Experience Using ”MOSS” to Detect Cheating On Programming Assignments

Kevin W. Bowyer and Lawrence O. Hall

Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620-5399

kwb@csee.usf.edu and hall@csee.usf.edu

Abstract – *Program assignments are traditionally an area of serious concern in maintaining the integrity of the educational process. Systematic inspection of all solutions for possible plagiarism has generally required unrealistic amounts of time and effort. The “Measure Of Software Similarity” tool developed by Alex Aiken at UC Berkeley makes it possible to objectively and automatically check all solutions for evidence of plagiarism. We have used MOSS in several large sections of a C programming course. (MOSS can also handle a variety of other languages.) We feel that MOSS is a major innovation for faculty who teach programming and recommend that it be used routinely to screen for plagiarism.*

1. Introduction

Probably every instructor of a programming course has been concerned about possible plagiarism in the program solutions turned in by students. Instances of cheating are found, but traditionally only on an ad hoc basis. For example, the instructor may notice that two programs have the same idiosyncrasy in their I/O interface, or the same pattern of failures with certain test cases. With suspicions raised, the programs may be examined further and the plagiarism discovered. Obviously, this leaves much to chance. The larger the class, and the more different people involved in the grading, the less the chance that a given instance of plagiarism will be detected. For students who know about various instances of cheating, which instances are detected and which are not may seem (in fact, may be) random.

A policy of comparing all pairs of solutions against each other for evidence of plagiarism seems like the correct approach. But a simple file `diff` would of course detect only the most obvious attempts at cheating. The standard “dumb” attempt at cheating on a program assignment is to obtain a copy of a working program and then change statement spacing, variable names, I/O prompts and comments. This has been enough to require a careful manual comparison for detection, which simply becomes infeasible for large classes with regular assignments. Thus, programming classes have been in need of an automated tool which allows reliable and objective detection of plagiarism.

2. What is MOSS?

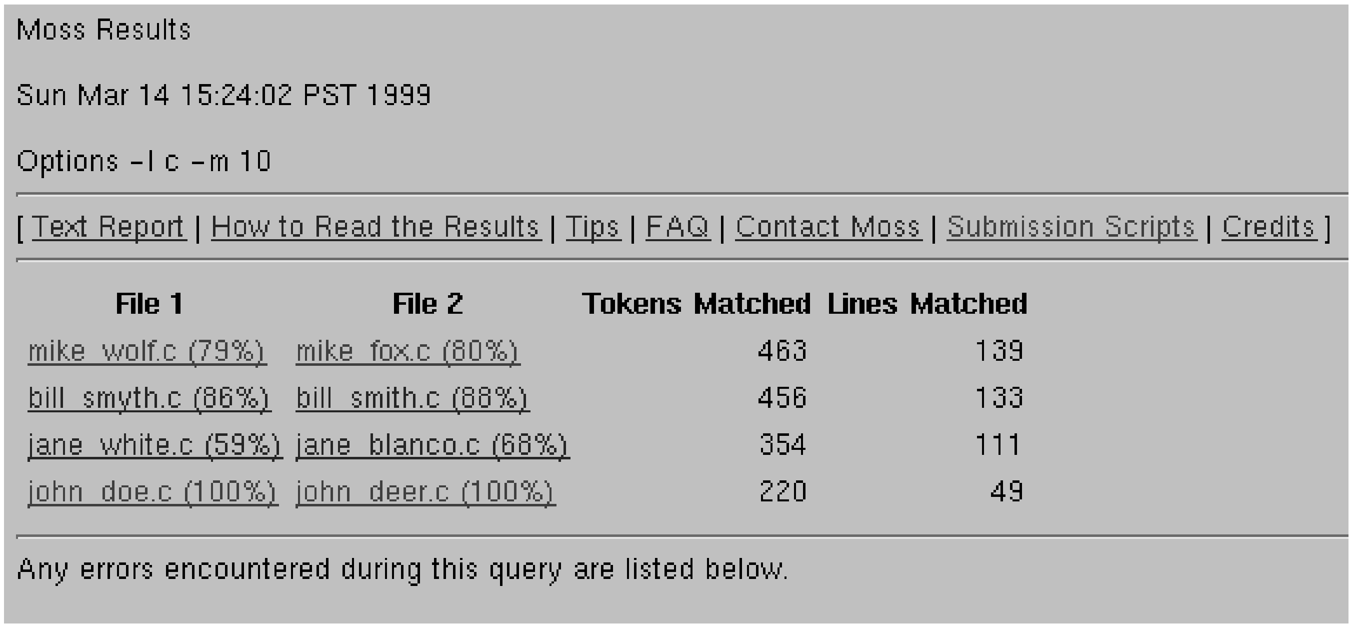
MOSS stands for ”Measure Of Software Similarity.” It is a system developed in 1994 by Alex Aiken, associate professor of computer science at UC Berkeley. MOSS makes it possible to objectively and automatically check all programs solutions for evidence of copying. MOSS works with programs written in C, C++, Java, Pascal, Ada and other languages.

