MAKING PROGRAM GRADING EASIER (but not totally automatic)

J. Archer Harris
James Madison University
MSC 4103
Harrisonburg, VA 22807
(540) 568-2774
ccsc04@jah.cs.jmu.edu

Elizabeth S. Adams
James Madison University
MSC 4103
Harrisonburg, VA 22807
(540) 568-1667
adamses@jmu.edu

Nancy L. Harris
James Madison University
MSC 4103
Harrisonburg, VA 22807
(540) 568-8771
harrisnl@jmu.edu

ABSTRACT

Many instructors believe that the introductory computer science course is one of the most difficult in the curriculum to teach. One of the biggest challenges is in providing meaningful feedback rapidly to students. Our philosophy requires that faculty be involved with all aspects of evaluating student work. This paper describes the pluses and minuses of our pedagogical approach and our tool that helps this faculty to meet the challenge.

CATEGORIES AND SUBJECT DESCRIPTORS

K.3.2 [Computing Milieu, Computers and Education]: Computer and Information Science Education, *Computer Science Education*.

1. INTRODUCTION

Teaching the course in which entering computer science students first learn to write programs to solve problems is perhaps a greater challenge than teaching any other computer science course. As teachers, we must begin the process of helping the students to learn a whole new way of thinking. We want to shape their untrained minds into the disciplined problem solving minds of the computer scientist. This paper will discuss tools used in our introductory programming course that assist in the program evaluation process.

As early as 1965, papers on the automatic grading of student programs appeared [4]. They provide an interesting mirror of the progression of languages taught in the intervening years: Algol [4], FORTRAN [3,16], Pascal [15,17], C [1], C++ [5], Java [18]. Some are philosophical and merely present the rationale for or results of automatic grading assistance [10,12]. Others are practical and focus their attention on specific areas such as: automatic submission [11], detecting plagiarism [1], compiling statistics on execution times and loop executions [8], program testing [14] or assigning a grade [6,13,5]. Still others require significant faculty interaction [7,9], student interactivity [2], or lack implementation detail. Our approach differs from these other tools in that it coordinates electronic submission, paper submission, and electronic testing, and guarantees that the same files are used in all three processes. This paper focuses on our rationale for creating and using an automatic submission and testing system and explains the pluses and minuses of our approach.

2. THE COURSE ENVIRONMENT

Our CS1 classes are all taught by full-time faculty members. Our class size is a maximum of twenty-five students and the course is a four-credit hour course. The course has two fifty minute lectures per week and two seventy-five minute labs. The lab assignments are intended to be completed within the lab period although students are frequently given additional time to complete them. In addition to the lab assignments, students are expected to solve six or seven programming problems on their own outside of lab time. Although we do use undergraduates who have done well in the course as teaching assistants, they do not do any grading for the course. All of the grading is the responsibility of the faculty member teaching the course.

The language in our introductory course is Java and programs are compiled and executed using Sun Microsystem's JDK. Students are provided with their own individual disk space on a University maintained Novell file server. The course laboratory contains Windows computers connected to the campus network. Students are provided with an account on the Novell file server and with an account on a small cluster of Linux servers. The Novell file space is accessible from the laboratory computers, from the Linux servers, and from computers in the students' dorm rooms. The Linux servers are accessible both on and off campus via ***telnet*** or ***ssh.***

The course incorporates the following five themes in the presentation of the course material: algorithmic thinking, the software engineering approach, professional ethics, coding practice, and reading specifications. Programming assignments concentrate on small but complete applications consisting of one or more classes. We require this because it has been our experience that allowing students to write only partial applications does not teach them what a program is or how to implement and test one.

Although students must be aware of all phases of the software engineering life-cycle, the introductory course focuses on implementing specifications with well-written code by a specified deadline, not on higher level design issues. Grading of assignments is based on program quality, program correctness, and timeliness of submission.

