Plagiarism prevention is discipline specific: a view from Computer Science

Ruth Barrett
Principal Lecturer, School of Computer Science
r.barrett@herts.ac.uk, ext 4363

James Malcolm
Principal Lecturer, School of Computer Science
j.a.malcolm@herts.ac.uk, ext 4310

Anna L. Cox
Lecturer, Department of Psychology, UCL.
anna.cox@ucl.ac.uk

Caroline Lyon
Senior Lecturer, School of Computer Science
c.m.lyon@herts.ac.uk, ext 4266

Summary
Many of the good-practice guidelines on tackling plagiarism and collusion are primarily relevant to essays and research projects. In Computer Science, and particularly in undergraduate first-year modules, there is an emphasis on understanding basic principles and standard techniques: students are often assessed by being required to apply these techniques to an example system. Constructing suitable examples is time-consuming, and the range of possible solutions is small, so we find that collusion is as much a problem as plagiarism. This is also true of Engineering and Mathematical disciplines where there is a foundation of laws and theories that must be mastered and sometimes only one right answer to a problem. We suggest guidelines for the design of in-course assessments and the procedures that accompany them that can help to reduce the opportunities for plagiarism while recognising the constraints imposed by limited staff time and large student numbers.

Introduction
This paper reports on a study carried out in the Computer Science department to examine ways in which plagiarism and collusion on in-course assignments could be reduced. A strategy to tackle plagiarism can have many aspects. For example:

1. Sound and open university procedures
2. Educating students in the use of information resources and in correct citation techniques
3. Factoring out opportunities for plagiarism
4. Plagiarism detection, including electronic detectors

Recognising that all these are important, we wished to follow the advice given by Carroll (2001): 'if you only change one thing on your course, change your assessments' and focus on the design of in-course assessments. Our aim was to create a document of good-practice guidelines that would be of particular benefit to the staff of our department. In order to achieve this aim, we conducted four different activities: a look at published guidelines on ways to minimise plagiarism, an analysis of assignments set in the previous year, a consideration of the type of learning that is characteristic of Computer Science and the students who choose to study this subject, and a synthesis of good practice in the department. The last two were initiated at an internal staff workshop and followed up by individual interview.

Good-practice guidelines

The first activity was to look at the many good-practice guidelines and to see how they could be applied to typical assignments on a Computer Science programme. Although many guidelines were appropriate we found that they were biased towards essay-type assignments and concentrated on suggesting ways in which assignments can encourage individuality. In disciplines such as Computer Science, Mathematics and Engineering, many assignments need to be very specific about what the students have to do and give little scope for individuality.

There are a number of sources of good-practice guidelines for dealing with plagiarism and collusion, including (Carroll 2001, Culwin and Lancaster, 2001, Harris 2001, Hricko 1998, McDowell and Brown 2002, McInnis and Devlin 2003). Many of the guidelines describe good practice in essay assignments: see, for example, the suggestions by Harris (2001). Others may concentrate on plagiarism from the internet and give advice on student education in citing references and paraphrasing. Hricko (1998), in the context of student education, says that 'in most situations, plagiarism occurs as a result of ignorance'. McDowell and Brown (2002) advocate an assessment process in which students are working towards divergent rather than convergent goals, by the use of, for example, individualised negotiated assignments.

However, when some of these guidelines were presented in a plagiarism workshop they gave rise to a general muttering to neighbours. There were comments about suggestions such as 'design in assessment-tasks with multiple solutions or set one that creates artefacts to capture individual effort' (Carroll and Appleton, 2001) and even 'never reuse an old assignment specification as previous submissions have a habit of being handed in again' (Culwin and Lancaster, 2001). Consequently, we wished to investigate which guidelines would be useful to lecturers designing typical Computer Science assignments.

A survey of assignments

We chose the thirty-six strategies to minimise plagiarism by McInnis & Devlin (2003) as they have been widely quoted and we considered these a good and recent synthesis of other published guidelines on this topic. We analysed the previous year's coursework assignments from one of our M.Sc. programmes in Computer Science in the light of these strategies. Although at M level, the types of assessment used here have also been used on many of the undergraduate modules. This M.Sc. programme is designed for graduates of disciplines other than computing-related subjects and provided a snapshot of many of the areas covered in typical computing courses. For interest, we have named the modules in which these strategies were implemented because different topic areas lend themselves to different types of assessment and so to different strategies.

