Introductory Computing Students' Conceptions of Illegal Student-Student Collaboration

Michael Stepp, Beth Simon
Computer Science and Engineering Dept.
University of California, San Diego
La Jolla, CA
{mstepp, bsimon}@cs.ucsd.edu

ABSTRACT
Academic integrity and cheating are issues of specific importance in computing courses due to the restricted nature of much of our assigned work. Additionally, use of valued pedagogical and professional practices such as pair programming can muddy the waters when it comes to students' understandings and experiences with collaboration. In this study we report on 112 students at the beginning of a second programming course being asked to describe a scenario of student-to-student collaboration that "crosses the line" in terms of what should be allowed in the course. We find that students describe inappropriate acts involving sharing of code and sharing of information, with the former being more prevalent. Additionally, about half of the scenarios include mitigating circumstances that should not affect the propriety of those acts. Finally, when presented with other students' (often vague) scenarios, students have little consensus on whether those reflect appropriate or inappropriate collaborations.

Categories and Subject Descriptors
K.3.2 [**Computer Science Education**]: Introductory programming – *academic integrity.*

General Terms
Human Factors.

Keywords
CS1, Cheating, Plagiarism, Academic Integrity.

1. INTRODUCTION
The academic integrity or collaboration policy for a computer science course is perhaps more interesting than in other disciplines, because programming assignments are often structured in restricted ways. Educators give introductory programmers straightforward tasks, and encourage them to write code in one particular style. Furthermore, the compiler catches a wide range of mistakes, giving greater similarity through enforced correctness. This greatly reduces the solution space, meaning that there are far fewer acceptable answers to a programming assignment than, say, an English essay.

Additionally, a clear and explicit policy is especially important for beginning students, since they may have little or no experience in "programming for a grade". Their previous experiences in other disciplines collaborating on homework may have a different "look and feel" than those with a programming assignment.

Though at first it may seem trivial to create a collaboration policy, there are many elements to consider. For instance, if Student A asks Student B for help, should B be allowed to look at A's code? Can A look at B's code? In either case, should the student be allowed to look at the whole program, or a limited portion? Can the helper describe the solution, or is she limited to explaining the high-level concepts involved? How would students know the difference between those two situations? Is debugging collaboration considered differently than code-creation collaboration? Most importantly, how can these various situations be described in a way that novice students can understand and, hopefully, apply in their actual programming experiences?

In this paper, we take a different tack from much of the related literature which asks students to evaluate pre-defined (and expert-defined) situations. We asked 112 students at the beginning of a second computing course to invent scenarios regarding "appropriate" collaboration practices between students. We find that students discuss a range of practices that generally fall into two categories regarding sharing of *code* or *information*, and that students are more likely to describe the former. We also find little consensus in students' ability to determine the propriety of other students' scenarios. Through this exercise, we identify students' lack of attention to key details when creating scenarios, and suggest value in both training of students in this area and class discussion of such scenarios. Our hope is that improved student ability at creating and evaluating scenarios may make their collaboration with each other more thoughtful, and perhaps more in line with local norms. Alternately, such scenarios, as coming from the student perspective, may be more effective in defining a collaboration policy that is meaningful and useful for students and we provide a handout for students to support such discussion.

2. RELATED WORK
[1] studied first year students' ability to detect plagiarism from predefined scenarios, in contrast to student-generated ones. In addition, the students rated 15 scenarios relating to copyright violation and plagiarism, based on the seriousness of the infraction. They found students tend to consider some forms of

plagiarism as more serious than others, and that they would be more likely to admit to the ones they considered less serious. Students also had difficulty identifying which scenarios involved plagiarism and which didn't.

[2] conducted a study of 1206 students and 190 academic staff asking about four issues (seriousness, penalty, prevalence, personal history) about 20 different scenarios relating to academic integrity. For every scenario, the students rated the seriousness as lower, the penalty as lower, and the prevalence as higher than the staff did. Staff underestimated the actual prevalence of these acts (as reported by personal history).

