STRATEGIES FOR PROMOTING ACADEMIC INTEGRITY IN CS COURSES

Eric Roberts[1]

Abstract ¾ *In recent years, plagiarism in computer science courses has become increasingly widespread. This paper describes the approach taken at Stanford University over the past ten years in an attempt to control this problem. Our approach consists of five steps. First, we have encouraged computer science faculty to become actively engaged in the university judicial process. Second, we have instituted the practice of using "expert witnesses" who are familiar with programming assignments as part of the judicial process. Third, we have redefined the most common transgression—students who copy assignments from one another—as "plagiarism" instead of "unpermitted collaboration." Fourth, we have adopted electronic tools to detect instances of plagiarism. Fifth, we have established consistent and explicit departmental policies about collaboration and plagiarism and made sure that those policies are well understood by students. By adopting this multifaceted approach, the computer science department at Stanford has been able to detect a larger fraction of the instances of academic dishonesty and prosecute more effectively the violations that occur.*

Index Terms ¾ *academic integrity, plagiarism.*

INTRODUCTION

Since at least the 1970s, plagiarism, excessive collaboration, and other violations of academic integrity have been recognized as serious problems for departments of computer science [10, 12]. As at many universities and colleges, computer science courses at Stanford account for the lion's share of the cases brought to the Office of Judicial Affairs, which is responsible for investigating and adjudicating violations of Stanford's honor code. In each of the past ten years, computer science has been the source of more honor code violations than any other department. As Table I illustrates, computer science represents between 20 and 54 percent of the total cases depending on the year, with an aggregate share of 37 percent over the past decade. Given that computer science enrollments are only about 6.5 percent of the total university enrollment, accounting for more than a third of the total violations is highly disproportionate.

Stanford, however, is by no means alone. Most colleges and universities experience a similar problem. At MIT in 1991, 73 students in a course entitled "Introduction to Computers and Problem Solving" were disciplined for excessive collaboration, out of a total enrollment of 239. This incident represented the largest cheating institute in the history of the institute and was written up in *The New York Times* [2]. Other references show that this problem is both widespread and international [3, 5, 6, 14, 15].

TABLE I
HONOR CODE CASES AT STANFORD (1991-2001)

	CS cases	Total cases	% CS
1991-92	10	20	50%
1992-93	11	35	31%
1993-94	12	28	43%
1994-95	9	45	20%
1995-96	12	40	30%
1996-97	8	22	36%
1997-98	11	33	33%
1998-99	13	33	39%
1999-00	21	58	36%
2000-01	32	59	54%
Total	139	373	37%

At Stanford and elsewhere, most violations of the honor code that arise in computer science involve homework assignments rather than exams or other aspects of course work. The two most common problems are as follows:

1. Two or more students work as a team to complete an assignment even though the rules for that assignment stipulate that the work be done individually.
2. One student obtains a copy of someone else's code and then submits that version, often after making simple edits in an attempt to disguise the source. Students obtain copies of code by a variety of methods, such as rooting through discarded program listings taken from a recycling bin or checking machines in public clusters to see whether a student previously using that machine might have left a solution lying around.

WHY IS CHEATING SO PREVALENT IN CS?

Before looking at the steps a department can take to reduce the level of unpermitted collaboration and plagiarism on programming assignments, it is useful to give some thought to the question of why these violations of academic integrity seem to be so much more prevalent in computer science than they are in other disciplines. After thinking about this question for many years, we have identified the following factors as potential causes of the high incidence of cheating in computer science:

1. *Economic factors create high levels of competition in computer science that in turn encourages certain types of abuse.* Particularly in the years of the Internet boom, many students have seen a computer science degree as the stepping stone to wealth and job security. As a result, many students have been attracted to computer

[1] Eric Roberts, Department of Computer Science, Stanford University, Stanford, CA 94305-9015 eroberts@cs.stanford.edu

science more by its marketability than by any intrinsic interest in the subject. The economic power that being able to program well confers generates significant pressure for students, who often see doing well in a computer science course as critical to their future livelihood. Those students have a tangible incentive to succeed that sometimes leads them beyond the bounds of acceptable behavior.

