NOVA

University of Newcastle Research Online

nova.newcastle.edu.au

Simon; Sheard, Judy; Morgan, Micheal; Petersen, Andrew; Settle, Amber; Sinclair, Jane; Cross, Gerry; Riedesel, Charles "Negotiating the maze of academic integrity in computing education". Published in ITiCSE '16 Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (Arequipa, Peru 11-13 July, 2016) p. 57-80 (2016)

Available from: <http://dx.doi.org/10.1145/3024906.3024910>

Accessed from: <http://hdl.handle.net/1959.13/1343482>

Negotiating the Maze of Academic Integrity in Computing Education

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

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

Michael Morgan
Monash University
Australia
michael.morgan@monash.edu

Andrew Petersen
University of Toronto
Canada
andrew.petersen@utoronto.ca

Amber Settle
DePaul University
United States of America
asettle@cdm.depaul.edu

Jane Sinclair
University of Warwick
United Kingdom
j.e.sinclair@warwick.ac.uk

Gerry Cross
Mount Royal University
Canada
gcross@mtroyal.ca

Charles Riedesel
University of Nebraska - Lincoln
United States of America
chuckr@unl.edu

ABSTRACT

Academic integrity in computing education is a source of much confusion and disagreement. Studies of student and academic approaches to academic integrity in computing indicate considerable variation in practice along with confusion as to what practices are acceptable. The difficulty appears to arise in part from perceived differences between academic practice in computing education and professional practice in the computing industry, which lead to challenges in devising a consistent and meaningful approach to academic integrity. Coding practices in industry rely heavily on teamwork and use of external resources, but when computing educators seek to model industry practice in the classroom these techniques tend to conflict with standard academic integrity policies, which focus on assessing individual achievement.

We have surveyed both industry professionals and computing academics about practices relating to academic integrity, and can confirm the uncertainty and variability that permeates the field. We find clear divergence in the views of these two groups, and also a broad range of practices considered acceptable by the academics.

Our findings establish a clear need to clarify academic integrity issues in the context of computing education. Educators must

carefully consider how academic integrity issues relate to their learning objectives, teaching approaches, and the industry practice for which they are preparing students. To this end we propose a process that fulfils two purposes: to guide academics in the consideration of academic integrity issues when designing assessment items, and to effectively communicate the resulting guidelines to students so as to reduce confusion and improve educational practice.

CCS Concepts

• Social and professional topics ~ Computing education

Keywords

academic integrity; programming education; plagiarism; collusion

1. INTRODUCTION

Ethical issues in computing education have long been acknowledged as problematic. More than 20 years ago, introducing some of the issues still discussed today, Harris [22] gave an unequivocal statement of why plagiarism is unacceptable:

Students who knowingly plagiarize cheat others by inflating their marks at the expense of students who are sincerely trying to do their own work. They cheat themselves by denying their own education and they cheat the author by claiming the work to be theirs [22 p133].

The subsequent years have seen much debate about the interpretation of ethical concepts such as plagiarism in the context of computing assignments, and a good deal of work has been conducted to probe the understanding of ethical concepts amongst academics and students in computing education [28, 53, 54]. These studies demonstrate confusion and a lack of consistency surrounding the issue of correct ethical behaviour in computing education. There have been many initiatives worldwide to educate students about ethical behaviour; similarity detection tools have been applied and penalty systems introduced; and still computing students engage in plagiarism, collusion, and other forms of

unauthorised aid, apparently not considering that doing so might be problematic.

Researchers have found that computing disciplines account for a disproportionately high proportion of plagiarism cases in tertiary education [7, 15, 18]. Roberts [48] found that 37% of academic misconduct cases at Stanford University over a decade involved computing students, although students in computing courses accounted for only 7% of the student population. Within a postgraduate computing degree, Alam [2] found a significantly higher incidence of plagiarism in programming assignments than in other forms of assessment. These imbalances might be due in part to the working practices and nature of the assignment tasks in computing. Activities such as collaboration and code reuse, core activities in professional program development [58], are often treated as punishable behaviour in academia, leading to potential conflict with the educational goal of preparing students for professional computing careers. In order for computing students to become useful members of the computing profession they need to learn to collaborate and also to become proficient in identifying and reusing appropriate existing code that meets the requirements of an element of a task.

Our ITiCSE working group was formed to investigate academics' assessment practices in the context of academic integrity and any possible disparity with work practices in the computing industry. This paper reports the results of a review of existing research and the results of a survey of computing academics and professional programmers. Based on our findings, we present a process that academics can use to articulate acceptable and unacceptable practices with regard to a particular assessment item. The key feature of the process is a visualisation to explain to students the expectations of how they will conduct their assessment. We present this as a potentially valuable technique for addressing the issue of academic integrity in computing education. The work is presented as a catalyst for further consideration and discussion within the academic computing community.

1.1 Working Group Goals

The main goal of the working group was to consider and clarify how we might articulate acceptable ethical practice in the context of computing education, given the potentially varied requirements of different assessment tasks. In addition, once an academic has designed an assessment task and specified acceptable practice, a process is needed to effectively communicate the guidelines to students in order to reduce confusion and to help them understand the educational justification for the rules adopted and their relationship to professional practice.

In order to achieve these goals the following research questions were proposed:

- 1: What areas of current academic integrity practice engender confusion or conflicting views in academics and/or students?
- 2: With regard to academic integrity, specifically in the use of code and assistance with coding, what are the differing perspectives and ranges of acceptable practices as perceived by computing educators and by computing professionals?
- 3: What aspects of educational and/or professional practice in computing conflict with standard academic integrity policies?
- 4: Can we outline a process by which educators can design assessment tasks while taking into account the interaction of

academic integrity, learning objectives, teaching practices, and professional practice in the computing industry?

- 5: Can a process be designed that will facilitate the effective communication of academic integrity rules to students once an academic has developed these rules for a specific assessment task?

In answering and reflecting on these research questions, we identified the need to rethink the approach that computing educators take toward academic integrity. We came to understand that certain principles underlying academic integrity in computing should actively guide the development of policies and practices. In this paper we provide evidence that answers our research questions, along with broad proposals that we hope will guide the future direction of academic integrity policy and practice in computing education.

2. BACKGROUND: LITERATURE REVIEW

The literature on academic integrity is extensive, and in this review we endeavour to summarise articles relevant to unauthorised aid in computing education. We identify key terms necessary to understand previous work, provide perspectives of staff and students, specify how computing may differ from other disciplines both in practice and in how policies are applied, and highlight important differences between professional practice and academic practice.

The term *unauthorised aid* is used in academic integrity discussion and policies to refer to the full range of unethical behaviour involving obtaining help or ideas in ways that contravene the prevailing policy. The particular aspects of unauthorised aid on which we focus in this review are plagiarism and collusion, which will be explained in section 2.3.

2.1 The issue of academic integrity

A culture of integrity in an academic institution is important for a number of reasons. Dick et al. [17] observe that breaches of academic integrity can harm the students and the educational environment, and can damage the reputation of the academic program, the institution, and the profession.

Arguably the most important issue with unauthorised aid such as plagiarism and collusion is its impact on a student's learning. It requires practice and application to develop high-level thinking skills such as creative problem-solving and critical thinking. A student who cuts and pastes solutions from the internet is not engaging in the problem in a way that is likely to develop these skills. Likewise, a student in a group project who relies on others to do the work will not develop the attendant collaborative and teamwork skills. Ultimately, avoidance of these learning experiences will diminish the student's employment prospects upon graduation.

Breaches of academic integrity also have broader implications. Students who avail themselves of unauthorised aid gain an unfair advantage over others in competition for scholarships, places in study programs, and jobs. Clarke and Lancaster [10] propose that when students receive academic credit for work they have not produced it can "lead to them being placed in employment which they are not qualified for, reflecting badly on the academic institution and potentially depriving a more deserving student of a job" [10 p219]. Observing others receiving unauthorised aid can cause honest students to become aggrieved and perhaps to

themselves consider plagiarising or colluding, harming the education environment for all [17].

2.2 Academic integrity policies

An ITiCSE working group in 2012 [47] studied academic integrity policies in computing education. The group concluded that university-level policies were too general to meet the needs of computing education, and found that, as a result, many computing departments have interpreted, extended, or even replaced their institutional policies. This supports a finding from an extensive survey of UK computing departments by Culwin et al. [14] which found that 80% of all institutions had policies specific to source code plagiarism. Riedesel et al. [47] also note that institutional, departmental, and even course-level academic integrity policies vary greatly among universities and colleges.

Academic integrity guidelines for an individual course or assessment may be used to clarify or override an institutional or department policy and can accommodate different teaching and assessment approaches. However, students then need to be aware of the rules for each separate context. Even where such policies and guidelines are carefully constructed, it is unclear if these specific variations of the standard rules are fully documented by academics and, as a consequence, if they are communicated effectively to students.

2.3 Aspects of academic integrity

This section introduces the topics of plagiarism and collusion, including the related topics of self-plagiarism, code reuse, and outsourcing, and considers these within the context of computing education.

2.3.1 Plagiarism

Plagiarism has long been recognised as an academic offence. Many definitions of plagiarism found in the literature are very general; for example, Fraser [18] defines plagiarism as “the act of copying someone else’s work whether or not the original author is aware of the act” [18 p183]. Often the definitions allude directly or indirectly to text-based work. For example, Jones et al. [27] define plagiarism as occurring when

...a student incorporates another person’s or body’s work by unacknowledged quotation, paraphrase, imitation or other device in any work submitted for assessment in a way that suggests that it is the student’s original work [27 p20].

A number of authors have attempted to capture the concept of plagiarism in the computing disciplines, relating plagiarism to the unacknowledged use of others’ code, ideas, designs or any other creative component [20]. A key point here is that the use of any of these elements is not in itself a problem: the integrity issue arises when the original source is not acknowledged [20].

Despite the apparent simplicity of defining plagiarism there are many ‘grey areas’ relating to levels of similarity, the definition of common knowledge, and the student’s intent [41]. Within computing education, studies have repeatedly exposed these grey areas and revealed students’ lack of understanding as to how the term *plagiarism* should be correctly interpreted [29, 52, 54]. As noted by Hawley, “the problem becomes one of determining the ‘ethical threshold’, i.e., the line of demarcation between acceptable and unacceptable behaviour” [24 p35].

2.3.2 Self-plagiarism

Self-plagiarism, also referred to as replication, is the act of including one’s own previously submitted work in a subsequent assessment [11, 27]. Many universities have regulations banning the reuse of work that has been submitted for credit in a previous assignment, so self-plagiarism may infringe assessment criteria as well as offending against an ethical code. However, it is also one of the consistently noted areas of misunderstanding among computing students [29]. In the context of computing education, common computer coding practices, such as object-oriented programming techniques, may increase students’ confusion regarding appropriate reuse of code they have previously created.

2.3.3 Code reuse

The concept of reuse merits explicit attention when discussing plagiarism in computing education. Reuse of software components and algorithms is a fundamental feature of successful software development. The task of identifying appropriate resources, and adapting and incorporating them in a systematic way to fit a new context, is challenging and creative work [20].

Code reuse might be viewed simply as a specific instance of including pre-existing material. However, given the central role of the practice in computing, a number of authors have explicitly considered the implications of reusing one’s own code or that of others. Gibson [20] suggests a code of practice to distinguish acceptable and unacceptable forms of computing code reuse. Petre [45] suggests that code writing can be viewed as a construction process in which tinkering, modification and repurposing are all part of legitimate experimentation and learning. It is therefore important to clarify the way in which the term *reuse* is constructed:

Drawing that line between legitimate reuse and misuse clearly and explicitly in a way that considers various learning goals and contexts is essential in a multi-cultural setting where there can be widely varying interpretations of when copying is legitimate or even encouraged, and when it is offensive [45 p15].

This challenges the computing education community to find a conceptualisation of reuse, and a way to explain the boundaries, that can help to foster learning while clarifying what constitutes misconduct.

2.3.4 Collusion

Activities where students discuss and work jointly on a task can be highly beneficial for their learning. If the joint work is permitted, this situation is called *collaboration*; if the joint work is not permitted, it is called *collusion* [18]. Jones et al. define collusion as:

The collaboration without official approval between two or more students (or between student(s) and another person(s)) in the presentation of work which is submitted as the work of a single student; or where a student(s) allows or permits their work to be incorporated in, or represented as, the work of another student. [27 p20].

The distinction between collaboration and collusion is often poorly defined [18]. It is troubling and confusing that a practice considered acceptable in the context of one institution, department, or assessment may be considered unacceptable in another. For example, one department may view it as acceptable collaboration for students to discuss an individual assignment, while another may label the same activity as collusion. Fraser [18]

notes the impossibility of attaching fixed meanings to these distinctions. He refers to these movable boundaries between concepts as ‘idiosyncratic’: that is, it is inevitable that instructors will have their own interpretations. This implies that the limits of permitted collaboration must be set and communicated effectively to students for each individual context.

