

JISC

Joint Information
Systems Committee



SOUTH BANK UNIVERSITY
·LONDON·

SCHOOL OF COMPUTING, INFORMATION SYSTEMS & MATHEMATICS

Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools

by

Fintan Culwin, Anna MacLeod & Thomas Lancaster

**Technical Report SBU-CISM-01-01
September 2001**

Contents

Introduction		Page 3
1. Source Code Plagiarism Detection Systems		Page 4
2. The Questionnaire Survey		Page 5
2.1 Methodology		Page 5
2.2 The Results		Page 6
2.3 Free Response Comments		Page 8
3. Internet Hosted Similarity Detection Engines		Page 9
3.1 Qualitative analysis		Page 10
3.2 Quantitative analysis		Page 12
4. Other Issues		Page 16
5. Conclusions and recommendations		Page 18
Appendix A	Source code plagiarism publications list	Page 19
Appendix B	The questionnaire	Page 22
Appendix C	Detailed survey results	Page 23
Appendix D	Disclaimer	Page 31

Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools

Fintan Culwin, Anna MacLeod & Thomas Lancaster
Centre for Interactive Systems Engineering (CISE)
School of Computing Information Systems & Mathematics (SCISM)
South Bank University (SBU)

{fintan macleoam lancaste}@sbu.ac.uk
Web server: cise.sbu.ac.uk

Introduction

In December 2000 the Joint Information Systems Committee (JISC) commissioned CISE to produce a report on 'Plagiarism Detection Methods in Computer Programming'. This project was a constituent part of the JISC Committee for Integrated Environments for Learners (JCIEL) electronic plagiarism detection project (see www.jisc.ac.uk/jciel/plagiarism).

The purpose of the report was to inform JCIEL of the extent and nature of source code plagiarism and advise upon any resources that might need to be developed or deployed. It was also commissioned with the intention of making the technology more widely known within the UK HE academic computing community.

The report was completed in May 2001 and consists of four parts. It commences with a short literature survey intended to establish a context for the subsequent parts of the report. The second part summarises the results of a questionnaire based survey of UK computing departments, designed to establish perceptions of the extent of, and attitudes towards, source code plagiarism. The third part gives a quantitative and qualitative analysis of two widely known and freely available similarity detection engines, MOSS and JPlag. The final part comments on a number of miscellaneous issues and the report finishes with a number of conclusions and recommendations to JCIEL.

The report was presented to JCIEL at its September 2001 meeting and was published as SBU SCISM Technical Report 2001-01, copies of which were sent to all UK HE computing schools. An updated copy of the report (hopefully incorporating more replies to the questionnaire) will be maintained on the CISE web server whose address is given above.

1. Source Code Plagiarism Detection Systems

A list of references that have been consulted in the preparation of this report is given in Appendix A.

Computer systems for source code plagiarism detection have been in existence for over twenty years (Halstead 1977, Ottenstein 1977) and are now routinely used in many academic institutions. The first systems were based upon attribute counting algorithms, extracting various superficial metrics from source code submissions, for example a count of the use of a particular reserved word, and then flagging those pairs of submissions which were significantly close for tutors to inspect further. Later developments saw the introduction of structure metric systems where each submission is reduced to a series of identifiers, or tokens, representing, for example, a function call or a variable declaration. Pairs of tokenised submissions are then recursively searched for the longest common token sequences and the proportion of the submissions matched used as a similarity metric. Structure metric systems have been shown to be more effective than attribute counting systems (Verco & Wise 1996) and all existing systems use those principles.

The earliest attribute counting system, developed by Ottenstein (Ottenstein 1977), used Halstead's software science metrics (Halstead 1977), and primarily counted the uses of operators and operands within a piece of code. Such metrics were originally developed to allow automated marking of source code for style. Subsequent research has introduced new metrics and tried to find the most effective combinations (Robinson & Soffa 1980). Parker and Hamblen reviewed the metrics that seem to be successful for finding plagiarism in source code written in different programming languages (Parker & Hamblen 1989).

Structure metric systems have developed from Donaldson, Lancaster and Sposato's early hybrid system (Donaldson et al 1981). More recent examples include Plague (Whale 1998) and YAP3 (Wise 1996). Joy and Luck showed how students could plagiarise and how they could be detected with reference to their own SHERLOCK system (Joy & Luck 1999).

Academic institutions have access to both their own locally developed systems, including TEAMHANDIN (Culwin & Naylor 1995) and Irving's Big Brother system (Irving 2000). They also have access to services provided over the Internet. Best known is Aiken's Measure of Software Similarity (MOSS). No technical details of MOSS are available, only an independent critique (Bowyer & Hall 1999) describing one approach for using the system. The JPlag system, accessible on the Web and described from a technical perspective (Prechelt et al 2000 & 2001) offers similar functionality. (See section 3 of this report.)

Aside from JPlag little research continues into source code plagiarism detection, since it is a well-understood problem involving the analysis of constrained text.

Recent developments in structure metric systems employ sophisticated pattern matching algorithms, developed as part of the human genome project, to establish similarity between pairs of programs.

2. The Questionnaire Survey

2.1 Methodology

The questionnaire reproduced in Appendix B was prepared both in text form and as an interactive Web page. A list of contacts was prepared; starting with a list supplied by the Learning and Teaching Support Network centre for Information and Computer Sciences (LTSN ICS) and supplemented by personal contacts and Web searches for the e-mail address of the computing head of school. An e-mail was sent early in December 2000 to the people on the list asking for the questionnaire to be completed.

For institutions who had not responded towards the end of January 2001 a reminder e-mail was sent to the same contact. In mid-February 2001 every institution that had not responded was contacted by telephone and asked to complete the survey. A total of 55 out of approximately 110 HE level computing schools completed the questionnaire. It is intended to continue collecting responses and the Web hosted version of this report will be updated with additional data as it becomes available.

The instructions for completing the survey emphasised that the data obtained would be processed anonymously, and the Web system was so successful in this respect that questions concerning differences between pre-92 and post-92 Universities cannot be answered. Despite this a small number of institutions refused to complete the questionnaire on the grounds that they regarded the information as being too sensitive to reveal. Another small group of institutions became embroiled in internal discussions regarding the information to be supplied and have not yet responded. The reasons why other institutions have not responded is unknown. It is presumed that the request was never transmitted to the appropriate person or that the request was not regarded as sufficiently important.

Two institutions (both Russell group) responded in detail, describing how their students were required to regularly attend small group tutorials and describe the work that they had done. Consequently they did not regard source code plagiarism as a problem as students who cheated would be unable to adequately explain the code they presented and so would be caught out. One institution responded that they emphasised the principle of software reuse to the extent that using other student's source code was regarded as totally legitimate. The same institution commented that the appropriate response to student cheating would be to educate them as to why it was unacceptable.

A version of the questionnaire was prepared and trailed with a small number of Further Education computing lecturers. The opinion obtained was that the nature of assignments that FE students were given made plagiarism much less prevalent. Additionally, the smaller number of students in a cohort made detection much more likely and the appropriate response was seen as more educative than punitive. Because of these initial results no large scale survey of FE staff was conducted. However, it is felt that the substantive results of this report on the characteristics of available technology would be applicable to any interested FE tutors.

