plagiarism. For one thing programs are constructed with a restricted vocabulary (there are only 69 reserved words in Ada95 and Ada is considered to be a *large* language) and, for another, programming language syntax is precisely defined. Aside from differences associated with proprietary compilers in some languages and versions of languages, in general there are no equivalents for vernacular, idiom or slang nor variations in spelling which make free-text documents distinctive. Furthermore, programming assignments are normally quite strict in their requirements specification (which reduces the opportunity for idiosyncratic variation) and the number of executable statements required is low (measured in tens rather than thousands of lines). As a result, especially in an *ab initio* unit, submissions have a high, and perfectly legitimate, level of similarity — there are only a few ways to write *'Hello World'*!

On the positive side, these very restrictions isolate those characteristics which make source code distinctive:

- Source Lines of Code:
  - Total number — elegance of solution;
  - Percentage of total text lines — clarity of solution;
- Layout style:
  - Use of white lines — delineate code artefacts;
  - Frequency of comment lines — define the purpose of code artefacts;
  - Frequency of in-line comments — describe phases of the process;
- Reserved words:
  - Number in use — extent of programmer's vocabulary;
  - Frequency of use — programmer's articulation of the algorithms;
  - Words used — can identify number of subroutines, iterations and conditionals;
- Operators:
  - Range, occurrence and identity — define the algorithmic solution;
- Identifiers:
  - Appropriate self-description — readability of the program;
  - Consistency — degree of forethought and design.

2.2.2 Detection Software

Programs to detect source code plagiarism have been around for decades. Early versions reported suspicious cases on the basis of counts of program characteristics like those listed above and of linguistic comparison metrics like Zipf's Law. [13, p.517] Later models worked on structure metric systems such as Halstead's Software Science Metrics which used counts of operators and operands and these types have been shown to be more effective. [10, p.2] Code plagiarism detection systems are available on-line, e.g. *'Measure of Software Similarity'* or MOSS (http://www.cs.berkeley.edu/~moss/general/moss.html). Free-text plagiarism detection software which scans millions of websites for text comparison is also available (for examples see http://www.plagiarism.org, http://www.integriguard.com or http://www.copycatch.freeserve.co.uk).

2.2.2.1 Mapping the Task to the Student Cohort

Of necessity, these early attempts presented their reports on character-screens or in line-printed documents but the ready availability and usability of graphics packages means that relatively simple programs can now present reports in graphical displays. While graphical presentation does not allow for easy automation of the detection process it provides a convenient and tangible indication of suspect code. Research has not revealed a detection application which does not, in the end, require a supervening human decision.

Table 1: Reasons why Students Don't Cheat [1]

Scenario	Self worth, Pride	Unaware, moral values, religious beliefs	Fear of punishment, consequences
Want to know what your work is worth	0.749	0.194	0.134
Pride in your work	0.880	0.160	0.081
Can get good marks without cheating	0.673	0.189	0.195
Against your moral values	0.453	0.464	0.192
Against your religious beliefs	0.220	0.552	0.181
Fear of being found out	0.140	0.306	0.653
Never thought about it	0.204	0.666	0.132
Don't know how to	0.070	0.739	0.132
Fairness to other students	0.252	0.475	0.382
Penalties if caught are too high	0.173	0.189	0.885

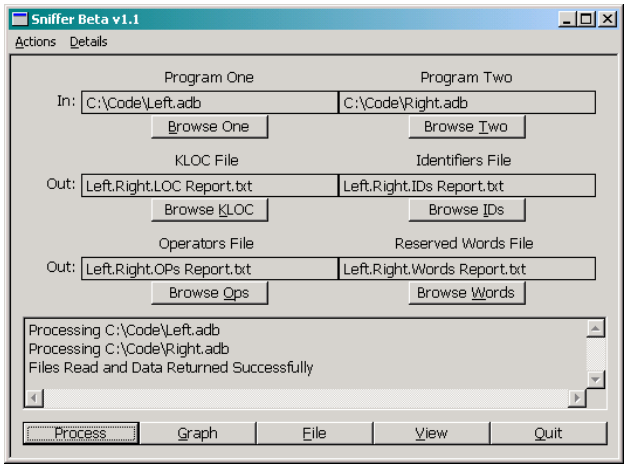

Figure 1: Sniffer GUI

Use of graphics also means that an assignment to devise relatively primitive plagiarism detection software can be tailored to the student body for whom it is set. If a GUI like that shown in Figure 1 is not within the capability of the students for whom the assignment is set it can be supplied as a support package. If the graphing window with its different graphing processes and the primitive *'Close'* button (Figure 2) is beyond their reach then it too can be supplied as a support package. For genuinely *ab initio* students for whom some of the more complex the text handling problems would prove insurmountable it would be simple enough to supply a support package with text comparison subroutines. for them to use. On the other hand, if the student group can make all that look easy, add the requirement for the program to test all files in a folder against each other automatically and/or report suspicious cases on the basis of some software metric.