2.3.5 Outsourcing

Another form of unauthorised aid is *outsourcing* or *contract cheating*, where a student obtains an assignment solution from another person and submits it as their own work. This aid might be freely offered in response to the student posting the assignment problem on a forum or it might involve paying somebody else to write the solution [31]. Contract cheating is a difficult problem to address. Typically, the outsourcing produces work that is original, and cannot be detected by detection tools, which look for similarities with other solutions [10].

Several well-established sites on the internet offer code outsourcing services for legitimate purposes. However, a study from 2005 to 2012 by Clarke and Lancaster [10] found that a number of these sites are used extensively by students for the purposes of contract cheating. This form of cheating appears to be increasing in prevalence, and is difficult to detect as the sites are quite sophisticated in their operations.

2.3.6 Seriousness of Plagiarism and Collusion

Although there is widespread agreement in the literature that practices such as plagiarism and collusion are offences against academic integrity, many authors consider some practices to be more serious than others, thus introducing the idea of severity or levels of infringement. McInnis and Devlin [41] consider that plagiarism is not all equal and may be graded along the three dimensions of extent, intent, and penalty. It is likely that most institutions would concur with this unofficially if not in a formalised way; for example, by treating plagiarism of a whole assignment more severely than unattributed copying of a few lines. Attempting to establish intent is also common but can be difficult in practice [61]. The perceived severity of an infringement might also be influenced by the year group of the student and whether previous offences have been recorded.

A number of studies have investigated perspectives on integrity offences that are ‘more’ or ‘less’ serious. Wilkinson [59] found that staff and students rated collusion as being a less serious offence than plagiarism. This supports earlier work by Nuss [42], which found agreement between staff and students that collusion (such as working on assignments with other students when not permitted) and minor forms of plagiarism (such as copying answers from a source and copying short pieces of text without footnoting) were the least serious offences. A rationale from staff, as proposed by Barrett and Cox, is that with over-zealous collaboration “at least they are learning something!” [5 p116].

The use of ‘essay mills’ and other forms of contract cheating has long been viewed as more heinous than other acts of plagiarism and collusion [24]. More recently the ready availability of sites such as Stack Overflow enables students to obtain complete solutions for computing tasks, with the outside assistance being provided free of charge.

Fraser [18] suggests a hierarchy of seriousness from collaboration to collusion to plagiarism. However, as noted above, the fluid or idiosyncratic boundaries of these concepts mean that any given

action can only be properly classified within the appropriate context of applicable policy.

2.4 Addressing academic integrity violations

The literature discusses many methods that have been proposed to address academic integrity in the field of computing education. For example, several studies have suggested that the way forward is to educate students about acceptable and unacceptable aid. Lampert [30] calls for discipline-specific approaches to plagiarism education, but the only details provided are for journalism, which limits the utility for computing educators. Dick et al. [17] discuss cultural change regarding cheating and recommend requiring students to sign work in order to indicate how much of it was their own and how much was from other sources. A study in South Africa used a ‘community of practice’ approach to educate students about textual attribution, a process that resulted in diminished numbers of breaches [40].

Another prevention strategy is defensive assignment-setting: designing the assessment task so as to reduce the likelihood of unacceptable behaviour. Suggested approaches include making the assignment unusual and local/specific; adopting a portfolio approach, in which work is developed and collected over time; requiring early drafts, so that the history of development is recorded; using collaborative tasks; using authentic assessment tasks; questioning students; and providing plagiarism feedback on a first draft [5, 21, 57]. These approaches may not always assess the aspect intended, as with questioning students about their code [57], or may introduce problems of their own, as with the need to assess individual contributions to group work tasks.

While prevention is preferable, detection and punishment are also generally acknowledged as necessary elements of deterring unacceptable behaviour [40]. Many papers have been written about the development and use of algorithms and tools for detecting and reporting code similarity. Le et al. [35] conducted a pilot study of the educational value of a code similarity detection tool that was integrated into a learning management system. Some students who participated in the study reported that seeing similarity between their submissions and other students’ submissions discouraged them from collaborating, a consequence that the authors noted might be problematic [35].

While the above approaches address some aspects of the problem of academic integrity in computing, we believe that approaches that simply seek to implement and/or enforce generic academic integrity policies in the computing context fail to address some fundamental aspects of the problem. The continuing prevalence of academic integrity issues involving computing students indicates that a different perspective is needed to address this issue.

2.5 Why is computing different?

Previous work has acknowledged that academic writing differs from writing in other contexts. Pecorari and Shaw note that “academic writing ... is subject to the sort of intertextual ties which inevitably arise when texts share features such as a common topic, purpose, or register” [44 p150]. This sort of commonality is even more pronounced in computing than in other disciplines, as computer programs differ in structure and purpose from essays, research papers, and other written texts. Some work in the computing education literature has focused on understanding these differences and their impact for academic integrity.

In a study of originality in student submissions, Mann and Frew [38] examine ‘normal’ variability in student programming assignments, in part as a way to clarify the threshold for what should be considered plagiarism. The authors used a normalised value for the number of lines of code, the cyclometric complexity, and the function count. Programs were compared pairwise to obtain a structural dissimilarity index for each submission. The authors suggest that students who have highly similar submissions may be those who are carefully following exemplars [38]. Louw and Pieterse [37] propose that similarity between submissions is more likely in the context of introductory courses with large enrolments since it is more likely that two students will arrive at the same solution independently. However, the authors do not justify this proposition.

Correct attribution of external sources is generally taken as the cornerstone of academic integrity. For text, there is a long-established tradition of referencing and citation with explicit formats set out for making appropriate acknowledgements [44]. However, the rules for textual documents seldom apply to computing assignments. Simon and Sheard [58] observe by way of example that there is no sense in placing externally sourced code in quotation marks, as this will give rise to syntax errors.

Inadequately acknowledging source-code authorship is one of the main categories of source-code plagiarism; however, there is no generally agreed way of providing such acknowledgement [28]. Gibson [20] lists some of the different ways in which code attribution might be made. These range from a vague statement that the code has been externally inspired, to a full reference, listing and difference file. These obviously imply differing levels of effort on the part both of the student and of the academic who must check that they are provided correctly. Even at the simplest level of providing comments of attribution within the source code, there is no agreement on how these should be provided or on how the extent of the content should be denoted (including aspects such as the length of the code, whether it is verbatim or ‘inspired by’, whether adaptations have been made, and so on). While referencing guidelines are sometimes given at a departmental level or by individual academics, this is often not the case. It is thus likely to be confusing to students, who will not know what is expected in any particular circumstance.

2.6 What is regarded as acceptable?

Computing academics, computing students, and indeed computing professionals must daily negotiate the territory of ethical practice. A number of studies have investigated how academics and students interpret the limits of appropriate behaviour in various practical situations.

2.6.1 Academics’ interpretations of plagiarism and collusion

A survey by Cosma and Joy [12] of UK computing academics (n=59) and a survey by Simon et al. [55] of Australian computing academics (n=70) used scenarios to gain the academics’ perceptions of what constitutes plagiarism and collusion. Both studies found a high level of agreement among the academics that submission of code from another source without acknowledgement constitutes plagiarism. Most considered that submission of a modified version of another person’s code without acknowledgement is plagiarism, but a few suggested that it depends on the extent to which the code is adapted. Some academics in Cosma and Joy’s study [12] raised concerns that the

ethos of component reuse in authentic programming environments, such as object-oriented development, mitigates against these scenarios being viewed as offences. They expressed the feeling that students should be encouraged to participate in ‘real’ programming practices without this being viewed as an ethical infringement. The comments highlighted potentially confusing grey areas and the need to define boundaries and contextualise for any given circumstance.

There was less agreement about the practice of self-plagiarism. About half the respondents in Simon et al.’s survey [55] considered using work that had been written and submitted for a previous assignment to be plagiarism. In Cosma and Joy’s survey [12] 81% of respondents saw a similar scenario as problematic, with 29% considering it to be plagiarism and 53% as some other offence. Again, comments from academics indicated that some felt such restrictions should not be enforced because they conflicted with the way they wanted their students to work in practice. One respondent asked “Would this ever happen in a programming-oriented module when we behave students not to reinvent the wheel?” [12 p97]. Another noted the need to prepare students for future teamwork.

The issue of collusion was covered in both surveys. Simon et al. [55] presented several scenarios about students discussing or showing their work to fellow students when they were meant to be working independently. Fewer than a quarter of the academics viewed these as collusion. Cosma and Joy found general agreement that “it is ‘pedagogically valuable’ for students to engage actively in sharing ideas while discussing assignments as long as they do not copy each other’s work” [12 p198]. This is another potentially confusing grey area that is often reflected in computing ethics statements: phrases such as “you may discuss but write alone” and “limited collaboration only is allowed” are employed but not clarified.

The issue of outsourcing was covered by the survey of Simon et al. [55]. In scenarios that described purchasing code written by others or paying someone to write the code there was clear agreement that these practices are unacceptable to academics.

2.6.2 Students’ interpretations of plagiarism and collusion

In addition to the studies of computing academics there have been scenario-based surveys of computing students to gain their perceptions of what constitutes plagiarism and collusion, and of the acceptability of these practices.

The studies showed that most students were able to identify obvious cases of plagiarism and considered these practices to be unacceptable. A survey by Joy et al. [28] investigated how UK computing students (n=770) interpret a range of scenarios covering possible incidents of plagiarism. Most students (81%) indicated that copying code from a book and using it without reference was plagiarism; however, a further 13% responded that they were not sure. A survey of Australian computing students (n=486) by Simon et al. [56] presented a similar scenario where an assignment was based on code that is freely available from a source, such as the web, without referencing. Although a high number (76%) of the students deemed this unacceptable, 10% saw it as acceptable and 14% were unsure.

A focus group study by Simon et al. [56] provided insight into the students’ ratings of the acceptability of various code-writing

practices A number of interesting perspectives emerged, such as the feeling that identifying, adapting, and incorporating existing code is in itself a creative activity, and that this should be allowed without negative consequences. Some students explained that they did not perceive copying code to be a serious offence if they had subsequently worked extensively on the code:

While some students were confident that using someone else's code without attribution was not acceptable, a number felt that unreferenced copying was justifiable if the code was changed sufficiently [56 p107].

Offering another perspective, some students in the focus group study indicated that copying code was legitimate so long as they understood the code [56].

Surveys of computing students at an Australian university conducted in 2000 (n=298) and 2010 (n=117) by Sheard & Dick [51] asked students to rate the acceptability of various practices on a five-point Likert scale, where 1 indicates *acceptable* and 5 indicates *not acceptable*. In 2000 the mean ratings of 3.81 for copying material from a text book and 3.85 for copying material from the internet indicate that most students viewed these practices as unacceptable. When the survey was repeated in 2010 the mean ratings had risen significantly, indicating that the students in 2010 viewed these practices as being even more unacceptable.

Some responses to the surveys suggest that some of the students may not have understood what plagiarism is, causing over-zealous interpretations in some cases. For example, nearly a quarter of the respondents in the Joy et al. survey [28] thought that reading a book for background information (but completing an assignment without the aid of the book) constituted an offence. It is, however, difficult to imagine that they would reference all such sources in practice. In the Sheard and Dick surveys [51] some respondents felt that showing an assignment to a lecturer for guidance was unacceptable. These surveys did not collect qualitative data, which could have provided explanations for the responses observed.

The Australian and UK surveys each presented scenarios that describe self-plagiarism. The students generally rated this practice as more acceptable than plagiarism from other sources, although their responses indicated some uncertainty. Only a few students (7%) in the survey by Joy et al. [28] identified copying code from a previous assignment as plagiarism or possible plagiarism. In the survey by Simon et al. [56] students were asked about the acceptability of submitting without referencing an assignment based largely on work that they had written and submitted for a previous course. More than half the students (58%) deemed this unacceptable, while 22% found it acceptable and 20% were unsure. The students in the 2000 survey by Sheard and Dick [51] rated resubmission of an assignment from a previous course as 2.82 on the five-point Likert scale, and this had not changed in the survey conducted a decade later.

Each survey explored the students' views of collusion. The students rated this practice as more acceptable than plagiarism but not necessarily as acceptable as self-plagiarism, although many were uncertain. In the survey by Joy et al. [28] a scenario described a situation in a group assignment where two students assigned to two different groups exchange code and submit it as part of their separate group assignments. More than half the students (62%) considered this unacceptable, but 28% were

unsure. Simon et al. [56] presented a couple of different collusion situations. In a scenario where a student discussed the detail of their code with another student while working on it, only 20% found this unacceptable and 57% deemed it acceptable, with 23% unsure. In another scenario, where a student while completing an assignment added features that they noticed when looking at another student's work, 36% found this unacceptable, 32% deemed it acceptable, and 32% were unsure. The respondents in the 2000 survey by Sheard and Dick [51] thought that collaborating with other students on an individual assessment was more acceptable (2.54 on the five-point Likert scale) than plagiarism or self-plagiarism; but a decade later such collaboration was seen as less acceptable. All of these examples indicate a high level of uncertainty and disagreement amongst students.

