AUTOMATIC TESTING PROGRAM

Denise Coke
Undergraduate Researcher
dfcoke@yahoo.com

Monica Brockmeyer
Faculty Mentor

Technical Report
Department of Computer Science
Wayne State University
Nov. 25, 2005

Abstract

One of the primary functions of a computer science degree program is to instill in students the ability to program. As a consequence, many courses in the undergraduate program involve significant student programming exercises, designed to reinforce the concepts taught in the course. However, programming assignments are notoriously difficult to grade in a fair, timely, and comprehensive manner.

Programs which appear to work on certain test cases may fail on others and programs which are partially correct must be graded on a relative scale which is consistent with academic objectives. Traditionally, human grading has involved careful examination of each line of code to determine a grade. This approach is error-prone and subjective, since it requires the human grader to serve as a mental "compiler" of the code and since it tends to reward solutions which are closer to a "model" solution well understood by the grader. Moreover, the effort required to manually grade programs has often resulted in the assignment of fewer programming assignments than would be desirable for student learning. Another challenge arises when the human grader executes student programs to evaluate them. Typically a Teaching Assistant executes student programs using their computer or account on the departmental computers, posing a significant security risk to the TA who is running unfamiliar code in his or her own user space. To alleviate these problems, many computer science departments have turned to automated testing mechanisms as a component in grading of student assignments.

1 Introduction

There are several components involved in the grading of student assignments:

1. Correctness - Given a legal input, the program is produces the desired output.

2. Robustness - Given an illegal input, the program produces an appropriate error message and continues to process input.

3. Efficiency- A program that performs its tasks without consuming too much processing time and memory. (The notion of "too much" depends upon the specific task.)

4. Design- A program that is easily understood, employs appropriate error checking, utilizes appropriate coding standards, and demonstrates other characteristics related to the maintenance and portability of the program rather than to the functional behavior.

5. Human Interface - A program that makes it easy for a human being to interact with.

While automatic testing programs have sometimes been referred to as autograders, it is clear that only some components of evaluation can be automated. However, evaluation of the first three characteristics listed, correctness, robustness, and efficiency, can be tremendously enhanced and facilitated by automatic testing. Therefore, the research was focused on these three with some consideration of the other two, design and human interface. The remaining characteristics can be more thoroughly evaluated by the grader who can devote more time to these tasks since the evaluation of correctness has been mitigated by automatic testing.

Automatic testing programs typically work by the use of an *oracle* or something similar, which accepts test input and provide the set of acceptable outputs corresponding to that input. The oracle can either be a reference implementation of the assignment or simply just a set of output files created for the comparison portion. For simple programming assignments, it is easy to use the reference implementation as the oracle which can be written by the instructor or TA. (The development of an oracle for programs where a single input can produce multiple correct outputs, is slightly more difficult, but can be addressed in some semiautomatic ways.) For other more lengthy assignments, a set of output files is usually more convenient. Student programs are tested by execution of the student's code using the test input and comparison of the resulting output with the output produced by the oracle. The grading procedure involves creation of a set of tests which demonstrate the various kind of functionality required by the program. The results of the tests can then be used by the human grader in assigning the final grade to the student project. (It should be emphasized that it is the human grader, not the testing mechanism, who is ultimately responsible for assignment of the project grade.)

In addition to easing the burden of determining program correctness, many other useful benefits can accrue to automatic testing of student programs. First, automatic testing of student programs can be used in advance of the final submission to provide the student with early feedback of the correctness of his or her code. This helps avoid the common frustration on the part of both students and instructors, where students submit code that they believe to be correct, yet receive low marks due to a lack of comprehension of program requirements. In this manner, the automated testing environment supplements the formal specification of the program assignment in helping the student to understand what exactly is required. Second, early feedback may promote earlier starting of student programs, since feedback about incorrect behavior can then be corrected before the due date. Third, automatic testing promotes a test-driven development process, characterized most recently by the concept of extreme programming. A test-driven development process promotes testing and it is easier for students to understand than traditional testing approaches and because it tests during the development phase by running different/incomplete versions of the code. In extreme programming, for example, new code is only developed in response to failed tests. This approach mitigates large-scale integration problems often observed when students try to write a comprehensive solution in its entirety before testing and increases the student's confidence in the code they have written.