2.2 The Results

The detailed results of the questionnaire are given in Appendix D, only an executive summary of the outcomes is presented here.

Questions 1 & 2 asked about the prevalence and extent of plagiarism on initial courses. The majority of the responses indicated that it occurred some, most or almost all the time and that it usually involved less than 20% of the cohort.

Questions 3 & 4 asked the same questions about subsequent courses and the responses indicated that plagiarism was a little less prevalent and involved a smaller proportion of students.

A personal communication from Alex Aiken asserts, on the basis of his experience of operating the MOSS server, that for any (USA) corpus 10% of submissions will be plagiarised.

Question 5 asked about the attitude towards plagiarism as a problem and the modal response was that it was 'a routine headache' with equivalent numbers indicating that it was a 'minor nuisance' or 'bad and getting worse'. Only one response was prepared to say that it was 'not a problem'.

The evidence of previous UK surveys on student behaviour (Newstead et al 1996) suggests that tutors under-estimate the extent of cheating behaviour. If this finding is applicable to source code plagiarism then a possible interpretation of the implications of questions 1 to 5 is that plagiarism is both more extensive and more prevalent than indicated.

Questions 6, 7 & 8 indicated that 36 institutions routinely check submitted source code but half of these (17) relied upon visual inspection with only 4 making use of a public service such as MOSS. Of those that do not use any system the usual (11/17) reason was that no one had ever thought of using one.

Question 9 indicated that four fifths of institutions have a written policy on source code plagiarism. However almost all of these are reactive policies that state what is to be done when plagiarism is found. Very few are pro-active in requiring all work to be checked.

Question 10 suggests that the overwhelming majority of institutions would possibly, probably or certainly use a detection service. Very few (4/55) stated that they would not use one.

Question 11 asked about attitudes towards plagiarism detection and the modal response was that it is required to validate the effectiveness of an assessment. Almost all the other responses indicated that it is either an unfortunate requirement or that without it examinations would have to be exclusively used.

Question 12 asked tutors to indicate why they thought students resorted to plagiarism. No tutor stated that the assignment specifications were too complex (although this might be the favoured response of students). The modal response was that students were too disorganised closely followed by the response that programming was too difficult for students to learn. Many respondents commented at the end of the questionnaire that they would have liked to make multiple responses to this question.

Questions 13 & 14 asked about the number of students in the school and the number caught plagiarising in the last academic year. The response to question 14 was very patchy with responses ranging from a very small fraction (0.0014%) to approximately 13% with an average of approximately 5%. There was small negative correlation (-0.20) between size of school and rate of offending. This might suggest that cheating is less common in larger schools or that it is easier to avoid being detected although this is likely to be statistically insignificant.

Questions 15 & 16 were free response questions asking what the modal penalties were for first time and repeat offenders. The responses were categorised and indicated that the standard penalty for a first time offender (38/50) was related only to the piece of work which was copied. Either the mark for that component was zeroed and/or a rework was required. For repeat offenders a more severe penalty was indicated, with half of all respondents stating that expulsion was a possibility. However many stated that such cases were uncommon.

Questions 17 asked which programming languages any proposed system should be configured to check. The most popular response was Java with almost as many requiring C or C++. This was followed by Visual Basic, Pascal, Prolog, Ada, Delphi, Haskell, Scheme and LISP which all achieved more than one mention. Ten other languages (listed in Appendix D) all gained a single mention.

2.3 Free Response Comments

The questionnaire concluded with a request for respondents to add any other comments they might like to make.

Several respondents commented that 'collusion' or 'substantial collaboration' is much more common than outright plagiarism as such and that theft of floppy disks or listings from printers is an increasing problem. Accordingly only those cases that are 'in your face' cheating are taken forward to formal action.

Some tutors faced with the extent of cheating and the effort required to take a case through disciplinary proceedings either ignore it or take informal action. However, it was noted that informal actions cannot be used as part of a subsequent formal procedure where, as question 16 suggests, expulsion would otherwise be a possibility.

Comments were made that students do find the problems set too difficult, but that problems of comparable difficulty were set five or ten years ago and did not present so many problems. This might be correlated with the increasing number of students who are having to work to support themselves and causing increased absence from lectures and tutorials, a group which were noted by one respondent to be more likely to cheat.

The institutional response is mentioned by some respondents with those who report plagiarism seen negatively by their managers and one tutor claiming that they were reprimanded for setting work that was too difficult. Others state that after making a clear case no action resulted from the disciplinary process. This is possibly because registry staff did not understand the nature of software development or because students flatly deny any wrongdoing despite the evidence and without an admission the institution feels it cannot proceed.

Two institutions have responded that in response to the problems of detection and processing they are abandoning continuous assessment and resorting to laboratory based exams, even though this is regarded as educationally less valid.

On a more positive note some institutions are implementing the ideas of pair programming from the extreme programming community in the hope that it will mitigate the problem and others are considering differentiated assessments and measuring value added per individual student.

The culture of cheating is mentioned by one response making the connection between the acceptance of cheating in the first year leading to cheating on final year projects. Other cultural comments relate to different expectations of students who arrive from non-UK institutions and their not realising that plagiarism is unacceptable.

3. Internet Hosted Similarity Detection Engines

A Web search, requests on e-mail lists and discussions with attendees at the 2001 SIGCSE symposium indicated that there were only two widely known, freely available source code similarity detection engines, MOSS and JPlag. There are a large number of private engines, produced and used within a single institution and tightly coupled with teaching or teaching administration systems, for example the BOSS system developed by Mike Joy at Warwick. There are also a smaller number of more generic engines in development, for example Big Brother developed by Rob Irving at Glasgow.

This evaluation is restricted to the two freely available engines and has been produced after using both systems, reading all available documentation and e-mail discussions with the service providers.

MOSS (Measure Of Software Similarity) is the better known of the two engines. It was developed by and is maintained by Alex Aiken at the University of California, Berkeley campus. Its home page is located at

<http://www.cs.berkeley.edu/~aiken/moss.html>

JPlag was developed by Guido Malpohl and is maintained by Guido Malpohl and Lutz Precheltand at the University of Karlsruhe. Its home page is located at

<http://www.JPlag.de>

Both of the engines are supplied and supported by the maintainers as a service to the academic community, with the passive support of their institutions. There is no formal guarantee of continued availability or statement of the level of support available. However, from experience, e-mail queries have been promptly responded to and both maintainers state that it is their intention to provide the service indefinitely. Neither supplier charges for the service but both require users to register in order to ensure that only bona fide academics are using the facilities.

3.1 Qualitative analysis

Rather than describe each service separately, both engines will be evaluated side by side using a number of criteria. Table 1 contains a summary of the evaluation.