The most unacceptable practice was outsourcing. In the survey by Joy et al. [28] almost all the students (97%) responded that paying a student from a previous year to write an assignment was unacceptable. The students in the survey by Simon et al. [56] had similar views, with 95% finding it unacceptable to pay someone to write code and 85% finding it unacceptable to purchase code written by others, but 11% were unsure. The students in the 2000 survey by Sheard and Dick [51] rated the practice of hiring someone to write one's assignment as highly unacceptable (4.51 on the five-point Likert scale), and a decade later it had become even more unacceptable.

The findings from these surveys indicate that many students find some aspects of academic integrity confusing. For text-based assignments there are rules and conventions for paraphrasing written text that help to determine when a text is altered sufficiently to be considered an original work and when attribution is required. There is no such consensus in the case of code modification and attribution. The lack of clear guidelines and of consistent rules on how to provide attribution were raised as potential issues. Opinions were also expressed relating to the purpose (and consequently the expected nature) of university assessment tasks, viewing them as "practice for when you can get a job so you should just have to understand and like work within the laws of copyright" [56 p107]. For students who think like this, the requirements of university codes of ethics may be seen not only as confusing, but as absurdly restrictive and detrimental to their learning and preparation for the world of work. The next section considers further the differences in accepted practice between industry and academia.

2.7 Differences between industry and academia

Many studies have examined the relationship between computing education and professional practice in the computing industry [3, 23, 36, 43, 46, 50]. Fuller et al. [19] observe that

Students studying computing fields often have strong career orientation and have come to study computing with the intent of getting a graduate job in computing. It is the responsibility of academics to acknowledge these aspirations in their teaching and assessment [19 p174].

Two important aims of university education are to prepare students for a professional career and to enable them to obtain the skills required in order to transition into the work place. Simon and Sheard challenge the validity of assessments that do not address "skill that is required in the normal working environment"

[57 p238] and argue for authentic learning contexts [25, 33, 34] and for tasks that replicate important aspects of the target professional practice of the industry, in this case the computing industry. They conclude that:

in much of programming education there is an unbreachable rift between the goal of authentic assessment, which necessarily entails collaborative work, and the need for summative assessment of individual effort, which typically requires work in isolation [57 p237].

Baugh et al. [6] discuss the alignment of academic practices regarding coding plagiarism and current computing industry practice, particularly in the area of teamwork. They focus on the lack of alignment between coding practice in the real world, which often revolves around coding in teams, and the academic practice of setting individual coding assignments. To illustrate this point they quote a 2010 interview by David Foote, CEO of Foote Partners:

In the real world, people write [software] code in teams where they are given pieces of a project to work on. The academic world should be mapping onto the real world . . . they shouldn't be handing out assignments where people are coding on their own [6 pp142-143].

Dick et al. [17] discuss the impact that educational goals can have on the amount of collaboration that is allowable, and specifically suggest that in certain circumstances, such as when individual assessment is not required, collaboration can be usefully encouraged. As Petre [45] writes,

A quick scan of SIGCSE proceedings makes it evident that many educators are adapting their pedagogy to embrace change in the discipline, and in particular to accommodate notions of reuse, composition and collaboration [45 p15].

Collaboration can greatly enhance student learning, but collaborating on individual assessments is often regarded as academic dishonesty.

Work practices in the computing industry have some unique characteristics, such as the prevalence of teams in coding activities, and explicitly designing code for reuse in order to improve productivity. Sant [49] discusses how the availability of online code resources has changed developer workflow:

The modern developer's workflow includes code foraging, the use of 'question and answer' sites and references to online API documentation [49 p295].

Sant also reports a study by Brandt et al. [9] which "found that programmers were using the web for 'just-in-time learning', to refresh their knowledge of complicated syntax and to clarify and extend their existing knowledge" [49 p295]. That is, professionals in industry are finding ways to increase productivity while at the same time updating their skills and learning new techniques. In the academic context, Sant [49] points out the costs of prosecuting plagiarism and the ambiguities in assessing what constitutes plagiarism in each case for computing students. He suggests giving a greater weight to controlled and restructured 'repurposing tasks' by careful assessment design rather than having students do 'repurposing' in an uncontrolled way.

Joy et al. [29] argue that code reuse is an integral aspect of professional computing practice. In fact, programming students are encouraged to design code to address this industry practice.

However, there is a fundamental conflict between these professional practices and the traditional rules concerning plagiarism, a conflict that both teachers and students find it difficult to resolve.

Programmers normally use existing software libraries to build new software, rather than building everything from scratch – hence, source-code reuse is acceptable practice by professional programmers who aim not to 're-invent the wheel'. For this reason, source-code reuse may be encouraged in some programming assignments (after all, the purpose is to prepare students for employment). It is therefore important for university policies to address acceptable software reuse, plagiarism and ethics in software development [29 p5].

Students undertaking assessment tasks are often required to develop new code from scratch. In view of the specific characteristics of the programming domain raised above, and of prevailing industry practice, students may find it difficult to understand the justification for the design of such assessment tasks. Louw and Pieterse [37] note that because the industry promotes code reuse, students should learn the skills required for responsible and effective reuse of code without being accused of plagiarism.

In the study by Simon et al. [56] there was a focus on code-writing practices, in particular a scenario-based discussion of code reuse practices and the use of open source code. The study reported that students often conflate issues of copyright and academic integrity. For professional practice in the computing field, code reuse practices are often guided by copyright considerations and legal issues. As previously noted, one student suggested that professional standards should apply in both situations:

[University assessments are] practice for when you can get a job so you should just have to understand and like work within the laws of copyright' [56 p107].

This statement indicates that some students fail to see the rationale and alignment with current professional practice of academic attribution practices, and are thus less likely to comply with the policy. Therefore it may be beneficial for academics to provide students with a rationale as to why this is important in the learning context. Joy et al. [28] suggest that for students to understand and comply with academic integrity rules, the issue of sharing code for individual assignments and the extent of allowable collaboration is seen as particularly problematic. They note that this is especially the case for students with current or past work experience who may routinely work in a way that is consistent with their professional experience but that conflicts with current academic policies.

The ethics of computing in industry is addressed by the standards of the professional bodies. The IEEE code of practice requires members "to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others" [26]. The ACM code of ethics states that

computing professionals are obligated to protect the integrity of intellectual property. Specifically, one must not take credit for others' ideas or work, even in cases where the work has not been explicitly protected by copyright, patent, etc. [1 s1.6].

These codes of ethics are deliberately broad, and cannot reasonably be expected to go into precise detail concerning attribution of algorithms or code. But even in their breadth, they might be seen as conflicting with accepted professional practice. Do the injunctions “to credit properly the contributions of others” and to “not take credit for others’ ideas or work” mean that professionals in these societies are indeed required to attribute every piece of code that might have been written by somebody else? If so, the codes of ethics might require revision to bring them into line with accepted practice. Alternatively, is it understood that the contributions, ideas, and work being discussed are at a substantially higher level than code? In that case, the codes of ethics might benefit from a clearer explanation of their intent.

There are good reasons to work towards a clear connection between academic and industry practices with respect to academic integrity in computing. McCabe et al. [39] note that

Dishonest behaviour in the workplace was lowest for participants . . . who currently worked in an organization that had a strongly implemented code of ethics [39 225].

While McCabe’s work is not specific to the computing field, there is evidence that adherence to ethics policies can have long-term effects with potential benefits for both academia and industry. Similarly, alignment of educational practice with industry practice could be beneficial in many areas of academic integrity that are currently problematic for both academics and students.

2.8 Summary of issues emerging from the literature

In summary, many students are unsure what is required of them in terms of academic integrity when programming [56]. This is not surprising, because many academics are also unsure [55]. There appears to be no baseline understanding of the meaning or requirements of academic integrity in the context of computing education.

There also appears to be a fundamental flaw in the computing context in the apparent goal of academic integrity to ensure that work submitted for credit is the student’s original work or ideas in the student’s own words. The literature review highlights the significant tension between the desire to encourage collaboration and reuse, which have been shown to benefit learning and engagement [8, 60], and many academic integrity policies.

The literature review documents a number of conflicting approaches between academic and industry practice in the area of integrity. Industry practice is evolving rapidly in terms of the availability of online resources and code reuse. Educational practice is also changing, with a greater emphasis on collaborative learning and authentic learning environments. It is therefore important to take a fresh look at these issues from both the academic and the professional perspective.

3. METHOD

This section describes the method used to investigate the issues raised in the literature by exploring the views of both academics and computing professionals regarding integrity issues. In order to further investigate these issues we designed an online survey that was built around a number of scenarios involving integrity issues in the context of programming. By inviting both academics and practising programmers to take part we aimed to gain a better understanding of the variety of approaches adopted in various

contexts and to develop proposals to improve educational practice in the computing context. We focus on aspects of unauthorised aid within the context of computing, and in particular on the topics of plagiarism and collusion. The online survey was circulated among computing academics and professionals, to determine their views regarding seeking help and using the work of others.

3.1 Survey questionnaire

The survey questionnaire was designed by one of the working group leaders and trialled by working group members. The questionnaire included a number of scenarios that described using the work of others and seeking assistance from others. The academics were asked to respond as they would hope their students would respond; the professionals to respond on their own behalf. The scenarios described practices that in some contexts could be seen as plagiarism or collusion; however, we did not use these value-laden terms, instead keeping the wording of the questions neutral.

The research instrument consisted of three sections: preamble questions and instructions, targeted at either academics or professionals; common scenario-based descriptions and questions; and demographic questions. The preamble questions for academics related to their knowledge of current academic integrity policies at their institution and the appropriateness of these for the computing discipline. The preamble for professionals asked about codes of practice and work practices. The scenario-based questions were common to both groups, with slight rewording to account for the different contexts. To illustrate, here are a scenario description and its associated question:

You have decided that your program needs to make use of a heap sort. You don’t remember the exact implementation details of heap sort, so you look up the algorithm on the internet and then code that algorithm in your program. Do you add a comment giving credit to the website where you found it, or to some other source that the website gives credit to?

yes / no / it depends (plus space for explaining)

The demographic questions were only slightly different for each group, and related to the country where they worked, a description of their institution or workplace, and years of experience.

The survey was promulgated to computing academics by way of the SIGCSE-members mailing list. In addition, members of the working group circulated it among their own contacts, both academic and professional. While a respondent’s path through the survey was determined by whether they identified as a programming educator or a professional programmer, there was no verification that respondents did indeed fall into one of these groups. The survey was open for five weeks in June and July of 2016.

3.2 Analysis

The responses were analysed in two parts. First, the yes / no / it depends responses were aggregated and counted. All counts are presented in Appendix A. Second, the free responses were examined and quotations were chosen to illustrate significant themes that emerged from them.

We note several potential threats to the validity of the survey instrument and the collected data.

1. Some respondents tried to answer some questions in general rather than for the specific example provided. They explained that they would cite in general, but that they wouldn't remember to cite in this case, or that they wouldn't cite trivial cases such as this one.
2. In the question about a basic algorithm, some of the respondents may have been thinking about copying code rather than looking up an algorithm. For example, the 'it depends' responses often draw a line between whether they are just copying the idea (the algorithm), which they would not cite, or copying pseudocode or code, which they would cite.
3. While educators were asked to respond as they would hope *their students* would respond, some appeared to answer for themselves.
4. While educators were asked to respond as they would *hope* their students would respond, some appeared to answer as they *believed* their students would respond.

Where possible we have indicated where responses fall into the above categories; however, we believe that the majority of responses provide valid and significant insights into the issues being examined.

4. SURVEY RESULTS

This section presents the responses to the survey of perceptions of academic integrity and professional practice. Each section of the survey is presented in turn, with demographics and responses about the existence of academic integrity policies and codes of professional practice first, results from scenarios next, and responses about appropriate formats for acknowledgments last. In each section, the responses from academics are generally presented first, followed by responses from professionals. Direct quotes from respondents are followed by the respondent number, such as A42 for an academic or P13 for a professional.

4.1 Demographics

Responses to the survey were received from 106 respondents identifying as programming educators and 38 identifying as professional programmers. The respondents had a range of backgrounds, and their experience ranged from 1 to 46 years, with a median of 20 years for academics and 12 years for professionals. Table 1 lists their geographic locations. Respondents were primarily from North America, with Australasia and the United Kingdom providing the majority of the remaining responses.

Table 1: Location of academic and professional respondents

Geographic region	Academic	Professional
Africa and the Middle East	3	0
Australasia	14	8
Europe	4	3
North America	75	24
South America	0	1
UK	9	2

A majority of academic respondents worked at four-year universities, with about half of those working at universities described as having graduate programs. The other half worked at liberal arts or primarily undergraduate institutions. A small

number of respondents (4) worked in high schools or two-year colleges. It should be noted that these characterisations are inherently northern American: in many other countries the norm is a three-year degree and there is no such thing as a liberal arts institution. Twenty-three respondents simply characterised their institution as 'university'; at least one provided a description meaningful only in Australia; and one, intriguingly, came from a 'Public Statue University'.