Sniffer was created using parallel programming (tasking) to bypass the inhibiting effect of relatively slow I/O on the speed of program execution and this requirement could add a further degree of difficulty to an assignment if it was applicable. If the principle focus of the assignment is plagiarism then programming may be completely secondary or of considerable importance as the teacher sees fit and as the class is capable.

3 GENERAL CONCEPT OF THE ASSIGNMENT

'Sniffer' was named after those cute Beagles which clamber all over the luggage at LAX and is designed to compare two source code files and display the comparison data. It was written in Ada95 for use on Ada95 source code using John English's JEWL (http://www.it.bton.ac.uk/staff/je/jewl/download.htm) for the GUI and Jerry van Dijk's AdaGraph (http://users.ncrvnet.nl/gmvdijk) for the graphical display. It could be set in any language with the GUI and graphics capability.

3.1 The Interface:

Beneath each of the six editboxes (see Figure 1) is a button which allows browsing for the relevant file. Browse for the program which will be the basis for comparison in the editbox labelled Program One then a different program in the editbox labelled Program Two. The remaining editboxes will fill with automatically generated filenames. All editboxes are editable so filenames may be altered or entered manually while Menus allow selection of all operations.

An action record is displayed in the memobox and a detailed log file is generated in the working folder.

3.2 The Functions:

- 'Process' reads in the two program files and extracts the data. You must 'Process' before using 'Graph', 'File' or 'View'.
- 'Graph' opens a graphics window and displays comparative graphs of the characteristics of the programs.
- 'File' stores the data in separate files. You must 'File' before you can 'View'.
- 'View' opens a push-button window which in turn allows you to select which set of data to peruse.
- 'Quit' (surprise ☺) ends the program run. You must close all child windows before you can 'Quit'.

3.3 Outputs

All text outputs in parallel columns showing comparative data for the code segments analysed:

- KLOCC file showing:
  - number of text lines;
  - number of executable lines of code;
  - number of comment lines;
  - number of in-line comments;
  - number of white lines.
- IDs file showing the identifiers used by the programmers and the number of times they appear in the code.
- Ops file showing the operators and the frequency of their usage in the code.
- Words file showing the language reserved words and the frequency of their appearance in the code.
- Graphical display showing the statistics from the files for both code segments (Figure 2).

3.4 Using Sniffer:

- Select base program in 'Program One'.
- Select program for comparison in 'Program Two'. This must be a different program from 'Program One'.
- Display Graphs. If there is obvious variation 'File' and 'View' written data..
- 'File' files the data in the named files in the working folder. Printing this data for study will clarify the extent of any similarity.
- 'View' gives a quick preview of the recorded data.

Whether or not an example like this is illustrated and/or demonstrated to the students is a matter for the teacher concerned but it is the author's opinion that students should be encouraged to express their individuality and flair. Giving students a specific example may have the effect of stifling creativity as students merely attempt to reproduce what they were shown(which may not set a great standard anyway).

3.5 Focus

Essentially, Sniffer is an exercise in text handling — words and other lexical tokens (e.g. the operators `>=`, `**` or `xor`) are found and counted. Data structures play a large role in the solution, however. One of the major problems is keeping track of the identifiers. In the case of operators or reserved words the number and identity of the lexical tokens is foreknown and the related data structures can be static, but this is not the case for identifiers. (Figure 2 shows a wide disparity in the number of identifiers used in the lower line graph.) Solution to this problem, therefore,

involves an abstract data type using dynamic memory allocation. Operators and reserved words constitute a different problem since (in Ada at least) they cannot be used for anything other than their cited purpose so, for example, an enumeration type cannot be defined using reserved words or operators to provide constants to which the lexical tokens may be compared (Ada does not have a predefined 'Set').

3.5.1 Learning about Plagiarism

Using a programming assignment to drive home lessons about plagiarism depends for its usefulness on the emphasis placed on the Requirements Analysis phase of Software Development. Put simply, it is not possible for someone to develop software to detect plagiarism unless they have a sound understanding of what plagiarism is. In the case of software intended to detect source code plagiarism, the developer must also have a sound understanding of the characteristics of source code which make an example of source code distinctive and make masking of plagiarised code difficult.

