:-) WHEN YOU GRADE THAT: USING E-MAIL AND THE NETWORK IN PROGRAMMING COURSES

David M. Arnow
Brooklyn College of CUNY

ABSTRACT: *We have been using network facilities and e-mail for submitting and responding to homework assignments, faculty-student communication and as a general means for providing instructional material (assignments, solutions, sample quizzes, etc.). The most important component of this is a homework programming assignment checker. These methods have been successful in alleviating a number of problems, including student difficulty in making faculty office hours, slow feedback on homework assignments, heavier teaching loads and larger classes. In addition, several unexpected benefits resulted as well.*

Keywords: *automated program checking, network classrooms, e-mail*

1. Introduction

Since the spring semester of 1992, several programming courses have made extensive use of e-mail and network facilities in connection with instructor-student communication. This has allowed us to overcome several problems related to the external demands placed on working students, limitations on computing facilities and faculty overload.

Although in retrospect the advantages of this arrangement seem obvious, a recent informal survey by the author of this paper of some 50 computer science faculty from as many institutions revealed that most made only incidental, if any, use of e-mail and other network facilities, even though such facilities were readily available. This is a bit odd, considering that fields outside of CS are beginning to use these facilities extensively (For an good review, see Brookshire[2]. Berman[1] describes e-mail and conferencing facilities in a computer literacy course setting).

One important component of our use of the network involves an automated homework program checker. There have been several reports such checkers in the computer science education literature (Kay[5], Burris and Darr[3], Iasscson and Scott[4], Reek[6]). Many of these were not network based and most importantly these have been used exclusively as a homework grading aid, not as a student learning tool as I describe below.

The purpose of this paper is, therefore, to alert the computer science education community to an extremely useful instructional device by presenting our experience with it.

2. The Context

The students are an ethnically diverse group of working-class students, many of whom are immigrants or members of minority groups. Many are considerably older than 22 years. A significant number (well over half) work more than 20 hours a week and/or have parental responsibilities. All are "commuters": our college has no dormitories.

The hardware facilities of the college are ample but not terribly diverse. We have a network of 35 Sun IPCs, a much larger Novell network of PS-2s and PCs, and access to an IBM 3090 mainframe. There are no experimental, supercomputer or parallel machines on campus.

The courses in question all used the Sun IPCs as their base machine. These courses are:

- a liberal arts "core requirement" computer literacy course for non-majors
- a second programming course for majors
- an undergraduate elective course on unix and network programming
- a masters level course on parallel programming

Each course made appropriately different and varying use of the facilities described below. In this paper, I focus on the second programming course for majors.

3. The Problems

There are several specific problems were overcome by using e-mail and other network facilities.

The discrepancy between faculty and student schedules. It is often impossible for faculty in my college to schedule office hours that all or even most of the students their classes can attend. This is due to the diversity of work and family responsibilities of the students.

The limits of available computing facilities. Some courses require or would benefit from either hardware that is not available on campus or expensive software of which only a single copy exists on the faculty's research network.

The slow turn-around of homework assignments. Many classes meet only twice a week. That means that the cycle of assigning a program, receiving a hardcopy source listing and output, and returning it to a student takes a minimum of a week. This long cycle makes it difficult to give the student an opportunity to redo his or her work in time to be relevant to the current class-work.

Increased faculty load. Because of budget cuts, faculty have been compelled to undertake higher teaching loads, both in the number of courses taught and in the number of students in their classes. This change makes it more difficult for faculty to comment critically on programming homework assignments or to ask students to resubmit problematic assignments.

It is true that all of these problems have political solutions (for example an increased budget could reduce faculty load and provide more equipment), but an instructor does not have the opportunity as an individual to achieve them. The solutions presented in this paper, though partial, are technological and available to the individual.