For beginning programmers, we believe it is important to have assignments with unambiguous specifications that must be met exactly. Any lack of precision, either in the assignment's specification or in the standard of evaluation, allow students some degree of latitude in substituting what they can accomplish as opposed to what they should accomplish. Particularly for beginning programmers for whom literal interpretation of instructions and precise requirements are foreign notions, the precision required of the students and their programs should match the precision with which the computer operates. While some might argue it is overly strict to require a program’s output to match a standard so closely that not even an extra blank line or space is permitted, we would argue that ANY difference should not be permitted because it confuses students to have some differences judged as acceptable while others are not.

Any evaluation process must address how it will handle assignments that are not completed on time. Particularly in an introductory course, we believe it is important to provide some leeway to the students. Such leeway can by granted by providing partial credit for either late or faulty programs. Given the emphasis in this course on writing code that generates precise output, we provide that leeway only by giving partial credit for late submissions; to receive any credit a program must meet its specifications exactly. The lateness policy calls for a late penalty of 10 points (out of 100) for each day the assignment is late.

For our evaluation of a student programming assignment to contribute most effectively to the learning process, it must be accomplished quickly. Ideally, each evaluation should be returned before the next program is due. To facilitate the grading process, we use two tools to automate the programming assignment submission and evaluation process: ***submit*** and ***progtst***. The ***submit*** utility provides a mechanism for sending to the instructor all required materials in electronic form with an accurate time-stamp. The ***progtst*** utility works in conjunction with ***submit*** to test programs when they are submitted. The submission fails if ***submit*** determines the submitted program failed to meet the specifications. The student receives immediate feedback that the submission failed so they know they must continue work on the assignment.

These tools only evaluate the objective factors of the assignment; program correctness, number of submissions, and the time of submission. The instructor is responsible for the subjective component of a program’s evaluation, the quality of the code.

3. THE SUBMISSION PROCESS

Students typically develop their program files on the Novell file server while working in the Windows environment, either on their own computer or on one of the laboratory computers. After they have successfully compiled their programs and have tested them until they believe that they are logically correct, they telnet to one of the Linux servers. The ***submit*** and ***progtst*** utilities are only implemented in the Linux environment. The lab computers are loaded with the Windows-based ***putty*** program to provide telnet capability (in addition to the native DOS-based ***telnet*** program that is supplied with Windows).

Once logged in to a Linux server, students can "mount" their Novell file space into the Linux file system. If their program files do not exist on the Novell file server, they can FTP their programs to the Linux systems.

The ***submit*** program is used to submit assignment files to the instructor. The ***submit*** program begins by displaying a menu of choices indicating to which faculty member and for which course the files will be submitted (the course-id). After choosing the course-id, the student is provided with a menu of assignments available for submission, specific to that course-id. After the assignment is specified, the student is prompted for the names of the files to be submitted. After specifying the file name(s), the student then must respond to an honor pledge declaration and indicate they have received no unauthorized assistance in completing the assignment. If they fail to do so, the submit aborts.

If the instructor has specified a due date and a schedule of penalties for late submissions, a "late penalty" (if any) is calculated based on the time of submission. The Linux servers have their clocks synchronized to the national time standard so the recorded time is accurate to within a few seconds.

The submitted files are copied to a directory that is created to store the submitted material. The created directory and its contents all belong to the instructor and are not accessible to the student. Since each submission results in a new directory being created, subsequent submissions do not overwrite earlier ones.

The ***submit*** program then compiles all submitted source code files (the instructor's copy). If the program fails to compile, an error message is output and the submission aborts. If the program compiles, the executable that is produced is supplied to the ***progtst*** program. For a number of test cases, the actual output generated by the user's executable is compared to the correct output. If the student's executable fails a test, an error message is output and the submission aborts. If the executable passes all the tests, a success message is output and the instructor's directory containing the submission is marked as correct.