`www.cs.berkeley.edu/˜aiken/moss.html` is the web page for brief summary information about MOSS. The automated mail server for requests for MOSS accounts (needed to use the MOSS server) is `moss-request@cs.berkeley.edu`. A mail to this address will result in a reply mail which contains a `perl` script which can be installed on the instructor’s system. Or, the latest MOSS script can be down-loaded from `www.cs.berkeley.edu/˜moss/general/scripts.html`. MOSS should run on UNIX systems which have `perl`, `uuencode`, `mail` and either `zip` or `tar`. The installed script will be referred to as the command `moss`. A comment in the script states – “Feel free to share this script with other instructors of programming classes, but please do not place the script in a publicly accessible place.” Accordingly, and in deference to possible copyright issues, we do not reproduce any of the script in this paper.

Program files to be submitted to MOSS can be in any subdirectory of the directory from which the `moss` command is executed. For example, to compare all programs in the current directory on a UNIX system, assuming that the programs are written in C and that `moss` is in the current directory, the following command could be used:

```
moss -l c *.c
```

The system allows for a variety of more complicated situations. For example, it allows for a “base file.” The base file might be a program outline or partial solution handed out by


Moss Results

Sun Mar 14 15:24:02 PST 1999

Options -l c -m 10

[Text Report | How to Read the Results | Tips | FAQ | Contact Moss | Submission Scripts | Credits]

File 1	File 2	Tokens Matched	Lines Matched
mike_wolf.c (79%)	mike_fox.c (80%)	463	139
bill_smyth.c (86%)	bill_smith.c (88%)	456	133
jane_white.c (59%)	jane_blanco.c (68%)	354	111
john_doe.c (100%)	john_deer.c (100%)	220	49

Any errors encountered during this query are listed below.

Figure 1: Opening web page of MOSS results.

the instructor. The degree of similarity between programs which is traceable to this base file should be factored out of similarity rankings of the programs. Also, MOSS allows for the programs that are compared to be composed of sets of files in different directories.

The `moss` command results in the programs being sent to the MOSS server at Berkeley. When the results are ready, an email is sent back to the login name that invoked the `moss` command. The email gives a web page address for the results. In our experience, sending approximately 75 to 120 C programs of a few hundred lines each, results of the similarity checking are available the same day. The return email from the similarity checking currently states that the results will be kept available for 14 days on the MOSS server.

Aiken does not supply explicit information about the algorithm(s) that MOSS uses to detect cheating. In keeping with his desire that the inner workings be confidential, we do not speculate on the algorithms involved.