4. Using E-mail and the network

Starting in the spring of 1992, an increasing number of courses allowed or required students to submit programming assignments via e-mail. In my second semester programming course for majors, for example, students were required to submit all their assignments electronically and encouraged to e-mail their questions about the course (including assignments, lecture, readings) to me. I, in turn, "guaranteed" 24 hour response time. In addition, all homework assignments, homework solutions, sample quizzes, answers to sample quizzes and in-class exams, and classroom examples were posted on the network and available to the students exclusively through that medium.

Creating an e-mail culture. Not all students who are new to e-mail approach it initially with eagerness. This was especially true of the course for non-majors, but also for some of the students in the second semester programming course. To overcome this, on the first day of class (in all courses) the students are given the assignment of submitting a resume. By responding speedily and in a personal way to the information in these submissions, an e-mail rapport was rapidly built up between instructor and student. In many cases, students responded to the instructor's response, thereby initiating a conversation that would often last through the whole semester.

The incentive to make use of this environment remained strong as it became clear very quickly that the fastest (and sometimes only) way to get information (assignments, announcements, solutions) was through the network.

In addition to rapidly creating a willingness to use e-mail and network facilities, this had an additional effect. It would not have been possible to so quickly get to know so many students in an ordinary classroom setting or office hour setting, both because of time considerations and because students, like many people, are more reticent in person than in e-mail.

5. The homework checker

Being in e-mail contact with my class mitigated a number of the problems mentioned above. However, initially, rather than addressing the problem of my already heavy workload, it exacerbated it. Much of my e-mail reading was devoted to partially-completed assignments that would fail on input cases not tested by the student. Much of my e-mail writing was answering (and trying not to be testy!) such submissions.

To overcome this, a homework checking program was set up. The student executes a script that wraps any number of files of any nature and e-mails them to a special account that uses a .forward file to pipe the mail to the program checker. The checker creates a temporary testing directory in which the files it extracts from the mail are placed. Upon receipt of the student mail, the checker analyses the student's homework submission and within minutes sends a response to the student and instructor.

The checker's analysis of the students' submissions is driven by a "homework configuration file" that allows the instructor to specify:

- the particular files required in the homework submission
- the files that should be compiled
- the libraries that should be created
- the object files and libraries that should be linked to form executables
- the executables that should be run (and their input and arguments)
- the run-time resource limits (execution time, maximum output size) on the executables
- the files to compare with the output of the executables that were run
- filters to apply to the output prior to comparison (e.g. remove blanks, tabs, etc.)
- the exit status code of particular test runs of the executables
- any functions or other externals that the student code must or must not contain
- scope requirements for C modules (to enforce static storage class for non-exported functions and variables in particular modules)
- shell scripts to conduct special customized tests of the homework submission.

An example of such a configuration file is given in Appendix I.

If the homework submission passes all the required tests, the student receives a message that "preliminary checks have been passed" and that the homework is being forwarded to the instructor for further consideration. If any problem is detected, the checker immediately sends mail to the student indicating that a problem has been detected. Very little detail is given as to the nature of the problem. Messages such as "cannot link", "incorrect output", "abnormal termination", "missing file: something.c" are typical. Only one failure is indicated— the checker stops as soon as a single failure is discovered.

The instructor receives more information than the student. In cases of success, all source code (and any text files for that matter) from the student is sent, along with "nm" and "ar" listings of any object modules and libraries. In case of failure, the instructor is sent all of the above, as well as a more detailed reason for failure and "diff" listings of the student output vs. the correct output.

Receiving the failures is pedagogically crucial so that the instructor can readily see what problems the students are having and most importantly what concepts have *not* been successfully conveyed in class. These problems can be corrected in the next class or, more often, by e-mailing a broadcast to all the students in the class.

The treat, from my point of view, however, comes with the successfully completed homework assignments, I *know* that they compile and link properly and execute successfully on all the test cases that I provide. If, for example, an assignment calls for them to write their own string handling routines, I know that there are now strcpy() and strcmp() library calls buried. I read the students' programs focusing on overall design, documentation and usage and I respond accordingly by mail. When I do accept a program, it is by invoking a script that automatically checks for plagiarism (see Appendix II) and updates a roster on the network that the students can monitor.