[3] had 103 students at Monash University rate 18 different scenarios relating to academic integrity violations. Students were very homogeneous in their views of what counted as academic integrity violations, but those views did not correspond well with university policy -- many students rated a variety of policy violations as acceptable. In this study, novices are not found to have homogenous views when shown student-described scenarios.

Our work seeks to bring greater authenticity and student reflection to the "scenario technique" by engaging students in developing their own collaboration or academic integrity scenarios. We also seek to develop instructional support in effective discussion and student application of academic integrity guidelines.

3. METHODOLOGY

3.1 Subjects

Data reported here is from two terms (Winter 2009 (Class A) and Spring 2009 (Class B)) of a CS1.5 course (2nd 10 weeks) at the University of California, San Diego. The previous CS1 course (for students with no prior programming experience) required the use of pair programming for assignments. Thus, this was students' nominal first experience programming independently. The same instructor (Simon) taught both CS1 and CS1.5 (both terms). A total of 164 students submitted answers but we report on 112 scenarios. We removed 16 student responses because they didn't answer the question, and 36 responses from Class A because we asked them a slightly different question (described below).

3.2 Assignment

Our experiment took the form of a survey presented to the students in both Class A and B. The exact text of the survey for Class A is given in Figure 1. Class A had the option of describing either an appropriate or an inappropriate collaboration scenario between two students in the class. We chose to focus on collaboration between students in the class based both on students' previous experience in pair programming and our common "lab culture" -- most students do their programming in a common room on campus.

For Class B, the assignment changed to exclude the "appropriate" response option, based on excessive vagueness in those responses from Class A. Specifically, it stated "[T]he goal of this description should be to outline the 'cross over line' where the kind of help you might seek from another student in the class is not allowable or fair." In addition, we presented 5 situations generated in Class A (4 "appropriate", 1 "inappropriate") to Class B students to be rated as one or the other. These situations, chosen for the "less than clear" situations they presented, were:

1. You've been trying to finish your assignment but are stuck on one of the methods and cant [sic] figure out how to do it. You call over another student and ask for help and they let you see what they did for there [sic] method to help you get started. [W35]
2. When it is a bug you have narrowed down where it can be to a small portion of the code. Just a couple of lines perhaps, or maybe its [sic] just in one method. This way hopefully you don't to show your whole code to the other person. [AltW31]
3. When my code does not work, I would ask my friend to give me a hint. So, when my friend comes by to help me on the code, he and I would generate a code as we discuss through the process. [AltW07]
4. Student A is working on his program but he has no idea what to do, then he asks student B about it. After listening to the explanation of the assignment and student B's idea of how to do it, student A is inspired and starts to do his own work without copying exactly what B tells him. [AltW34]
5. Student A and Student B are both assigned a program. Student A completes it and it works perfectly. Student B is struggling with the assignment and asks Student A for help. Student A comes over and skims the code, gives advice, but not directly telling Student B what to do. (i.e. "Take another look at your loops") or, Student A can also draw a diagram or something to help Student B understand the assignment more clearly. [AltW22]

The purpose of this activity is for everyone in the class to consider the reasons and value of having rules for what constitutes legal or "OK" ways of collaborating with other CSE8B students on programs. This does NOT address interactions with tutors for CSE8B. On one side, we know that it can be very valuable to get a "second set of eyes" when going after a difficult bug. It can be wasteful to spin your wheels forever when you might make progress much faster with someone else's quick comment. On the other side, we know that you, as a professional, have an expected level of personal effort and debugging that you are expected to be able to perform. This DOES involve struggling and trying to debug on your own. Additionally, program grades are part of your assessment in this course, and your personal hard work should be rewarded fairly. Below, **tell us about a debugging scenario you can imagine encountering in this class and tell us whether YOU think it should be legal to collaborate with another 8B student in that scenario (or not).**

Figure 1. Assignment posed to Class A (emphasis added)

3.3 Analysis Methodology

The authors developed an initial set of categories by reviewing the entire data set. Refining the categories led to complete agreement in coding responses. Based on students' descriptions of "inappropriate acts", we found 9 common categories of inappropriate acts, with a 10th included as "Other" to capture the remaining. The categories are given by example on the final page. Every student response fit into at least one inappropriate act category, but we coded them with as many categories as were applicable (max: 3).