Most professionals described their roles as being in software development, with small numbers of individuals further identifying their fields as chemical, education, energy, financial, healthcare, or manufacturing.

4.2 Academic integrity policies

As shown in table 2, students are almost universally required to abide by an academic integrity policy, and academics strongly hold this belief: "Absolutely all students must abide by SOME academic integrity policy. The question is ridiculous." (A108). However, some note that the policies are not uniformly enforced or, more commonly, that the policies are customised – officially or not – for a discipline, a course, or even an individual assessment. As one academic observes:

The policy can be interpreted to apply to computer programs, and has been interpreted to apply to programs, but that interpretation isn't always clear. This is why it's important (and common) for instructors to include more specific interpretations in their syllabi and on their assignments. (A84)

Table 2: Are your students / you required to abide by an academic integrity policy / code of practice?

Integrity policy / code of practice?	Academic (n=106)	Professional (n=38)
Yes	92%	50%
No	1%	39%
It's not that simple	6%	11%

Table 3 shows the sources of academic integrity policies identified by respondents and provides additional evidence that policies are often defined, re-defined, or interpreted at multiple levels. While academic integrity is an institutional concern, with 72% of respondents' institutions setting academic integrity policy, 44% of respondents report that policy can be set or interpreted at multiple levels. After the institution, responsibility for defining or interpreting academic integrity falls to the department (35%) and/or the individual instructor (45%).

Table 3: Sources of academic integrity policies

Source	Percent reported
Institution	30%
Institution and instructor	17%
Instructor (course/assessment)	13%
Institution, faculty/department, and instructor	13%
Institution and faculty/department	12%
Faculty or department	8%
Faculty/department and instructor	2%
Multiple (unspecified)	1%
Not reported	5%

The need to define integrity at multiple levels is driven by differences in context. Institutional policies should be inclusive. As one academic writes, their policy is “written broadly enough to account for academic misconduct in essentially any type of course, including programming courses” (A97). As a result, several respondents noted that it was difficult to communicate the policy to students: “because it is written so broadly, it is not effective in communicating to students the sorts of issues they should be concerned with” (A43). As a result, many find their role to be one of contextualisation: “The high-level policy is general enough to cover software dishonesty. But students need concrete examples/applications from the local policy.” (A75)

This issue of communication is made more urgent by the perceived need for different rules in different courses or even in different assessments. One academic notes that “The details vary depending on the course learning outcomes” (A111). Another indicates that rules vary across years of study:

... in first year students are required to work individually and are not to share code. They may discuss ideas on how to solve problems but not specific code. In second year students are allowed to reuse code from other sources, but MUST cite where such reused code comes from. (A117)

Table 4: Is the academic policy at your school appropriate for assessments such as computer programming?

Response	Percentage reported
Yes	49%
No	21%
It’s not that simple	31%

As table 4 shows, about half of the academic respondents believe that the policies in place at their school are appropriate for computer programming assessments. Academics who were satisfied generally described a policy that is sufficiently broad or high-level to permit the concepts to be transferred and applied to a range of courses. They also mentioned the existence of local refinements that helped to apply the policy to the computing context. Dissatisfaction occurs where the institutional policy is perceived to be written narrowly (often specifically for essays) or where more specific definitions and procedures have not been defined at a lower level: “The general principles at the faculty level aren’t sophisticated enough -- they apply reasonably straightforwardly to essays, but don’t cover programming well” (A111).

As found in policies, the term *plagiarism* give rise to frustration. One academic notes that “Students often don’t understand that plagiarism applies to code as well as to their English essays” (A51). But it’s not just students who find the term challenging. Academics also disagree with one another, with one writing that “Essays and code can both be plagiarized. However, the means for determining plagiarism can be different ...” (A106) and another saying “There is not an applicable high-level policy for programming. Just for plagiarism.” (A101). The latter opinion may be rooted in the manner in which plagiarism is defined and taught about:

... [the policy] is careful to define plagiarism quite broadly. However, the aids produced by the Office of the Dean and the education done around plagiarism focus on citation and other aspects of academic communication which don’t necessarily apply to computer science. (A114)

This difficulty potentially extends to other terms that may need specific interpretation within a computing context, such as collusion, unauthorised aid, and self-plagiarism.

4.3 Professional codes of practice

In contrast to the near-universal adoption of academic integrity policies, only about half of the professional respondents indicated that they are governed by a code of practice. The adoption of a code appears to be related to legal liability and the protection of intellectual property. Several professionals noted a requirement “to follow copyright law and associated software licenses” (P5), and another noted the existence of “a moonlighting policy that prevents use of company code in side-projects” (P28).

As shown in table 5, the majority of codes of practice are set by the employer. The level of specificity provided varies widely, with one professional noting, “There are general guidelines. However they cannot cover everything without limiting the ability to do our work.” (P17). Another reports on a policy with more detailed processes: “Certain licenses are ok to use without legal department consultation, though we need to add to a list so legal can be aware and verify them” (P18).

Table 5: Sources of professional codes of practice

Source	Percent reported
Copyright law	3%
Employer	46%
Professional society	3%
Self-defined	9%
No code of practice	40%

Even in the case where no formal policy is defined, some professionals feel that there is an implicit code of practice: “I’m not sure there’s an explicit policy. It’s understood we won’t incorporate external code that may belong to other corporations and we shouldn’t include code whose license will violate our business model.” (P29). Several reported that they defined their own codes of practice, often with past experience as a guide: “... informed by my previous workplace where it was set by the company legal department” (P27).

4.4 Perspectives on use of code

This section contains results from the scenarios presented in the survey. Each scenario begins with a general question or description, then proceeds with a sequence of questions varying one or more factors that might change perceptions. Fixed responses were limited to ‘yes’, ‘no’, and ‘it depends’; and every question had a field for comment, elaboration, or explanation. The complete counts of the quantitative responses for all scenarios are presented in Appendix A.

4.4.1 Using standard algorithms or code

Do you provide attribution when you use standard code or algorithms in your program?

Specific scenarios exploring code attribution practices are discussed below.

Code for a basic loop

You have coded a loop that adds all the values in an array. The loop is exactly the same, symbol for symbol, as one that was used in the teaching materials when you were taught programming. Do

you add a comment explaining where you first encountered this code?

Fewer than a quarter of the academics responded ‘yes’ to this question. Some academics commented that material presented in class is free to use and is considered ‘common knowledge’ unless otherwise specified. One comment related to the level of the students:

... as students progress in programming they will be using similar code often for this purpose and I wouldn't expect [them] to cite it every time. But in an intro course if they copied such a thing symbol for symbol they should cite it. (A118)

More than half of the academics responded ‘no’ to the question. A number commented that the example was too trivial to require acknowledgement and it would be difficult to recall when you first encountered such a simple construct. One academic explained:

... I don't see the point in crediting the source, as the source probably wasn't originally the instructor or the textbook author anyways when viewed from the perspective of a MOSS run. (A51)

A number of academics gave the ‘it depends’ response. For most it depended on the source of the code: “If it was in the textbook, yes; if from my lecture slides, handouts, or their class notes, no” (A42).

Only one professional responded ‘yes’ and most responded ‘no’ to this question. As one professional commented, “Simple algorithms that anyone could reinvent and are likely to be repeated by accident due to convergent style standards are not worthy of attribution” (P27). Very few professionals chose ‘it depends’ and the main comment raised was that that they would only use an attribution for documentation purposes.

Heapsort algorithm

You have decided to use a heapsort and don't remember the exact implementation details, so you look up the algorithm on the web and then code that algorithm in your program. Do you add a comment giving credit to the website where you found it, or to some other source that the website gives credit to?

More than half of the academics responded ‘yes’ to this question and fewer than a quarter responded ‘no’. Academics who responded ‘no’ mostly commented that heap sort is a common algorithm and therefore attribution is not necessary. As one remarked, “I believe that algorithms considered to be ‘simple’ or ‘standard’ in the field need no attribution” (A52).

There were many ‘it depends’ responses from the academics. Some wrote that attribution was necessary only if there was something unique or significant about the algorithm. As one explained:

For something as generic as a heap sort, this would not be required. For nonstandard, nontrivial code, this would be different. If it's something novel enough to be on Stack Exchange or Stack Overflow, then we expect students to add a comment with the URL. (A54)

For others, attribution depended on how the information was used. As one academic explained:

If you read a general explanation of a commonly-known algorithm like heap sort and then write your own code, I

don't think a citation is necessary. If you copy code or translate detailed pseudo-code into code, you should include a citation. (A118)

A few academics related their responses to learning objectives. For example:

It depends on whether the purpose of the assignment is to code the heapsort or whether the heapsort is a small part of a larger project and this work was just relying on a standard algorithm. In the latter case, I would imagine the documentation should reference heapsort but not necessarily the particular source consulted. This would also be different if it were copied verbatim vs. coded from an algorithmic description. (A114)

The professionals were more divided in their responses than for the previous question. Nearly a third indicated that they would add a comment to attribute the source; however, nearly half answered that they would not. A number of the ‘no’ respondents wrote that they would attribute only if required by the source of the algorithm or by law. One respondent wrote “Most of my code is copied from online examples” (P16). Comments from the ‘it depends’ respondents showed similar views to the academics. As one explained “I might add a link to a more specialized or obscure textbook algorithm. But heapsort is common knowledge among programmers.” (P28)

Importing a library

You need to import a particular library into your program. Do you add a comment giving credit for the library, with details of who wrote it, when it was written, and where you found it?

The academics were almost equally divided in their responses to this question. Only a few academics who responded ‘yes’ gave an explanation. One academic stated “absolutely yes you are literally taking someone else's work, their very words, and incorporating it into your own” (A138). Another stressed the importance of comments for documentation: “What happens when you look at the code 10-20 years later and the library has changed? You'd better have a record of the source/version number/etc.” (A51). Most of the academics who responded ‘no’ commented that the explicit import statement is sufficient citation and the library should be documented by the original author.

The academics who responded with ‘it depends’ mostly wrote that the need for attribution depends on the licensing of the library. As one academic explained:

For a widely used, standard library, the name of the library should suffice for credit. If the library is not widely known, e.g., the personal work of an acquaintance, then the source should be credited. (A73)

A number of academics related their responses to the purpose of the assessment.

If the library is provided as part of the assignment or is part of the language's standard library, no. If this is a library I have obtained myself, linked to my code, and included in my submission, then I give credit in a README or similar. (A84)

A couple of academics maintained that attribution was not needed for standard library imports, typically used by first year programmers; however, “In more senior courses students are expected to provide complete build instructions including any libraries they have used.” (A100)

The professionals were also divided in their responses to this question. More than a third wrote that they would add a comment to attribute the code, and fewer than a quarter wrote they would not. Comments from the 'it depends' respondents showed similar views to the academics. Most commented that adding attribution would depend on the licensing laws of the library. A couple also commented that if the library was not a 'standard' library then it was good practice to include details for future maintainability.

4.4.2 Using code written by others

You are required to design and write a program. Would you use code written by others and would you acknowledge this use?

Specific scenarios exploring code reuse practices are discussed below.

Modifying another's program

Would you find a related program written by somebody else and copy and modify some of its code?

The academics were divided in their responses to this question. Almost a third answered 'yes'. One commented, "If the program can be found through public sources and is clearly free to share, and if the use of found code is not specifically prohibited for the assignment" (A81). However, more than a third answered 'no' and the few comments indicated that they considered this to be plagiarism.

We would consider this unacceptable. Yes, many professionals do it, but as students, they need to be writing their own code. Again, this is stricter in the lower-level courses than the upper-level courses. In some upper level courses we may make allowances for copying code, particularly for things that aren't the 'heart' of what we're trying to teach with that assignment. (A55)

Almost a third of the academics responded with 'it depends', adding that the practice would depend on such factors as whether the instructor gave approval, the rules of the assignment, whether the code was central to the assignment, and the amount of code copied. For one academic it depended on the level of the course: "In early courses this would be expressly forbidden, but later on this wouldn't be an issue" (A142).

Nearly half of the professionals responded 'yes' to the question. As one stated, "If the code is made public then I believe I can use all or parts of it" (A35). Few responded 'no' and nearly half responded with 'it depends'; the comments indicated that copying and modifying another's program would depend on the code license, whether the code was open source, and whether the code was from another company.

If you used this code would you add a comment explaining what you have done and acknowledging the source?

More than two-thirds of the academics responded 'yes' to this question. A number stated that it would be dishonest not to acknowledge the source. One academic suggested, "If not substantially different, the student's marks could be reduced as no original work was completed" (A104). The few academics who answered 'no' proposed that this scenario should not happen and would be considered cheating. "Sadly, most of these cases are probably outright cheating and therefore would not be acknowledged. They are caught by code comparison utilities." (A101)

To a number of academics, whether the code should be attributed depended on the assessment question or how close the code was to the original code.