	MOSS	JPlag
Languages supported	8	4
Algorithm used	Not stated	Token pattern matching
Independent verification	No	No
Self validation	No	Yes
Collection tool	No	No
Multiple File submissions	Yes	Yes
Platforms supported	Unix	Any
Submission method	Command line	Command line Stand alone & from browser
Reporting method	Remote Web pages	Remote or local Web pages
Report contents	Ordered list Unparsed files	Submissions processed Histogram Clustered Ordered list
Metrics produced	percentage extent tokens matched lines matched	percentage similarity
Visualisation tool	Cross linked listings	Cross linked listings
Security	User id & e-mail	User id & password
Faq & support	Minimal	Minimal
Miscellaneous	Self adjusting Counter intuitive	N° of pairs can be specified

Table 1 Qualitative analysis summary

Languages supported. The programming languages currently supported by the engines are listed in Table 2. Although MOSS supports a larger number of languages results from the UK survey, as given above, indicate that Java and C/C++ predominate demand. Both services report that it would be relatively easy to support additional languages.

	Java	C++	C	Pascal	Ada	ML	Lisp	Scheme
MOSS	√	√	√	√	√	√	√	√
JPlag	√	√	√					√

Table 2 Programming languages supported by the engines

Algorithm used. Details of the algorithm used by the MOSS engine are not given in order to ensure that it is not circumvented. The only details given are that it actually examines program structure and does not rely upon superficial metrics. It is believed to be tokenisation followed by fast sub-string matching. The JPlag engine's algorithm is described in a technical report (Prechelt et al 2000) and is also tokenisation followed by sub-string pattern matching.

Independent verification. A search of the ACM and IEEE digital libraries and other available journal search engines resulted in only one publication that contained either 'MOSS' or 'JPlag' in its title, keywords or abstract. This paper (Boywer & Hall 1999) proved to be a descriptive paper relating how the MOSS engine was used in a single institution for analysis of a single submission. As such it was not considered adequate verification of the efficacy of the service. Both services report anecdotal comments from their users on the efficacy of the engines and of their value in plagiarism detection and deterrence.

Self validation. There has been no publication by MOSS of any attempt to validate the engine. JPlag has published a technical report (Prechelt et al 2000) and a paper (Prechelt et al 2001) describing a validation exercise. The exercise consisted of submitting four different corpora of student submissions to the engine and investigating the effects of tuning various parameters. Additionally for one of the corpora a number of deliberately plagiarised submissions were added and the engine was shown to be capable of detecting all of them. It would be valuable to conduct a replication of the black box aspects of this study with both the MOSS and JPlag engines, and a start at this has been made in the quantitative analysis study reported below in section 3.2.

Collection issues. Neither service supplies or supports a collection tool requiring the user to arrange for the corpus of submissions to be assembled in some way. Both tools support both single file and multiple file submissions, with the multiple files having to be placed into their own sub-directory in both cases.

Platforms supported: Submissions are sent to MOSS by executing a PERL script, which itself relies upon the existence of an environment that supports uuencode, mail and either zip or tar utilities. Although this does not preclude any platform, in practice it limits the system to Unix environments. JPlag, minimally, relies upon a Java run time system being available and so should run on almost any platform.

Submission method: MOSS only supports submission by execution of the PERL script from the command line. JPlag can also execute from the command line for maximum flexibility, but additionally it can be run as a stand-alone program or as an applet within Netscape (explicitly not Internet Explorer, Mosaic, Opera, etc.).

Reporting method: After processing the submissions MOSS sends an e-mail to the address from where the submission was made. The e-mail contains the URL of a page on the MOSS server where the results can be consulted. There is no provision for the results to be downloaded (for reasons of operational efficiency) and the results remain available for 14 days. When JPlag is used from the command line the results are placed directly onto the machine that was used with parsing and processing logs shown on the terminal. When JPlag is used stand

alone or within Netscape the results and the logs are available on a private page on the JPlag server, with an option for them to be downloaded as a zip archive.

Report contents: The main report page supplied by MOSS consists of an ordered list of pairs followed by a list of the submissions which could not be parsed, but which have been included in the processing with possibly inaccurate results. The report supplied by JPlag consists of a list of all the files which were processed, this may not be the same as all the files which were submitted as files which could not be parsed are excluded from processing. The list of excluded files is not contained in the report but it is available in the parsing log. The JPlag report also contains a histogram showing the number of pairs of submissions in each 10 percentile range and an ordered clustered list of pairs. The ordered clustered list commences with the most similar pair and then lists all other files, in order of similarity, to the first member of the pair which has the fewer matches with other submissions. This constitutes a cluster, and is followed by all other clusters.

Metrics produced. The JPlag engine produces a single metric for each pair, percentage similarity, which is defined as the proportion of the combined submissions which were shown to be identical. Full details of the metric are contained in (Prechelt et al 2000) and (Prechelt et al 2001). The MOSS engine produces four metrics for each pair. A percentage extent for each submission, presumably the same as the JPlag percentage similarity but reported for each submission individually. These are accompanied by the number of tokens which have been matched and the number of lines that have been matched. MOSS suggests that the tokens matched metric is the most indicative and the list is ordered by these values.

Visualisation tool. The visualisation tools for both engines are essentially identical having been first developed by JPlag and then adapted for use by MOSS. They consist of a HTML page containing three frames. One, at the top, contains a table which lists by line numbers, and colour codes, the fragments in each submission which were shown to be identical. Each line of the table is a hyperlink which, when activated, will cause the corresponding parts of the submissions, displayed alongside each other in the two remaining frames, to be shown. The corresponding parts of each submission are themselves appropriately colour coded and contain a graphical hyperlink that can be used to synchronise the view in the other frame. This latter mechanism allows the user to scroll through one of the submissions and easily locate the identical fragment in the other.

Security. As mentioned above it is necessary to register with each service before first use as a precaution to ensure that only academic staff have access to it. Registering with MOSS requires an e-mail address to be supplied and a numeric user-id is provided, this id has to be included within the script used to submit the program listings. The JPlag registration process requires the user to

indicate a password, and also issues a numeric id. The id is then used as a parameter to the URL (or encoded within the JAR archive supplied via this URL for standalone use) and the password has to be supplied with every submission.

FAQ and support. Both systems include a short FAQ and will respond to individual queries by e-mail.

Miscellaneous. One feature of the MOSS system not described above is the self-adjusting nature of the processing. Although it is possible for the user to indicate a 'base file' which was supplied to the students and which will cause the system to exclude its contents from consideration, a similar process operates automatically with code that appears in a high proportion of submissions being excluded from consideration. This has the counter-intuitive consequence that two submissions which are 100% identical on visual inspection are reported as being less than 100% in the results and may actually be reported as being less similar than a pair that are not 100% identical. However a command line option can be used to turn this feature off.

The command line options for the JPlag tool allow the user to indicate the number of pairs to be returned or the minimal percentage similarity to be reported upon. The MOSS tool seems to return either 500 pairs or fewer depending upon the possible number of pairs or the extent of the similarity. Both tools contain an option that allows the sensitivity of the matching to be tuned by indicating the minimum number of tokens in a match to be specified.