Of their own accord, some students noted "mitigating circumstances." Only 55 students included mitigating circumstances. These are also given by example on the final page.

4. RESULTS

4.1 Generated Scenarios

Inappropriate Acts. Figure 2 shows the breakdown of inappropriate acts as identified from student-generated scenarios in the two classes. There were two main categories of inappropriate collaboration, involving either sharing *code* or *information.* Examples of categories are shown on the last page.

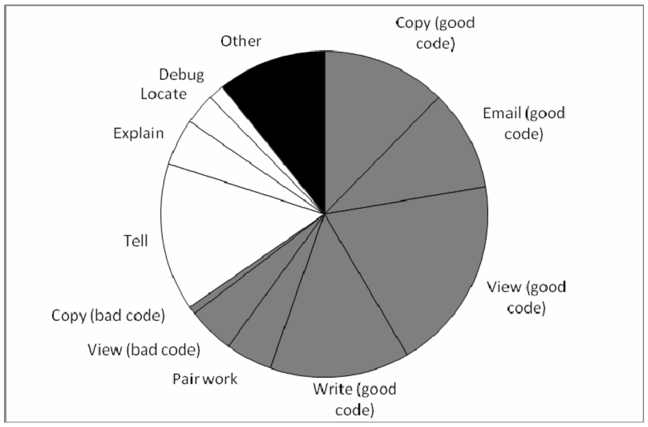

Figure 2. Freq. of Inappropriate Acts in Student Scenarios

65% of the inappropriate acts identified in student scenarios involved sharing of *code* (dark grey). The grand majority of code sharing acts (85%) occur "directionally" from a knowledgeable student to a less knowledgeable student (Copy, Send, View, or Write "good code"). Relatively few acts described students working together (often as a holdover from the previous class' pair programming technique) or having a more knowledgeable student working in some way with the less knowledgeable students' code (View or Copy "bad code").

24% of acts involved sharing of *information* (light gray) from a knowledgeable student to one less knowledgeable (Tell, Explain, Locate or Debug a bug). We classified the remaining 11% of acts as Other, either because they were too vague to classify or because they did not reflect two students collaborating (e.g. "google for some codes" [S01]).

Mitigating Circumstances. Just under half of all students (47%) also went on to give, as part of their scenario, a notable mitigating circumstance that might more clearly explicate when an inappropriate act may occur. The variety of these mitigating circumstances is interesting in helping to understand the expected situations in which students may be pressured to cheat.

Table 1. Mitigating Circumstances Frequencies

Mitigating Circumstance	**Frequency**
Stuck	31
Being at Different Places	16
Procrastination	15
Amount of Code	6
Asking Fails	2
Not Exhausting your own Resources	2

We categorized mitigating circumstances into six groups, shown in Table 1. Examples of the categories are shown on the last page. *Stuck* was one of the most common mitigating circumstances. Additionally, variations on being stuck sometimes mentioned frustration as a result of being stuck or, specifically, a student having exhausted their resources (having asked a TA, or having no TA available) in addition to being stuck. A surprising and common circumstance involved when two students have worked "different amounts" or were in "different places" with respect to completing their code – often cited as an aspect that makes collaboration inappropriate. We did not code as "different places" the most common case of one student being "completely done" with their program, as this was ubiquitous in most responses (either explicitly or implicitly).