This should not be done in first year since students are to be writing programs from first principles. If done in second year then the source of the basic code should be cited with an explanation of the changes. (A117)

Fewer than half the professionals responded 'yes' to the question and more than a quarter responded 'no'. About a quarter responded 'it depends', and indicated in their comments that attribution would depend on the amount of code used from other sources and on whether knowing the source of the code would be useful for future coding or debugging.

Using another's program

Would you find a program written by somebody else that performs the required task, and submit it more or less as is?

Most academics responded 'no' to this question. Many commented that this practice would be dishonest and would be specifically prohibited in course policies. A number mentioned that it would defeat the learning goals of the assignment. For example, one academic wrote "This would not be consistent with policy that states that submissions must be an honest representation of my own understanding" (A81). Another remarked:

I wouldn't do this, but I would be annoyed if I found out that I was being asked to write a program that was already out there, but I wasn't permitted to look at it - partly because I imagine some of my fellow students would find and copy the existing program, putting me at a disadvantage. (A38)

A few academics responded 'it depends' to this question. Whether they would use a program written by someone else depended on whether this was permitted by the assignment and course guidelines. For one academic it depended on how central the program was to the assignment:

Unless it were a subsidiary piece. Our students often build projects that use specific tools but involve novel combinations, etc. If this were, e.g., library use, of course. If this were representing the project built by another as one's own, of course not. (A134)

For another academic it depended on the level of the students. "In some upper-level courses I allow students to search for libraries that accomplish parts of an assignment (never an entire assignment, however)" (A72).

The professionals were almost equally divided in their responses to this question. Although about a third responded 'yes', their responses were qualified. For example, one professional would use a program written by someone else "Only if it saves significant amount of time and its license permits and it's the right thing to do" (P17). One professional argued:

If I have an employer that wants me to accomplish a task, s/he doesn't care how I accomplish it as long as I don't violate his/her business model. Obviously if I'm hired as a programmer and just use other people's code all the time there may be an issue, but only inasmuch as I may have been hired for the wrong position. If I'm producing results s/he will probably be happy. (P29)

Just under a third of the professionals responded ‘it depends’. Many comments mentioned licensing and gaining permission from the author. One professional wrote that they would seek the acceptance of their team and another remarked that it would be a project management decision.

If the decision was made to develop an in-house solution, then this is inappropriate. If the decision was made to use an existing program, it must be properly licensed and credited. (P28)

A couple of professionals were sceptical that this situation would arise, suggesting it would be highly unlikely that “an off-the-shelf program would fit the problem definition precisely” (P19).

If you used this code, would you add a comment explaining what you have done and acknowledging the source?

Nearly two thirds of the academics responded ‘yes’ to this question. Most argued that although they shouldn’t use code in this way, if they did then it should be acknowledged and the work should earn no marks. Almost a quarter of the academics responded ‘no’ and their comments indicated that they considered that this scenario should not happen or the students would try to hide it. A few gave the ‘it depends’ response. One related to the learning objectives:

I would not do this as I would consider it inappropriate in terms of assessment as the idea is that I would have actually done the work and thinking myself; getting assistance does not mean that you have not learnt and achieved the learning outcomes yourself BUT taking something verbatim from another source is not acceptable - UNLESS the assignment allows it and that source forms only a minor part of the assessment (e.g. using a library to read in a mesh model in a graphics assignment). (P121)

The distribution of responses from professionals was very similar to that of the academics, but the underlying reasons were different. A number claimed that they would provide a comment as documentation to assist with further development. One mentioned that they would never use a program in its entirety.

4.4.3 Reusing Your Own Code

You are required to design and write a program. Would you reuse all, or substantial parts, of a program that you have written previously?

More than two thirds of the academics responded ‘yes’ to this question and a number indicated strong support for reusing their own code. As one remarked:

I know some people think of this as ‘self-plagiarism’, but I wouldn’t agree. In fact, if my students write their code in such a way that they can reuse it, that seems like a good thing. (A42)

Another gave an example illustrating where code reuse is acceptable:

In many courses, as an instructor, I OK reuse of students’ own code or of posted solution code. Normally, this is cases where assignment work is based on incremental development across assignments and where the focus is on the introduction of some new technology. (e.g. convert your single threaded server to a multi-threaded server). (A101)

Very few academics responded ‘no’; one who did observed that this conflicted with the institution’s practice.

A quarter of the academics responded ‘it depends.’ Some wrote that such reuse was acceptable if prior approval was obtained. A number mentioned that such reuse would depend on the learning objectives.

Yes! This makes sense - after all it is what we would do professionally - why reinvent the wheel? But of course this is dependent on any IP agreements or laws related to code written for another company but as a student I would do so unless otherwise instructed. (A121)

The distribution of professionals’ responses was similar to that of the academics, but the explanations were different. A typical comment for the ‘yes’ response was “Why re-invent the wheel?” (P37). For others, reusing code would depend on such things as the project requirements, copyright status or proprietary status of the code. As one professional explained:

If the program was written at a previous employer, then no. If it was personal or at the current employer, then yes. Generally if the code was written for a previous employer, we are not allowed to use it unless it is just general algorithms that are not specific to the previous employer. (P36)

If you reused your own program code, would you add a comment explaining what you have done?

The academics were divided in their responses to this question. Nearly half responded ‘yes’ and their comments showed their concerns about self-plagiarism as being intellectually dishonest. However, more than a third responded ‘no’, explaining that they took quite a different view. For example,

I wrote the code; I thought it out and implemented it; I even remembered it, found it, and adapted it for this assessment item. Why would I need to reference that? (A38)

Another adds, “If it’s mine, it’s always been mine, and that’s ok. No need to attribute my past self.” (A138)

To other academics, however, whether they would reference the code depended on the amount of code reused, where the code had been written, and whether reuse was the intent of the assessment:

Of course, we would not tolerate substantial work done for one class to be handed in for another class without discussion to ensure that the work had been extended appropriately. (A54)

Nearly a third of the professionals responded ‘yes’ to this question, while more than half responded ‘no’. As one argued, “If I was the author previously, I’m still the author now - who do I attribute now, as if my current self is somehow different than my past self?” (P37). For others it depended on factors such as the complexity of the code and whether knowledge of the original source was necessary to help others understand the code.

4.5 Perspectives on seeking help

4.5.1 Seeking help with program design

You are required to design and write a program. Would you ask somebody else to help you work out the algorithm?

This section relates to issues of unauthorised aid and in an academic context could be equated with the offence of collusion,

which constitutes a significant proportion of academic discipline cases. More than two fifths of the academics responded 'yes' to this question. A few commented that in their course asking for such help is acceptable practice. One academic provided the caveat that 'help me work out' is very different from 'tell me what is' (A123). Another proposed that "Students can talk about solutions, but should take a break after discussion and write up their solutions privately" (A142). Few responded 'no' to this question. A couple wrote comments indicating that they considered that the students should produce their own solutions.

More than two fifths of the academics responded 'it depends'. A number stated that help was acceptable from the teaching staff but not from other students. A few remarked that whether they would ask for help would depend on what the instructor allows:

Generally, our rule of thumb is that you can talk through problems but your hands should be the ones on the keyboard (unless you are doing pair programming or a group project, which is often the case). (A134)

A number of academics commented that whether asking for help was acceptable depended on alignment with the course goals. For example, "If one of the goals is truly to *design* the program, then no, I wouldn't ask for help" (A114). Another factor was the level of help provided; as one remarked, "Discussing general algorithm ideas is permitted. Obtaining detailed algorithm information is not." (A106). Another commented that asking for help was acceptable only after students had done exhaustive work and they were stuck.

Almost two thirds of the professionals responded 'yes' to this question. One commented "working in groups is the best way to hash out an algorithm. One person may come up with a creative solution to the problem." (P23). Few responded 'no', and more than a quarter responded 'it depends'. Some of the professionals wrote that whether they would seek help would depend on the complexity of the problem, and one mentioned that it would depend on whether it affected the rest of the system. A few noted that they would ask for help only if they had someone nearby with the specialised knowledge. A couple mentioned that it would depend on the proprietary nature of the code.

If so, would you add a comment explaining what you have done and acknowledging the assistance?

Nearly half of the academics responded 'yes' to this question. One academic saw a difference between debugging and designing activities: "Debugging and designing feel like different activities. I would acknowledge assistance with the design of a program." (A114). More than a third responded 'no'. Several explained what they saw as acceptable; for example, "You are allowed to seek assistance in trying to get an understanding towards aspects of the algorithm you must develop [as] part of the learning process" (A96).

Fewer than a fifth of the academics responded with 'it depends'. A few stated that the requirement to acknowledge assistance would depend on course policies. A number stated that acknowledging assistance would depend on the level of assistance given, the usefulness of the assistance, and whether the algorithm was the main part of the problem to be solved. As one academic observed:

... you've got the loop termination wrong and you are leaving out the last value - then no. If it was - you should use this algorithm and here's a sketch of it, then yes. (A61)

One commented on the difficulty of getting students to acknowledge assistance in code comments:

... I often see students put information about who they talked to in reflective questions, but not usually in code comments (just getting them to write comments is often like pulling teeth...). (A51)

Fewer than a fifth of the professionals responded with 'yes'. However, one pointed out that "the comment is in the form of a kudos card, but not in the form of the comment in the code" (P30). Nearly two thirds responded with 'no'. One professional explained, "As we are all working on the same project adding in these kinds of comments would generally be considered clutter" (P12).

More than a fifth of the professionals responded with 'it depends'. A few stated that acknowledging assistance would depend on the complexity of the algorithm and the level and significance of assistance given. A few mentioned that such acknowledgement belongs not in code but in the design documentation. Acknowledgement could also be given via other means such as emailing their manager or in team meetings.

4.5.2 Seeking help with debugging

Your program has a bug that is preventing you from proceeding any further, and you have spent considerable time trying to work out what is causing it. Would you seek assistance in any of the following ways?

Specific scenarios exploring practices in seeking aid are discussed below.

Seeking help from a fellow student or colleague

Would you ask a fellow student / colleague who is working nearby to help spot the problem?

Almost two thirds of the academics responded 'yes' to this question. Many commented that they allowed or expected students to seek help from each other but some added provisos that students should first try to resolve the problem themselves and the help should not extend to actually fixing the coding problem. One academic related this to industry practice:

I encourage this sort of basic interaction as it is typical of actual practice in industry. This is advertised as something that is OK, only after you have made significant attempts to debug the code yourself. (A101)

Few academics responded 'no'; some who did added that this sort of help was not permitted in their courses.

More than a quarter of the academics responded with 'it depends'. Many commented that it depended on the goals of the assessment or the rules of the course. As one academic explained:

As the instructor, my written policy disallows this, but I recognize there is a lot of value in helping each other debug. As the instructor, I am generally okay with looking at others' code as long as it isn't lifted wholesale. Copying solution outlines / techniques is more problematic, and I don't have a clear idea of where to draw a line for that. (A84)

A few remarked that asking for this sort of assistance was forbidden in introductory courses but was allowed in the intermediate and advanced courses.

Most professionals responded ‘yes’ to this question and one commented that it was actually an expectation that people would ask their colleagues for help. Very few responded ‘no’; one commented that they always tried to help people but never asked for help themselves. There were a number of ‘it depends’ and a couple of comments indicated that seeking this help would depend on whether their colleagues were busy or whether the code was proprietary.

If your fellow student / colleague does help to resolve the problem, would you add a comment explaining and acknowledging the assistance?

The academics were fairly divided in their responses. Nearly a third responded ‘yes’ and the few comments provided indicated that acknowledging assistance was an expectation in their courses. More academics nominated ‘no’. A few commented that solving a small bug does not warrant acknowledgement. One academic wrote that “In the courses where students can assist one another, they are not required to note when/where this happens” (A54).

Nearly a third responded with ‘it depends’. Most comments concerned the extent of the fix. As one academic argued:

If the fellow student gave substantial help, such as pointing out a critical flaw in the algorithm – that would merit a reference. Most syntax errors, and minor flaws such as off-by-one errors, would not. (A38)

Another stated that the need for acknowledgement would depend on whether the help was in the form of code or an idea. “Code is owned as copyright, ideas are not” (A71). One academic mentioned that acknowledging such assistance would depend on the course policy of the instructor.

Very few professionals responded ‘yes’, and nearly three-quarters responded ‘no’. A number explained that helping colleagues is normal practice and no one would expect credit in this case. Only a few professionals nominated ‘it depends’, most adding that acknowledging assistance would depend on the extent of the contribution. A couple suggested other ways in which the help could be acknowledged, for example in an email or a project meeting.

Sending code to a fellow student or colleague

Would you send the code to a fellow student / colleague who is generally very helpful at debugging code, explain the problem, and ask for help in finding the cause of the error?