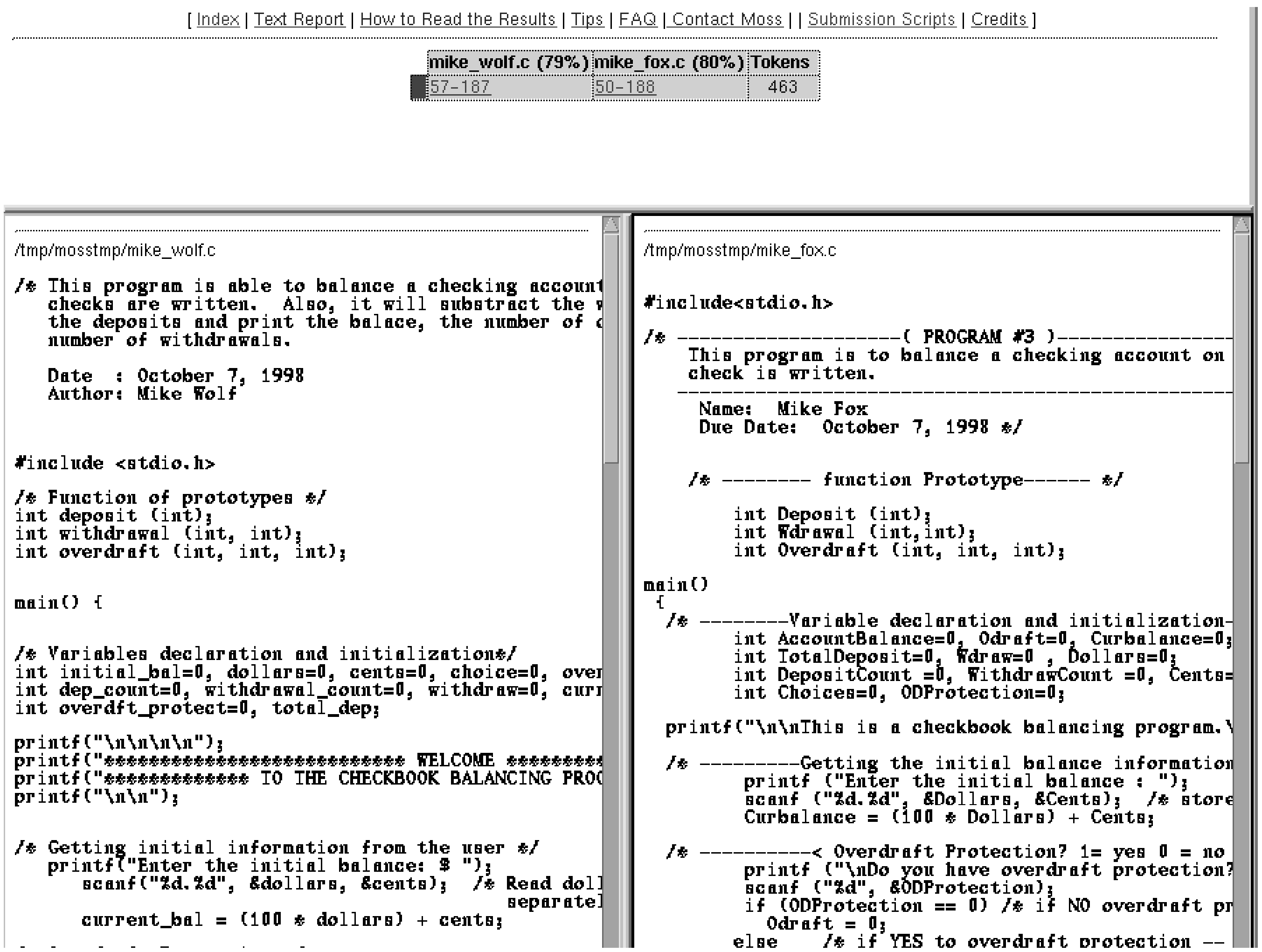
3. Plagiarism Detected by MOSS

Figure 1 shows the MOSS results web page for some actual program pairs involved in cheating incidents in one of our classes in the Fall semester of 1998. The file names have been changed to hide the individuals' identities. The results page lists pairs of programs which were found to have substantial similarity. For each such pair, the results summary lists the number of tokens matched, the number of lines matched, and the percent of each program source that is found as overlap with the other program. In our experience, with C programs of a few hundred lines, anything over 50% mutual overlap is a near-certain indication of plagiarism. However, our experience is that accusations of plagiarism should not be made “mechanically” solely on the basis of MOSS ratings. It is important for the instructor to consider the similar sections of the programs in the context of how the course is taught.