2. *The material in computer science is highly cumulative and requires consistent effort on the part of the student.* From talking with students over the years, it is clear that a significant fraction of the students who violate the honor code are those who have fallen hopelessly behind. Students who neglect their work in the early part of a term often are completely confused by the later assignments. If an assignment is due and you have absolutely no idea where to start, the pressure to cheat increases significantly.
3. *The computer is a relentlessly unforgiving arbiter of correctness.* In traditional disciplines, students sometimes submit work that they know isn't very good in the hope that the grader either won't read it carefully or will be feeling generous that day. Assignments that involve programming work very differently. Before you turn in a programming assignment, you've got to get that program running on a machine—a machine that forces you to fix your errors before it executes your code. As long as your program doesn't work, the computer simply rejects it. What's more, a human grader that tries to run your submission will know that your program doesn't work, because it is so easy to test it on the machine. Having the computer continually tell you your program is wrong generates significant anxiety.
4. *There is no shortage of sources for working solutions.* The high level of frustration associated with computing provides the motive for honor code violations, but the large number of cases also reflects the widespread opportunity most students have to gain access to working solutions. Students work in close proximity to one another in terminal clusters. If there are other people working on the same assignment, it is tempting to ask them for direct assistance, peek over their shoulder at the screen, or retrieve their program listings from a recycling bin. One very common problem is that many students accidentally leave copies of their work on the hard disk of the machines they use in the public clusters, which makes it easy for students to acquire a complete electronic copy, which removes even the necessity or retyping the program. Increasingly, the web also serves as a source for solutions, since search engines will often uncover code that matches closely the demands for many traditional assignments.
5. *Computer science courses often reuse past assignments.* The problem of having solutions be generally accessible is exacerbated by the fact that computer science courses often reuse assignments from year to year. The incentive for doing so is not laziness, but rather the desire to maximize the quality of the assignment. Assignments are rarely very effective the first time they are used because assignments, like programs, need to be debugged. The conventional wisdom in computer science education is that a program must be assigned three or four times before it acquires the polish that makes it a good assignment. The downside risk, however, is that students can easily acquire solution sets from past offerings of a particular course.
6. *Plagiarism is easier to detect in computer science.* Although the other factors cited in this list offer plausible explanations as to why programming assignments might encourage students to cheat, it may simply be that computer science courses discover a larger fraction of the violations that occur. It is relatively easy to detect copying on an assignment, particularly given tools for analyzing program similarity [1, 5, 11, 14]. Perhaps more to the point, it is certainly easier to detect copying on a programming assignment than most students imagine it will be. As a result, students may be lulled into thinking that they will get away with their transgression by failing to realize how accurate plagiarism-detection software can be.

STRATEGIES TO COMBAT CHEATING

Over the last decade, the Stanford computer science department has taken several steps to combat cheating in our courses. In brief, the five most significant steps are as follows:

1. Encourage computer science faculty to become actively engaged in the university judicial process.
2. Institute the practice of using "expert witnesses" who are familiar with programming assignments as part of the judicial process.
3. Redefine the most common transgression—students who copy assignments from one another—as "plagiarism" instead of "unpermitted collaboration."
4. Use electronic tools to detect instances of plagiarism.
5. Establish consistent and explicit departmental policies about collaboration and plagiarism and make sure that those policies are well understood by students.

Each of these strategies is described in more detail in the sections that follow.

Faculty Involvement in the Judicial Process

When I took over as director of undergraduate studies in 1990, the faculty in the computer science department were generally quite disillusioned by the processes that were then in place for handling violations of the honor code. One of the most significant complaints was that the administrative officers and judicial panels hearing computer science cases often had very little understanding of what actions would represent a violation of the honor code in computer science and still less as to what evidence would constitute a reasonable standard of proof. This lack of experience had led to several cases where charges were dismissed in the face of what the computer science faculty considered to be

essentially incontrovertible evidence. In the wake of these decisions, some of the faculty withdrew from the existing judicial system and began imposing their own penalties when they discovered students were violating the honor code.

Abandoning the system, however, had the effect of weakening it further. With fewer computer science faculty taking part, the judicial process was even less connected to the specific concerns of the department with the largest case load. To reverse this trend, I worked hard to have the department become *more* involved in judicial affairs. Over the last ten years, computer science faculty have played significant roles on the committee to rewrite the judicial charter, the policy board that governs judicial affairs, and the individual judicial panels that make the determinations of fact when students are charged with violations. The net result of the increased involvement has been that the system is now much more attuned and responsive to the specific needs and problems of computer science.