3.2 Quantitative analysis

The intention of this analysis was to investigate the extent to which the two available detection engines, MOSS and JPlag, agreed with each other. As any decision to ask an individual student to explain undue similarity in their work starts with the list of pairs produced by a detection engine, it would seem essential that the order of this list be as accurate as possible. Although it is difficult, if not impossible, to validate the ordering in any absolute manner, it might be hoped that two independent engines would produce a largely similar list.

To accomplish the analysis a corpus of Java source code obtained from the 2000/2001 SBU first year BSc computing students completing a specification known as waypoint 5, was submitted to each engine. The specification was in two parts. The first of which required the students to construct an accompanying class for a pre-supplied class hierarchy that allowed the user to initialise the state of a new instance by inputting values for its attributes from the terminal. Once this has been accomplished the second part required them to use this facility in a more complex specification which maintained a collection of objects.

The waypoint 5 specification closely shadowed the contents some taught material known as waypoint 4, and students were encouraged to adapt the provided waypoint 4 material when completing waypoint 5. Hence it was anticipated that there would be a large degree of similarity within the entire cohort. Only submissions for the second part of the specification were used in the qualitative analysis as the first part had proved so uncomplicated that all submissions were highly similar. A typical submission was approximately 200 lines long and contained a main() method and six to eight subsidiary methods.

A total of 102 submissions were available at the time of the investigation of which 88 were determined by JPlag to be syntactically correct and processed. For the same 102 submissions MOSS reported that only 4 had syntactic problems, which were processed with a warning that the results for those files may be inaccurate. As neither of the engines actually uses a compiler to ensure that a submissions will compile this inconsistency must be caused by differences between the parsers used. Unfortunately there was no time before the production of this report to investigate this further and the analysis was restricted to the 88 submissions which were acceptable to both engines.

The MOSS engine returned a list of the 500 pairs of submissions that had the highest values on the tokens matched metric; there was no apparent mechanism to obtain any more pairs. To facilitate processing the two extent values, that is the percentage of each file that was implicated as being similar, were combined to give a single percentage extent metric. The third metric, lines matched, was also used in the analysis. 2000 pairs of submissions were requested from the JPlag engine each of which had a single percentage similarity metric.

Consequently for a pair of submissions returned from either MOSS or JPlag there could be only the JPlag percentage similarity metric if it was reported by JPlag and not by MOSS. Alternatively if it was reported by MOSS and not by JPlag there would only be the MOSS percentage extent, tokens matched and lines matched metrics. Finally if the pair were reported by both engines then all four metrics would be available.

To facilitate discussion the four metrics will be identified as j1, m1, m2 and m3 corresponding to the percentage similarity, percentage extent, tokens matched and lines matched metrics respectively. The MOSS site suggests that the m2 metric is the most indicative of undue similarity.

percentage extent and tokens matched (m1 and m2)	0.729471
percentage extent and lines matched (m1 and m3)	0.74123
tokens matched and lines matched (m2 and m3)	0.83344

Table 3 Moss metric correlations

The first stage in processing was to compute the correlations between the three MOSS metrics for all 500 pairs. The results are given in Table 3 and the values,

particularly the m2 & m3 correlation, suggest that metrics are measuring approximately the same factors within the submissions.

The analysis continued by taking approximately the top 100 pairs from all four lists and combining them into a consolidated list by removing duplicates. It was not possible to take exactly the top 100 pairs due to sequences of pairs having the same value, in these cases more than 100 pairs were taken. If all four metrics were measuring the same factor in the same manner this should result in a consolidated list of approximately 100 pairs. The process produced a list that contained approximately 240 pairs, already indicating a degree of dissimilarity between the individual lists.

In the consolidated list there were 63 pairs which were contained within the top 100 pairs from the JPlag list that were not included within the top 500 pairs of the m2 MOSS list. There were also 28 pairs from the top 100 pairs obtained from the MOSS lists which were not contained in the top 2000 pairs from the JPlag list. When these unmatched pairs were removed from the this resulted in a pruned consolidated list that contained only 150 pairs. The Spearman rank correlation coefficients computed from this list are given in Table 4.

percentage extent and tokens matched (m1 m2)	0.467093
percentage extent and lines matched (m1 and m3)	0.458347
tokens matched and lines matched (m2 and m3)	0.573241
percentage extent and percentage similarity (m1 & j1)	0.348746
tokens matched and percentage similarity (m2 & j1)	0.087894
lines matched and percentage similarity (m3 & j1)	0.00194

Table 4 Spearman rank correlations for the 100 pair pruned list

Of the correlations between m and j factors only that between m1 and j1 was significant at the 1% level. (All the m intercorrelations were significant at this level.) The lack of significance between the favoured m2 and j1 metrics gives no evidence that the engines are in accord with each other. The significance of the m1 and j1 metrics is explicable when the nature of the two metrics, essentially the proportion of the files which are shown to be identical, is considered.

The conclusion from this part of the analysis is that, particularly when the 91 excluded pairs are taken into account, there does not appear to be a large degree of consensus between the two engines. However it could be argued that this lack of agreement might be caused by considering the top 100 matches from each metric and a greater degree of agreement might be obtained if the range were more restricted.

Consequently the analysis was repeated using approximately the top 25 entries from each list. This produced a consolidated list containing 71 pairs and a pruned consolidated list that contained 50 pairs. The Spearman rank correlation coefficients computed from this list are given in Table 5.

percentage extent and tokens matched (m1 m2)	0.399856
percentage extent and lines matched (m1 and m3)	0.341224
tokens matched and lines matched (m2 and m3)	0.596879
percentage extent and percentage similarity (m1 & j1)	0.363745
tokens matched and percentage similarity (m2 & j1)	0.088211
lines matched and percentage similarity (m3 & j1)	-0.18824

Table 5 Spearman rank correlations for the 25 pair pruned list

The level of significance of these correlations is identical to those of the previous analysis (the first four are significant at the 1% level, the remaining two are not). These results would tend to suggest that, if anything, there is less agreement between the two engines at the very top of the similarity list than in the upper part of the list.

Assuming that suspicious pairs are examined in the sequence indicated by the engine and that the resources for physical inspection are limited. The implication of these findings is that the chances of a pair being inspected, and hence the chance that a student will be asked to explain undue similarity, depends upon which engine has been used. This would seem to be unduly capricious. However the analysis has only been conducted upon a relatively small first year corpus that was expected to contain a high degree of similarity and so would need to be repeated with more corpora before this conclusion could be confirmed.

The providers of the services would like it to be pointed out that the tools are not intended as tests for plagiarism. They supply an ordered list of apparent similarities that allow a tutor to more efficiently locate the pairs that should be examined to determine if they require further investigation.

4. Other Issues

The UK QAA computing benchmark statements (available from www.qaa.ac.uk/crntwork/benchmark/computing.pdf) includes, for both threshold and modal levels, the following statement:

Produce work involving problem identification, the analysis, the design and the development of a system, with accompanying documentation. The work will show problem solving and evaluation skills, draw upon supporting evidence and demonstrate a good understanding of the need for quality

The British Computer Society (BCS) Guidelines on Course Exemption & Accreditation (available from www.bcs.org.uk/educat/guide.htm) includes the following statement:

All exempt courses will: . . . have a strong emphasis on design throughout the course . . . [stress] the need for . . . an emphasis on design and the development of appropriate practical skills.