MOSS makes it easy to examine the corresponding portions of a program pair. Clicking on a program pair in the results summary brings up side-by-side frames containing the program sources. See Figure 2 for an example. This page allows scrolling through the program sources to read each and consider the similarities. It is also possible to click on a line range listed under the program name and jump straight to that section. For example, clicking on ”57-187” and ”50-188” in Figure 2 brings up the matching sections as in Figure 3. The similar sections are marked with a dot at the start, and are given color-coded highlighting. The plagiarism in Figure 3 is obvious. Variable names and spacing of statements have been changed, but that is about all that is different.

MOSS just as easily uncovers more sophisticated attempts at cheating. Multiple distinct similar sections separated by sections with differences are still found and given color-coded highlighting. Functions may be given different names, and placed in a different order in the program and they are still matched up. Students who have changed all variable names, the statement spacing, the comments, the function names and the order of appearance of the functions stand out just as readily as students who turn in exact duplicate programs!

To summarize, the actual detection of plagiarism on program assignments is made relatively painless and simple us-



[Index | Text Report | How to Read the Results | Tips | FAQ | Contact Moss | | Submission Scripts | Credits]

mike_wolf.c (79%)	mike_fox.c (80%)	Tokens
57-187	50-188	463

/tmp/mosstmp/mike_wolf.c

```
/* This program is able to balance a checking account
   checks are written.  Also, it will substract the w
   the deposits and print the balace, the number of c
   number of withdrawals.

   Date  : October 7, 1998
   Author: Mike Wolf



#include <stdio.h>

/* Function of prototypes */
int deposit (int);
int withdrawal (int, int);
int overdraft (int, int, int);


main() {


/* Variables declaration and initialization*/
int initial_bal=0, dollars=0, cents=0, choice=0, over
int dep_count=0, withdrawal_count=0, withdraw=0, curr
int overdft_protect=0, total_dep;

printf("\n\n\n\n");
printf("**************************** WELCOME *********
printf("************** TO THE CHECKBOOK BALANCING PROC
printf("\n\n");


/* Getting initial information from the user */
   printf("Enter the initial balance: $ ");
      scanf("%d.%d", &dollars, &cents);  /* Read doll
                                            separatel
      current_bal = (100 * dollars) + cents;
```

/tmp/mosstmp/mike_fox.c

```
#include<stdio.h>

/* --------------------( PROGRAM #3 )---------------
    This program is to balance a checking account on
    check is written.
   ------------------------------------------------
     Name:  Mike Fox
     Due Date:  October 7, 1998 */


    /* -------- function Prototype------ */

        int Deposit (int);
        int Wdrawal (int,int);
        int Overdraft (int, int, int);

main()
 {
  /* --------Variable declaration and initialization-
        int AccountBalance=0, Odraft=0, Curbalance=0;
        int TotalDeposit=0, Wdraw=0 , Dollars=0;
        int DepositCount =0, WithdrawCount =0, Cents=
        int Choices=0, ODProtection=0;

  printf("\n\nThis is a checkbook balancing program.\

  /* ---------Getting the initial balance information
         printf ("Enter the initial balance : ");
         scanf ("%d.%d", &Dollars, &Cents);  /* store
         Curbalance = (100 * Dollars) + Cents;

  /* ----------< Overdraft Protection? 1= yes 0 = no
         printf ("\nDo you have overdraft protection?
         scanf ("%d", &ODProtection);
         if (ODProtection == 0) /* if NO overdraft pr
           Odraft = 0;
        else    /* if YES to overdraft protection --
```

Figure 2: Side-by-side frames of suspect programs.

ing MOSS. Once the MOSS script is installed, plagiarism detection is just a matter of the faculty member invoking a one-line command, waiting a short time for an email from the MOSS server, and then browsing a web page that has color-coded the corresponding sections in pairs of suspect programs. The real difficulties for the faculty member arise in processing the cases of plagiarism through the grading and appeals process.