For each module, and each of the thirty-six strategies, we picked out examples of implementation of a strategy. We found the following strategies were most common:

> Strategy 9: 'Randomise questions and answers for electronic quizzes/ assignments.' *Used in the open systems and networks module.*
>
> Strategy 10: 'Ensure assessment tasks relate to the specific content and focus of the subject (and therefore the students) so students are less tempted to simply copy something from the web.' *Used in programming, systems analysis and design, computer architecture, databases, human computer interaction, internet environments and open systems and networks modules.*

> Strategy 12: 'Use essay/assignment topics that integrate theory and examples or use personal experience. For example, a field trip report, a task with no right answers or a personal reflection on a task.' *Used in human computer interaction, open systems and networks and internet environments modules.*
>
> Strategy 13: 'Use assignments that integrate classroom dynamics, field learning, assigned reading and classroom learning.' *Used in programming, systems analysis and design, open systems and networks, databases and human computer interaction modules.*
>
> Strategy 14: 'Use alternatives to the standard essay, such as case studies, which present more difficulties in locating suitable material to plagiarise.' *Used in open systems and networks, internet environments, databases and human computer interaction modules.*
>
> Strategy 27: 'Make a virtue of collaborative work in subjects with large student numbers and common assignments.' *Used in systems analysis and design, and internet environments modules.*

What the department does:

> Strategy 20: 'Minimise the number of assessment tasks – continuous assessment and over-assessment contribute to plagiarism.'
>
> Strategy 34: 'Use deterrence penalties. For example, a first offence results in failing the assignment, a second means failing the subject.'
>
> Strategy 35: 'Request that all work outside of examinations be submitted with a cover sheet defining plagiarism and requiring the student's signature.'

What was more interesting in this sample of modules was the number of strategies which were not followed rather than the ones which were. One of the reasons for this could be that many of the strategies are not very relevant to Computer Science programmes and are biased towards disciplines which normally assess through essays. Nevertheless, there were recommended practices that could be implemented.

There are a number that would help in reducing plagiarism in major project work, but are often not tackled until the project module because many other modules do not require students to acquire these skills:

> Strategy 3: 'Teach the skills of summarising and paraphrasing.'
>
> Strategy 4: 'Teach the skills of critical analysis and building an argument.'
>
> Strategy 5: 'Teach the skills of referencing and citation.'
>
> Strategy 6: 'Include in assessment regimes mini-assignments that require students to demonstrate skills in summarising, paraphrasing, critical analysis, building an argument, referencing and/or citation.'

And one strategy that could be implemented to a greater extent:

> Strategy 7: 'Design out the easy cheating options, for example, using the same essay/prac questions year after year.'

Characteristics of Computer Science assessment and students

We were surprised to see that some obvious strategies were so little used when other aspects of good practice were adopted widely in the department. There are a number of characteristics of Computer Science that appear to make the remaining strategies difficult to implement:

- Understanding is often tested by application of the theory to a specific problem and as a consequence there is a small solution space or even the possibility of a 'correct' answer.

- The assignment is often to test skills that are acquired by application and practice, for example in programming and solving numerical problems. Students who do not keep up with the work cannot tackle the assignment and a last-minute search of the literature will not help.

- Some assessment regimes are introduced to keep students working, for example, a series of small programming tasks or the use of lab-books. These assessments are often small, attracting few marks. The standard plagiarism penalties for these small assignments (which can be worth as little as 5%) may be excessive but must be applied.

- Assignments are written by lecturers with a complete solution, both to see if the problem is do-able in the specified time and to provide a detailed marking scheme. Re-use of these assignments is a survival strategy of many lecturers.

In addition, many Computer Science courses are taught to large cohorts (~250) and this exacerbates the problem of designing assignments that allow the students some leeway in originality. This is not a characteristic unique to Computer Science programmes but issues such as time to mark and consistency of feedback do influence assessment design.

We did not include the major project in this study. This requires students to undertake a substantial piece of investigative work and is assessed both on the artefact produced and the project report. Many of the guidelines on choice of topic, student education in the use of sources, and regular meetings to discuss a student's work (Culwin and Lancaster, 2001, McDowell and Brown, 2002) are appropriate to this module.

Some reasons why students plagiarise and collude

Lecturers in the department were asked why they thought Computer Science students plagiarise and collude in assignments. The reasons given coincided with those reported by students in Sheard et al (2002): not enough time, will fail otherwise, too great a workload at university, can't afford to fail, assignments are too hard, afraid of failing. Franklin-Stokes and Newstead (1995) also found that the students reported 'time pressure' and 'to increase the mark' as the main reasons for all types of cheating behaviour, and 'to help a friend' for cheating on coursework. As many of the assignments require students to work to a well-defined specification, the likelihood of collusion is increased. Research has shown that students do not perceive this as a serious crime in a range of cheating behaviours (Franklin-Stokes and Newstead, 1995, Dordoy 2002).

Additionally, from a lecturer's perspective it was felt that the previous educational experience in schools and colleges may not have helped students to take responsibility for

their own learning. Examination-result league tables may have led to over-coaching through coursework and a leniency on deadlines. The time-management skills of many students are poor and they do not always do the prerequisite work for an assignment. Of course, from the students' perspective, they may be strategic learners. Some assignments require writing skills and students do not know how to summarise, paraphrase, or make use of sources without directly quoting huge chunks. This may be true in many disciplines, but in general Computer Science assignments are technical rather than discursive and so students do not get many opportunities to learn and practise these skills.