In addition to the onscreen messages, ***submit*** generates a submission report. A copy of the report in text format is created both in the instructor's submission directory and in the working directory of the student. The student is also provided with a PDF copy of the report. The report provides the submission information (student name, account, date, assignment, late penalty, honor pledge, etc) and a listing of the submitted source files. If the program compiled correctly, a summary of the ***submit*** output is included. If a test failed, detailed information on that test is included. This information includes a copy of the input supplied to the program, the actual output the program generated, and the correct output the program should have generated. A statement indicates the first place the actual output differs from the correct output. Since submission aborts as soon as any test fails, the report can contain information on only one failed test. Figure 1 contains a copy of the cover page of a submission report.

```
SUBMIT REPORT
User: studenac
Name: Any CS139 Student
Course: cs139h
Assignment: pa3
Version: 3
Date: Wed Nov 12 01:36:20 PM 2003
Files: ThreeN.java ThreeN_Helper.java

Late Penalty: 20 (submitted after 11/11/2003,22:05)

PDF-spec: 90/99
Honor Pledge: I have NOT received unauthorized assistance

Compiling ...
j139c ThreeN.java ThreeN_Helper.java

/aux/bin/progtst -C -v -f test-report pa3

*** PROGRAM TEST ***
Assignment: pa3
Test Dir: /fs/home-f/harrisnl/cs139.d/assign.d/pa3.d
Test Cnt: 6
Test 01: (pa3 <tst-01 )
Exec prog...; output: right; SUCCESS.
Test 02: (pa3 <tst-02 )
Exec prog...; output: right; SUCCESS.
*** TEST RESULT FOR ‘pa3’: SUCCESS ***
```

Figure 1: Submit report cover page

To receive full credit, programs passing all tests must be submitted by the due date and a copy of the PDF version of the report file must be printed and turned in at the next class meeting. Submissions that do not pass all tests are not considered for a grade. If an assignment has been submitted more than once, the printed report file determines which submission is to be used in determining the grade.

```
<--- Test 01: input data follows (WITHOUT line numbers) --->
inches
12
2
<--- Test 01: correct output follows (WITH line numbers)--->
    1
    2
    3   What units are your measurements in?
    4
    5   Enter the length as a whole number:
    6
    7   Enter the width as a whole number:
    8   The area is: 24 square inches
<--- Test 01: your incorrect output follows (WITH line numbers)--->
    1
    2
    3   What units are your measurements in?
    4
    5   Enter the length as a whole number:
    6
    7   Enter the width as a whole number:
    8   The area is: 24
<--- First difference after line 8, char 15: Expected=' ', Observed='\n'. --->
<--- Test 01: model err-output is empty --->
<--- Test 01: your err-output is empty --->
```

Figure 2: Submit error report example

4. WHY WE USE *submit* AND *progtst*

Using ***submit*** and ***progtst*** provides a number of positive benefits for both the student and the instructor. Pluses from the student perspective include the following:

S1. The tool compiles and executes the student's submitted Java program. The student receives immediate feedback on their monitor screen and in a detailed submission report.

S2. The submission report is in the form of both a text file and PDF file, so the text file can be read within a telnet session and the "prettier" PDF format can be printed or viewed in a graphical interface.

S3. If an execution fails to successfully pass a test, the submission report provides detailed information about the test input data, the correct output, and the incorrect actual output along with an error message telling the student the exact place where the output data did not conform to specifications. We feel this is an important aid for beginning students who are still learning how to deal with precise specifications.

S4. Because the student sees the input data that resulted in errors, they have the opportunity to learn something about the nature of good test data.

S5. All submissions are electronically dated and a formal document is produced that specifies the time of submission. If a program is submitted late, immediate feedback is delivered indicating the penalty to be assessed.

S6. Programs can be returned more promptly with more meaningful feedback about coding style and the use of appropriate programming techniques and adherence to standards.

S7. The specifications of programming assignments must be very precise with this system. Students are not left to guess about how a requirement should be met. Because of the way the output is evaluated, faculty must be much more precise with their requirements specification than they might otherwise be. In other words, it requires the faculty to exhibit the discipline and rigor they expect of their students.

Pluses from the faculty perspective include the following:

F1. Students learn to read specifications more precisely.

F2. More time can be spent on the evaluation of technique and style and more meaningful feedback can be provided to the students.

