Carnegie Mellon University

From the SelectedWorks of Mary Shaw

1983

Cheating in Computer Science Courses: Problems and Some Solutions

Mary Shaw, Carnegie Mellon University

Available at: https://works.bepress.com/mary_shaw/9/
Cheating in Computer Science Courses: Problems and Some Solutions

Mary Shaw
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA, 15213, USA

Many Computer Science Departments report problems with cheating in their courses. Some of these problems can occur in any university course, but others arise from the technological nature of computing courses and computing equipment. These special problems require special techniques for prevention and detection. In this talk, I will review the general cheating problem and the problems peculiar to computing courses, discuss techniques for preventing cheating incidents and for dealing with incidents that do occur, and present a policy statement we distribute to students in courses at Carnegie-Mellon. I will draw heavily on our experience at Carnegie-Mellon, especially on the Computer Science Department's formal policy [13].

1. Cheating in University Computing Courses

Although detailed data on the incidence of cheating in university computing courses is scarce, anecdotal evidence indicates that the problem is fairly common. Many tales circulate privately, and the subject occasionally attracts public attention (for example, at ACM SIGCSE meetings [6, 10]). These discussions of cheating problems focus on computer science courses, but the same issues arise in other courses that use computing facilities heavily. Indeed, as computer use becomes common in university courses, other departments will look to Computer Science Departments for leadership in dealing with computer-related cheating. Some of the problems in computing courses are essentially the same as the problems in any course, but others are directly related to shared computing power.

Computer Science is as susceptible as any other discipline to the ordinary sorts of cheating. Sometimes the cheating is accidental, as when two students start discussing conceptual material but accidentally slip into extensive discussion of specific assignments. At other times it is deliberate, especially when students are unqualified for the courses, when they feel overworked, and when they feel no intellectual commitment to courses they are required to take. Although these latter circumstances may arise in many departments, they may be particularly prevalent in computer science departments at this time:

"Many students must take computer programming whether they have an interest in the subject or not." (O’odrill, in [10])

"We suspect that the strongest contributing factor . . . [in cheating] . . . is the computing profession’s reputation for being able to offer such high-salaried positions." (Hwang, in [6])

Our main interest here is in the problems that are particularly severe in computer science courses, so we will not dwell on these common problems.

When we set out to develop an explicit policy for computer science courses at Carnegie-Mellon, we asked a number of other computer science departments for information about their experiences with and policies on cheating [13]. The respondents generally felt that the factors most conducive to cheating are large class size, beginning students, and out-of-class programming assignments. Unfortunately, all three factors are usually present in the most common computer science course, introductory programming. Our colleagues uniformly believe this is where the most severe cheating problems occur. Most of them reported one or two blatant cases per course and a feeling that many more less obvious cases went undetected. They widely agreed that the major difficulty is with out-of-class assignments, not with exams. Some respondents also mentioned that illegal use of computer facilities had caused serious problems in their departments.

The cheating problems that are particularly severe in computer science departments seem to stem from technological opportunities for copying information, the pedagogical needs for independent work to provide hands-on experience, and the presence of students who are more interested in job opportunities than in computer science. Since computer facilities are almost always shared, students have ample opportunities for copying solutions -- with or without permission of the owner -- and for sabotaging the work of other students. Various schemes for protection and security can be devised, but virtually all of them
interfere with the need to teach cooperation and teamwork.

Although the study of computer science involves much more than just programming skills, students must have mastered programming in order to grasp principles and to learn advanced techniques that depend on teamwork or the use of programs written by others. As a result, there is often a conflict between the needs for independent and cooperative work. Further, the balance between the two varies from course to course.

"With the dramatically-expanding use of computing systems has come the need to provide introductory programming courses for greater numbers of students and for students with increasingly diverse backgrounds, interests, and capabilities. Computer programming is not a skill which can be taught effectively. It is an ability which is acquired and enhanced through practice. Programming courses must, therefore, provide an opportunity for students to write, test, and perfect meaningful programs." (Lidtke, in [10])

The result is that instructors must be able to expect independent work on occasion, but the facilities that enforce that independence must not interfere with cooperation at other times. Students must also learn to distinguish these two cases.

2. Prevention and Cure

The environment in which a student studies computer science should be adapted to eliminate factors that might motivate a student to cheat or to pressure another student into cheating. However, the environment should not make students constantly aware that they must not even give the appearance of cheating; that would be oppressive. Moreover, students must learn to cooperate if they are to be effective as professionals, and there is a direct tension between the need to teach teamwork and the need for independent performance for individual evaluation.

The general strategies we can pursue are to reduce the need for cheating, the opportunity for cheating, the payoff for cheating, and the chance of successfully cheating. The most interesting methods for preventing and detecting cheating involve course organizations that guard against cheating and programs that examine student programs for signs of copying. Other approaches include supportive instruction facilities, punishment, and protection schemes. This section surveys some of the specific policies and actions that have been explored.

