Academic integrity and computing assessments

Simon
University of Newcastle, Australia
simon@newcastle.edu.au

Judy Sheard
Monash University, Australia
judy.sheard@monash.edu

ABSTRACT
A recent Australian project has investigated academics' and students' understandings of and attitudes to academic integrity in computing assessments. We explain the project and summarise some of its findings, which have been presented in a number of prior papers. In an extended discussion section we then raise a number of questions that we believe must be addressed by the computing education community if it is to be seen to take academic integrity seriously. We question the value and the validity of a number of current educational practices, and urge the community to work towards resolutions of the unanswered questions.

CCS Concepts
• **Social and professional topics~Computing education**

Keywords
Academic integrity; computing assessment; programming assessment.

1. INTRODUCTION
Many, if not all, universities in Australia and New Zealand have institution-level academic integrity policies. Reading them can be a salutary experience for computing academics, as they can appear to be designed entirely for essays.

A typical set of 'guidelines for avoiding plagiarism' includes suggestions to

- "make sure you place the quote marks correctly;
- note down all the necessary details of the source for the references or bibliography section;
- make sure you use the correct citation or referencing standard … such as Harvard, Chicago, APA, [or] MLA;
- submit your draft assignment to [a text-matching program]" [6].

These suggestions are excellent for essay-type assessments, and are found in various forms on the web pages of most Australasian universities.

Some universities do mention other forms of assessment item. Victoria University of Wellington [24] indicates that sources to be acknowledged include "software programs and other electronic material"; Charles Darwin University [1] writes that "material that is subject to plagiarism includes, but is not necessarily limited to, written work, such as essays, books, reports, theses, journal articles and computer programs, whether published or unpublished . . ."

Some universities go further still. The University of Western Australia [23] notes that "in relation to forms other than written assessment, such as visual and digital media, computer codes, musical composition and performance, and oral presentations, an estimate of the level of seriousness is made in relation to the extent to which the plagiarism breaches the intention of the assessment and the guidelines provided for that assessment item." It is rare for a university's policy to acknowledge that the level of seriousness of academic misconduct can depend on the intention of the assessment and the guidelines for the particular assessment item. We found no university website that went further to suggest that the nature of academic misconduct might vary according to the discipline in which the assessment item is set.

Even when they mention non-essay assessment items, universities typically proceed to explain at length how to summarise other people's writings in one's own words, how to use direct quotes, how to combine the works of several other authors, how to apply text-referencing techniques, and so on: skills that appear to have no relevance to non-essay assessments. Most universities' policies and procedures appear to have little bearing on the use of externally sourced code in a computer program, the use of a formula in a spreadsheet, the use of a database query, or of the many other non-essay-type assessments found in computing education.

Narrowing the focus just to computer programming assessments, there are no guidelines about how much of another person's code can be included in one's own; about what sort of referencing is expected if one uses a standard algorithm; about how much help one can legitimately obtain from one's fellow students; and so on.

It is in the context of these glaring gaps that we have undertaken an exploration of students' and academics' understandings of, beliefs about, and approaches to academic integrity in non-essay-type computing assessments. Findings from this exploration have been presented in a number of papers. In this paper we collect the main findings, and proceed to a more thorough discussion of those findings than was possible in the individual papers.

The questions that we set out to answer are as follows:

- What do computing academics and students think constitutes a breach of academic integrity?

- Are there behaviours that are not breaches of academic integrity in computing, even though they might be in text-based assessments?
- How do computing academics educate students about academic integrity?
- How do computing academics detect and deal with breaches of academic integrity?
- Are there academics and students who think that copied programming assessments cannot be detected because all correct answers are effectively identical?
- Do university policies and procedures adequately cover academic integrity in computing assessments?

2. BACKGROUND

A wealth of literature world-wide, as summarised by McCabe et al [10], confirms that given the opportunity, many university students are willing to cheat in various ways. Being so different from disciplines that employ text-based assessment items, the computing discipline has attracted its own literature in academic integrity to mirror that carried out in the world of essays.

In Australia, Sheard et al [12] surveyed first-year undergraduate students in computing at their university, presenting them with a number of scenarios that the researchers considered to be cheating, and found that the students were far less likely to consider the scenarios as cheating. A decade later, after introducing a number of measures to raise students' awareness of academic integrity, Sheard and Dick [11] surveyed their students again and found substantial improvement, but with large numbers still considering certain behaviours not to constitute cheating.