3.6 Suggested Deliverables

- Major prose assignment (40%) due the Monday before mid-semester break — including a bibliography with abstracts and a reading list of material not cited — examining and explaining:
  - any topic-specific terminology in your essay.
  - attitudes to plagiarism;
  - authorship attribution;
  - collaboration;
  - collusion;
  - double jeopardy;
  - factors in the modern world which aid or inhibit plagiarism;
  - forensic linguistics;
  - intellectual property;
  - methods of plagiarism detection;
  - paper mills;
  - penalties applicable to plagiarism;
  - plagiarism and the courts;
  - self plagiarism;
  - signs of plagiarism;
  - the borderline between plagiarism and legitimate research;
  - the legal, professional and moral implications of plagiarism;
  - the treatment of plagiarism and the application of copyright law in different countries;
  - which excuses for plagiarism are not excuses for plagiarism.
- Minor prose assignment (15%) due first teaching day after mid-semester break covering:
  - distinctive characteristics of code
  - problems with proving source code plagiarism;

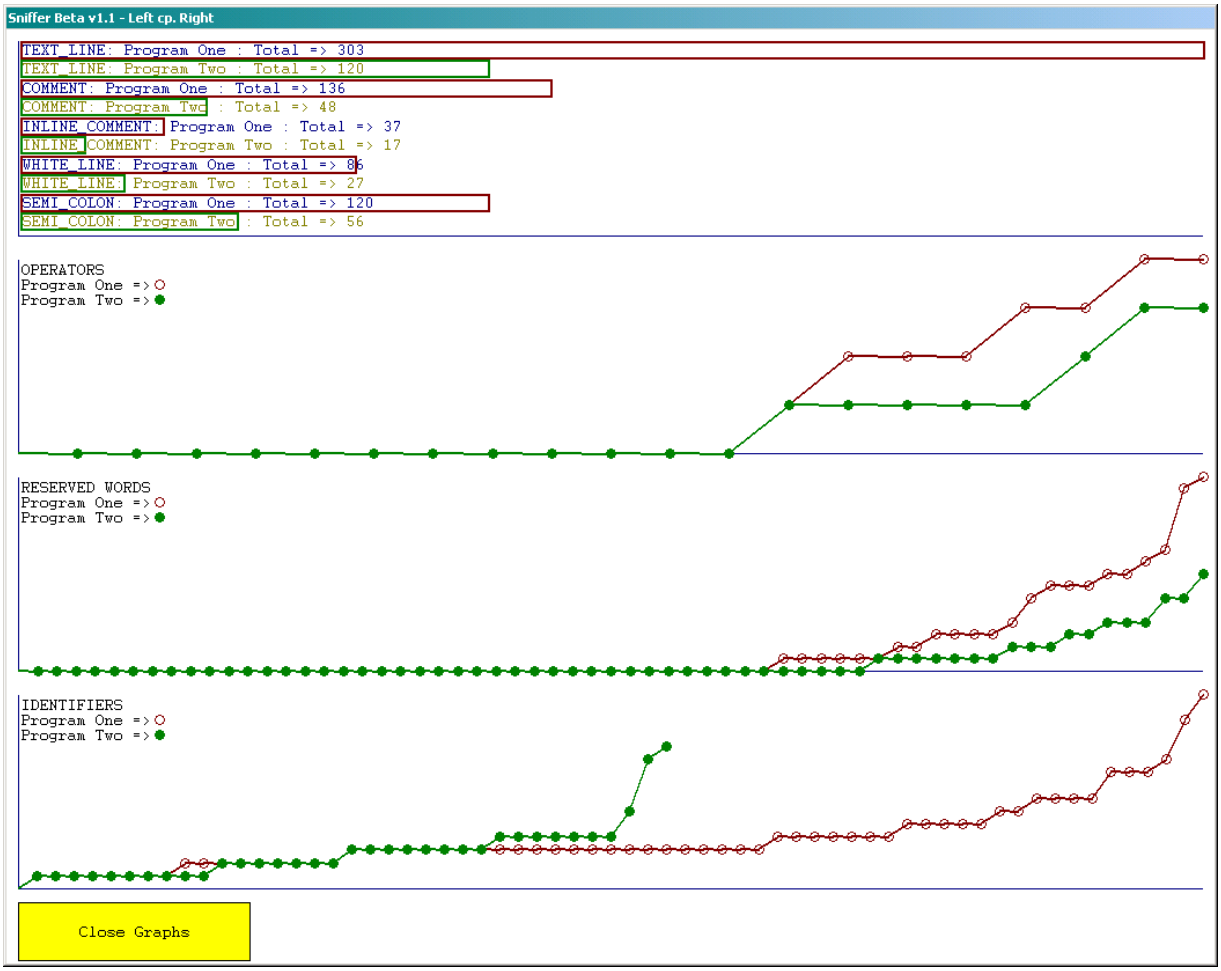

Figure 2: Sniffer Graphical Output

- Software Application (40%) due Monday prior to demonstration, including
  - Soft copy of all source code including:
    - main body of your program;
    - all supporting modules written by you;
    - all supporting modules used by you which are:
      - not written by you (including source information and checkable access);
      - not supplied in the LAN as part of the unit;
      - not specifically recommended as part of the unit.
  - Hard copy of:
    - List File of the main body of your program;
    - List Files of all supporting modules written by you;
    - printed data output from comparison of the sample source code supplied;
    - graphical data output from comparison of the sample source code supplied.
  - Documentation:
    - Technical Documentation;
    - User Documentation.
- Demonstration (5%) in normal Lab before swot vac — not available for applications handed in late.