This strong emphasis upon the design process and its accompanying documentation includes the production designs expressed using notations such as Jackson Structured Programming (JSP) schematics or Unified Modelling Language (UML) notations. These should be an integral part of any software development submission and be accorded a significant proportion of marks. As with program source code, the production of these artefacts is a time consuming and laborious, yet very necessary, process and a superficial disguise of a plagiarised copy is for some students a very attractive alternative.

From experience in SBU SCISM the submission of plagiarised designs is not as prevalent as the submission of plagiarised source code but is significant. However, as far as is known, no similarity ranking engines exist to support detection of cheating in this aspect of software development.

The visualisation tools supplied as part of the MOSS and JPlag systems are considerably more effective for verifying the existence of undue similarity than unaided manual inspection. However they lack the ability to scale the visualisation from a detailed view to an overview and are limited to examining only a pair of submissions, providing no support for the investigation and validation of a cluster. Again practical experience of source code plagiarism detection suggests that the vast majority of similarities are part of a cluster rather than a simple pair.

The only known approach to improved and alternative visualisation tools are the Composite Categorical Patterngrams (CCP) reported by Ribler and Abrams (Ribler & Abrams 2000) which allow clusters of plagiarised submissions to be detected and investigated.

5. Conclusions and recommendations

The survey provides a snapshot of the extent of and attitudes towards source code plagiarism provided by approximately 50% of UK HE schools of computing. It suggests that there is a definite problem and that automated techniques to assist detecting similarity could be more widely used.

While the MOSS and JPlag services remain available under the current conditions of use there seems to be little point in JISC replicating either service or supplying an alternative. However this recommendation is made without regard to the requirements of UK data protection and copyright legislation and should the export of identifiable data outside the UK (or EU) prove to be problematic then this conclusion may need to be revised.

A watching brief should be maintained upon both services and JISC may have to consider taking action if one or both of them becomes unavailable at some time in the future.

The UK computing academic community owes a debt of gratitude to Guido Malpohl and Alex Aitken, and their institutions, for developing and supporting the services.

The JPlag service would seem to be the easiest to use and produces more comprehensible results. However it supports a smaller number of languages and will not process programs which do not parse. Whilst the MOSS server will process programs which do not parse the results produced in these circumstances may be invalid. It is not JISC's purpose to endorse either of these services but to make their existence and characteristics known to the UK community.

The reasons why some files that are successfully parsed by MOSS are refused by JPlag and the reasons why the results returned from each system are apparently widely different need further investigation. This would seem to be suitable for a final year undergraduate or master's level project.

Appendix A

Source code plagiarism publications list

This list contains an entry for every reference that was consulted in the preparation of this report. It is not claimed that this is an exhaustive literature survey and the authors would be most interested in learning of any other suitable entries.

Kevin W. Bowyer and Lawrence O. **Hall 1999**

Experience Using "MOSS" to Detect Cheating on Programming Assignments
Proc. 29th ASEE/IEEE Frontiers in Education Conference, 13b18-22

Janet Carter 1999

Collaboration or Plagiarism: What Happens when Students Work Together?
Proc. ITICSE 1999, Cracow, Poland, pp52-55.

Paul Clough 2000

Plagiarism in Natural and Programming Languages: an Overview of Current Tools and Technologies
http://www.dcs.shef.ac.uk/~cloughie/plagiarism/HTML_Version/index.html

Fintan Culwin. & Jeoff Naylor 1995

Pragmatic Anti-Plagiarism
Proc. 3rd All Ireland Computer on the teaching of computing, DCU Dublin

Padraig Cunningham & Alexander N. Mikoyan 1993

Using CBR Techniques to Detect Plagiarism in Programming Assignments
Available from Department of Computer Science, Trinity College, Dublin

Donaldson, John L., Lancaster, Ann-Marie & Sposato, Paula H. 1981

A Plagiarism Detection System
Proc. Twelfth SIGSCE Technical Symposium, St. Louis, Missouri, pp. 21-25.

Faidhi J. A. W. & Robinson S. K. 1987

An Empirical Approach for Detecting Program Similarity and Plagiarism Within a University Programming Environment
Computer Education, v 11 No. 1

Andrew Gray, Philip Sallis & Stephen MacDonnel 1998

IDENTIFIED (Integrated Dictionary-based Extraction of Non-language-dependent Token Information for Forensic Identification, Examination and Discrimination): A Dictionary-based System for Extracting Source Code Metrics for Software Forensics
Proc. 1998 International Conference in Software Engineering, Education and Practice, pp252-259.

James O. Hamblen, Alan Parker & Stephen R. Wachtel 1998

A New Undergraduate Computer Arithmetic Software Library
IEEE Transactions on Education, Volume 31, No. 3

James K. Harris 1994

Plagiarism in Computer Science Courses
Proc. 1994 Ethics in Computer Age, pp133-135.

Jonathan Isaac **Helfman** 1994

Similarity Patterns in Language

Proc. IEEE Symposium on Visual Languages, pp173-175

Rob **Irving** 2000

Plagiarism Detection: Experiences and Issues

Presented at JISC Fifth Information Strategies Conference:

Focus on Access and Security, London

Jankowitz H. T. 1988

Detecting Plagiarism in Student Pascal Programs

The Computer Journal Vol. 38 No. 1

Mike **Joy** & Michael **Luck** 1999

Plagiarism in Programming Assignments

IEEE Transactions in Education, Vol. 42, No.2, pp129-133

G. R. **Lund** 1995

Controlling Plagiarism in Student Programs

Proc. 3rd All Ireland Computer on the teaching of computing, DCU Dublin

Newstead S.E., **Franklin-Stokes** A., **Armstead** P. 1996

Individual Differences in Student Cheating,

Journal of Educational Psychology Vol. 88 No 2. 229-241.

Ottenstein, Karl J. 1977

An Algorithmic Approach to the Detection and Prevention of Plagiarism

SIGCSE Bulletin vol. 8 No. 4, pp. 30-41.

Alan **Parker** and James O. **Hamblen** 1999

Computer Algorithms for Plagiarism Detection

IEEE Transactions on Education, Volume 32, Number 2, pp94-99.

Lutz **Prechelt**, Guido **Malpohl** & Michael **Phillippsen** 2000

Jplag: Finding Plagiarisms among a Set of Programs

Universitat Karlsruhe, Germany, Technical Report 2000-1

Lutz **Prechelt**, Guido **Malpohl** & Michael **Phillippsen** 2001

Finding Plagiarisms Among a Set Of Programs with Jplag

Submission to Journal of Universal Computer Science

Ribler R. L. & **Abrams** M. 2000

Using Visualisation to Detect Plagiarism in Computer Science Classes

IEEE 0-7695-0804-9/2000, pp173-177.

Robinson, Sally S. & **Soffa**, M. L. 1980

An Instructional Aid for Student Programs

SIGCSE Bulletin Vol. 12 No. 1, pp. 118-129.