Expert Witnesses

The first structural change that we were able to incorporate into the judicial process occurred in 1991, when we lobbied to have the investigative officer consult with members of the computer science faculty whenever a case hinged on the likelihood that one program submission was copied from another. In an attempt to disguise the fact that their work is copied, students often make insignificant changes in the program source, such as renaming the variables or rewriting the comments. Those changes typically leave in place massive similarities that the trained eye—or electronic tool, for that matter—has no trouble distinguishing. Someone from outside the field, however, might look at two such submissions and see them as different. The only way to recognize the similarities would be to call in as an expert witness someone who has the necessary experience to make a proper determination.

To emphasize the need for expert witnesses, I offered as an example to the judicial affairs board the question as to what they might do when confronted with two expository papers written in some language using a non-Latin alphabet that no one on the judicial panel could read. No one would presume to say the two papers were different because some curlicue on one letter was a little longer than its counterpart or because the line breaks occurred in different places. Faced with that sort of problem, the only way to determine similarity would be to bring in someone who could read the language involved. The same is true for programs.

From 1991 until the new judicial charter was adopted in 1997, the judicial affairs office would ask faculty members unconnected with the case to provide an estimate as to the likelihood that two programs had an independent genesis and a short exposition of their reasons for making that determination. The inclusion of these expert witness reports had a significant effect on the fraction of cases in which the students were found to be in violation of the honor code. The new judicial charter provides other avenues for incorporating such testimony into the judicial proceedings, which has led to a decline in the need for expert witnesses.

Plagiarism vs. Collaboration

The most significant change we have made in our approach to pursuing honor code violations at first seems too subtle to have much impact: we stopped using the category of "unpermitted collaboration" (which was one of the possible violations enumerated in the judicial code) and replaced it with the more definitive charge of "plagiarism." The major advantage behind this change is that it turns a highly nuanced decision into a clear-cut binary one. Unpermitted collaboration is a slippery slope. By its very construction, the notion of "unpermitted collaboration" implies that there must also be "permitted collaboration," which is precisely what we want to convey. We encourage students to brainstorm, discuss strategies, and ask questions of other students because doing so makes the learning process far more active. At the same time, we insist that students write and debug their own code, because the value of those experience is lost if one simply gets the answer from someone else. Thus, the concept of collaboration represents a gradient in which some activities are clearly permitted, some are clearly prohibited, and some fall in the gray area in between.

The uncertainty in the notion of collaboration makes it difficult to use that standard as the basis for an honor code charge. Students who were brought up on that basis could and indeed did argue that they were unaware that their actions had crossed that nebulous dividing line between permitted and unpermitted collaboration. Particularly under a judicial system that uses "beyond a reasonable doubt" as its standard of proof, this level of uncertainty makes it hard to secure a conviction if there is the least concern in the mind of the review panel on the question of intent.

In most cases, however, we found that students who had actually copied someone else's work—and even many of those who had simply collaborated with others on an entire assignment—clearly understood that they were doing something wrong. It may be reasonable to argue that you didn't realize that you were crossing the line between permitted and unpermitted collaboration. That distinction, after all, is vague. On the other hand, if you *change* the code to disguise the fact of your collaboration, that action undercuts the legitimacy of your claim. In the honor code cases that represented the greatest threat to overall academic integrity, we found that students were very likely to cover their tracks.

Based on this observation, we decided to approach honor code cases as instances of plagiarism rather than as collaboration that had gone too far. We tell students at the beginning of class, as you can see from the handout in Figure 1, that they must indicate on their assignment submission any assistance they received. If they get help from a classmate or a teaching assistant, all they need to do is cite that assistance in the comments to the program. As long as people believe themselves to be on the safe side of

Computer Science and the Stanford Honor Code

Since 1921, academic conduct for students at Stanford has been governed by the Honor Code, which reads as follows:

THE STANFORD UNIVERSITY HONOR CODE

A. The Honor Code is an undertaking of the students, individually and collectively:
   (1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   (2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

In the Computer Science Department, we take the Honor Code seriously and expect you to do the same. The good news is that the vast majority of you will do so. The bad news is that all historical evidence indicates that some students in computer science will submit work that is not their own, shortchanging not only their own learning but undermining the atmosphere of trust and individual achievement that characterizes Stanford's academic community. Each year, the Computer Science Department accounts for somewhere between 20 and 50 percent of all Honor Code cases, even though our courses represent only about seven percent of the student enrollment.