F3. The source code and results of every submission are kept in faculty directories in an organized structure. A faculty member can easily find student submissions. Queries such as how many correct submissions have been received to date can be easily answered.

F4. Having an archive of all submissions can provide a useful record. For example, if unauthorized collusion is suspected between two students, patterns of submission can show who did the work and who may have copied, stolen, or used the particular program. These source code files may also be used with tools to detect collusion.

F5. There are no disputes over dates and times of submission because it is all recorded electronically.

F6. Faculty do not have to deal with the issues that are inherent in e-mail or floppy disk submission, namely students forgetting attachments, students saying they submitted when they did not, etc. If a submission is done properly in this system, the student has the record of that work as does the faculty member.

F7. Faculty do not have to spend time determining partial credit for partially running programs. By requiring students to meet exact output specifications, they will either satisfy the standard or they will not be able to successfully submit their programs. The students quickly learn to write solid code that meets specifications.

F8. Faculty receive paper copies of the PDF report containing all required information in a consistent format. The cover page contains all necessary summary information about the student and submission. All reports contain source code listings using the same font size and layout. Source listings are formatted to include line numbers and headers with file names.

F9. In cases where there were multiple submissions, there is no ambiguity over which submission is to be evaluated. Responsibility for choosing the appropriate report lies with the student.

5. STUDENT SURVEY RESULTS

Two student surveys were conducted using two distinct populations of students. One survey tested the population of students currently enrolled in the introductory programming course who had used ***submit*** for their first three major programming assignments. The second survey tested the population of students who successfully completed the introductory programming course using ***submit*** and who are now taking the advanced programming course. The results are provided in Appendix A, and the analysis provided below.

5.1 Introductory Programming Survey

The survey consisted of five opinion questions and two open ended questions. The opinion questions asked the students to quantify their experience with submit on a 4 point scale with 1 indicating a negative response to the question and a 4 indicating a positive response.

A summary of the questions and responses from 28 students indicates that in general, students liked the use of submit for class assignments. To the question “Submit only allows programs to be submitted if they have been verified as working correctly on all the test cases. How much did you like knowing that your program, which had been submitted for evaluation had been verified as working correctly?”, 89% of the respondents replied with a 3 or 4 response indicating that they liked the fact that they knew their program was working correctly. To the question, “How useful have you found the diagnostics provided by submit in helping you determine why your program was functioning incorrectly?”, 72% of the students replied with a 3 or 4 response indicating that submit was helpful. And to the question,”How useful has it been to have had graded programs returned within a week of being submitted?”, 88% of the students responded with a 3 or 4 indicating that they thought it was useful.

Two questions are indicative of why the professors find this a useful tool. To the question, “Is it likely there was at least one lab that you got working correctly (possibly late) that, if you had been allowed to submit a partially working program for partial credit, you would have done so?”, 52% of the students responded that it would be likely that they would submit a partially working program. And to the question, “Is it likely there was at least one lab that you did not get working correctly (so you were never able to submit it) that, if you had been allowed to submit a partially working program for partial credit, you would have done so?”, 50% of the students responded in the affirmative. (On this question, students could also choose a response that indicated that they never had a program that they could not get to work and 42% chose this response.)

Some examples of responses to the question “What one thing do you like about submit?” are indicative of why the responses were so positive. “It assures my program is working and guarantees me at least an 80 on the programming assignment.” “The fact that it allows you to submit at anytime since it was activated.” (s***ubmit*** is available 24/7 upon set-up.) “I like the fact that it does test for every single case that could possibly arise. It covers some cases that I probably would not think of testing.”

The negative comments were more homogeneous. Students did not like the precision that is required of them in the programming environment and the fact that ***submit*** only highlights the first discrepancy between expected and actual output. “If more than one error could be highlighted at a time, rather than just telling you the first instance, it would make submitting go much faster.” “Sometimes the error messages it gave you were very vague.” “I don't like the fact that our program has to look 100% like the program in ***submit***. It takes away from creativity and makes it feel like we have no creative powers within our own program.”