A number of case studies have reported on the detection of particular forms of cheating in computing courses, and the consequences for those involved. Zobel [26] describes the extraordinary lengths to which he and others went to deal with a 'tutor' who was effectively offering to write students' assignments in exchange for payment. Simon [13] applied an electronic watermark to an individual online exam and found that at least a third of the students collaborated with others on the exam. D'Souza et al [5] describe two cases of students placing programming assignments on commercial code-tendering websites, and subsequently paying bidders from those sites to write the programs for them.

Joyce [8] surveyed the Australasian literature on academic integrity in computing, and observed that "most staff are 'digital immigrants' whereas most students are 'digital natives', used to freely accessing and sharing electronic information, so there may be a clash of perceptions and values as well as a technological gap".

Of course there is also much research in these areas outside Australia. Cosma and Joy [4] surveyed computing academics in the UK, finding that there was much uncertainty among the academics as to what constituted plagiarism in computing assessments. Subsequently they surveyed students [7] and again found little agreement as to what practices were academically inappropriate. Clarke and Lancaster [3] devote much of their work to identifying university assessment items placed on code-tendering websites.

Vogts [25] suggests that students who cheat are crying for help. However, this defies the findings of others [10, 13] that given the opportunity, some students will cheat regardless of whether they might be capable of doing the work. Chuda et al [2] consider the reasons for cheating from the perspective both of academics and of students. One-third of their student survey respondents admitted to plagiarising, most of them on programming assessments; and fully two-thirds said that they had allowed their work to be copied from.

3. RESEARCH APPROACH

The research summarised in this paper was funded to be conducted in the Australian higher education sector. The project, known as Prianit (Plagiarism and Related Issues in Assessments Not Involving Text) began with document analysis, examining the publicly-accessible academic integrity policies and procedures of as many Australian universities as possible, with a view to assessing how pertinent they were to computing assessments. Almost all of the universities yielded some public documentation.

The next step was to conduct telephone interviews with computing academics at most of the Australian universities that offer degrees in computing (for example, computer science, information technology, information systems). Interviewees were asked what materials they used to educate their students about academic integrity in computing, and whether they were willing to share those materials with academics at other universities. Notwithstanding the difficulty of finding suitable people for these interviews and the subsequent logistical arrangements, we were able to interview 28 computing academics from 23 universities.

Focus groups were conducted among academics and students at the project members' universities and at two computing education conferences, to start to document understandings of academic integrity, attitudes to academic integrity, and relevant practices and procedures, and also to inform the development of questions for the survey. Three focus groups involved 18 computing academics, and two focus groups involved 12 computing students.

Finally, a wide-ranging survey was conducted among academics and students Australia-wide. Once the survey responses had been cleaned, there were responses from 70 computing academics and 486 computing students from most universities in Australia. A large part of the survey presented various scenarios in the same manner as earlier researchers [4, 12], asked whether they were plagiarism/collusion, and also asked whether they were acceptable. The survey was long; to thank them for their time, participants were invited to enter a draw for one of five iPads or comparable devices.

4. SUMMARY OF FINDINGS

In this section we briefly summarise the findings of the Prianit project, many of which have been published in detail in other papers [14-19]. The point of this section is not to present the findings, but to summarise them in order to set the context for the subsequent discussion.

4.1 University policies

As indicated in the introduction, a number of universities mention in their policies non-essay-type assessment items, and some explicitly mention computer code. Nevertheless, all of them then tend to give instructions and examples showing approved ways of referencing externally sourced text in essays.

We found no university-level instructions or examples dealing with any type of computing-specific assessment.

Not all of a university's policies and procedures are accessible to the public, so we also discussed the adequacy of these instruments in the focus groups and asked about them in the survey. In the focus groups there was general agreement that the policies and procedures were inadequate to deal with computing assessments. In the survey, only 30% of academics and 42% of students agreed that the university policy on academic integrity adequately addresses non-text assessment items, and 24% of academics and 49% of students agreed that the university adequately educates students in relation to academic integrity for such items. These and related findings have been reported elsewhere in more detail [15].