The purpose of this handout is to make our expectations as clear as possible in the hope that we will reduce the number of Honor Code violations that occur. The basic principle under which we operate is that each of you is expected to submit your own work in this course. In particular, attempting to take credit for someone else's work by turning it in as your own constitutes plagiarism, which is a serious violation of basic academic standards.

From the attention that the department pays to the Honor Code, some of you will get the idea that any discussion of assignments is somehow a violation of academic principle. Such a conclusion, however, is completely wrong. In computer science courses, it is usually appropriate to ask others—the TA, the instructor, or other students—for hints and debugging help or to talk generally about problem-solving strategies and program structure. In fact, we strongly encourage you to seek such assistance when you need it. The important point, however, is embodied in the following rule:

Rule 1: You must indicate on your submission any assistance you received.

If you make use of such assistance without giving proper credit, you may be guilty of plagiarism.

In addition to providing proper citation—usually as part of the comments at the beginning of the program—it is also important to make sure that the assistance you receive consists of general advice that does not cross the boundary into having someone else write the actual code. It is fine to discuss ideas and strategies, but you should be careful to write your programs on your own. This provision is expressed in the following rule:

Rule 2: You must not share actual program code with other students.

In particular, you should not ask anyone to give you a copy of their code or, conversely, give your code to another student who asks you for it. Similarly, you should not discuss your algorithmic strategies to such an extent that you and your collaborators end up turning in exactly the same code. Discuss ideas together, but do the coding on your own.

The prohibition against looking at the actual code for a program has an important specific application in computer science courses. Developing a good programming assignment often takes years. When a new assignment is created, it invariably has problems that require a certain amount of polishing. To make sure that the assignments are as good as they can be, Stanford's department—like most others in the country—reuses assignments over the years, incorporating a few changes each time to make them more effective. The following rule applies in all computer science courses:

Rule 3: You must not look at solution sets or program code from other years.

Beyond being a clear violation of academic integrity, making use of old solution sets is a dangerous practice. Most assignments change in a variety of ways from year to year as we seek to make them better. Each year, however, some student turns in a solution to an assignment from some prior year, even though that assignment has since changed so that the old solution no longer makes sense. Submitting a program that solves last year's assignment perfectly while failing to solve the current one is particularly damaging evidence of an Honor Code violation.

Whenever you seek help on an assignment, your goal should be improving your level of understanding and not simply getting your program to work. Suppose, for example, that someone responds to your request for help by showing you a couple of lines of code that do the job. Don't fall into the trap of thinking about that code as if it were a magical incantation—something you simply include in your program and don't have to understand. By doing so, you will be in no position to solve similar problems on exams. The need to understand the assistance you receive can be expressed in the following rule:

Rule 4: You must be prepared to explain any program code you submit.

Although you should certainly keep these rules in mind, it is important to recognize that the cases that we bring forward to Judicial Affairs are not those in which a student simply forgets to cite a source of legitimate aid. Most of the students we charge under the Honor Code have committed fairly egregious violations. Students, for example, have rummaged through paper recycling bins or undeleted trash folders to come up with copies of other students' programs, which they then turn in as their own work. In many cases, students take deliberate measures—rewriting comments, changing variable names, and so forth—to disguise the fact that their work is copied from someone else. Despite such cosmetic changes, it is easy to determine—and we have tools for doing so—that copying has occurred. Programming style is highly idiosyncratic, and the chance that two submissions would be the same except for variable names and comments is vanishingly small.

We have no desire to create a climate in which students feel as if they are under suspicion. The entire point of the Stanford Honor Code is that we all benefit from working in an atmosphere of mutual trust. Students who deliberately take advantage of that trust, however, poison that atmosphere for everyone. As members of the Stanford community, we have a responsibility to protect academic integrity for the benefit of the community as a whole.

FIGURE 1
DEPARTMENTAL HONOR CODE HANDOUT

the collaboration boundary, they will be happy to cite their sources. By contrast, students who have taken someone else's assignment from an unemptied trash can are far less willing to indicate where they got their program.

The major effect of adopting plagiarism as the standard basis for honor code cases is that the more egregious violations are now far easier to prove. In a plagiarism case, the issue is not whether students know that they are crossing an ethical line in terms of collaboration, but simply whether they cited their sources properly. If we can establish that one submission was substantially copied from another, the presence or absence of the appropriate citation is the critical issue in the case. That issue is a binary one: the citation is either there or it isn't. This approach avoids entirely the gray area of collaboration and simplifies the process of establishing responsibility.