The typical student who is attracted to a Computer Science course does not relish the thought of reflecting on the learning process and writing about how problems in their assignments were solved. Investigations by Chandler, Carter and Benest (2003) using a Myers-Briggs personality questionnaire on 264 Computer Science students suggested that the students: 'mentally live in the present, naturally use targets, dates and standard routines to manage their lives, are motivated internally, are cautious, quiet, diligent and conscientious'. Consequently we should widen the range of assignments we set in order to require students to develop reflective skills and to think laterally.

Good practice in Computer Science

In this section, we report the scenarios given to Computer Science lecturers during a plagiarism workshop and a summary of the good practice they reported. We suggest that these would be of use to Computer Science lecturers in other universities and to those teaching similar subjects.

Problem: A lecturer sets the same assignment every year. How can that assignment be changed without requiring extensive work on the lecturer's part?

1. Many assignments are based on a case study that requires the student to model the problem and provide a solution (for example those in Databases, Programming, SAD, HCI and Web design projects). It is possible to change the scenario, for example from ticket-selling to theatre-booking, car hire to holiday-cottage hire, a book library to a video library, a patient queue to a supermarket queue. These case studies will often have a similar data structure and operations to add, delete, change and query the data. These isomorphic systems will look very different to students because, whilst the expert can recognise the patterns, the novice will focus on the surface details. It is recommended that a module team should build up a number of different case studies to rotate over the years.

2. Variations on this theme include using the same case study with different questions or different requirements. The assessment tasks can be swapped around. For example one year students may design an interface using HCI principles and the next year evaluate an interface.

3. Other ways of changing an existing assignment include changing the numerical values in questions, asking for a critique of a 'wrong' answer, and changing the assumptions the students are allowed to make.

4. The assignment can have a unique feature that cannot be copied from previous years (such as 'include a frog as

part of your system'), or one part that is left to the student to devise, such as an added requirement.

5. In addition, the mode of submission can be changed, for example from a report to a poster presentation.

Problem: A large number of students are taking this particular module and are consequently all completing the same assignment. What can be built in to minimise plagiarism and collusion?

1. Give the students some choice. Allow the students to choose a topic and define the assessment criteria quite tightly. For example, in a level 1 programming course: design a system to implement a questionnaire, but the criteria could be 'must use an array', 'must find an average' etc. Give a range of requirements in the same case study from which the students have to choose. A successful piece of coursework allowed the students to choose the example for a semantic network to illustrate their understanding of a number of concepts. Marking was easy because the students were given a number of rules the network had to obey.

2. Include a written element, checked electronically, that asks for personal reflection or evaluation. For example: write about the decisions you had to make in the design process, evaluate the final product against your own criteria, give examples of where X is useful, show how concept Y has been applied, evaluate the modelling technique Z.

3. Assess students' understanding under test conditions. For example: a compulsory examination question based on the coursework, a time-constrained and supervised practical assessment (in programming, say) based on a piece of coursework, or an objective test marked using an optical mark reader.

4. Give help. Students may be tempted to plagiarise or to collude with others because they don't know how to start the assignment. Students are more likely to cheat if they have left the work to the last minute. Use tutorials to help students get to the point where they can tackle the pass level part of the assignment. Have a non-assessed review point (for example, a review of Z state schemas before the operation schemas are written); regular project meetings discussing a student's work. Build in milestones to break up large, monolithic assignments. Use group work with well-defined joint and individual work so that students support each other. Allow students to work in 'optional pairs' which allows students to work collaboratively or to choose to work alone. Optional pairs work well in programming assignments that do not contribute substantially to the final mark.

5. Build in processes to deter and detect. Use deterrence strategies such as adding in a component of the assignment that is assessed under test conditions such as an oral or a demonstration, changing the test data for different groups of students. Conduct a session in which you demonstrate the effectiveness of electronic plagiarism detectors. Finally, use detection strategies such as the same marker for each question or topic or online submission so that work can be submitted to plagiarism detectors.

6. Use multiple instances of the same problem with different variables. An example from a course on computer networks asked the students to do a Cyclic Redundancy Check calculation on their student record number expressed in binary.

Problem: It is considered appropriate that a particular module is to be assessed by 100% coursework. How can the assessment be designed so that the lecturer can have confidence that it is the person's own work?

1. Use a graded set of tasks with increasing levels of difficulty. An assignment in which students accumulate marks by doing a number of tasks of equal type and difficulty will also be one that will encourage students to copy from one another. On programming assignments it has been found that the most difficult part needs very few marks to inspire the best students and to deter the average student from spending too much time on it or from cheating.