5.2 Advanced Programming Survey

The survey for the advanced programming students consisted of a single opinion question and the same two open ended questions asked of the introductory students.

The 45 advanced student respondents were asked: “How helpful was using ***submit*** in CS139 in teaching you how to program?” This question was answered on a 1 to 4 scale where 1 was not useful and 4 very useful. 64% responded with a 3 or 4 indicating that they felt the tool was useful. The free form answers followed the same pattern of responses as the introductory students, with immediate feedback and confidence that the program was correct being the primary positive statements and the precision of the checking mechanism the major negative comment.

5.3 Instructor Response to Survey

The instructors using the ***submit*** tool agree with the students on the positive aspects of ***submit***. But, the students’ negative responses are actually viewed positively by the faculty. Many students come to us believing it is okay to write code that is good enough, and not necessarily correct. This tool does reinforce the idea that there is a correct way to write their software and that if it is not correct, then it will not be accepted.

The system does take away creativity in designing program function from students. Beginning students must learn the distinction between program specification and implementation and allowing such creativity would blur that distinction. Creativity would also allow students to avoid generating code they find difficult to create, compromising the primary objective of teaching students how to code.

While students like that they had little responsibility for testing their programs, some might argue this is a deficiency in the system. For a first course in programming, we disagree. Testing is an important part of program design and is covered in the course. Separate class exercises require students to design test cases. But the focus on the programming assignments is on code creation. Intensive design of test cases is left to following courses. (It should be noted that ***submit*** is flexible enough to be used in courses where test case design is the responsibility of the student. The submit system can be configured to deduct points for each failed submission and/or test case. It can also be configured to provide little, if any, information about the test cases used by the submit system.)

6. IMPLEMENTATION DETAILS

The basic ***submit*** script obtains all the required information from the user, resets itself so it is executing with the permissions of the instructor, checks to see if a file exists that specifies a late penalty schedule, creates a directory to store the submitted files, and copies the files to that directory. It then checks to see if a course specific script exists. As it executes, ***submit*** generates the report file in the directory containing the submitted files. After executing the course specific script (if it exists), ***submit*** copies the report to the student's directory and creates a PDF copy of the report from the text copy. Without a course specific script, ***submit*** provides a mechanism for work to be submitted to a convenient, secure location. The report would just list the summary information about the submission.

For our introductory programming course, the course specific script executes the following tasks. It obtains the honor pledge declaration. It then identifies all submitted files whose names end in .java. Those source files are compiled by the java compiler. If compilation is successful, the course script calls ***progtst***. After ***progtst*** executes, a listing of each source file is output to the report file.

By using a different course script, ***submit*** has been used in other courses to handle the submission of C++ code, the submission of microcode to be run on a machine simulator, and to provide automatic grading of multiple-choice quizzes.

For automated program testing, the instructor must create an assignment directory for each assignment to be tested by ***progtst***. Within that directory, for each test, files must be created indicating the input to be supplied to the program and the correct output the program should generate when provided with that input. When ***progtst*** executes the submitted program, it arranges for the process's execution to occur under the submitter's access permissions, not the instructor's. This change in permission is necessary since ***submit*** and therefore ***progtst*** are executing with the instructor's access permissions. A misbehaved submitted program running with the instructor's access permission would have the ability to compromise an instructor's files. The executing program also runs with limits on its execution time and size of its output. If execution terminates normally, the output generated is compared to the correct output using the Linux ***cmp*** program. Abnormal program termination or output that is not identical to the correct output is identified as an error by ***progtst***. An alternative version of the correct output can be specified in the assignment directory, in which case actual output matching either version of correct output is considered correct. In addition, a correct error output file can be specified which must match the error output generated by the program.

7. CONFIGURATION FILES

The flexibility of the system comes from the series of files and directories that control the system.

7.1 Submit List

The initial submit menu is derived from information in the file, **submit-list**. Each line in this text file specifies three items of information about a course. The first item is an identifier ***submit*** uses to identify the course. The second item specifies the "course directory" where additional information about submissions for the course will be found. The third column specifies the user id of the faculty member who will own the submitted files. Items in **submit-list** are separated by white space.