4.2 Interviews with academics

In interviewing computing academics around Australia we hoped to discover and gather the resources that they and their academic units use to educate their students about academic integrity in computing. However, very few resources emerged from this process, and most of those that did were used not by academic departments or schools but by individual academics. These were generally guidelines to be followed when using externally sourced code in a program, including some ways of noting a reference in the program's documentation.

4.3 Focus groups

The focus groups elicited very much the same perceptions from students and from academics: in terms of academic integrity, computing assessments differ substantially from essay-type assessments. Participants tended to believe that the boundaries between acceptable and unacceptable practices are harder to define with computing assessments, and that different standards should be applied to these two different forms of assessment.

Participants differed on the need to reference program code taken from external sources. Some felt that such code should be referenced, whereas others thought that the work required to incorporate other people's code into an assignment was sufficient to make it one's own work. It was observed that it is generally neither possible nor desirable to write code entirely from scratch, but students are seldom required to reference 'standard' code when using it in their own work.

Academics generally had stricter views than students, and were less generous in their understanding of what actions might constitute academic misconduct.

There was a general feeling that students in computing are more likely to engage in collusion, working inappropriately with their peers, than in plagiarism, taking code from publicly-accessible sources.

These and related findings have been presented elsewhere in more detail [14].

4.4 Survey – computing assessments vs essays

The scenarios in the survey were presented both for essays and, as comparably as possible, for computing assessments. We were thus able to compare the responses for these two different contexts, and a number of clear differences emerged. All of the differences summarised in this subsection were found to be statistically significant.

The use of other people's words or code without referencing was widely seen as plagiarism/collusion, but by 84% of students with regard to essays and only by 76% with regard to computing.

In contrast, many other practices were considered as plagiarism/collusion by more participants in the computing context than in the essay context. These include discussing the detail of work in progress; posting work to an online forum and asking for feedback; and changing completed work to incorporate features seen in another student's work.

A number of computing practices were considered less acceptable by academics than their essay equivalents. These include fully-referenced copying from an external source; discussing the detail of assessment work in progress; and asking another student for advice on how to fix troublesome code.

Comparing the responses of academics and students, more students than academics thought that it was acceptable to base an assessment on externally-sourced code, whether or not that code was referenced; and to base a program substantially on one written for a previous course without acknowledging this.

A number of practices were considered by some respondents to be plagiarism/collusion but nevertheless to be acceptable. These include discussing the detail of code while working on the assessment and asking another student to help fix troublesome code. Conversely, the reuse of one's own prior work was considered by some respondents to be unacceptable despite not being considered as plagiarism/collusion.

These and related findings have been reported elsewhere in more detail [17].

4.5 Survey – attitudes to particular practices

The scenarios presented in the survey can be considered in groups representing variations on particular practices.

The use of code from external sources was considered acceptable by 73% of students and 58% of academics if the code was appropriately referenced, and only by 10% of students and 3% of academics if it was not referenced. Yet only 37% of students indicate confidence that they know how to reference external material in computing assessments, and only 25% of academics are confident that their students know how to reference material.

There was general agreement that it is not acceptable to take another student's work and submit it as one's own, but 12% of students thought this was acceptable, and a further 22% were unsure, if the student took an early draft of the other student's work and subsequently developed it alone.

Re-use of one's own prior work without acknowledgement was generally considered unacceptable, but 22% of students thought it acceptable, and a further 20% were unsure.

More than half of the students, and 40% of the academics, thought it acceptable to post troublesome code on a message board and ask for help with it. Asking other students for help was considered acceptable by even more: 69% of students and 57% of academics. However, acceptability ratings were far lower when it came to giving the troublesome code to another student and asking them to fix it.

These and related findings have been reported elsewhere in more detail [16].

4.6 Survey – academic and professional practice

Some of the survey's computing scenarios were presented not just in the context of academic integrity but again in the context of professional practice in the computing industry. The decision to do this arose partly from the focus group observations that certain practices should be acceptable in academia because they are acceptable in the workplace.

While it was almost universally agreed that it is not acceptable to pay somebody else to write the code for an academic assessment, only about a third of the academics and a fifth of the students saw the same practice as acceptable in the workforce – where it forms a large part of the business model known as outsourcing.

Both academics and students agreed that many practices not considered acceptable in academia are completely acceptable in the workplace. An example is getting help from colleagues or message boards.