2. Design assessments that involve individual creativity by a student. For example, in creating their own example system to illustrate some principle as in databases, systems analysis and design, creation of web pages or interactive systems. Do not over-specify the marking scheme so that students notch up marks rather than applying themselves to the assessment criteria.

3. Use in-course tests as a reliable and valid measure. However, this should not be the only type of assessment.

4. Include a requirement to find and use new information resources, for example, an application of artificial intelligence or neural networks.

5. Know your students. Check on drafts or interim work. Ask students to keep a diary or logbook.

Except perhaps under examination conditions, we can never be certain that a piece of coursework is a person's own work, but what we can do is design the assessments to motivate the majority of students and deter them from cheating. The deliberate cheaters will still have to be detected and punished, but adoption of the strategies discussed here should help as part of a holistic approach to the problem of plagiarism.

Conclusion

Assignments in Computer Science are varied in type and assessment mode, but they tend to be technical and practical rather than discursive, especially in the undergraduate first year. Very often, when there are definite skills to be learnt, students are not required to read widely around the subject but to concentrate on understanding the principles and techniques, often through a recommended textbook. A typical assignment will then require students to apply their knowledge, understanding and practical skills to a particular example system. Consequently there is not a wide variety of possible solutions and thus more potential for collusion from within the student body than for plagiarism from literature sources or the Web.

In this paper we have identified some characteristics of Computer Science assessment regimes and developed some

specific guidelines to aid lecturers in designing in-course assignments that minimise plagiarism and collusion. The intention was to improve rather than change existing practices as we recognise that there is a great deal of good practice currently employed by our colleagues who were generally very positive about what was possible, even within the constraints of large student numbers and lack of staff time.

Although we have focused on Computer Science, the design of appropriate assessment tasks to reduce the opportunities for plagiarism and collusion is relevant to all disciplines. In particular, we would recommend the following guidelines: avoid an assignment that is too similar to one set in previous years, require some individuality in the student's work such as an evaluation, opinion or unique design, and don't give the student too much scope to find the content elsewhere.

Acknowledgements
We thank members of the Computer Science Department for their contributions. This work was supported by a grant from the University of Hertfordshire's Learning and Teaching Development Fund 2002-2003.

References

Carroll, J. (2001) *What kind of solutions can we find for plagiarism?* URL: http://www.ilt.ac.uk/1120.asp (accessed 5 Dec 2002)

Carroll, J. and Appleton, J. (2001) *Plagiarism, A Good Practice Guide*. URL: www.jispas.ac.uk (accessed Sept 2003)

Chandler, J. & Carter J, Benest I, (2003) Extrovert or Introvert? *The Real Personalities of Computing Students*. 4th Annual LTSN-ICS Conference, NUI Galway. URL: http://www.ics.ltsn.ac.uk/pub/conf2003/janet_carter.htm (accessed 12 Jan 2004)

Culwin, F. and Lancaster, T. (2001) *Plagiarism, Prevention, Deterrence & Detection.* URL: http://www.ilt.ac.uk/1108.asp (accessed 21 May 2002)

Dordoy, A. (2002) *Cheating and Plagiarism: Staff and Student Perceptions at Northumbria.* Proceedings of the Northumbria Conference 2002 URL: http://online.northumbria.ac.uk/faculties/art/information_studies/lmri/Jiscpas/site/pubs_student.asp (accessed September 2003)

Franklin-Stokes, A. & Newstead, S.E. (1995) *Undergraduate cheating: Who does it and why?* Studies in Higher Education, 20, 159-172.

Harris, R. (2001) *Anti-plagiarism strategies in research papers.* URL: http://www.virtualsalt/antiplag.htm (accessed 30 Aug 2002)

Hricko, M. (1998) *Strategies to Deter Academic Misconduct.* URL: http://www.mtsu.deu/~itconf/proceed98/mhricko.html (accessed 30 Aug 2002)

McDowell, L. and Brown, S. (2002) *Assessing students: cheating and plagiarism.* URL: http://www.ilt.ac.uk/1144.asp (accessed 5 Dec 2002)

McInnis, J.R. and Devlin, M. (2003) *Minimising Plagiarism.* URL: http://www.cshe.unimelb.edu.au/assessinglearning/03/plagMain.html#36 (accessed May 2003)

Sheard, J., Carbone, A. and Dick, M. (2002) *Determination of Factors which Impact on IT Students' Propensity to Cheat.* Australasian Computing Education Conference (2003). URL: http://crpit.com/confpapers/CRPITV20Sheard.pdf (accessed Dec 2003)

Biographical notes

The authors are, or have been, lecturers in the School of Computer Science. They have published widely in the area of plagiarism prevention and detection. An electronic plagiarism detector, called Ferret, has been developed in the School and is available by contacting the authors.