Philip **Sallis**, Asjborn **Aakjeer** and Stephen **MacDonnal** 1996

Software Forensics: Old Methods for a New Science

Proc. International Conference in Software Engineering, Education and Practice, pp481-485

Robert **Sanders** 1998

Online Plagiarism Detector Helps CS Professors Bust Cheating Programmers

Berkeley <http://www.coe.berkeley.edu/EPA/EngNews/98S/EN1S/aiken.html>

Steve Saxon 2000

Comparison of Plagiarism Detection Techniques Applied to Student Code
Trinity College, Cambridge Part II Computer Science Project

John Traxler 1995

Cheating In Pascal Programming Assessments with Large Classes
Proc. 3rd All Ireland Computer on the teaching of computing, DCU Dublin

Kristina L. Verco & Michael J. Wise 1996

Plagiarism a la Mode: A Comparison of Automated Systems for Detecting Suspected Plagiarism
The Computer Journal, Vol. 39, No. 9, pp741-750.

Kristina L. Verco & Michael J. Wise 1996

Software for Detecting Suspected Plagiarism: Comparing Structure and Attribute-Counting Systems
Proc. First Australian Conference on Computer Science Education, Sydney Australia

Whale G. 1990

Identification of Program Similarity in Large Populations
The Computer Journal, Vol. 33 No. 2

Laurie Williams 1999

But Isn't That Cheating?
Proc. 29th ASEE/IEEE Frontiers in Education Conference, San Juan, Puerto Rico

Michael J. Wise 1996

YAP3: Improved Detection of Similarities in Computer Programs and Other Texts
Proc. SIGCSE '96, Philadelphia, USA, Feb 15-17 1996, pp130-134

Appendix B The questionnaire

For questions 1 to 12 please delete the responses that DO NOT apply until ONLY ONE response is left:

Question 1 I think that outbreaks of source code plagiarism, involving any proportion of students, on initial programming courses is:

- * Extensive almost all the time
- * Prevalent most of the time
- * Moderate some of the time
- * Occasional on occasion
- * Rare almost unknown

Question 2 I think that the proportion of students involved in a typical outbreak of source code plagiarism on initial programming courses is:

- * more than 50%
- * between 35% and 50%
- * between 20% and 35%
- * between 10% and 20%
- * less than 10%

Question 3 I think that outbreaks of source code plagiarism, involving any proportion of students, on subsequent programming courses is:

- * Extensive almost all the time
- * Prevalent most of the time
- * Moderate some of the time
- * Occasional on occasion
- * Rare almost unknown

Question 4 I think that the proportion of students involved in a typical outbreak of source code plagiarism on subsequent programming courses is:

- * more than 50%
- * between 35% and 50%
- * between 20% and 35%
- * between 10% and 20%
- * less than 10%

Question 5 I think that the problem of source code plagiarism is:

- * bad and getting worse
- * under control
- * a routine headache
- * a minor nuisance
- * not a problem

Question 6 Does your institution routinely check all submitted source code for indications of copying?

- * yes (please answer question 7 and ignore 8)
- * no (please answer question 8 and ignore 7)

Question 7 (if your answer to Question 6 was yes) The checking system used is:

- * non-automated, by hand and eye
- * a public service such as MOSS
- * a service developed and/or operated in house
- * a part of an integrated teaching environment
- * some other technique (please specify at the end)

Question 8 (if your answer to Question 6 was no) A checking system is not used because:

- * nobody has ever started to use one
- * the consequences of using one are thought horrendous
- * a decision has been taken not to use one
- * it is believed that no or little cheating takes place
- * the group sizes are so small it is not needed

Question 9 Does your school/department have a written policy on source code plagiarism:

- * no
- * yes - a pro-active policy that requires work to be checked
- * yes - a re-active policy that states what to do when it is found

Question 10 If a national source code plagiarism detection service was established do you think that your school would use it?

- * no - we do not use one and would not use one
- * no - we are happy with our current system
- * no - due to cost and staffing issues
- * possibly - it would depend upon the type of service
- * probably - providing it met our needs
- * certainly - when will it start?

Question 11 Which of the following statements best describes your attitude towards source code plagiarism detection?

- * it is not needed
- * it is not an activity that we should get involved in
- * it is an unfortunate requirement
- * it is required to validate the effectiveness of assessment
- * without it examinations are the only valid form of assessment

Question 12 I think that the major reason why students resort to plagiarism is:

- * programming is too difficult for them
- * the programs they are asked to do are unreasonably complex
- * there is too much pressure from other subjects
- * there is too much pressure from work and/or family
- * they are too disorganised to complete the work in time

Question 13 The number of FTE students in my school is approx. _____

Question 14 The number of students caught plagiarising source code in the last academic year was approx. _____

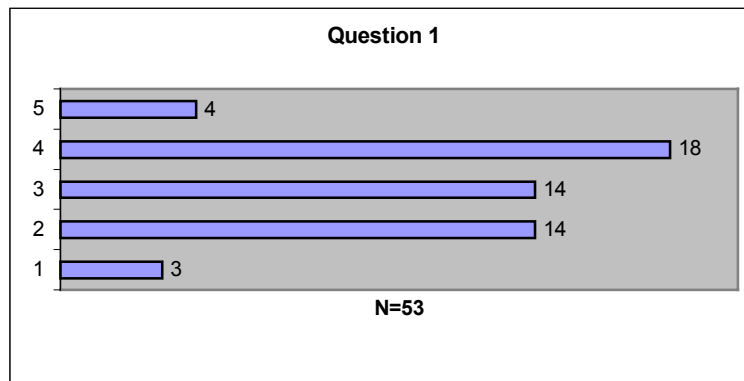
Question 15 The modal penalty for a first time offender who was adjudged to have committed significant plagiarism would be _____

Question 16 The modal penalty for a repeat offender who was adjudged to have committed significant plagiarism would be _____

Question 17 If a national service were to be set up, in order from most to least important, the programming languages needed would be _____

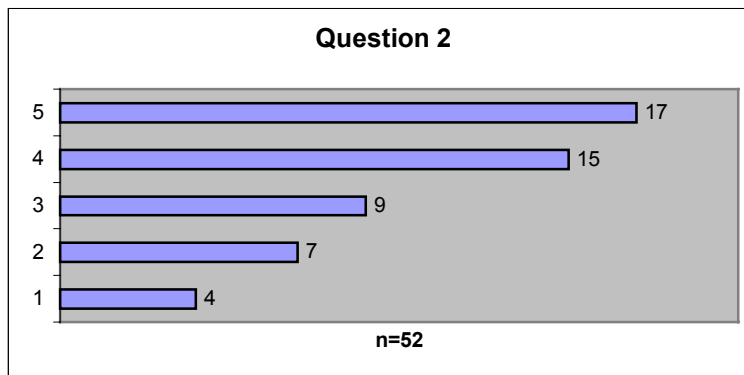
Please add any further comments here.