These and related findings have been reported elsewhere in more detail [18].

4.7 Survey – detecting and dealing with issues

Academics taking part in the survey were asked what they do, if anything, to detect assessment items that involve plagiarism or collusion. A number mentioned various code similarity detection programs, which are used more to detect collusion than plagiarism: programs are generally compared with those of other students in the same class, not with a large corpus of external programs.

Very few reported using these programs. In contrast, between 60% and 80% reported that they detect plagiarism or collusion by closely inspecting students' work, by looking for similarities between submitted assessments, by observing sudden improvements in the work of individual students, or by noticing the use of content that had not been covered in the course. In summary, despite the availability of suitable tools, the detection of excessive similarity is almost invariably manual.

Suspected or confirmed cases of academic misconduct are sometimes dealt with by the academic, but are more often required to be escalated to a person or committee with particular responsibility for dealing with academic misconduct. In the latter case, some academics complain that the person or committee fails to deal with the case adequately, often because of a failure to understand what constitutes excessive similarity in computer programs.

5. ISSUES AND DISCUSSION

The findings summarised in the preceding section raise a number of questions to which the answers are far from obvious. In this section we will present and discuss a number of the questions. We do not have the answers: they are for the computing education community to resolve. However, in order to stimulate discussion, at the end of this paper we do propose one set of possible answers.

5.1 Using externally sourced code

Students writing essays are generally expected to read a number of suitable external sources, then to write their own thoughts in their own words, with reference where appropriate to the ideas and perhaps the words of the sources. It is in the nature of a natural language such as English that even if two people have exactly the same ideas, they are unlikely to express them in the same way. Therefore the detection of academic misconduct such as plagiarism relies on finding the same ideas expressed in the same words.

In programming, by contrast, little value is placed on the skill of expressing things differently from the way other people have expressed them. Computer programming is to a large extent an exercise in assembling the right components in the right way, where the components are also known as plans [21] or design patterns [20]. The skills that we seek to teach and assess are those of knowing what existing components can be used in solving a particular problem, combining them in a suitable way to solve the problem, and creating new components from scratch where suitable ones do not seem to exist.

Most computing educators seem to teach their students by example, showing them existing code, explaining how it was developed, and also explaining how it might be adapted to other situations. Many of us also teach students not to reinvent the wheel: if you want some records sorted, don't devise a new sorting algorithm; instead use one of the existing sorting methods that you have been shown, and integrate it into your program.

This being the case, one would expect a great deal more similarity between two computer programs designed to carry out the same task than between two essays designed to discuss the same topic; consequently, code similarity detection will typically be of less value than text similarity detection in essays. Of course there is some value in seeing that two students' submissions are essentially identical: but we must accept, as pointed out by Mann and Frew [9], that code similarity can often be ascribed to the nature of programming, the value of code reuse, and the fact that all students are typically working on solutions to the same task.

The similarity between students' programs for a given task might be somewhat reduced by requiring the students to create their programs from scratch, without looking at the many code samples that they have been shown in their classes. But why would we enforce such a requirement when it defies the very nature of the programming task?

5.2 Reuse of one's own code

As an extension of the preceding argument, if a student has previously written some code that might be useful in a current assessment task, is there merit in requiring the student to ignore that previously written code and write a completely new solution? If so, what is that merit?

Many universities explicitly bar the use in assessment items of material that has been previously submitted for assessment. Such barring will sometimes be found in the university's policies and procedures, and sometimes in the standard cover sheets that students are required to submit with assessment items. Is this a useful practice for computing assessments, and particularly for programming assessments? Or should we be urging students to adapt and reuse code that they originally wrote for some other purpose?

If a student can submit the same complete answer to two different assessment items in two different courses, there might be a problem with either or both of those courses, or the linkage

between them, or the communication between their coordinators. But given the component nature of programming, it should not be at all surprising if a student decides to reuse and adapt a particular module or algorithm that they have successfully written and debugged for a prior course.

Some might consider a requirement that bars substantial reuse, the idea being to permit the reasonable reuse of components but not the reuse of complete programs. However, this would almost certainly lead to difficulties in the definition of substantial reuse. Any specification of how much reused code is too much is bound to be somewhat artificial, arbitrary, and indeed very difficult to measure.

