Introducing the Test Harness: Automating the Test Suite

Ronald Finkbine, Ph.D., rfinkbin@ius.edu, Indiana University Southeast
Nicholas Kraft, nkraft@cs.clemson.edu, Clemson University

ABSTRACT

Software testing is an essential component in the development of quality software. It is important for students to have a solid introduction to this in their academic career. Since students are not often excited to write code, they certainly will not be interested in testing their code. Students will often only submit test executions for a small handful of test cases and rarely will they regression test the entire test suite. The repetition necessary for proper testing is an obvious contributing factor to this problem. Enter the test harness. Automation of a test suite is an effective way to allow for quick and repeatable testing of a program. Though, it requires effort to build the test harness, it can be used to test the program throughout the development and maintenance process. Allowance for quick regression testing in every test harness execution is an added bonus.

1. INTRODUCTION

Software testing, much like verifying answers in the field of mathematics, is a much reviled and often overlooked task. However, testing is becoming increasingly important for development of reliable software and currently accounts for a substantial and growing portion of the cost of software in industry. But, regardless of its importance in industry, the subject of software testing gets very little coverage in the typical undergraduate curriculum.

As the field of computer science matures, the size, complexity, and legal risk associated with computer programs increase. These factors contribute to the increasing need for adequate software testing. Students often encounter testing for the first time after their academic careers have ended. Whether they are testing their own or another programmer's code, students must have the knowledge and tools to show the correctness of the code.

While testing alone cannot prove the correctness of a piece of software, it can be used to compare the actual performance of a program to the expected performance. A summation of the goals of testing, as laid out by Myers [3], follows:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one having a high probability of finding an as yet undiscovered error.
3. A successful test is one that uncovers an as yet undiscovered error.

Avoidance of learning the intricacies of software testing concepts and procedures is expected of the typical undergraduate student since testing is, by necessity, tedious and repetitive. Yet, it can enhance software development skills in that it requires integration of concepts and practice [2]. So, how can testing be transformed into something less mundane? Carrington [1] makes the case that students who have been involved with the toil of testing recognize the need to automate the process with creation of a test harness.

2. TEST HARNESS DESCRIPTION

The test harness consists of a number of items which we shall define, test cases, a test suite, a set of input files, a set of expected output files and the test script to drive the entire process. This section shall define each of these items.

2.1. Test Case

A single test case consists of an input file that contains inputs to specifically test certain characteristics of the subject program. The execution of the test case will produce an output file, absent some form of run-time error that interrupts execution. A test case should test one certain behavior of the subject program. For example, five test cases for a linked list program would be:

1. Can you insert into an empty list?
2. Can you insert into the front of a list?
3. Can you insert into the end of a list?
4. Can you insert into the middle of a list?
5. Can you insert a duplicate item into the list?

2.2. Test Suite

A test suite is the collection of test cases for a given program and its creation requires more experience than coding since the test suite must use a sequence of increasingly difficult test cases to fully test a program. A test suite alone cannot prove the correctness of a given program, but a good test suite can be used to uncover unknown defects.

2.3. Input Files

An input file consists of the input test data for a single test case. Test cases, like functions in a program, should concentrate on a single task. Therefore, an input file should contain test data for testing one specific capability of the subject program. Input files should be named using a consistent convention (i.e. test001.input). Documentation explaining the condition being tested must accompany each test input file in the test suite.

2.4. Output Files

Each test case produces an output file and execution of the entire test suite produces a collection of these files. Each should be named as associated with the test case such as test001.output. These output files need to be analyzed to determine if their associated test cases were successful.

Figure 1 is an MS-DOS command line example of executing one test case, showing the program name, input file being redirected, and output file creation.

```
C:\>program1 <test001.input >test001.output
C:\>program1 <test002.input >test002.output
```

Figure 1: Input and Output

Redirection operators allow files specified in the test script to serve as input and output to the subject program rather than having the files specified from within the subject program, itself.

```
%1 <test001.input >test001.output
…
C:\testharness program1
```

Figure 2: Parameterized

Figure 2 shows the addition of a parameter to the test script previously shown in Figure 1. This will allow the entire test suite to be run against a number of versions of the same program which is the main characteristic of mutation testing [4].

2.5 Control Files

The result of executing the program given each test input file must be captured and compared byte-for-byte to a control file. Initially, there are no control files though the systems analyst should have an idea what should be in the file. The control files will each be created during the first totally successful execution of the test case.