4 FINAL TWIST

If there is a danger in this assignment it is that it might give some of our rebels a cause. For some there will always be a fascination in going against the system merely for the thrill of getting away with it. Hopefully, nothing will ever change that, such students are an integral part of the fabric of university society. However, there is a tactic which might lessen that risk and have all the students *'on the side of the angels'* — include in Section Four the announcement that the demonstration will consist of each application being used by someone other than the developer to test four or six randomly assigned submitted programs against each other. Tabulation of the collected results should reveal any submissions which merit further investigation but one can expect two other effects. Firstly, students will be more reluctant to cheat knowing that their work will be examined (by their peers) and, secondly, students who are assigned to play the part of the *cop* may be less likely to play the *robber* at the same time. (**Note:** it may be necessary under local laws and university regulations to have the students sign a copyright waiver at the commencement of the unit to permit the storage and handling of their intellectual property in this manner.)

5 SUMMARY

Setting a programming assignment in plagiarism detection can serve two purposes — to examine the programming skills of the student body in the normal manner and also to expose the students to the subject and effects of plagiarism in a manner calculated to turn them against the practice and to encourage them to develop their skills in research and citation. It would also give them experience in including researched material in their work and in being part of Harris' *'Great Conversation'*. Such an assignment can be tailored to suit students at virtually any level above *ab initio* and, if set early in the course, would have a spill-over effect into all the other subjects the student studies in that course. Universities stand to benefit from a decrease in student plagiarism, students stand to benefit from improved assignment-writing skills early in their careers while examination of student programming skills is unaffected.

6 REFERENCES

[1] Sheard, J., Dick, M., and Markham, S. Questionable Work Practices: Comparison of the Views of Undergraduate and Postgraduate Students in IT Courses.: Published - 2001. [Web Page] Accessed: September 20, 2002; URL http://www.csse.monash.edu.au/~mdick/ITICSEWorkingGroup/index.html

[2] Moon, J. (Staff Development Officer) How to stop students from cheating. The Times Higher Education Supplement: Published - 99. [Web Page] Accessed: November 1, 2002; URL http://www.thes.co.uk

[3] Standler, R. B. Plagiarism in Colleges in USA.: Published - 2000. [Web Page] Accessed: October 15, 2002; URL http://ww.rbs2.com/plag.htm

[4] Ryan, J. J. C. H. Student Plagiarism in an Online World.: Published - 98. [Web Page] Accessed: October 17, 2002; URL http://www.asee.org/prism/december/html/student_plagiarism_in_an_onlin.htm

[5] Clough, P. Plagiarism in natural and programming languages: an overview of current tools and technologies. University of Sheffield, Sheffield (UK): Published - 2000. [Web Page] Accessed: October 17, 2002; URL http://www.dcs.shef.ac.uk/~cloughie/papers/Plagiarism.pdf

[6] Why do Students Plagiarise. Joint Information Systems Committee [Web Page] Accessed: November 1, 2002; URL http://www.jisc.ac.uk/plagiarism/whyplagiarise.html

[7] Fain, M. and Bates, P. (Kimbel Librarians) Cheating 101: Paper Mills and You. Coastal Carolina University, Conway SC (USA): Published - 99. [Web Page] Accessed: October 14, 2002; URL http://www.coastal.edu/library/papermil.htm

[8] Carroll, J. and Appleton, J. Plagiarism: A Good Practice Guide. Joint Information Systems Committee (UK): Published - 2001. [Web Page] Accessed: October 31, 2002; URL http://www.jisc.ac.uk

[9] Harris, R. Anti-Plagiarism Strategies for Research Papers.: Published - 2002. [Web Page] Accessed: October 16, 2002; URL http://www.virtualsalt.com/antiplag.htm

[10] Culwin, F., MacLeod, A., and Lancaster, T., "Source Code Plagiarism in UK HE Computing Schools: Issues, Attitudes and Tools," South Bank University, London (UK), SBU-CISM-01-01, Sep 2001.

[11] Ketchell, M., Plagiarism: Fresh claims against Monash Uni head *The Age*, vol. News, News. pp. 1Jul 6, 2002. Fairfax. Melbourne, VIC.

[12] Ketchell, M., Quiet rejoicing in Monash corridors *The Age*, vol. News, News. pp. 1, 8, Jul 13, 2002. Fairfax. Melbourne, VIC.

[13] J.F. Peters and W. Pedrycz. *Software Engineering: An Engineering Approach*, New York, NY (USA): John Wiley & Sons Inc., 2000.