5.3 How to reference reused code

We have argued in the preceding subsections that reuse of existing code, both externally sourced and previously written by the student, is integral to the nature of programming as it is normally practised and taught.

Most university academic integrity policies express the insistence that any use of the words or ideas of others be properly acknowledged, and then direct students to text-based referencing guides for systems such as Harvard, APA, and IEEE. We were unable to find any university guides for referencing program code. Some academics shared individual guides that they had prepared, but most indicated that they do nothing to teach students how to reference code – principally because they do not require students to reference reused code.

It is clear from our focus groups and survey that most computing educators do not expect students to acknowledge every use of code from lectures, books, and the internet. There is no clear agreement on the need to reference code taken from external sources, and we suggest that this is why no referencing standards have emerged.

If our understanding is wrong, if most computing educators do require their students to formally acknowledge the source of every component used in their programs and other assessment items, we call on the computing education community to make this clear, and subsequently to develop agreed guidelines for referencing reused code.

5.4 Collusion and help with debugging

Another aspect of the nature of the programming task is that it is collaborative. Particularly in the debugging phases of a project, but also in the design phases, students and professionals alike consult widely with one another. They show one another their code, ask for suggestions as to what might be causing perceived problems, and freely share their own knowledge and experience. In our interviews and focus groups, many academics explicitly indicated that they teach their students to work in this collaborative manner.

Then it comes time to assess the students, and many academics change the rules of engagement. Students are now not permitted to show their code to others, to ask for help with explicit problems, or to see how other students have overcome the same problems. They are now required to work in a manner that is alien to the way they have been taught to work.

The principal reason for this appears to be that as academics we are required to assess individual students, in effect certifying each student as capable of contributing in the workforce. If we permit students to work together indiscriminately, we can no longer be sure of the capability of the individual, and therefore we cannot assess and certify the individual. Unfortunately, the solution of insisting on individual work, if it can be enforced, means that we assess the individual student in a skill that is markedly different from the one that we teach and that students will be expected to practise in the workplace.

Why do students seek help from others even when explicitly told not to do so? For many it is a case of having nowhere else to turn. Academics are not always available to assist; the student cannot find helpful information on the internet or in the textbook; and, most important, the development simply cannot proceed while the error remains undiagnosed and unfixed. A student writing an essay is seldom in a comparable position, because it is possible to leave that section of the essay, work on another, and seek the teacher's help when it is available. With programming, where a single error can prevent any further meaningful development, that error must be fixed as soon as possible; and if other students are available and willing to help, they are the obvious solution. And, of course, seeking such help is acceptable both professionally and – except with assessment items – academically.

Unfortunately, seeking debugging help from fellow students can be seen as the thin end of the wedge of collusion. Where should the line be drawn between helping with a single syntax error and proceeding to 'fix' the remaining syntax errors, and then any logical errors that emerge? In the end, have the students colluded? Who has done how much of the work, and how can the academic reasonably assess the individual student?

5.5 Workplace practice and authenticity

If collaborative development as a work practice is acceptable in the computing professions, why is it not acceptable in academia? The principal answer is the one presented in the preceding subsection: that it makes it difficult or even impossible to assess individual students and certify their preparedness to join the professions.

This leads to an interesting conclusion about assessment in computing: almost all individual computing assessment is necessarily inauthentic, because it requires students to work in a manner that is very unlike the way they will be expected to work when employed.

The conclusion applies to written examinations even more than to individual software development. Whatever a written programming exam assesses, it is not the skill of developing software. Many of the academics in our interviews and focus groups acknowledged this, but gave two reasons for continuing to use exams for assessment. First, with an invigilated exam one can be reasonably confident that the work being assessed is the work of the student who will get the marks. Some take this further and require students to pass the exam in order to pass the course; but some universities explicitly prohibit such requirements. Second, practical programming exams can be extremely poor indicators of students' software development abilities, because it is easy for some students to become completely stuck on a single syntax error; unable to proceed past that error, they end up failing to demonstrate the work that they would have produced had the error not arisen.

5.6 Detection of academic misconduct

There are many similarity detection packages for program code, working in a number of different ways, and for a number of different programming languages. We have found very few

academics who use these packages. Perhaps one reason is that the academics do not disapprove of plagiarism and collusion in computing to the extent that their university policies suggest.