Here is how we handled the incidents of plagiarism. Where the professor feels that cheating is likely, an e-mail is sent to the students involved to request a written summary of any information that might be important in understanding what has happened. See Figure 4 for an example of this email. In a small portion of the cases, this first e-mail elicited a confession from one student that they somehow copied the other student's program. Copying may occur through lost or stolen diskettes, discarded printouts, unprotected files, or other means. In cases where one student copied another student's program without their knowledge, only the one student who copied the program received an "F." In cases where it was clear that one student gave their program to another student, each student received an "F."

In an additional portion of the cases, the first response to the e-mail was a denial, but then a confession came before the scheduled meeting with the professor. Of the cases which went as far as a meeting with the professor, laying out the two program listings and outlining the similarities resulted in a confession in all but one case. In this case, two students admitted talking together about the program and agreed that the programs were strikingly similar, but insisted that they did not cheat. This insistence was maintained even when it was pointed out that the programs contained non-functional elements of similarity: un-needed curly brackets, `const` values passed to functions and not used, and so on. In this case, both students were assigned an F.

The USF handbook provides for several levels of appeal if students are unhappy with a decision in grading. In our experience, about half the plagiarism incidents are not appealed.

```
From: The Professor
To: Student_1, Student_2
Subject: Similar solutions on assignment N.

This is about the solutions for assignment N.
The "copy checker" utility suggested that
there was enough similarity in your two
solutions that they should be looked at.
I have looked at them, and there is some
unusual and striking similarity.

I would like for each of you to send me an
email, or leave me a written note, with any
information that you feel may be relevant
to this situation.  Then, please come to
see me during office hours on Wednesday.

Thank you.

The Professor
```

Figure 4: Example of initial e-mail to students.

The remaining half are appealed at the Department level, and only a small percentage continued appeals to higher levels. Most appeals are not on the basis of denying that plagiarism occurred, but arguing for a lesser penalty. The most common premise for the argument was simply that an "F" for the course was too harsh, even if it was specified in the syllabus. Additional premises sometimes offered were that it would hurt the student's cumulative GPA, chances of getting into grad school, and/or chances of getting a desired job.

Each cheating incident typically requires several hours of the professor's time. Examining the MOSS comparison results is a small part of this. Additional time is spent communicating with the students, documenting the incident and, in some instances, meeting with appeals committees.

4. Discussion

Our Department policy calls for an "F" for the course as a result of a first cheating incident. A student who cheats a second time is typically dismissed from the Department and possibly also from the College of Engineering. (We did have one student caught in both Fall '98 and Spring '99.) Students are informed of the policy at the first meeting of each course, both in the syllabus and a separate handout.

We routinely used MOSS with all program assignments in two sections of a Program Design course in the Fall of 1998 and another section in Spring of 1999. This particular course is used as part of a "gate" for entry to the Department. Students must achieve a certain GPA in three specified gate courses in order to major in the Department.

In the first semester we used MOSS, in one section of about 75 students, a total of ten received an "F" for plagiarism. In a section of over 140 students the next semester, nine received an "F" for plagiarism. Thus it seems that the rate of detected plagiarism decreased. In the first semester, students may not have initially believed the warnings that all programs were checked for plagiarism. It is possible that, as word spread, some plagiarism was prevented by the knowledge that all programs are carefully checked. However, there is another less-pleasant possible interpretation.

MOSS is a wonderful tool, and a major advance for faculty who teach programming courses. However, by nature, it can only detect cheating that is evidenced in the program solutions turned in. If a student has a person who is not in the course write the solution for them, it will not normally be detected. This point was brought home to us by one incident. In this incident, two students whose programs were nearly identical insisted that they had not cheated from each other. Further investigation revealed that both had obtained their program outline from the same third person. This third person was not in the course, and in fact was not currently a student at the university.