The development of the control files is most critical and must be verified by the systems analyst and not by only the junior programmer. A wrong answer in a control file indicates either a bug in the software or a bug in the test script, either of which is very bad. But a bug in the test script also ensures that the bug will continue to be present, never detected nor corrected. Once the desired and verified output is achieved for a given test case, the output file should be deemed a control file, and have its filename changed accordingly (i.e. test001.control).

2.6. File Comparison Program

Analysis of the output files consists of comparing each output file to its corresponding control file. The file comparison utility program is used to compare the control files to the output files from the most recent execution.

A file comparison program with a ‘quiet’ mode is preferred over a built-in shell command. ‘Quiet’ mode keeps unneeded information from being recorded in the error logs. Also, with a ‘quiet’ mode being used, a lack of feedback from test case execution indicates success (no news is good news). The UNIX utility program *diff* and MS-DOS utility program *comp* are two such file comparison programs and *diff* will be used to construct the sample test harness below.

```
C:\>diff test1.control test1.output >>errorlog
```

Figure 3: Append

Figure 3 shows the usage of the file comparison utility *diff* (this UNIX utility has been ported to MS-DOS) verifying that the control file and the just-produced output file are equivalent. The utility runs by default in quiet mode and if it generates any output at all, that is not good. But the output error messages indicating that the test case was not passed will be appended to the file *errorlog*. If all executions of the *diff* command during one execution of the test suite append to one file, then the condition of file *errorlog* at the completion of the test script determines the success of the test script. An empty *errorlog* indicates no errors were detected in any of the test cases; a non-empty *errorlog* indicates at least one test case failed.

```
%1 <test001.input   >test001.output
diff   test1.control  test1.output >>errorlog
```

Figure 4: Addition of file comparison

2.7. Test Script

Either a batch file or a shell script file (dependent on system) automates the test harness. The test script contains all command line statements necessary to execute the test suite. In addition, the test script will perform the comparison of the control and output files. Parameterization of the script is simple and will greatly aide reusability. A positional parameter can be used in either a batch file or a shell script, although the syntax differs slightly.

The test script can be described as a driver script (program) that directs a testing sequence; supplying inputs, capturing outputs, comparing those outputs to the control (expected) outputs and appending outputs to error logs.

A detailed test script developed by the systems analyst can assist junior programmers in development of the program. If the tests cases are sequenced by complexity level, the junior programmer can begin programming with the simple test cases, which hopefully will be the easiest to understand and require the least coding effort.

2.8 Test Harness

The test harness is the compendium of parts describe thus far, a test suite of test cases, each with an input file, a just-produced output file (from the last execution), a control file to compare for test case success and a file comparison utility for determining file equivalence.

Figure 4 shows a parameterized test script which contains a single test case. The %1 will be replaced by the program name by the command shell interpreter. Each test case will read from the input file and write to the output file. After creation the output file will be compared by the *diff* command.

Construction of the test harness is relatively straightforward with just a basic knowledge of command line redirection and batch/script file authoring. The harness itself can also be a template from which others can be constructed for future test scripts. It is also noteworthy that regression testing is performed each time the test harness is executed thus insuring that code modifications made to correctly execute the current test case do not as a side effort (effect?) cause the program to incorrectly process a previously passed test case.

3. CONCLUSION

Testing is an extremely important part of any software project and should be part of any computer science curriculum. Resistance to learning and carrying out testing is to be expected, however automation can alleviate the pains normally associated with the testing process. The building and usage of a test harness is beneficial to both software professionals and students alike.

4. REFERENCES

1. Carrington, D. "Teaching Software Testing." In *Proceedings of the second Australasian conference on Computer science education* (The Univ. of Melbourne, Australia, 1996), pp. 59-64. ACM Press, New York, 1997.
2. Jones, E. "Software testing in the computer science curriculum -- A holistic approach." In *Proceedings of the on Australasian computing education conference* (Melbourne, Australia, 2000), pp. 153-157. ACM Press, New York, 2000.
3. Myers, G. "The Art of Software Testing." Business Data Processing. Wiley-Interscience, 1979.
4. Wong, W. E., “Mutation Testing for the New Century”, Kluwer Adacemic Publishers, ISBN 0-7923-7323-5, 2001.