If the computing education community does agree that current policies and procedures should apply to computing in the same way that they do to essays, it should possibly start using code similarity detection software more broadly, in the way that Turnitin and similar software is used for essays.

It should be noted that while text similarity detection software compares written submissions with vast databases of other written work, program similarity detection software tends to compare all of the submissions to a single assignment in a single class, or perhaps in several offerings of the same course. This use is clearly intended to detect collusion rather than plagiarism. This is almost certainly a good thing: given the existence of countless pieces of 'standard' software, similarity detectors would probably find it difficult to identify any truly unique programs among a sufficiently large set of programs.

Outsourcing of program code and other assessment items appears to be a growing problem in computing education. Outsourced code is notionally custom-written and unique, and so will not be detected by similarity detection software. Does the computing education community need to start trying to find ways of detecting outsourced materials? If so, what methods would be appropriate?

5.7 Explaining vs writing

Many academics mention that they interview students to help establish whether the students wrote the code that they have submitted for assessment. There are clearly pragmatic consequences of this practice for large classes, but there are other problems with it.

To a certain extent, the practice uses the ability to explain the code as a proxy for, or as evidence of, the ability to write the code. There is little evidence that this practice is used for other forms of assessment, such as essays. We have not heard of academics asking students to explain an essay, and using the explanation to decide whether the student wrote the essay. We cannot imagine an art teacher deciding that a student painted an artwork because the student can explain aspects of the artwork.

It appears that the use of the practice in computer programming is based on a genuine belief that if students cannot explain the code, they cannot have written it. Unfortunately, this belief is flawed. There is evidence in the literature [22] of students who code almost by random mutation, continually changing things more or less arbitrarily until they eventually happen upon a combination of symbols that satisfies the compiler and does the required task. Such students have written the code, but will be completely unable to explain it. Therefore the belief behind this practice is faulty, and the value of the practice is at best questionable.

If, on the other hand, the interviews are explicitly used to assess the students' understanding of the code, and understanding of the code is one of the assessment criteria, the practice is sound. Our concern is the sometimes tenuous connection between understanding the code and having written it.

5.8 Consensus and education

One thing that was evident both in our focus groups and in the survey is that there are high levels of uncertainty, among academics and students, about many aspects of academic integrity in computing.

There is a pressing need for some sort of consensus in the computing education community regarding academic integrity, what it is in the context of computing, and how it applies to the different forms of computing assessments that are not essay-like in form.

If that consensus can be reached – and it will not happen quickly – the next step will be to persuade academics to apply it in their own assessment. If they have contributed to the consensus, they are of course more likely to apply it. The academics will then be able to start passing the knowledge to their students.

One pressing need in this development will be a set of exemplars of educational materials. We should not expect each individual academic to develop materials to explain the same broad concepts. Rather, we would propose that some academics will develop materials and share them with the rest of the community, to reduce the individual load and to enhance the prospect of uniformity of practice.

Until that stage can be reached, every academic, for every assessment item, must include really clear explanations of the academic integrity requirements for that particular item.

5.9 Policies and procedures

Assuming that it is possible to resolve all of the questions posed in this section, how do we go about persuading our universities to change their policies and procedures – if we agree that the lower-level principles embodied in those documents do not apply to computing assessments? As indicated in section 4.4, our survey uncovered evidence of practices that some considered to be acceptable although they were plagiarism or collusion, and of practices that were not acceptable even though they were not plagiarism or collusion. Does this evidence suggest that the definitions of plagiarism and collusion should be changed to conform with perceptions of what is acceptable? Or should the perceptions of what is acceptable be altered to conform with the definitions?

Should there, indeed, be a new definition of academic integrity that acknowledges the differences we have found for computing assessments? For example, academic integrity might be defined as ensuring that one's work is conducted in the manner expected by the discipline. All we would then need is to agree as a discipline on what our expectations are. Given the nature of professional practice, the expectations regarding reuse and referencing of material, and the way such material is referenced, what does it actually mean to behave with academic integrity in the context of a computing assessment, and, more specifically, of a computer programming assessment?

6. CONCLUSIONS AND FUTURE WORK

The Prianit project set out to answer a number of questions about academic integrity in computing, exploring how the concepts are understood by students and academics, and what practices are applied to address academic integrity.