Use of Electronic Tools

The various changes in our approach to honor code cases described in the preceding sections are not in themselves sufficient to reduce the level of plagiarism. Particularly with the growth in computer science enrollments over the last five years, the number of submissions and the number of graders have expanded to the point that many, if not most, of the assignments that were copied from another source went entirely undetected. Over the last two years, we have moved to a system of electronic submission that allows us to use Alex Aiken's MOSS (Measure Of Software Similarity) program [1] to uncover massive similarities that suggest copied code.

In the courses that have used MOSS, we have been saddened to find that the rate of unpermitted and uncited copying is typically in the range of three to five percent of the class. This discovery, and the honor-code cases that arose from it, is the principal reason behind the increase visible in Figure 1 of the number of honor-code charges filed in computer science classes during the last two years.

We believe that the continued use of MOSS in the department will lead in a relatively short time to a decline in the number of honor code cases that we see. For too long, students have had a perception that those who cheated on their assignments were getting away with it and therefore doing better than their more honest classmates. As that perception changes, the number of cases should decline.

Definition and Distribution of Plagiarism Policies

The final element of the strategy is an effective campaign to ensure that students have access to and understand both the policy and the department's collective determination to enforce it. The handout shown in Figure 2 is distributed on the first day of the introductory classes and is reinforced by similar discussions in the undergraduate departmental handbook and orientation meetings.

CONCLUSION

The perception about the effectiveness of the judicial system at Stanford has changed remarkably over the last ten years. With the changes we have put in place, both faculty and students now believe that the system works. We believe that a much larger fraction of honor-code cases are being caught and that the prosecution of those who violate the honor code is proceeding both fairly and to greater effect. We believe that this change in perception has created increased respect for academic integrity that will quickly lead to a decline in the number of cheating incidents in computer science courses.

REFERENCES

[1] Alex Aiken. Moss: A system for detecting software plagiarism. Berkeley Computer Science Department. **http://www.cs.berkeley.edu/~aiken/moss.html**.

[2] Fox Butterfield. Scandal over cheating at MIT stirs debate on limits of teamwork. *The New York Times.* May 22, 1991.

[3] Janet Carter. Collaboration or plagiarism: What happens when students work together? Proceedings of the 4th Annual Conference on Innovation and Technology in Computer Science Education. Cracow, Poland. June 1999.

[4] Arlene Franklin-Stokes and Stephen E. Newstead. Cheating: Who does what and why? *Studies in Higher Education*. June 1995.

[5] James Harris. Plagiarism in computer science courses. *Proceedings of the Conference on Ethics in the Computer Age*. Gatlinburg, TN, November 1994.

[6] Thomas Lancaster and Fintan Culwin. Towards an error free plagiarism detection process. *Proceedings of the Sixth Annual Conference on Innovation and Technology in Computer Science Education,* June 2001.

[7] Angela Lipson and Norma McGavern. Undergraduate academic dishonesty at MIT. Paper presented at the 33rd Forum of the Association for Institutional Research, Chicago, IL, May 1995.

[8] Sheilah Maramark and Mindi Maline. Academic dishonesty among college students. *Issues in Education.* 1993.

[9] Donald McCabe and Sally Cole. Student collaboration: Not always what the instructor wants. *American Association for Higher Education Bulletin.* November 1995.

[10] Karl Ottenstein. An algorithmic approach to the detection and prevention of plagiarism. *SIGCSE Bulletin.* December 1976.

[11] Robert Sanders. On-line plagiarism detector helps computer science professors bust cheating programmers. University of California, Berkeley, news release, November 19, 1997.

[12] Mary Shaw, Anita Jones, Paul Knueven, John McDermott, Philip Miller, and David Notkin. Cheating policy in a computer science department. *SIGCSE Bulletin.* July 1980.

[13] Scott Stebelman. Cybercheating: Dishonesty goes digital. *American Libraries.* September 1998.

[14] Kristina Verco and Michael Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. *Proceedings of 1st Australian Conference on Computer Science Education.* Sydney, Australia, July 1996.

[15] Neal Wagner. Plagiarism by student programmers. Unpublished article, **http://www.cs.utsa.edu/~wagner/pubs/plagiarism.html**.