The academics were divided in their responses to this question. Almost a quarter responded ‘yes’. One commented that sending code to a fellow student is allowed, as long as it is acknowledged. However, more than a third responded ‘no’, some adding that this practice is specifically prohibited. One academic was concerned with the educative value:

This would suggest a more involved process separate from the learner who would therefore be unlikely to benefit from the process of learning to debug by working *with* another student. (A101)

About a quarter of the academics responded ‘it depends’; from their comments, the decision to send the code would depend on how well they knew the person, the goals of the assessment, and whether policies allowed this. One academic commented:

If both parties are operating honestly, then this could be an acceptable allegory to the co-located ‘hey take a look at

this’ situation ... [but] I would caution students to be careful, and not let the file get out of their hands. (A138)

More than two thirds of the professionals responded ‘yes’ to this question, with a couple commenting that it was standard practice in their company. Only a small number responded ‘no’. One commented that they were co-located and face to face was faster and less hassle. A small number answered with ‘it depends’, proposing that it would depend on the complexity of the problem and whether the code was proprietary.

If your fellow student / colleague does help to resolve the problem, would you add a comment explaining and acknowledging the assistance?

More than a third of the academics responded ‘yes’ to this question. One academic stressed “Whether the fellow student was successful or not, the collaboration would need to be acknowledged” (A110). Fewer than a third responded ‘no’, with one commenting that this would not occur as the practice of sending code is forbidden. The same number gave ‘it depends’ responses, their explanations all referring to the level of help given. For example:

... citations are for sources of content. If the other student is only helping me to produce my own content, then there’s likely no need to cite anything. (A108)

The distribution of professionals’ responses to this question was similar to that of their responses to the previous question about acknowledging help given in person. The explanations were similar in that providing a comment to acknowledge assistance would depend on the level of assistance given, or mentioning that the assistance could be acknowledged in other ways.

Posting a request for help on a website

Would you post a request on a website such as Stack Overflow, include the problematic code, and ask for help in finding the cause of the error?

Fewer than a quarter of the academics answered ‘yes’ to this question. One academic wrote that he actually encouraged students to use this but few did. Another wrote that this would be allowed only for later-year students.

For the level of sophistication of problems that actually get answered on Stack Overflow, we would allow it, especially in the context of an individual project. We would not allow CS0/CS1/CS2 students to seek help like this, though. (A54)

More than half of the academics answered ‘no’. A number of academics were concerned about the lack of control over what the students would find and how much help would be provided. As one wrote, “Many people on such websites re-write the code to make it correct and provide it in the answer to the question” (A98). Several remarked that there were much better resources available for debugging help. A couple were concerned about others using the posted code: “my posted problems become public and searchable by others, facilitating all-too-easy plagiarism” (A52).

More than a quarter of the academics responded ‘it depends’, indicating that the decision to post the code would depend on policies and on the goals of the assessment. As one academic observed: “For a homework problem, no. But for some code that’s

part of a large project, perhaps.” (A62). A number mentioned that posting would depend on the level of help sought. For example, Debugging is fine - I have no problem with that, but sometimes the ‘help’ might end up being a rewrite of the code using different approaches and techniques. In other words, it might go far beyond debugging. (A89)

For one academic, whether they would post would depend on what they needed to post: “If I can generalize the problem then – yes, but [I] would not post [an] actual assignment” (A102).

Nearly half of the professionals answered ‘yes’ to this question. One remarked that they make liberal use of Stack Overflow. Fewer than a third responded ‘no’. One stated that they very rarely ask for help online and another commented that they were unsure of their company policy on this practice. More than a quarter responded with ‘it depends’, giving a variety of reasons. Some said that they would post only if they were really stuck, having tried to solve the problem internally and having thoroughly researched the problem online. A few maintained that they would post only if they could anonymise the code so that their company was not identifiable and no company secrets or proprietary information would be revealed.

If somebody responds and helps to resolve the problem, would you add a comment explaining and acknowledging the assistance?

More than half of the academics responded ‘yes’ to this question. The few comments provided stated firmly that attribution is expected. Fewer than a quarter of the academics responded ‘no’; their comments indicated that they considered the actual practice of posting such a request was unacceptable and outside traditional academic practice.

Fewer than a quarter of the academics provided ‘it depends’ responses. A few wrote that whether they would add a comment would depend on the level of assistance given. One academic provided an interesting perspective:

Only if it was permitted would I cite it. If I was violating the honesty policy, I probably wouldn’t. However, my professor tells me if I do cite it’s not an honesty case (which carries stiffer penalties), it’s only a matter of whether I get credit on the assessment. (A50)

More than a quarter of the professionals responded ‘yes’ to this question. One wrote that they would attribute the assistance by including a link to the question and answer. However, half the professionals responded ‘no’, with several commenting that they would not post to a website so the question of attribution was not relevant. One asked “What’s the point in crediting an anonymous internet person?” (P16). Fewer than a quarter responded with ‘it depends’. One professional wrote that attribution would be required only if the helper spent significant time solving the problem. A few remarked that they would attribute only if they thought that their other colleagues working on the source code might have queries about the code.

Outsourcing your work to another programmer

You are required to design and write a program. Would you pay somebody else to write part or all of the program for you?

Most academics responded ‘no’ to this question. Their comments clearly showed that they considered this practice to be academic dishonesty, with several using the term *contract cheating*. As one remarked: “This would seem to be a textbook example of

academic misconduct” (A97). However, one academic was not sure whether this practice was against their institutional policy. Only a couple of academics responded ‘yes’, with one remarking, “It’s allowed in my class. The purpose is learning, not assessment.” (A56)

A few academics answered with ‘it depends.’ A couple remarked that although sub-contracting was not part of their assessment, it is a well-established practice in industry. One stated:

Except in our senior capstone course, where the use of paid peers to complete specific parts of the project is sometimes a part of emulating industry practice, or other cases where the outsourced piece is not a part of the expected coursework, this would not be done. (A134)

Almost two thirds of the professionals responded ‘no’ to this question. Several commented that programming was their job: “I’m the one getting paid to program...” (P16). More than a quarter answered ‘it depends’. For some it depended on their role. “As part of my day job as a software engineer, no. As part of my job developing software for my own private company, yes.” (P15). A couple commented that this could be an economic decision within their company: “If we don’t have the resources to tackle a project that we can comfortably outsource, then this is a possibility” (P24).

If you commissioned another person to write your code would you add a comment explaining what you have done and acknowledging the source?

Fewer than half the academics responded ‘yes’ to this question. One commented that outsourced work should be acknowledged but this would only happen in specific circumstances, for example, if allowed in a senior capstone project. Another wrote:

If acknowledged, the work would almost certainly get a low grade; however, by being open about the source, the student would not risk expulsion from the college. If not acknowledged, suspension or expulsion would be likely outcomes. (A91)

There were fewer ‘no’ responses and these all indicated that this practice would not happen and the question was therefore not relevant. There were a number of ‘it depends’ responses and the comments indicated they considered students who paid someone else to write their code were unlikely to admit this. As one academic responded, “Using code written by someone else is disallowed by the policies in the course, so if the students did this they would likely try to hide it” (A110).

The distribution of professional responses was similar to that of the academics. More than a third of the professionals answered ‘yes’ to this question. A few mentioned that such acknowledgement would be for the purpose of documentation to assist future development. Almost a quarter of the professionals responded ‘it depends’. They indicated that acknowledgement would depend on the agreement with the outside source. For example,

As long as I retain the intellectual property of the written code, there is really no value to acknowledging the source. If it makes sense to credit the source then I would do so. This is very situational. (P33)

4.6 How to attribute coding assistance

Substantial majorities of both academics and professionals agreed that there was no standard format for acknowledging the source of

code or assistance. Informally, there was consensus that if attribution is documented in the code, the source location (ideally a URL) is required, but there was less agreement regarding other fields, such as the name of the author or the date of access. In general, academics required more information, such as a comment describing the code being used, the name of the author, and the date of access. Several academics noted that they did not have a standard form but that it would be a good idea.

The issue of licensing is a factor in the form of attribution: “If the code is licensed, I would follow the terms of the license and leave a note as to its origin” (P14). The licensing issue is related to where acknowledgements are placed. Some licenses require attribution in specific ways, and the license often needs to be placed in a specific file in the source distribution.

Some domains and employers also have norms. For example, “I make games. Code contribution is not documented in file, but credit is given in the game credits.” (P15). Some companies also provide mechanisms for recognising contributions: “Coworkers are given opportunities to acknowledge the contribution of others in reviews every few months. Source code comments are for documenting the code.” (P28)

4.7 Summary

Findings from the survey confirm the findings from the literature that there are many differences of interpretation and expectation concerning acceptable practice in computing assignments among both academic computing staff and students. There are in practice many different de facto policies implemented by academics in many different contexts, particularly with regard to what should be attributed, what reuse is acceptable, what aid is allowed, from whom help might be sought, and how it should be attributed. Further, there is a tension in all of these dimensions between what might be viewed as academic necessity (for example, pedagogic needs and the requirement for individual assessment) and the reality of professional practice. While academics may be doing their best to balance the two, the range of approaches documented above is problematic in terms of providing clear guidance to students on how to comply with academic integrity policies.

Our work appears to break new ground by surveying computing professionals as well as academics. This has permitted us to observe that while both academics and professionals justify their practices in a variety of ways, there appears to be a fairly clear partitioning of justifications between academics and professionals. Many students are familiar with the professional perspective: perhaps because they already work in the computing profession, or perhaps because they have become familiar with the type of work that will be expected of them [54]; and therefore they sometimes struggle to see the rationale for specific academic integrity practices.

5. PROPOSALS FOR CLARIFYING ACADEMIC INTEGRITY PRACTICES IN COMPUTING EDUCATION

In light of the overwhelming uncertainty and confusion suggested in the literature and confirmed by the survey, there is a clear need for ways in which different, contextualised academic integrity practices can be articulated and conveyed to students in a simple and effective manner. In this section we propose a scheme designed for that purpose. At this stage the scheme is offered not as a comprehensive framework, but as a first step towards

recognising and systematically addressing the complexity that exists in this area, and as a catalyst for discussion within the academic computing community.

In an ideal world, this paper would include a tutorial showing academics how to devise academic integrity guidelines that suitably combine their learning objectives, the nature of their assessment items, and the year level of their course, while at the same time resolving any mismatch with whatever academic integrity policies their students are required to follow. Unfortunately, such a tutorial would be well beyond the scope of this work, and is unlikely to be achievable at all. The range of learning objectives, the range of assessment items, and the range of academic integrity policies are all so great that the task cannot be simplified to a simple guide.

Instead we outline a flexible process that individual educators can use to clarify acceptable practices in terms of academic integrity for each specific assessment task and to effectively communicate the guidelines to students. One intended consequence of these proposals is that academics will be led to consider the relationship between academic integrity policy and the learning objectives for each specific assignment and context. This will encourage clarification and justification of the specific rules that will be applied to each assessment task. Academics are encouraged to reflect on the alignment between the relevant academic integrity policy and the learning process they are creating for the students. If standard academic integrity practices cannot be aligned with the learning objectives and pedagogy that they have adopted, they must make the case for specific guidelines to be created to enable such an alignment – or to argue how and why the alignment is not possible.

Another intended benefit of these proposals is that expectations will be clearly communicated to students, thus avoiding confusion. Knowing how the rules will be applied in specific cases makes it easier for students to comply and ensures a level playing field. If students are required to appear before academic discipline committees, they can point to specific guidelines that they have received as to what was considered acceptable practice in the context of the specific assignments involved.

The proposals are also designed to ensure that the academic provides an explanation of the learning objectives and a justification of the educational benefits of the specified academic integrity rules. A number of academic practices make sense only in the context of a learning environment, and may conflict with standard industry practice. If students are informed of the purpose of these practices in terms of facilitating learning, they are more likely to embrace and comply with them.

The proposals can be codified as follows:

- **Proposal 1:** For each assessment task there should be a consideration of what constitutes acceptable academic integrity practice for this assignment on the dimensions we have specified, including attribution of code, code reuse, seeking aid, attribution of help, etc.
- **Proposal 2:** The defined acceptable practice should be linked to specific learning objectives and a justification should be created of how these enhance the students’ learning.
- **Proposal 3:** The academic should explain to the students the academic integrity rules that apply to this situation, how they will be applied, and why they are justified in terms of learning

objectives. This may include constructing diagrams for relevant dimensions to inform students in a simple and readily interpreted visual form of the types of action and behaviour that are expected, allowed, and disallowed in the situation.

The authors of this report suggest that the above policies should be communicated to the students in a positive form, emphasising what is encouraged and allowed, rather than in a negative form, focusing on what is prohibited.

When implementing the above proposals, academics will determine a set of dimensions that illustrate the range of current practice within which they can locate the preferred, allowed and disallowed practices. For example, one assignment might have two dimensions:

- Assistance (i.e., sources of help, discussion, or clarification)
- Resources (i.e., artefacts or people who provide or work with implementation details such as code, UML diagrams, etc.)