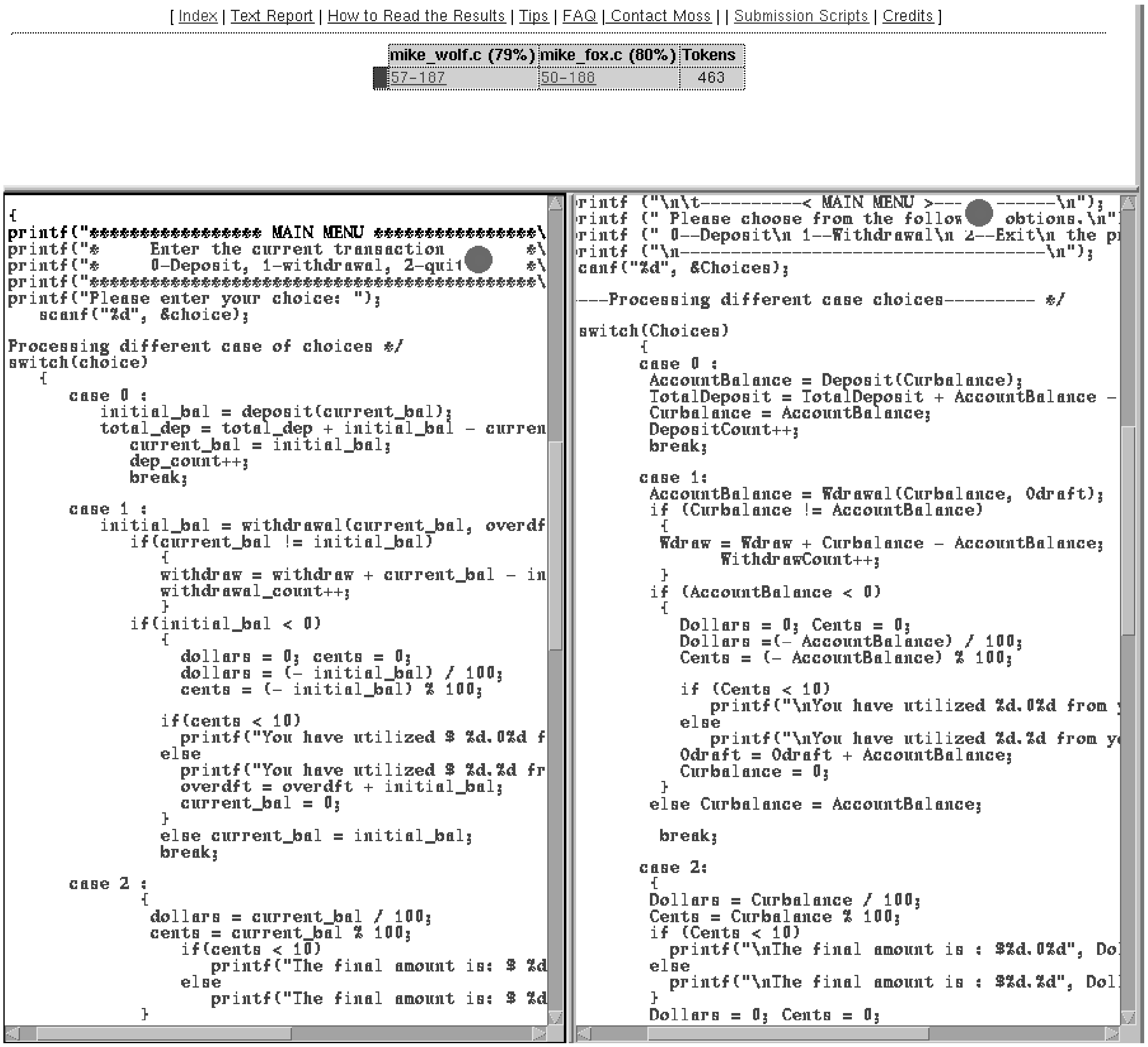
We suspect that the "ghost author" phenomenon is more widespread than just the incidents that we uncover. We have noted the phenomenon of students who consistently receive near-perfect scores on program assignments yet also consistently receive low scores on in-class quizzes which require writing short program segments. We have adjusted our grading scheme for the class to reduce the contribution of program assignment grades to the final grade. Also, we have seriously considered possible grading schemes in which only work that is done in class would count toward the final grade.