6. Results

Overall and not surprisingly, the most important effect this environment had was in improved teacher-class communication. As soon as I become aware of misunderstanding concerning lecture material, an assignment or anything else, I post it on the network. Students who might otherwise, because of schedule or shyness, not have communicated concerns and questions were able to do so.

A considerable amount of class-time that would otherwise have been devoted to handing out and going over assignments and solutions is saved. Furthermore, students who missed or were late to a class could get their assignments accurately with no cost to the instructor.

The impact of the homework checker included both anticipated and unanticipated results. As expected, students appreciated the immediate feedback from the checker and I appreciated the ability to look at their programs from a design and documentation perspective, rather than one in which I tried to imagine whether the program worked or not. Additionally, many of the checking programs expected program output and input files to be in a very particular format, thus forcing *students to pay closer attention to specifications*. By the same token, I was forced to do the same, that is be clearer than in the past in my assignments. Any ambiguities were rapidly discovered and corrected as a result of this arrangement. Students learned not only to follow specs, but to read them critically.

Due to the homework checker and the generally improved environment for communication, it was possible to give more programming assignments and be stricter on time-limits. Being stricter on time-limits made it possible to post solutions to the problems earlier than would otherwise have been the case. This made it possible for the homework solutions to serve as a genuine supplement to class work.

One curious side-effect of being a commuter institution is that even those students who do have the capability of being on campus every day of the week often fall into a pattern where they, like their working and parenting counterparts, come only on days when they have classes. A gratifying aspect of the network environment is that it seems to reverse this trend, that is, a surprising number of students were coming in to work on their assignments on a daily basis,

7. Concerns

There are a number of concerns that are raised by this approach. The most serious of these is whether the approach is an instance of technology-fetishism, the inappropriate substitution of technology for need human action. Clearly this would be the case if this approach were used to replace direct professor-student contact. Less clear is its use in augmenting such contact in a climate of diminishing resources. Can solutions of this kind be used to justify further cutbacks?

The program checker, with its five minute (or less) response time raises the spectre of students mindlessly trying one thing after another to get the program accepted, and thus reduced to a condition of, as Dijkstra put it, "pavlovian slobber". If this turned out to be a problem it would be quite easy to modify the script to count submissions and reject or penalize after a threshold. In our experience, pavlovian slobber was infrequent and where it existed, it was generally the least of the students' problems.

Finally there are security concerns. Letting a student submit a program that is run on another account poses an obvious potential problem that should be approached with caution and the assistance of a systems administrator.

8. Conclusion

In summary, by making use of e-mail and network technology, the problems described earlier in this paper were alleviated:

- Although many students could not see the instructor during official office hours, communication in between class was extensive.
- Turn-around of homework assignments was much faster than in a course not employing e-mail.
- Faculty time spent on repetitive tasks relating to increased numbers of students was reduced.

Furthermore, class-time was spent more productively, greater attention was paid by the students to analyzing and precisely meeting specifications, more programming assignments could be given.

9. Availability

For information about obtaining the software described here, send mail to *arnow@sci.brooklyn.cuny.edu*. Currently it assumes a network of workstations running SunOS 4.X. It requires the presence of a special course account that contains scripts that are automatically executed by the students' login process. Plans are underway to set up a straight-forward installation script and make this environment available via ftp and the web.

10. References

[1] Berman, A. M.: "Class Discussion By Computer: A Case Study", Proceedings of the 23rd SIGCSE Technical Symposium, Kansas City (Mar. 1992).

[2] Brookshire, R.G.: Electronic Bulletin Boards as Teaching Tools in a University Setting", Proceedings of the Tenth Annual Research Conference Office Systems Research Association, Washington, DC (Mar., 1991).

[3] Burris, H., and M. Darr, "The PROGRAMS Package for Integrated Grading," Program in Computing, Department of Mathematics, University of California, Los Angeles (1988).

[4] Isaacson, P. C., and T. A. Scott, "Automating the Execution of Student Programs," SIGCSE Bulletin vol. 21 no. 2 (Jun. 1989).