Another assignment might have two quite different dimensions:

- Activity (i.e., types of task involved, such as debugging, implementation, design, etc.)
- Source (i.e., groups or sets of people who might provide assistance with these activities)

As academics work through the design of their learning objectives, learning activities, and assignment specifications, the intention is to guide them to clarify the rules of academic integrity that will be applied and the purpose of applying these rules.

An important aspect of the process is to create a visual representation of acceptable practice. We have selected circular diagrams that list applicable practices and that are colour coded to indicate the desirability of the practice: practices that are to be encouraged, practices that are acceptable, practices that are potentially problematic for which advice should be sought from the instructor if they are being considered, and finally practices that are not allowed in this context. Colours are assigned to each type of practice to indicate its categorisation. Three examples are documented below to show how the process can be applied to specific contexts.

5.1 Assessment design examples

Below we present specific examples of assessment tasks and their context to show how the process could be applied. We provide examples for three courses at three levels of a degree.

Example 1: First-year programming course

The learning goals of introductory courses typically include the development of skills for code writing and for the design of solutions to problems. The emphasis is on individual assessment. Collaboration may or may not be allowed depending on the context for the course.

Figure 1 shows two diagrams that could be placed on a syllabus (course outline) or that could accompany an assignment specification. The diagrams describe the people and resources allowed for the assignment. In the example, yellow is used to indicate an encouraged practice, blue indicates a practice that is acceptable but requires attribution, brown indicates a practice that is not allowed, and green indicates a practice that requires further consultation -- either with the instructor or by referring to another chart that provides more detail about relevant scenarios. As explained in section 5.2, the colours were chosen not with any cultural meaning in mind, but simply to be distinct.

Assistance: Who can you get help from? Use this diagram to determine from whom you may seek help with your programs.

Resources: Where can you get code from? Use this diagram to determine where you may find code to use in your programs.

This assignment, which is to be completed individually, is your chance to gain an understanding of fundamental concepts of program structure and coding syntax on which later learning will be based. It is important that you master these concepts yourself.

Since you are mastering fundamental skills, you are permitted to work from course examples, but you must acknowledge assistance from other textbooks or classmates. In particular, you must not use online material or help from others, as this would prevent you from mastering these concepts.

Figure 1: Example from a first-year programming course

The diagrams are intended to be applied to all individual assignments in the course, for example, homework assignments. A different policy might be developed for assessments that allow tighter collaboration between students, such as lab exercises and activities. As an example of an assignment, students might be asked to write a program that reads a text version of a book stored in a file line by line and create an index for the text, that is, a list of words of interest in the text and associated line numbers in which those words appear.

The words of interest would be provided as input to the program or function, and the structure containing the words and lines on which they appear would be the output or return value of the program or function.

In keeping with the emphasis on individual assessment, the diagrams describe a situation where only limited collaboration is acceptable. The diagrams also encode an assumption about the ability of first year students to differentiate between different programming activities. For example, the students may not clearly understand the distinction between writing code and debugging it. As a result, the diagrams do not differentiate between different types of aid. Instead, they focus on who (or what) is providing the information, and some sources, such as people, appear in both diagrams. In creating the charts that arise out of the relevant dimensions and actions, academics will need to make a distinction regarding the hierarchy of the charts. In this example each chart is independent of the other, and which chart is referenced depends on which type of activity is being performed. If the student is discussing the assignment, the Assistance chart is referenced. If the student is doing something that directly involves code, the Resources chart is referenced.

Example 2: Upper-year operating systems course

The learning goals of upper-year courses with programming components usually de-emphasise basic code writing and debugging skills in favour of outcomes related to the algorithms or mechanisms being studied. Upper-year courses may also emphasise teamwork and assess collaborative projects. As a result, the instructions given to upper-year students may assume that students can differentiate between different activities such as designing, implementing, and debugging, and may give students more freedom to find and use a wide variety of resources. The example in Figure 2 is intended for a third-year systems course.

The figures and text in the example are placed on the syllabus and are applied to all assignments in the course. Each assignment requires that students understand relevant, pre-existing code and implement new functionality based on theoretical material from within the course. For example, the second assignment requires that students understand existing page table structures and use them to implement paging, including eviction and replacement.

In contrast to the first-year example, which differentiates between people and resources, this example provides direction based on the type of assistance being sought.

To use the charts, the student first consults the Source chart. Depending on what resource is being used, students either have permission to consult the resource (encouraged) or need to investigate what kind of aid is allowed (it depends). In the latter case, they consult the Topic chart to determine the proper course of action.

Figure 2: Example from an upper-year systems course

Example 3: Professional experience capstone course

As we know from the survey findings presented in Section 4, professional settings are more focused than academic settings on

results and team orientation. They place less emphasis on attribution and more on ownership and licensing.

The example in Figure 3 derives from the responses of programming professionals in our survey, and is proposed for a professional experience capstone unit where a project team is working on a real-life project with an external client. Unlike the previous two examples, the diagrams show how standard academic integrity rules may be adapted to provide an authentic learning experience. Like the example in Figure 2, which is intended for upper-year students, this third example assumes that students can differentiate between different activities. However, the order of the diagrams is reversed in the two examples. In this example, the student would first consult the Activity chart, and then, if necessary, consult the Source chart to determine an appropriate course of action.

5.2 Summary

Although these three examples all use pictorial wheels to represent resources and types of collaboration allowed in specific academic contexts, there is a fundamental difference in approach between them. The first-year programming example assumes that students lack context for making decisions about the kinds of behaviour they may be considering. For example, a novice student might find it difficult to distinguish between writing, editing, and debugging code. The wheels are therefore devised to be used independently. A student should consult the Assistance wheel when working with someone and the Resources wheel when searching for or referencing code examples. By contrast, in the other two examples the assumption is that students have sufficient knowledge to understand the types of behaviour in which they are engaging. For this reason there is a dependency between the two wheels in each of these examples. Students start by consulting the first wheel, then move to the second wheel if told ‘it depends’. These wheels encompass a level of analysis and understanding that may elude first-year programming students.

Readers are reminded that these charts are in the early stages of development. It is clear that the representations are very broad-brush, and cannot capture the nuanced aspects of many academic integrity guidelines. For example, contextual clarification is clearly required when academics mention “acceptable bounds of collaboration” (A18) or that students should apply “the Gilligan’s Island Rule” (A67) when working together.

While the circular form of the charts is preferred by the authors of this paper, it is by no means the only usable form. Indeed, the information could equally well be presented as a simple textual table, with no graphical element at all. The content is clearly more important than the form, and individual users should feel free to use whatever form they believe would be most effective for their students. We also caution readers against imposing a traffic-light interpretation on the colours; such an interpretation would make no sense, especially in diagrams with four colours. Our colour set has been chosen simply on the basis of distinctness when viewed either in colour or in greyscale, and people adopting the system should feel free to choose their own colours.

Furthermore, the charts give no indication of how code should be attributed when attribution is indicated. However, the authors hope that educators will help us to develop the approach by applying it to existing assessment tasks, identifying any issues that arise, and alerting us to those issues.

Figure 3: Example from a professional experience capstone course

6. DISCUSSION

In section 1.1 we proposed five research questions that our work was intended to address. In this section we shall discuss each of these questions in turn.

1: What areas of current academic integrity practice engender confusion or conflicting views in academics and/or students?

Both our literature review and our survey analysis highlight specific areas where both academics and students experience significant confusion and disagreement as to what practices are acceptable. These include areas such as reuse of one's own code, modifying existing code, perspectives on seeking help, and attribution of both assistance and code use.

2: With regard to academic integrity, specifically in the use of code and assistance with coding, what are the differing perspectives and ranges of acceptable practices as perceived by computing educators and by computing professionals?

In each of the areas mentioned in the answer to question 1, particularly in the areas of code reuse, seeking assistance, and attribution practices, academics and computing professionals have conflicting approaches and justifications for current practice. Computing professionals indicate that productivity drives code reuse, that contractual arrangements and work teams influence help-seeking behaviours, and that copyright and licencing arrangements influence attribution practices.

In contrast, academics tend to focus on motivators such as the learning objectives of the assessment task and the progression of the learner (from novice in first year to experienced in upper years) to modify standard academic integrity rules. For example, help-seeking practices deemed appropriate for first year students are often inapplicable to more advanced students completing capstone projects.

3: What aspects of educational and/or professional practice in computing conflict with standard academic integrity policies?

While many academics indicate that they try to emulate professional practice when designing learning experiences, this often conflicts with standard approaches to academic integrity policies. For example, programming practices promoting the design of code for reuse and the use of software libraries tend to conflict with the goal of assessing the student's original contribution. There is no consistent approach to attribution of code segments sourced from others, and students are unclear about what needs to be referenced. Integration of industry practice into the classroom is also thought to be more appropriate for more advanced students, even where this is in conflict with academic integrity practices.

4: Can we outline a process by which educators can design assessment tasks while taking into account the interaction of academic integrity, learning objectives, teaching practices, and professional practice in the computing industry?

We have outlined a process that helps academics to align academic integrity rules with learning objectives in specific contexts while giving due deference to current professional practice. An important part of this process is to justify specific academic integrity rules to students in terms of the learning objectives of the assessment task. Our proposal falls short of trying to explain how academics should design their assessment tasks and the corresponding academic integrity guidelines, but we are confident that our proposed procedure and examples will provide considerable help with the design process.

5: Can a process be designed that will facilitate the effective communication of academic integrity rules to students once an

academic has developed these rules for a specific assessment task?

The charts that we have outlined in section 5 are designed both to help academics clarify their academic integrity expectations and to facilitate the communication of these expectations to students. Once academic integrity rules and associated charts and informational descriptions have been written, it is important to communicate them effectively with students. At a minimum the charts and text need to be provided to students, but ensuring that they are used effectively is best done by working through an example with them. For example, one could develop the solution to a small activity or exercise in a class session, referring to the charts as the work proceeds. It might even be appropriate to provide students with an assignment that involves interacting with the charts and then describing how and when they consulted the charts.

The suggested visual representation of academic integrity rules is one method that can be customised for a variety of contexts. Further research is needed to evaluate this method and perhaps to develop other effective visual methods.

Our work seeks to facilitate ethical practice among students by suggesting ways of clarifying the requirements in different contexts and by increasing the perceived relevance of assessments by linking them to academic and professional activity. This will not prevent plagiarism and collusion; however, it can contribute towards providing greater clarity for both academics and students, encouraging the former to set out a statement of acceptable practice by way of a standard, graphical means of representation; and explaining to the latter that there are good reasons for following the recommendations.

7. CONCLUSION

For decades computing education researchers have sought ways to educate for, detect breaches of, and enforce standard academic integrity rules for our students. The work presented here indicates that the efforts have had limited success, as both academics and students still hold widely divergent views as to what constitutes acceptable practice. Academic integrity rules created for other contexts do not translate well into a computing context. In addition, common practice in the computing industry, as demonstrated by the views of computing professionals, does not align well with the individualistic assessment practices on which standard academic integrity rules are based. This working group report seeks to bring these issues into focus and to provide greater clarity for both academics and students.

ACKNOWLEDGEMENTS

Many thanks to those who took the time to respond to the survey and to those who talked to us about the issues at ITiCSE 2016. Thanks, too, to the reviewers whose observations have led us to clarify a number of aspects of this paper.

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.