Appendix C - Detailed survey results



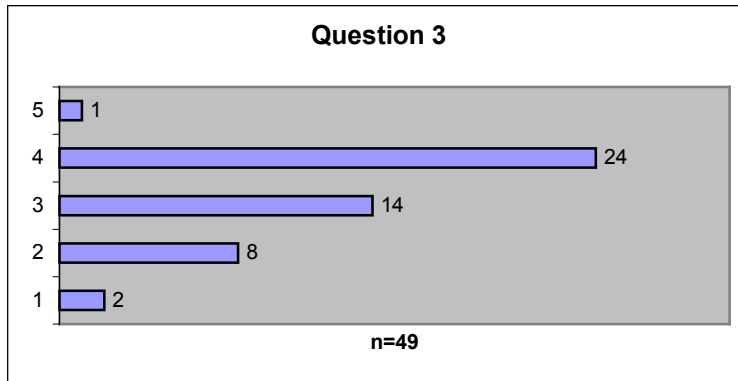
I think that outbreaks of source code plagiarism, involving any proportion of students, on initial programming courses is:

- 5 rare almost unknown
- 4 occasional on occasion
- 3 moderate some of the time
- 2 prevalent most of the time
- 1 extensive almost all the time



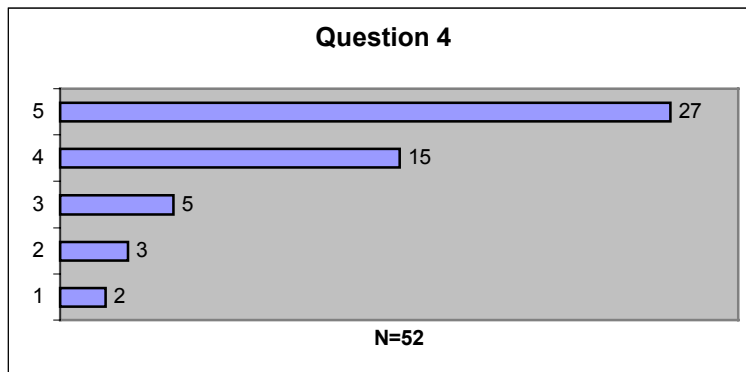
I think that the proportion of students involved in a typical outbreak of source code plagiarism on initial programming courses is:

- 5 less than 10%
- 4 between 10% and 20%
- 3 between 20% and 35%
- 2 between 35% and 50%
- 1 more than 50%



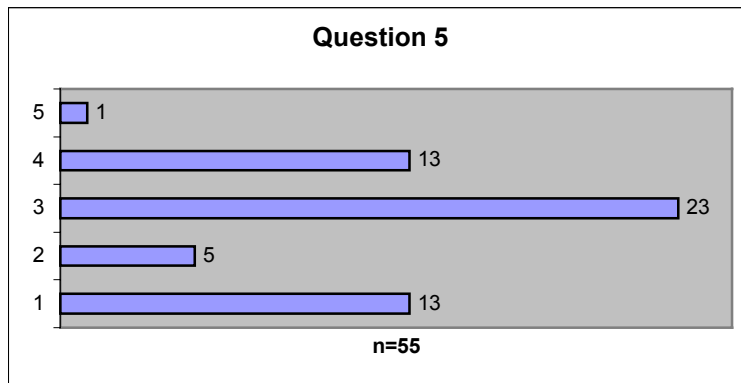
I think that outbreaks of source code plagiarism, involving any proportion of students, on subsequent programming courses is:

- 5 rare almost unknown
- 4 occasional on occasion
- 3 moderate some of the time
- 2 prevalent most of the time
- 1 extensive almost all the time



I think that the proportion of students involved in a typical outbreak of source code plagiarism on subsequent programming courses is:

- 5 less than 10%
- 4 between 10% and 20%
- 3 between 20% and 35%
- 2 between 35% and 50%
- 1 more than 50%



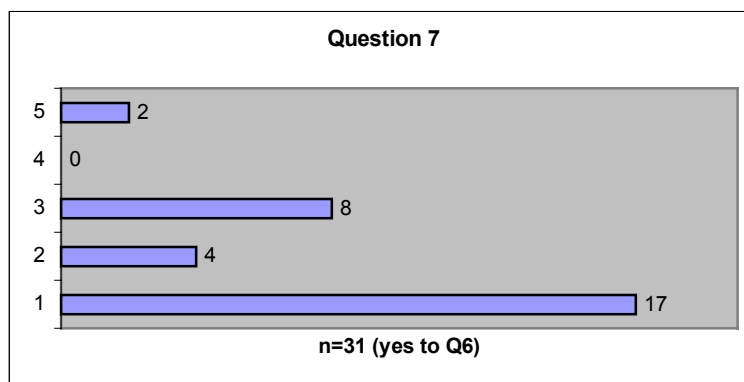
I think that the problem of source code plagiarism is:

- 5 not a problem
- 4 a minor nuisance
- 3 a routine headache
- 2 under control
- 1 bad and getting worse

Question 6

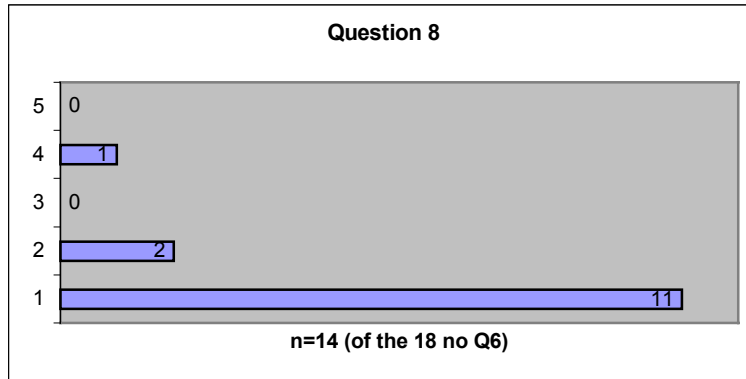
Does your institution routinely check all submitted source code for indications of copying?

- yes 36
- no 18



The checking system used is:

- 5 some other technique (please specify at the end)
- 4 a part of an integrated teaching environment
- 3 a service developed and/or operated in house
- 2 a public service such as MOSS
- 1 non-automated, by hand and eye



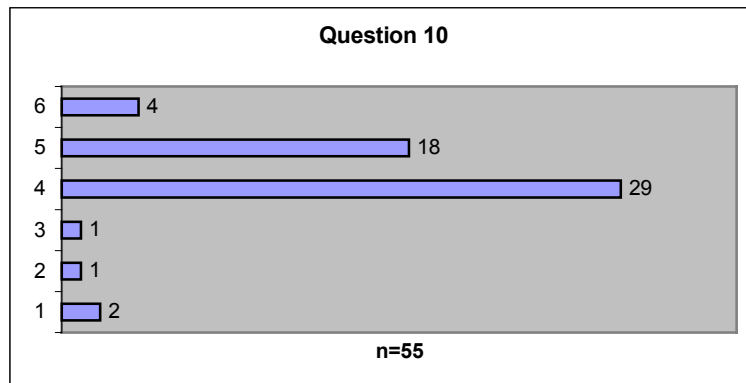
(if your answer to Question 6 was no) A checking system is not used because:

- 5 the group sizes are so small it is not needed
- 4 it is believed that no or little cheating takes place
- 3 a decision has been taken not to use one
- 2 the consequences of using one are thought horrendous
- 1 nobody has ever started to use one

Question 9

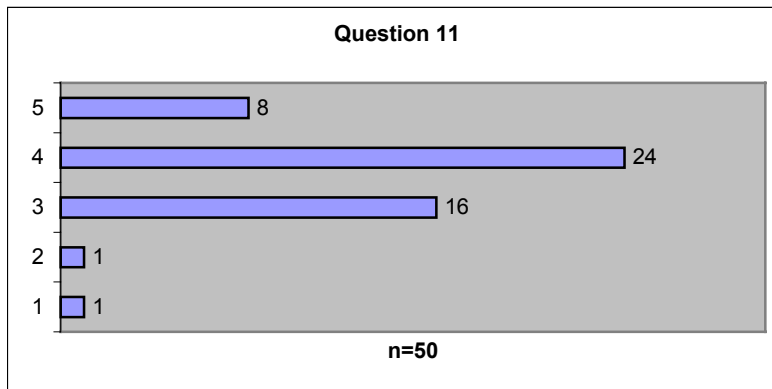
Does your school/department have a written policy on source code plagiarism:

- | | |
|--|-----------|
| no | 10 |
| yes - a pro-active policy that requires work to be checked | 4 |
| yes - a re-active policy that states what to do when it is found | 39 |



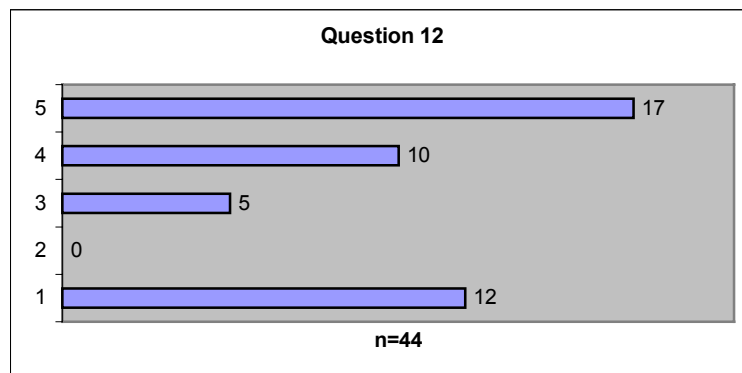
If a national source code plagiarism detection service was established do you think that your school would use it?

- 6 certainly - when will it start?
- 5 probably - providing it met our needs
- 4 possibly - it would depend upon the type of service
- 3 no - due to cost and staffing issues
- 2 no - we are happy with our current system
- 1 no - we do not use one and would not use one



Which of the following statements best describes your attitude towards source code plagiarism detection?

- 5 without it examinations are the only valid form of assessment
- 4 it is required to validate the effectiveness of assessment
- 3 it is an unfortunate requirement
- 2 it is not an activity that we should get involved in
- 1 it is not needed



I think that the major reason why students resort to plagiarism is:

- 5 they are too disorganised to complete the work in time
- 4 there is too much pressure from work and/or family
- 3 there is too much pressure from other subjects
- 2 the programs they are asked to do are unreasonably complex
- 1 programming is too difficult for them

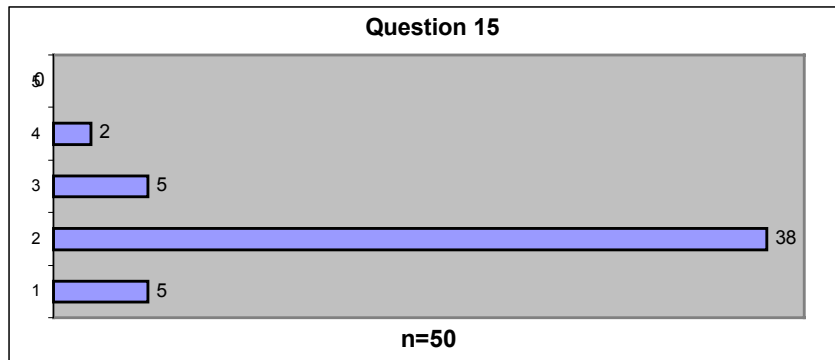
Questions 13 & 14

Q 13 The number of FTE students in my school is approx. _____

Q14 The number of students caught plagiarising source code in the last academic year was approx.

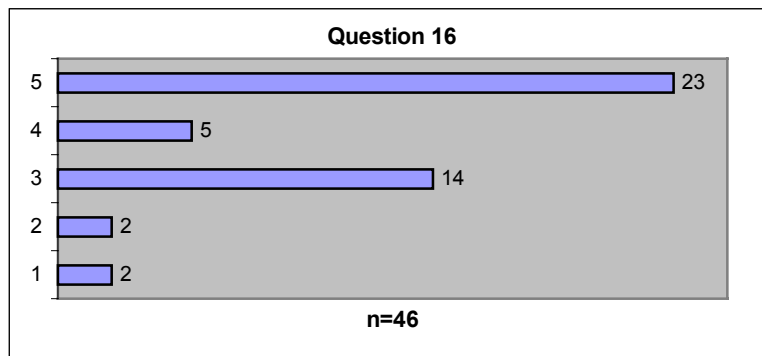
Q13 produced answers in the range 20 (actually a cohort on a course) to 1800. Q14 gave very varied answers, the modal appearing to be that exact records were not kept.

For the values which were given the percentage rate of offending ranged from a fraction (0.0014) of a percent to approximately 13%, with an average rate of approximately 5%. There was a small negative correlation (-0.20) between the size of a school and the percentage rate of offending.



The modal penalty for a first time offender who was adjudged to have committed significant plagiarism would be:

- 5 possible expulsion
- 4 course level penalty (e.g. repeat year)
- 3 unit penalty (e.g. fail part of year)
- 2 coursework penalty (e.g. zero work submitted & repeat)
- 1 warning only



The modal penalty for a repeat offender who was adjudged to have committed significant plagiarism would be:

- 5 possible expulsion
- 4 course level penalty (e.g. repeat year)
- 3 unit penalty (e.g. fail part of year)
- 2 coursework penalty (e.g. zero work submitted & repeat)
- 1 warning only

Question 17

If a national service were to be set up, in order from most to least important, the programming languages needed would be. (Languages listed without regard to priority).

Java	46
C++	27
C	13
VB	14
Pascal	6
Prolog	5
Ada	3
Delphi	3
Haskell	3
Lisp	2
Scheme	3
Eiffel	1
javascript	1
ML	1
Modula 2	1
Occam	1
Oracle	1
Perl	1
QBasic	1
SML	1
SQL	1
VHDL	1

Appendix C

Disclaimer

This report and the judgments contained within it are published in good faith but no responsibility is accepted for the accuracy of any of the information contained in the report or for the performance of any of the products evaluated in it.

Purchasers must satisfy themselves that any product referred to in this report will be suitable for their individual needs.