[5] Kay, D.G.: "Don't Give Grades Without It: A Comprehensive Automated Grading Assistant For Student Programs", SIGCSE (1993).

[6] Reek, K. A., "The TRY System, or How to Avoid Testing Student Programs," SIGCSE Bulletin vol. 21 no. 1 (Feb. 1989).

Appendix I

Below is an example of a homework-checker configuration file. This was a homework in which the student had to write two source files and use them to create a library. Along with the configuration file, the instructor had to prepare the following files:

drivea.c, driveb.c	driver programs for the library routines
in1, in2	test input
out1, out2	desired output

In this case there are six sections, each started by a section name on a line by itself: NAMES, LIB, SCOPE, LINK, TEST, CONTAINS.

The NAME section of this particular file indicates that among other files, the student must submit tm.h, tm_time.c, tm_money.c and libtm.a.

The checker will compile all .c files automatically. including two driver files, drivea.c and driveb.c.

The LIB section instructs the checker to create a library "libtm2.a" from the object files that resulted from the compilation of tm_time.c and tm_money.c.

The SCOPE section indicates that there must be exactly 6 exported externals in both libraries.

The LINK section creates 3 executables from the driver files. Two of these use libtm.a, the library the student submitted. Another uses libtm2.a, the library that was created from the student's submitted source files.

In the TEST section, three tests are specified. For example, in the first test, the command

```
pa -d -x <in1
```

is executed, The standard output of that command is filtered through a script "squish" that reduces all sequences of spaces and tabs to a single space. The result is compared with a file out1. (More tests would actually be given, but their display would add little to this presentation.)

Finally, the CONTAINS section specifies that both libraries must contain the indicated list of externals "tm_24, tm_ctod, ...".

Appendix II: Addressing plagiarism

Every technology has its unfortunate aspects and the networked classroom is no exception. Just as instructor-student and student-student communication is facilitated, so is the ability for students to mindlessly copy each others programs.

To counter this, a plagiarism detector was written. It carries out a number of transformations on each student's code and then makes pairwise comparisons. First, the program is separated into two files, one containing only the comments, the other containing the code without the comments. Then every external definition and declaration is joined onto one line a piece, spaces and tab sequences are reduced to single spaces and the resulting file is sorted by line length. From this file, two additional files are created: one containing only the identifiers, the other only the keywords and operators (including all special characters and symbols). The pairwise comparison is based on the number of lines resulting from the diff utility in Unix— smaller numbers suggesting plagiarism.

The detection of similarity is imperfect and only calls to the instructor's attention possibilities of plagiarism. Because I hate "playing cop", I examine only the most egregious-looking results. I do make sure to pursue a few of these *early* in the semester. The well-advertised fact that the instructor possesses this sort of tool helps deter some of the worst abuses.

Another way that plagiarism is addressed in the course is to individualize assignments. This does not mean giving completely different assignments, but rather where possible to parameterize assignments based on the students unix user id. For example, in one assignment, the students are asked to read standard input, encode it and write the result to standard output. The particular encoding that each student is given is unique, making it impossible to simply copy another student's code. (A thoughtful kind of plagiarism is certainly possible here— but at least it would require understanding someone else's work before making the necessary modifications!)

A Configuration File

```
NAMES
tm.h tm_time.c tm_money.c libtm.a

LIB
libtm2.a tm_time.o tm_money.o

SCOPE
libtm2.a 6
libtm.a 6

LINK
pa drivea.o libtm.a
pa2 drivea.o libtm2.a
pb driveb.o libtm.a

TEST
pa arg -d arg -x stdin in1 comp1 out1 filter squish
pa2 arg -d arg -x stdin in1 comp1 out1 filter squish
pb stdin in2 comp1 out2 filter squish

CONTAINS
libtm.a  nm tm_24 tm_ctod tm_ctoc tm_pND tm_pED
libtm2.a nm tm_24 tm_ctod tm_ctoc tm_pND tm_pED
```