REFERENCES

- [1] ACM (1992). ACM code of ethics and professional conduct. Retrieved from <https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct>
- [2] Alam, L.S. (2004). Is plagiarism more prevalent in some forms of assessment than others? Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference, 48-57.
- [3] Bailey, J. & Mitchell, R.B. (2007). Industry perceptions of the competencies needed by computer programmers: technical, business, and soft skills. *The Journal of Computer Information Systems*, 47(2), 28-33. Retrieved from <http://search.proquest.com/docview/232576272?accountid=12528>
- [4] Barrett, R. & Cox, A.L. (2005). 'At least they're learning something': The Hazy Line between Collaboration and Collusion. *Assessment & Evaluation in Higher Education*, 30(2), 107-122.
- [5] Barrett, R. & Malcolm, J. (2006). Embedding plagiarism education in the assessment process. *International Journal for Educational Integrity*, 2(1), 38-45.
- [6] Baugh, J., Kovacs, P., & Davis, G. (2012). Does the computer programming student understand what constitutes plagiarism? *Issues in Information Systems*, 13(2), 138-145.
- [7] Bowers, W.J. (1964). Student dishonesty and its control in college. Columbia University, New York CRP-1672.
- [8] Braught, G., MacCormick, J., & Wahls, T. (2010). The benefits of pairing by ability. 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10), 249-253. <http://dx.doi.org/10.1145/1734263.1734348>
- [9] Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M., & Klemmer, S.R. (2009). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. *SIGCHI Conference on Human Factors in Computing Systems*, 1589-1598.
- [10] Clarke, R. & Lancaster, T. (2013). Commercial aspects of contract cheating. 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13), 219-224.
- [11] Collberg, C. & Kobourov, S. (2005). Self-plagiarism in computer science. *Communications of the ACM*, 48(4), 88-94.
- [12] Cosma, G. & Joy, M. (2008). Towards a definition of source-code plagiarism. *IEEE Transactions on Education*, 51(2), 195-200.
- [13] Culwin, F. (2008). A longitudinal study of nonoriginal content in final-year computing undergraduate projects. *IEEE Transactions on Education*, 51(2), 189-194.
- [14] Culwin, F., MacLeod, A., and Lancaster, T. (2001). Source code plagiarism in UK HE computing schools: issues, attitudes and tools. Technical Report SBU-CISM-01-02, School of Computing, Information Systems & Mathematics, South Bank University London.
- [15] Davis, S.F. & Ludvigson, H.W. (1995). Faculty forum: additional data on academic dishonesty and a proposal for remediation. *Teaching of Psychology*, 22, 119-121.
- [16] Dexter, S., Buchanan, E., Dins, K., Fleischmann, K.R., & Miller, K. (2013). Characterizing the need for graduate ethics education. 44th ACM Technical Symposium on Computer Science Education, 153-158.
- [17] Dick, M., Sheard, J., Bareiss, C., Carter, J., Joyce, D., Harding, T., & Laxer, C. (2003). Addressing student cheating: definitions and solutions. *ACM SIGCSE Bulletin*, 35(2), 172-184.
- [18] Fraser, R. (2014). Collaboration, collusion and plagiarism in computer science coursework. *Informatics in Education*, 13(2), 179-195.
- [19] Fuller, U., Little, J.C., Keim, B., Riedesel, C., Fitch, D., & White, S. (2010). Perspectives on developing and assessing professional values in computing. *ACM SIGCSE Bulletin*, 41(4), 174-194.
- [20] Gibson, J.P. (2009). Software reuse and plagiarism: a code of practice. *ACM SIGCSE Bulletin*, 41(3), 55-59.
- [21] Hansen, B., Stith, D., & Tesdell, L.S. (2011). Plagiarism: what's the big deal? *Business Communication Quarterly*, 74(2), 188-191.
- [22] Harris, J.K. (1994). Plagiarism in computer science courses. *Conference on Ethics in the Computer Age*, 133-135.
- [23] Havelka, D. & Merhout, J.W. (2009). Toward a theory of information technology professional competence. *The Journal of Computer Information Systems*, 50(2), 106-116. Retrieved from <http://search.proquest.com/docview/232573038?accountid=12528>
- [24] Hawley, C.S. (1984). The thieves of academe: plagiarism in the university system. *Improving College and University Teaching*, 32(1), 35-39.
- [25] Herrington, J. & Herrington, A., (1998). Authentic assessment and multimedia: how university students respond to a model of authentic assessment. *Higher Education Research & Development*, 17(3), 305-322.
- [26] IEEE (2016). IEEE Code of Ethics. IEEE Policies, Section 7.8, page 7.3. Retrieved 2 July 2016 from https://www.ieee.org/documents/ieee_policies.pdf
- [27] Jones, K.O., Reid, J., & Bartlett, R. (2008). Cyber cheating in an information technology age. *Digitum*, 10, 19-29.
- [28] Joy, M., Cosma, G., Yau, J.Y.K., & Sinclair, J. (2011). Source code plagiarism—a student perspective. *IEEE Transactions on Education*, 54(1), 125-132.
- [29] Joy, M., Sinclair, J.E., Boyatt, R., Yau, J.K., & Cosma, G. (2013). Student perspectives on source-code plagiarism. *International Journal for Educational Integrity*, 9(1). Retrieved from <http://www.ojs.unisa.edu.au/index.php/IJEI/issue/view/140>
- [30] Lampert, L. D. (2004). Integrating discipline-based anti-plagiarism instruction into the information literacy curriculum. *Reference Services Review*, 32(4), 347-355.
- [31] Lancaster, T. & Clarke, R. (2007). Assessing contract cheating through auction sites – a computing perspective. 8th Annual Higher Education Academy Conference for Information and Computer Sciences, 143-150.
- [32] Lancaster, T. & Clarke, R. (2012). Dealing with contract cheating: a question of attribution. 1st Annual Higher

- Education Academy Conference in Science, Technology, Engineering and Mathematics.
- [33] Lave, J. (1988). *Cognition in Practice*. Cambridge, UK: Cambridge University Press.
- [34] Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. New York: Cambridge University Press.
- [35] Le, T., Carbone, A., Sheard, J., Schuhmacher, M., de Raadt, M., & Johnson, C. (2013). Educating computer programming students about plagiarism through use of a code similarity detection tool. *Learning and Teaching in Computing and Engineering Conference (LaTiCE)*, 98-105.
- [36] Little, J.C., Granger, M.J., Boyle, R., Gerhardt-Powals, J., Impagliazzo, J., Janik, C., Kubilus, N.J., Lippert, S.K.W., McCracken, M., Paliwoda, G., & Soja, P. (1999). Integrating professionalism and workplace issues into the computing and information technology curriculum: report of the ITiCSE'99 working group on professionalism. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '99)*, 106-120. <http://dx.doi.org/10.1145/349316.349558>
- [37] Louw, D. & Pieterse, V. (2015). Dealing with plagiarism in introductory programming. *International Conference on Computer Science Education Innovation & Technology (CSEIT)*, 4-13.
- [38] Mann, S. & Frew, Z. (2006). Similarity and originality in code: plagiarism and normal variation in student assignments. *8th Australasian Conference on Computing Education*, 143-150.
- [39] McCabe, D.L., Treviño, L.K., & Butterfield, K.D. (2001). Cheating in academic institutions: a decade of research. *Ethics & Behavior*, 11(3), 219-232.
- [40] McKay, T.M. (2014). Combating plagiarism using a community of practice approach. *South African Journal of Higher Education*, 28(4), 1315-1331.
- [41] McInnis, R. & Devlin, M. (2002). Assessing learning in Australian universities. *Centre for the Study of Higher Education, University of Melbourne*. [29 Dec 2002, verified 23 Oct 2004] <http://www.cshe.unimelb.edu.au/assessinglearning>.
- [42] Nuss, E.M. (1984). Academic integrity: comparing faculty and student attitudes. *Improving College and University Teaching*, 32(3), 140-144.
- [43] Parnas, D.L. (1996). Education for computing professionals. *Computer*, 23(1), 17-22.
- [44] Pecorari, D. & Shaw, P. (2012). Types of student intertextuality and faculty attitudes. *Journal of Second Language Writing*, 21(2), 149-164.
- [45] Petre, M. (2012). Academic integrity in a changing environment. *ACM Inroads*, 3(2), 15-16.
- [46] Radermacher, A., and Walia, G. (2013). Gaps between industry expectations and the abilities of graduates. *44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*, 525-530. <http://dx.doi.org/10.1145/2445196.2445351>
- [47] Riedesel, C.P., Clear, A.L., Cross, G.W., Hughes, J.M., Simon, & Walker, H.M. (2012). Academic integrity policies in a computing education context. *Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups*, 1-15.
- [48] Roberts, E. (2002). Strategies for promoting academic integrity in CS courses. *32nd Frontiers in Education*, 2, F3G-14.
- [49] Sant, J. (2015). Code repurposing as an assessment tool. *37th International Conference on Software Engineering*, 2, 295-298.
- [50] Shapiro, S. (1994). Boundaries and quandaries: establishing a professional context for IT. *Information Technology & People*, 7(1), 48-68.
- [51] Sheard, J. & Dick, M. (2011). Computing student practices of cheating and plagiarism: a decade of change. *2011 Conference on Innovation & Technology in Computer Science Education*, 233-237.
- [52] Sheard, J., Dick, M., Markham, S., Macdonald, I., & Walsh, M. (2002). Cheating and plagiarism: perceptions and practices of first year IT students. *ACM SIGCSE Bulletin*, 34(3), 183-187.
- [53] Sheard, J., Markham, S., & Dick, M. (2003). Investigating differences in cheating behaviours of IT undergraduate and graduate students: the maturity and motivation factors. *Journal of Higher Education Research and Development*, 22, 91-108.
- [54] Simon, Cook, B., Sheard, J., Carbone, A., & Johnson, C. (2013). Academic integrity: differences between computing assessments and essays. *13th Koli Calling International Computing Education Research Conference (Koli Calling 2013)*, 23-32.
- [55] Simon, Cook, B., Sheard, J., Carbone, A., & Johnson, C. (2014). Academic integrity perceptions regarding computing assessments and essays. *10th International Computing Education Research Conference (ICER 2014)*, 107-114.
- [56] Simon, Cook, B., Sheard, J., Carbone, A., & Johnson, C. (2014). Student perceptions of the acceptability of various code-writing practices. *2014 Conference on Innovation & Technology in Computer Science Education*, 105-110).
- [57] Simon & Sheard, J. (2015). Academic integrity and professional integrity in computing education. *2015 ACM Conference on Innovation and Technology in Computer Science Education*, 237-241.
- [58] Simon & Sheard, J. (2016). Academic integrity and computing assessments. *Australasian Computer Science Week Multiconference*, 3.
- [59] Wilkinson, J. (2009). Staff and student perceptions of plagiarism and cheating. *International Journal of Teaching and Learning in Higher Education*, 20(2), 98-105.
- [60] Wilson, J.D., Hoskin, N., & Nosek, J.T., (1993). The benefits of collaboration for student programmers. *24th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '93)*, 160-164. <http://dx.doi.org/10.1145/169070.169383>
- [61] Yorke, J., Lawson, K., & McMahon, G. (2009). Can we reliably determine intent in cases of plagiarism? *International*

Appendix A: Quantitative Summary of Survey Scenario Responses

This appendix should be read in conjunction with sections 4.4, 4.5, and 4.6, where the academic versions of the questions are included. The versions for computing professionals are as similar as possible to those for academics: for example, the words “a fellow student” would be replaced by “a colleague”.

	Academic (106)			Professional (38)		
	Yes	No	Depends	Yes	No	Depends
Use of Code						
<i>Section 4.4.1: Using standard algorithms or code</i>						
Code for a basic loop	24 (23%)	63 (59%)	19 (18%)	1 (3%)	33 (87%)	4 (11%)
Heap sort algorithm	57 (54%)	24 (23%)	25 (24%)	11 (29%)	17 (45%)	10 (26%)
Importing a library	39 (37%)	37 (35%)	30 (28%)	15 (39%)	7 (18%)	16 (42%)
<i>Section 4.4.2: Using code written by others</i>						
Modifying another’s program	32 (30%)	40 (38%)	34 (32%)	17 (45%)	4 (11%)	17 (45%)
Acknowledging use of modified source	66 (69%)	14 (15%)	15 (16%)	18 (49%)	10 (27%)	9 (24%)
Using another’s program without modification	7 (7%)	81 (79%)	15 (15%)	13 (34%)	13 (34%)	12 (32%)
Acknowledging use of unmodified source	47 (62%)	18 (24%)	11 (14%)	22 (61%)	9 (25%)	5 (14%)
<i>Section 4.4.3: Reusing your own code</i>						
Reusing your own program	75 (71%)	4 (4%)	27 (25%)	27 (71%)	0 (0%)	11 (29%)
Acknowledging program reuse	45 (43%)	36 (35%)	23 (22%)	13 (34%)	19 (50%)	6 (16%)
Seeking Help						
<i>Section 4.5.1: Seeking help with program design</i>						
Seeking help with program design	46 (43%)	15 (14%)	45 (42%)	22 (58%)	5 (13%)	11 (29%)
Acknowledging design help	45 (46%)	36 (37%)	17 (17%)	6 (18%)	20 (61%)	7 (21%)
<i>Section 4.5.2: Seeking help with debugging</i>						
Seeking help from nearby colleague	69 (65%)	7 (7%)	30 (28%)	31 (82%)	1 (3%)	6 (16%)
Acknowledging help from a nearby colleague	31 (30%)	42 (40%)	31 (30%)	4 (10%)	27 (66%)	10 (24%)
Seeking help from colleague by email	22 (21%)	56 (53%)	28 (26%)	25 (66%)	5 (13%)	8 (21%)
Acknowledging help from an emailed colleague	39 (41%)	28 (29%)	29 (30%)	5 (14%)	26 (70%)	6 (16%)
Seeking help online	22 (21%)	55 (52%)	29 (27%)	16 (42%)	12 (32%)	10 (26%)
Acknowledging online help	56 (58%)	21 (22%)	20 (21%)	11 (23%)	17 (36%)	9 (40%)
<i>Section 4.5.3: Outsourcing</i>						
Paying another person to write your program	4 (4%)	95 (90%)	7 (7%)	6 (16%)	22 (58%)	10 (26%)
Acknowledging payment for code	28 (44%)	26 (41%)	10 (16%)	11 (37%)	11 (37%)	8 (27%)
<i>Section 4.6: Attribution of help with coding</i>						
Existence of standard acknowledgment format	16 (16%)	79 (78%)	6 (6%)	5 (13%)	31 (82%)	2 (5%)