2.1. Innovative Teaching and Testing Formats

Innovative teaching strategies are begun tried in a number of universities, especially for introductory programming courses. The objective of these innovations is usually to improve the effectiveness of the courses, but the new organizations may also reduce the frustrations inherent in writing and debugging code, provide ways to measure individual performance on joint work, and reduce the incidence of cheating. This section examines the ways some of these innovative course structures reduce cheating problems; the pedagogical benefits of the innovations are discussed in the source papers.

At Carnegie-Mellon, we have adopted a "mastery examination" as the primary means of grading the introductory programming course [9]. The proposal for this reorganization was made in 1980:

The introductory programming courses have been a source of difficulty in the department for many years. The major problems include manpower requirements, tension between students and instructors (especially regarding the goals of the course), cheating, uniformity of grades and standards, course format, and software support...

We recommend introducing a mastery exam as an alternative means for students to demonstrate programming skills. . . . We propose an examination lasting several hours in which the student is required to design an algorithm, code the algorithm, debug the code, and document the working program. The reasons for the format . . . include meeting course objectives, accommodating varying rates of learning, reducing academic cheating, using instructors' time productively, and instituting clear and consistent public standards regarding the competence required in our basic course.

The course is now partially self-paced; an advanced program development environment simplifies the mechanics of programming, and students can take the mastery exam whenever they feel prepared -- multiple times if necessary, and either before or after the end of the semester in which the student enrolls in the course. The course grade is largely determined by the mastery exam: a student cannot pass the course without passing the mastery exam, but if the student's homework performance has been significantly better than the mastery exam grade, the course grade will be based on both. The effect of the change has been to increase the amount the students learn while substantially decreasing the opportunity and payoff for cheating -- and the observed incidence of cheating.

Individualized programs (e.g. self-paced CAI [1, 4, 12]) can offer some protection from cheating. The nature of such a program is that it requires regular student participation or response in direct contact with an instructor. If the instructors or tutors know the students and discuss the course work individually, then a student cannot cheat by simply submitting work that he or she has not authored.

Similarly, the Keller-plan organization at the University of Texas [2] keeps student files off-line on floppy disks and controls access to files and self-paced exams by requiring both positive identification and a demonstrable plan for access. Since it is mastery-based, grades are based on proctored demonstrations of competence.

It is also possible to prevent cheating by basing grades entirely on in-class exams or other proctored performance. Under this philosophy, home programming exercises resemble exercises for the music student -- a way to prepare for a public performance. The master exams described above have this character. A less radical alternative is to use "structured walk-throughs" or other events that require students to demonstrate understanding of their programs. Exams with questions about programs that have already been turned in can be used to force students to understand their solutions, though not necessarily to do their own
work. Unfortunately, basing grades on in-class performances (as recommended, for example, in [8]) tends to place the most emphasis on the activities and skills that are the least important. Subverting the thrust of a course is a high price for cheating protection. Hwang [7] discusses some mixed strategies that may strike a reasonable balance.

2.2. Similarity-Detection Programs

A number of schools have developed programs that examine student solutions to detect sets of solutions that are similar enough to raise suspicions of plagiarism. The reports on these systems indicate that they are helpful, but no hard numbers seem to be available [3, 5, 11, 14].

The general strategy in these programs is to derive some lexical or syntactic information about a program and then to compute a function that reduces a program to a single value, a vector of values, or a short string. Equivalence classes of these results are then defined, and programs that land in the same equivalence class are manually examined for signs of cheating.

Thus the "detection" programs should not be expected to provide definitive accusations or "proof" of cheating. They should be used only to simplify the task of comparing large numbers of programs. Actual accusations must be based on human judgments about reasonable and unreasonable degrees of similarity.

Ideally, the similarity detector should be just one of several facilities available to instructors. Other related facilities might provide for running test data against student programs, collecting statistics, and recording grades [14].

2.3. Auxiliary Instruction Facilities

In some cases, students may find themselves in questionable situations because they turn to classmates for help with assignments and their classmates overstep the bounds of reasonable assistance. Some of these situations could be avoided by providing consultation services as extra support for students in courses. Such consultation can often be justified on educational grounds; its influence in reducing the need and temptation for cheating is a bonus. Some considerations for auxiliary course consulting:

- Provide an adequate number of available, knowledgeable consultants to advise students in the lower level courses. Where feasible, staffing these positions with the undergraduates who have already shown affinity for spending long hours in the terminal rooms would allow us to channel their efforts, potentially avoiding situations in which these people are tempted to do substantial portions of other students' assignments.
- Provide guidelines for user consultants, indicating what kinds of help they should and should not give to students in each course.
- Coordinate the activities of user consultants provided by the computation center with those employed by academic departments to support courses. It is important that the information a student receives from consultants not be at cross purposes with either the intent of a programming assignment or with the style with which the courses are taught.
- Upgrade on-line assistance including "help" facilities, debuggers, and on-line explanation of routinely encountered errors.