Another incident provides a warning against too-quick accusations. Two students had very similar program solutions. However, after investigation, it appears that both had independently discovered the same way to adapt an example in the textbook into a solution for the assignment. Thus, their programs were constrained to be highly similar by design. In this case, no accusation of plagiarism was made.

Professor Aiken is to be congratulated on having produced a very nice system that fulfills a real need of programming instructors everywhere. We use MOSS routinely now, as do essentially all instructors in all programming courses in our Department.

References

[1] Kevin W. Bowyer, *Ethics and Computing*, IEEE Computer Society Press, 1995.


[Index | Text Report | How to Read the Results | Tips | FAQ | Contact Moss | | Submission Scripts | Credits]

mike_wolf.c (79%)	mike_fox.c (80%)	Tokens
57-187	50-188	463

```
{
printf("****************** MAIN MENU *****************\
printf("*     Enter the current transaction           *\
printf("*     0-Deposit, 1-withdrawal, 2-quit         *\
printf("**********************************************\
printf("Please enter your choice: ");
   scanf("%d", &choice);

Processing different case of choices */
switch(choice)
   {
      case 0 :
         initial_bal = deposit(current_bal);
         total_dep = total_dep + initial_bal - curren
            current_bal = initial_bal;
            dep_count++;
            break;

      case 1 :
         initial_bal = withdrawal(current_bal, overdf
            if(current_bal != initial_bal)
               {
               withdraw = withdraw + current_bal - in
               withdrawal_count++;
               }
            if(initial_bal < 0)
               {
                 dollars = 0; cents = 0;
                 dollars = (- initial_bal) / 100;
                 cents = (- initial_bal) % 100;

               if(cents < 10)
                 printf("You have utilized $ %d.0%d f
               else
                 printf("You have utilized $ %d.%d fr
                 overdft = overdft + initial_bal;
                 current_bal = 0;
               }
               else current_bal = initial_bal;
               break;

      case 2 :
             {
              dollars = current_bal / 100;
              cents = current_bal % 100;
                 if(cents < 10)
                    printf("The final amount is: $ %d
                 else
                    printf("The final amount is: $ %d
             }
```

```
rintf ("\n\t----------< MAIN MENU >---    ------\n");
rintf (" Please choose from the follow   obtions.\n"
rintf (" 0--Deposit\n 1--Withdrawal\n 2--Exit\n the pr
rintf ("\n-------------------------------------\n");
canf("%d", &Choices);

---Processing different case choices--------- */

switch(Choices)
      {
      case 0 :
       AccountBalance = Deposit(Curbalance);
       TotalDeposit = TotalDeposit + AccountBalance -
       Curbalance = AccountBalance;
       DepositCount++;
       break;

      case 1:
       AccountBalance = Wdrawal(Curbalance, Odraft);
       if (Curbalance != AccountBalance)
        {
        Wdraw = Wdraw + Curbalance - AccountBalance;
              WithdrawCount++;
        }
       if (AccountBalance < 0)
        {
          Dollars = 0; Cents = 0;
          Dollars =(- AccountBalance) / 100;
          Cents = (- AccountBalance) % 100;

          if (Cents < 10)
             printf("\nYou have utilized %d.0%d from y
          else
             printf("\nYou have utilized %d.%d from yo
          Odraft = Odraft + AccountBalance;
          Curbalance = 0;
        }
       else Curbalance = AccountBalance;

        break;

      case 2:
       {
       Dollars = Curbalance / 100;
       Cents = Curbalance % 100;
       if (Cents < 10)
         printf("\nThe final amount is : $%d.0%d", Do
       else
         printf("\nThe final amount is : $%d.%d", Dol
       }
       Dollars = 0; Cents = 0;
```

Figure 3: Side-by-side frames, cued to matching sections.