4.2 Student Evaluation of Scenarios

As shown in Figure 3, when asked to evaluate other student-generated collaboration scenarios, only once was there solid consensus from students. Again, in this question, we ask students to say whether acts were "appropriate" rather than "legal" because we encouraged the students to think for themselves about the value and consequences of collaborating with others. The instructor and TA for the course (the authors) both agreed on the kind of collaborations they could expect to allow in this course. They agreed with the majority of students on all scenarios except Scenario 3, where they thought the phrase "he and I would generate a code" was beyond what was acceptable.

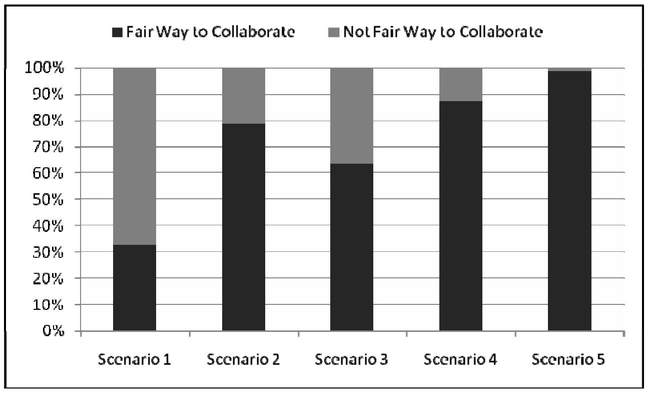

Figure 3. Spring Student Assessment of Provided Scenarios

5. DISCUSSION

Generating Collaboration Policies. Most previous work has looked at student ability to identify and rate violation of academic integrity policies and has shown students struggle to appropriately apply such rules. Might we reach a better outcome if we involved students in the process more generatively? Here we engage students in "defining" policy by "consider[ing] the reasons and value of having rules for what constitutes legal or 'OK' ways of collaborating" as part of consideration of their professional development and academic value of course assessment.

This process for generating local or course-specific policy has potential benefit for *instructors*. Students had, in a previous course, signed a computing-specific academic integrity statement which included standard wording on plagiarism and prohibited specifically "providing, procuring or accepting assignments in part or in whole" to/from other students, required that code be "student's original work", and outlined that "collaboration with other students to develop, complete, or correct course work is limited to activities explicitly authorized by the Instructor". Recognizing that she lacked a list of "explicitly allowable" collaboration activities to provide to students, the instructor asked students to help generate such a list that she would then endorse for the course.

This process for generating local or course-specific policy has potential benefit for *students*. Another consideration is to engage students in considering the implications of collaboration activities on their overall learning process, given recognized positive aspects of collaborations for novice programmers. Part of the motivation for pair programming in CS1 is to help students struggle less with getting stuck (especially debugging), and to develop skills in discussing and evaluating multiple solutions to a problem specifically as that process positively influences learning [4]. From this viewpoint, the instructor approved collaboration decisions (based on the 5 scenarios in Spring and their discussion in class) help students draw the line in interpreting the phrase "student's original work" (e.g., discussion up to a point is valued, generating code together crosses the line). Viewing others' code to help debug is "reasonable collaboration" as long as the amount of code viewed is small. While these scenarios leave significant room for improvement, their discussion does allow for reflection on positive valuation of collaboration targeted at improved understanding, versus "simple sharing" of code or information.

Code versus information. By instructor standards for this class, all code sharing activities (except possibly those that have the more knowledgeable student viewing the less knowledgeable student's code, for say restricted debugging) were clearly inappropriate acts. As **clearly** inappropriate acts, it is concerning that so many student responses are of this type, rather than exploring the more interesting "borderline" cases. Certainly, discussion of the difference in "tell" and "explain" scenarios would be quite valuable in class. Students described "Telling" as a form of "speaking in code", compared to "explain" which implied more algorithmic or higher level discussion, and also often implied more 2-way discussion rather than "dumping" of information from a knowledgeable source to a less knowledgeable one. In this class, many cases of "explain" (and also "locate" and "debug") were in fact considered allowable by the instructor – or at least could be with small modifications to the described scenario (see [S38] on last page).