2.4. Punishment

When we surveyed other computer science departments, we asked about explicit definitions of and policies on cheating. All departments reported the same basic system: no single, explicit definition or policy at the departmental level and a brief definition and elaborate policy at the university level. Typically, individual instructors deal with detection and prevention of cheating using whatever means they happen to know. When cheating is suspected, it may either be handled locally or thrown into the official university channels. (In all cases, the student may file an appeal using the university mechanism.) There was some praise for the idea of always handling such matters at the university level. It was often pointed out that successful prosecution of cheating at a university court requires that non-computer scientists who judge such cases must be sensitized to what should constitute program plagiarism, and that explaining program plagiarism to a non-computer scientist is often a difficult task. Some specific actions on the part of the faculty can help by making the rules clear and by raising student awareness of the reasons cheating is harmful.

- Establish a record of vigorous prosecution of cheating cases. Student confidentiality must be respected, but the fact that a penalty has been levied in a matter of academic dishonesty can be made public without including details.
- Provide students with a hand-out stating the cheating policy and disciplinary action that will reward those who violate it. If modifications are made for a particular course, state them in writing.
- Spread the word that cheating is not condoned. For example, the campus radio station and the campus newspaper can be used to communicate such information.
- Maintain records on cheating incidents in departmental courses in order to take note of repeat offenders.

2.5. Protection Schemes

With the cooperation of the university computer center, opportunities for cheating can be reduced by implementing various security measures. In some cases, however, the security provisions may interfere with legitimate cooperation.

- One department has adapted the operating system to disallow copying of files between student disk areas.
- Another department provides closed trash cans for the disposal of old program listings. Removing a listing is therefore either difficult or awkwardly obvious to the others in the terminal room.
• Where listings are printed centrally and filed for distribution, some measures should be taken to ensure that listings are picked up only by their owners.

3. A Policy Statement for Students

At Carnegie-Mellon, we distribute an explicit policy statement at the beginning of each computer science course. The full text of that statement is part of the departmental policy[13]. It is presented below as a model for other departments to draw on.

** Student Information: Cheating Policy **

Computer Science Department
Carnegie-Mellon University

Since the Computer Science Department is part of the University, the general academic policies on cheating and plagiarism apply within the Department. These policies suffice for much of our work, including examinations and written assignments. However, they do not deal explicitly with course work involving computers; thus the policies must be extended to cover those cases.

The decision as to whether a student cheated depends on the intent of an assignment, the ground rules specified by the instructor, and the behavior of the student. Two guidelines help an instructor decide if cheating has occurred:

Program plagiarism will be suspected if an assignment that calls for independent development and implementation of a program results in two or more solutions so similar that one can be converted to another by a mechanical transformation.

Cheating will be suspected if a student who was to complete an assignment independently cannot explain both the intricacies of his or her solution and the techniques used to generate that solution.

It is unreasonable to expect a complete definition of cheating; each case is important enough to be given careful, individual scrutiny. It is, however, helpful to have guidelines and precedents. Here are some examples of cases which are clearly cheating and clearly not cheating.

**Cheating**

• Turning in someone else's work as your own (with or without his or her knowledge). Turning in a completely duplicated assignment is a flagrant offense.

• Allowing someone else to turn in your work as his or her own.

• Several people writing one program and turning in multiple copies, all represented (implicitly or explicitly) as individual work.

• Using any part of someone else's work without the proper acknowledgement.

• Stealing an examination or a solution from the instructor. This is an extremely flagrant offense.

**Not Cheating**

• Turning in work done alone or with the help of the course's staff.

• Submission of one assignment for a group of students if group work is explicitly permitted (or required).

• Getting or giving help on how to do something on the TOPS-20 operating system.

• Getting or giving help on how to solve minor syntax errors.

• High-level discussion of course material for better understanding.

• Discussion of assignments to understand what is being asked for.

The Computer Science Department faculty will not condone cheating. When cheating is suspected, instructors will take reasonable action to establish whether it actually occurred. If it has, the instructor will apply appropriate disciplinary policy.

The ordinary University penalty for cheating is failure in the course. (See Student Handbook). Penalties less severe than the recommended penalty may, when appropriate, be imposed. A list of possible disciplinary actions is given below.

**Actions within the course:**

• Negative credit for the assignment

• No credit for the assignment and loss of a letter grade for the course

• Makeup assignment over the same material; no credit

• Forced drop in the course

• Failure in the course

**Actions within the Computer Science Department:**

• Suspension from Departmental courses for a designated period

• Expulsion from Departmental courses

**Actions by the University:**

• Warning

• Probation

• Suspension from the University for a designated period

• Expulsion from the University

The following policies apply to all cases of cheating and plagiarism:

For a first offense, the penalty will always be more severe than the penalty for failing to turn in the assignment (or take the exam) in question.

For either repeated offenses or a flagrant offense by any student, the instructor shall either refer the incident directly to the University for action or assign a penalty no less severe than failure in the course.

In the event that a faculty member accuses a student of cheating and imposes a penalty, the student who believes that the accusation is unjust has the right to request that the charge of cheating be heard before the University Committee on Discipline.