7.2 Course Directory Files

The course directory contains a number of files and directories and directories important to submit.

submit.d - Files are submitted into the sub-directory named **submit.d** within the course directory. Within this "submit" directory, a subdirectory for each student is created as needed by ***submit***. Within each student directory, ***submit*** also creates a subdirectory for each submission.

Submit-script - The course specific script that is executed by ***submit*** is the file named **Submit-script** in the course directory. If not present, ***submit*** just deposits the submitted files into the submit directory.

submit-assigns - The **submit-assigns** file within the course directory specifies the assignments for which submissions are being accepted. Each name in the file serves as an assignment identifier. Optionally, the identifiers may be followed by a comma and a more complete description of the assignment.

section - For courses with multiple sections, the **section** file in the course directory associates student account names with course section numbers. If present, this file allows submit to include on the cover page of the report a line indicating which section the submitting student is registered in. If grading is not done by section or there is only one section of the course, this file does not need to be used.

assign.d - Additional information about each assignment is located in the **assign.d** directory within the course directory. Information about a particular assignment is located in a sub-directory within **assign.d** for that assignment. The subdirectory name is generated by appending "**.d**" to the assignment identifier specified in the submit-assigns file. Thus, assignment **lab2** uses subdirectory **lab2.d** within **assign.d**.

7.3 Assignment Directory Files

Files within the assignment directory can specify a late penalty schedule and the specify test cases that ***progtst*** is to perform on submitted assignments.

late - This file provides a due date for the program and the amount of penalty for each day late. The first column in the file specifies a date and time, the second column specifies the penalty. Alternative the first column may specify a plus sign followed by a number indicating the specified number of days following the previous entry.

tst-*id* **-** A test case is defined by creating a text file containing the input to be supplied for that test case. The files name is generated by appending an identifier *id* to the string "**tst-**" (for example, **tst-01**). Attributes for the test case can be specified in the file prior to the lines containing the test case input. Attributes include command line arguments to be supplied to the program (default: none), directory the program should run in (default: the directory in which the files were submitted), maximum execution time (default: 15 seconds), maximum output file size (default: 10K), and whether or not a listing of error output should be included in the test report (default: ignore error output).

tst-*id***.cout-correct** - This text file specifies the correct output for test case *id*.

tst-all - This file provides a way to specify alternate default attributes for all tests.

8. OTHER CONSIDERATIONS

Since the tests run with strictly text based processing, every prompt message or explanatory message that is displayed on the screen is captured in the output. The report lists the input separately from the output. If a program generates a series of input prompts that do not contain newline characters, the information displayed on the screen will appear reasonable since a newline is supplied when the user types in the input. However since input is listed separately from output in the report, the listing for the output in the report would have all the prompts on the same line. To improve readability of the report files, it is advantageous to have all prompts terminate with a newline. Or better yet, have a command line argument that controls whether or not the program issues any prompts since when a program obtains its input from a file, prompting for input is not necessary. Therefore, about mid-semester students are taught how use command line arguments.

9. CONCLUSION

While initially frustrating to the students, they come to like the immediate feedback that they receive from the tool. The faculty like the fact that the students are submitting correct programs and they do not need to grade sloppy, imprecise work. The reduced workload in grading means that more time is available for one-on-one interaction with students and on comprehensive evaluation of their work.

The software is freely available at `http://www.cs.jmu.edu/users/harrisja/software/submit`.

10. REFERENCES

[1] Arnow, David, :-) When you grade that: using e-mail and the network in programming courses, *Proceedings of the 1995 ACM Symposium on Applied Computing*, 10 – 13., 1995.

[2] Dawson-Howe, Kenneth, Automatic submission and administration of programming assignments, *ACM SIGCSE Bulletin, Volume 28*, (Issue 2), 40-42, 1996.

[3] Deimel, L.E., & Clarkson, B.A., “The TODISK-WATLOAD system: a convenient tool for evaluating student programs”, (ACM) *Proceedings 16th annual SE regional conf*, 168-71, 1978.