Often two scenario characteristics applied that made View (bad code), explain, locate, and debug reasonable: limited scope of assistance and lack of "use" of knowledgeable student's "good" code. Specifically, having a more knowledgeable student view, explain or work with (in a limited way) a less knowledgeable student's code was more likely to be seen as appropriate, valuable collaboration – compared to engaging the less knowledgeable student with the "good code".

Evaluation of Scenarios. Though students achieved little consensus evaluating each other's scenarios (replicating reports when evaluating pre-defined scenarios), we chose those scenarios for their vagueness and, hence, discussion potential. In future work, this sort of "peer review" might be a valuable process in helping students develop better scenarios. Future work would then be required to see if resultant student-generated scenarios are, in fact, more likely to be understandable and consistently evaluated by other students, perhaps even at different institutions.

6. IMPLICATIONS FOR INSTRUCTORS

Developing Skill in Describing and Assessing Collaboration. From reviewing the scenarios we find that students commonly lack skill in specifying details of collaboration scenarios to make them clearly assessable. Often, the degree of interaction and other important characteristics are missing or muddied. This can leave open issues for discussion (see below), but makes it difficult to define explicit, easily-followed rules. If we want students to accurately follow a set of behaviors, we should consider explicitly developing their skills in both describing and evaluating such scenarios. Perhaps introductory computing courses should engage students in writing detailed scenarios as both a goal in itself and to provide them greater opportunity to reflect on their behavior. One might provide students with a set of "sample" scenarios that are worded too vaguely, then ask each student to "elaborate" both an appropriate and an inappropriate version of them. Making this a class assignment could help develop more contextualized, institutional, or course-specific sets of guidelines.

Class Discussion. Even without adding explicit educational goals as outlined above, having students generate their own scenarios regarding collaboration may have value. For an instructor, review of such responses may highlight common behaviors at your institution or specific cases where students are clearly mistaken in understanding local accepted practices. For students, a valuable class activity could be a discussion of their own results, or even those provided by students in this study (which may make the conversation less loaded, especially in small classrooms). The last page of this paper provides student-generated scenarios as a starting point for classroom discussion, which we invite instructors to use verbatim or modify for their own purposes.

7. CONCLUSIONS

In this work we report on the experience of asking students in an introductory computing class to define their own inappropriate collaboration scenarios. Results feature the sharing of *code* and *information* as distinct situations. We consider the possibility of this first experience for future, more comprehensive instructional processes for improving both collaboration policies and students' abilities to recognize and evaluate them.

8. REFERENCES

[1] Marshall, S., Garry, M. (2005). How well do students really understand plagiarism?. In H. Goss (Ed.), Proceedings of the 22nd annual conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE) (pp. 457-467). Brisbane, Australia, 4-7 December.

[2] Brimble, M., Stevenson-Clarke, P. (2005), Perceptions of the prevalence and seriousness of academic dishonesty in Australian universities, *Australian Educational Researcher*, Vol. 32 No.3, pp.19-44.

[3] Dick, M., Sheard, J., and Markham, S. 2001. Is it okay to cheat? - the views of postgraduate students. *SIGCSE Bull.* 33, 3 (Sep. 2001), 61-64.

[4] Simon, B., Hanks, B. First Year Students' Impressions of Pair Programming in CS1. JERIC 7, 4. 2008.

Student guide to unfair collaboration scenarios

These scenarios are students' own words describing possible situations where they might work together in ways that would not be fair based on university standards and their goals for professional development.
Do you agree with them -- are these inappropriate ways to collaborate? Do they provide enough detail for you to decide?