2 Outline

Given the simple description of the project the outline is still very complex.

1. Create a program in Perl that executes another inside a shell specifically created for this execution.
2. Adjust the program to accept redirection and Inter-process Communication (IPC) using pipes.
3. Modularize program for reuse of common functions.
4. Create the web interface of the program.
5. Set up a server for Web interface (Abyss).
6. Create a login setup for students and professor's to have access to their folders.
7. Modify web page to allow forms and scripts to pass through the server.
8. Modify program to execute the build portion of the testing with the test file as the input.
9. Port the program from Windows to Unix OS.
10. Configure the Unix server to runs scripts (Apache).
11. Run a set of standard tests and compare the outputs.
12. Add the hidden and efficiency tests to the professor's portion and test the outputs.
13. Create a Demo version of autograder for presentation purposes
14. Last but not least, make sure the information sent across the server is secure.

3 Fine Tuning

Other desired features for the automatic testing program include:

1. **Submission mechanism.** Submission of student programs for evaluation by the grader has posed some challenges not found for other kinds of assignments. Currently, students compress their programs into a tar or zip file and submit it via email or the digital drop box in blackboard. Then a TA typically downloads the submission for execution and un-compresses the file, using email to communicate with the student if the submission is incomplete. Some students, especially those in beginning programming courses, may experience difficulty in creation of the compressed file, leading to incomplete submissions or corrupted directory structures. An ideal submission mechanism would include the following features:
    - Support for accurate and complete submission of required files, with automatic feedback to the student in the case of problems.
    - An accurate times tamping mechanism to determine timeliness of the submission.
    - Automatic integration with the secure execution environment.

1. **Student Feedback.** As described above significant educational advantages are possible if the system provides student feedback. Various mechanisms for student feedback are possible and are closely related to the front-end used to develop the submission mechanism. Several types of front ends are possible. Many traditional systems have involved email front-ends, using reply email for student feedback. A web interface offers the possibility of more synchronous feedback and support for more complex interactions between the student and the automatic testing mechanism.

2. **Plagiarism Detection.** Detection of plagiarism is an important consideration and there are automated comparison mechanisms that are extremely effective. Such mechanisms provide pairwise comparison of program code and can report the percentage of similarity. While plagiarism detection is not a primary focus of the project, integration with an existing plagiarism detection mechanism would offer a tremendous convenience.

4 Inspiration

The common practice in computer science departments is to use home-grown systems, which have not frequently been supported at level that makes them appropriate for general use. However, a few systems have been made available for download. One of these systems is the Online Judge and it was used as a starting point for the research. The Online Judge was developed for use in programming contests for computer scientists and provides support for most of the test types we describe above, including support for testing the efficiency of student code. Extensions are needed to provide more useful feedback to students and to improve the security of the execution environment, as well as to provide support for nondeterministic and multi-process programs developed in more advanced courses. Fortunately, the use of scripting languages to develop Online Judge and other systems makes them easy to extend. Also, many automatic testing systems have been developed for use in industry and are generally very expensive. In addition, they are not designed to support evaluation of programs from an academic perspective. Hence the designing of our own autograder system.

5 Conclusion

Overall, the automatic testing mechanism will be very helpful once it is fully developed. The time allotted for the completion of this project was insufficient so it is still being developed at this time. Once the development is complete, the additions listed in the outline will be added and the project will be prepped for presentation.

6 Acknowledgements

The first part of this research was done through the sponsorship of the Distributed Mentor Project (DMP) program during the summer 2005 year at Wayne State University, Detroit, MI. Please visit my website for more information on this project. http://paris.cs.wayne.edu/~ak5270

7 References

[1]Brockmeyer, Monica. *Description of the Automatic Testing Program.* Department of Computer Science, Wayne State University, June 2005.

[2]Castro, Elizabeth. HTML for the World Wide Web. California: Peachpit Press, 2003.

[3]Lowe, Vincent D.. Perl Programmer's interactive workbook. New Jersey: Prentis Hall PTR, 1999.

[4] Pierce, Clinton. Perl in 24 hours. Indiana: Sams Publishing, 2005.