[4] Forsythe, Goerge E., Wirth, Niklaus, Automatic grading programs, *Communications of the ACM, Volume 8*, (5), 275-278, 1965.

[5] Hitchner , Lewis E., An automatic testing and grading method for a C++ list class, *ACM SIGCSE Bulletin, Volume 31*, (Issue 2), 48-50, 1999.

[6] Isong, Julia, Developing an automated program checker, *Proceedings of the 7th annual CCSC central plains*, 218 – 224, 2001.

[7] Jackson, David, A semi-automated approach to online assessment, *Proceedings of the 5th Annual SIGCSE/SIGCUE ITICSE conference*, 164-167, 2000.

[8] Jackson, David, Usher, Michelle. Grading student programs using ASSYST, *Proceedings of the 28th SICCSE technical symposium* 335-339, 1997.

[9] Jones, Edward, Grading student programs – a software testing approach, *Proceedings of the 14th Annual CCSC Southeastern Conference*, 185-192, 2000.

[10] Kay, David G, Scott, Terry, Isaacson, Peter, Reek, Kenneth, Automated grading assistance for student programs, *Proceedings of the 25th SIGCSE technical symposium*, 381-382, 1994.

[11] Macpherson, P.A., A technique for student program submission on UNIX systems, *ACM SIGCSE Bulletin*, *Volume 29*, (Issue 4), 54-56, 1997.

[12] Malmi, Lauri , Korhonen, Ari and Riku Saikkonen, Experiences in automatic assessment on mass courses and issues for designing virtual courses, *Proceedings of the 7th Annual ITiCSE Conference*, *Volume 34,* (Issue 3), 50-59, 2002.

[13] Pardo, Abelardo, A multi-agent platform for automatic assignment management, *Proceedings of the 7th ITiCSE Conference*, *Volume 34*, (Issue 3), 60-64, 2002.

[14] Reek, Kenneth, The TRY system –or- how to avoid testing student programs, *Proceedings of the 20th SIGCSE technical symposium*, 112-116, 1989.

[15] Rees, Michael J., Automatic assessment aids for pascal programs, *ACM SIGPLAN Notices*, *Volume17,* (Issue 10), 33-42, 1982.

[16] Robinson, Sally S., Soffa, M.L., An instructional aid for student programs, *Proceedings of the 11th SIGCSE technical symposium*, 118-129, 1980.

[17] Schorsch, Tom, CAP: an automated self-assessment tool to check Pascal programs for syntax, logic and style errors, *Proceedings of the 26th SIGCSE technical symposium*, *Volume 27*, (Issue 1), 168-172, 1995.

[18] Tremblay, Guy, Labonté, Éric, Semi-automatic marking of Java programs using JUnit, *International Conference on Education and Information Systems: Technologies and Applications (EISTA '03),* 42—47, 2003.

Appendix A – Survey Results

Introductory programming course – n = 28

Question	**1**	**2**	**3**	**4**	**5**
Question 1- Submit only allows programs to be submitted if they have been verified as working correctly on all the test cases. How much did you like knowing that your program, which had been submitted for evaluation had been verified as working correctly?	0%	10%	25%	64%	
Question 2 - Is it likely there was at least one lab that you got working correctly (possibly late) that, if you had been allowed to submit a partially working program for partial credit, you would have done so?	21%	18%	21%	39%	
Question 3 - Is it likely there was at least one lab that you did not get working correctly (so you were never able to submit it) that, if you had been allowed to submit a partially working program for partial credit, you would have done so?	7%	0%	11%	39%	43%
Question 4 - How useful have you found the diagnostics provided by submit in helping you determine why your program was functioning incorrectly?	7%	21%	43%	29%	
Question 5 - How useful has it been to have had graded programs returned within a week of being submitted?	4%	11%	25%	61%	

Advanced Programming Survey – n = 45

How helpful was using submit in CS139 in teaching you how to program?	11%	24%	40%	24%	