The project has found answers to those questions, as presented in the prior papers referred to in this one [14-19]. But more important, arising from those answers it has found a new set of extremely serious questions, questions that must be addressed by the computing education community if it is to proceed with integrity.

We, the authors, cannot answer the questions posed here. Instead we urge the community to consider them seriously, and to work collaboratively to resolve the questions.

Unfortunately, the work that lies ahead is not, and cannot be, a set of tasks to be carried out by a small group of researchers: it is work that must be carried out by the whole discipline. Whenever Prianit papers have been presented at conferences, there has been an air of agreement that these are problems that must be addressed, but nobody has proposed solutions to the problems. Rather, the impression has been of people keen to be told what the solutions are.

In essence, the community must now begin the process of deciding what constitutes academic integrity in computing assessment, perhaps starting with computer programming assessment. In detail, this will mean answering many of the questions posed in section 5 of this paper, questions such as the following.

- Are there circumstances in which it is legitimate to incorporate externally-sourced code or algorithms in an academic programming assessment?
- If so, is it obligatory to reference that code and those algorithms?
- If so, how is the referencing to be done? Can we develop referencing standards that will be applied by the whole computing education community?
- Does the same referencing requirement apply to code previously developed by the same student?
- Are there circumstances in which it is legitimate for students doing assessments to seek debugging assistance from their peers and/or from message boards?
- If so, is some sort of referencing required? What sort of referencing, and how would we expect students to quantify the assistance received?
- While it might be interesting to see whether students can explain the programs they submit for assessment, do we accept that this does not indicate whether they wrote the programs, and is often not listed among the assessable criteria for the assessment item?
- Do we understand that in most cases our students have no idea what constitutes academic misconduct in a computing assessment, and that we need to be highly explicit about our requirements in this regard?
- Do we accept that our universities' policies and procedures are almost completely ineffectual for computing assessments, and are we willing to try to have them changed accordingly?

For the sake of argument, let us propose the following answers.

- Programming is to a large extent a component-assembly exercise, with the components ranging from simple plans such as an input loop, through standard algorithms such as a published sorting method, to major library modules. There is a general expectation that existing components will be incorporated where appropriate into a program, and there is no general requirement to reference such components, whether externally sourced or written by the same writer.
- There is therefore no need for a standard referencing method for reused program code.
- It is standard practice, and acceptable, to seek the help of others when developing and debugging code, and therefore this should not be prohibited for students completing coding assessments.
- In the light of the above observations, there is no known way of establishing just how much assistance a student required or accepted in developing the code for a given assessment item.
- Student interviews can be used to establish whether a student understands a program, but not whether the student wrote the program.
- It is not always possible to assess the programming ability of an individual, or of a group, by examining the programs produced by that individual or that group.
- If a program is produced in invigilated circumstances, we have some assurance of the program's authorship, but must accept that the student might have performed very differently if given more time for the task.
- The universities' notions of academic integrity, plagiarism, and collusion need to be altered to reflect these realities.

It is now up to the computing education community to decide whether these answers are acceptable. If they are, the community needs to work out how to adopt them into educational practice. If they are not, the community needs to formulate alternative answers that it does deem acceptable, and work out how to adopt them into educational practice. What should be clear, though, is that it is no longer acceptable for computing educators to believe that they are acting in accordance with their universities' academic integrity policies and procedures, when in fact they are defying those policies and procedures with almost every computing assessment that they set.

ACKNOWLEDGMENTS

Support for this project has been provided by the Australian Government Office for Learning and Teaching, grant SP12-2312. The views expressed in this publication do not necessarily reflect those of the Australian Government Office for Learning and Teaching.

The authors are grateful to the academics and students who gave up their time to take part in the interviews, focus groups, or survey conducted by this project. We are also grateful to the other members of the project team.

REFERENCES

[1] Charles Darwin University, 2015. Academic and scientific misconduct policy. http://www.cdu.edu.au/governance/policies/pol-001.pdf. Viewed September 2015.

[2] Chuda, D, P Navrat, B Kovacova, and P Humay, 2012. The issue of (software) plagiarism: a student view. *IEEE Transactions on Education* 55:1, 22-28.