Illegal Acts: sharing information

- ***Tell*:** One student encounters a failure with his code and seeks help from surrounding peers. He finds another student who has had the same failure and that student has identified the defect. The student asks this student to **tell him exactly what it is that he changed** to make it work. [W23]
- ***Explain*:** I am working on an assignment that is due in 2 hours and panicking and just need to do one last part before I finish. I see my friend in the lab as well and I go over to him and ask him for some quick help. I notice he is stuck on a part that I finished and he did the part I need. I tell him we should just **explain in words how we did our own sections in words away from the code** and he agrees. This is crossing the line because it is not friendly debugging help (which might be ok) but it is actually an unfair help in a crucial step in completing the code. [S38]
- ***Locate*:** Let's say you get an out of bounds error. You don't know why that is. You can ask your friend for possible reasons, but if you get your friend to actually **go up to your computer and point out, "your problem is here"**, that crosses the line. [S32]
- ***Debug*:** **Asking someone else for help debugging** when you have not exhausted your own resources of finding the bug. [W29]

Illegal Acts: sharing good code

- ***View*:** You've been trying to finish your code but are stuck on one of the methods and can't figure out how to do it. You call over another student and ask for help and **they let you see what they did for their method** to help you get started. [W35]
- ***Copy*:** One individual is having trouble figuring out an assignment. He has started on it and has been working on it for awhile, but cannot seem to finish the last part. He asks another student in the class for help. **He copies/pastes the last section to his program directly from the other student's program.** [S07]
- ***Send*:** Student A asks student B to debug a code. **Student A gives student B the whole code through e-mail** because it's late at the lab. Student B does debug some of the code, but there is a risk of student B keeping most of the code for himself. [W44]
- ***Write*:** You are having a problem with your code because it is having an out of bounds error and you are not sure why. You ask a friend and **he fixes the line that is messing up your code**. [S17]

Illegal Acts: coding together

- ***Pair*:** Student A needs help debugging and Student B realizes he has made a similar mistake. Both students have completed their program. **They work together to solve the bug** but student

Illegal Acts: sharing bad code

- ***View*:** Classmate A has a runtime error on his code and doesn't know what's wrong. He is about 90% done, and needs someone to **skim over his code to see what he did wrong**. He contacts his friend, classmate B, to check over his code. He sends it to Classmate B via e-mail, and just asks him to find out what's wrong and email him back. [W06]
- ***Copy*:** A scenario that would not be okay would be if the debugging involves errors in most parts of the program so that none of the program is executing. This is because then a peer reviewer would have to go through and understand all parts of the students program and would see exactly how they did their work. I think that would make it too easy and tempting **for the reviewer to copy if they hadn't started working on their project yet**. [W33]

Mitigating Circumstances

- ***Stuck*:** Student A is finished with the body of the code but when he/she runs the code the expected result does not occur. Student A has **spent a considerable amount of time trying to figure out where the bug is** and what he/she has done wrong; however, he/she is unsuccessful in fixing the program. Student A asks Student B for assistance in locating where the code is going wrong. [S63]
- ***Different Places*:** Sally, a good friend of mine, had been working on her program for a couple of days. When she compiles her code, everything is alright except for a logic error. **She wants me to check over her code when I haven't even written mine yet.** I should tell her she should look for a tutor for her problems because I'd be tempted to use some of her code. [S21]
- ***Procrastination*:** **It's late at night and the assignment is due tomorrow**, I email a classmate and ask if they can help out by sending part of the program they made to help me figure out my error and finish my code. [S60]
- ***Amount of Code*:** If someone has a code that is riddled with errors which they cannot spot and they ask for help it's ok, but as soon as they start letting the other person **rewrite parts of it beyond a line or two** it starts becoming questionable. [S09]
- ***Asking Fails*:** Let's say a person gets stuck on their code for quite some time. They **have been asking for help from fellow students, and finally the person helping gets frustrated** and lets them see their own code. The person then understood his/her mistake and used what he/she saw to fix his bug. [S18]
- ***Not exhausted resources*:** When you have **spent minimal time examining the problem yourself** and ask for help after like 5 minutes of looking at it. When you know the other person is already done and maybe willing to help push you in the right direction. [W34]