[3] Clarke, R, and T Lancaster, 2006. Eliminating the successor to plagiarism? Identifying the usage of contract cheating sites. *Second International Plagiarism Conference*.

[4] Cosma, G and M Joy, 2008. Towards a definition of source code plagiarism. *IEEE Transactions on Education* 51:2, 195-200.

[5] D'Souza, D, M Hamilton, and MC Harris, 2007. Software development marketplaces – implications for plagiarism. *Ninth Australasian Computing Education Conference (ACE 2007)*, 27-33.

[6] James Cook University, 2015. Academic acknowledgement and plagiarism policy. www.jcu.edu.au/student/assessmentexams/JCU_090852.html. Viewed September 2015.

[7] Joy, MS, G Cosma, JY-K Yau, and J Sinclair, 2011. Source code plagiarism – a student perspective. *IEEE Transactions on Education* 54:1, 125-132.

[8] Joyce, D, 2007. Academic integrity and plagiarism: Australasian perspectives. *Computer Science Education* 17:3, 187-200.

[9] Mann, S, and Z Frew, 2006. Similarity and originality in code: plagiarism and normal variation in student assignments. *Eighth Australasian Computing Education Conference (ACE 2006)*, 143-150.

[10] McCabe, DL, LK Treviño, and KD Butterfield, 2001. Cheating in academic institutions: a decade of research. *Ethics & Behavior* 11:3, 219-232.

[11] Sheard, J and M Dick, 2011. Computing student practices of cheating and plagiarism: a decade of change. *16th ACM conference on Innovation and Technology in Computer Science Education (ITiCSE 2011)*, 233-237.

[12] Sheard, J, M Dick, S Markham, I Macdonald, and M Walsh, 2002. Cheating and plagiarism: perceptions and practices of first year IT students. *Seventh ACM conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)*, 183-187.

[13] Simon, 2005. Electronic watermarks to help authenticate soft-copy exams. *Seventh Australian Computing Education Conference (ACE 2005)*, 7-13.

[14] Simon, B Cook, J Sheard, A Carbone, and C Johnson, 2013. Academic integrity: differences between programming assessments and essays. *13th International Conference on Computing Education Research – Koli Calling 2013*, 23-32.

[15] Simon, B Cook, A Carbone, C Johnson, C Lawrence, M Minichiello, and J Sheard, 2014. How well do academic integrity policies and procedures apply to non-text assessments? *Sixth International Integrity and Plagiarism Conference (6IIPC)*.

[16] Simon, B Cook, J Sheard, A Carbone, and C Johnson, 2014. Student perceptions of the acceptability of various code-writing practices. *19th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2014)*, 105-110.

[17] Simon, B Cook, J Sheard, A Carbone, and C Johnson, 2014. Academic integrity perceptions regarding computing assessments and essays. *Tenth International Computing Education Research Conference (ICER 2014)*, 107-114.

[18] Simon and J Sheard (2015). Academic integrity and professional integrity in computing education. *20th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2015)*, 237-241.

[19] Simon and J Sheard (2015). In their own words: students and academics write about academic integrity. *15th International Conference on Computing Education Research – Koli Calling 2015*, 97-106.

[20] Shalloway, A and JR Trott, 2001. *Design patterns explained: a new perspective on object-oriented design*. Addison-Wesley.

[21] Soloway, E, 1986. Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM* 29:9, 850-858.

[22] Spacco, J, D Hovemeyer, and W Pugh, 2004. An eclipse-based course project snapshot and submission system. *OOPSLA Workshop on Eclipse Technology eXchange 2004*, 52-56.

[23] University of Western Australia, 2015. University policy on academic conduct. www.teachingandlearning.uwa.edu.au/staff/policies/conduct/university-policy-on-academic-conduct-ethical-scholarship,-academic-literacy-and-academic-misconduct. Viewed September 2015.

[24] Victoria University of Wellington, 2015. Academic integrity and plagiarism. http://www.victoria.ac.nz/students/study/exams/integrity-plagiarism. Viewed September 2015.

[25] Vogts, D, 2009. Plagiarising of source code by novice programmers a "cry for help"? 2009 *Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, 141-149.

[26] Zobel, J, 2004. "Uni cheats racket": a case study in plagiarism investigation. *Sixth Australasian Computing Education Conference (ACE 2004)*, 357-365.