Writing Grading Software to Create an Automated Testing Environment for Improving Software Test Coverage

Lauren V. Gaber
University of Michigan – Dearborn
lgaber@umich.edu

***Abstract* – the problem this research project addresses is student software testing. The goal is to see whether or not a grading program that focuses on unit testing improves student's testing skills and assignment grades by making them pass the tests in order to get a higher grade on their assignments. A blind study using the software will be performed and the results presented at a later date.**

This software was made to test student's programs by applying unit tests (which are functionality tests for code segments) with data input compiled with the student submitted code for an assignment. The professor must submit correlating input and answer files in order for the students to have tests to pass. The program takes each submitted student program and the test inputs, goes through compilation and run procedures to create output files, and then compares that output to the answer for the test. The students can get hints in the grading response if they fail a test, but only if their professor chose to submit hint files. It is designed to encourage the students to make their code maximally functional by thinking more critically about testing their code and the importance of fixing errors.

I. INTRODUCTION

Software system failure can be costly to business reputation, time, and money. Therefore, software testing is an important practice to teach students who plan to work with software development and testing. There have been other software tools written to encourage unit testing, such as CxxTest, JUnit , and NUnit. Among them, Web-CAT [1, 2] is used for grading student programs by running student-written tests. But beyond basic code functionality and assignment requirements, it is difficult to get students to think about software testing when their main goal is to complete an assignment; thus this grading software marries the two objectives in an effort to improve student performance.

This software will go beyond Web-CAT and the others by using both black box and unit testing to evaluate the student's code, keep records of student scores and assignment data, and provide more tools to the professor to view and advise their student's code practices. Lastly, these improvements will be tested for effectiveness in the blind study. The program is written with Visual Studio 2012, in C#.

II. SOFTWARE MODEL AND SPECIFICATIONS

The software model presented to the programmers consists of three main components: the user interface, file class and database class structures. The user interface is tailored for three types of users. The first are administrators, who run the website and can control user and administrator access to the site. Administrators are typically the programmers and research project directors. The second type of user is a professor who can create courses, create assignments for those courses, upload files used in grading, view their courses' grades, code and output, and manage their student's access and their own access to the software. This type of user is defined as professors of computer science students at the University of Michigan – Dearborn and Oakland University. The last type of user is a student, who can submit their code for grading and see the test results. At this time, students are defined as undergraduate computer science students at the University of Michigan – Dearborn and Oakland University.

The file class performs the main file creation actions for the users as well as the compilation and grading functions. The accepted source code languages for student program submission are C++, C#, and Java. The C++ compiler accepts coding conventions that appear in Visual Studio 2011 and older versions. An additional way to submit code is available where the professor submits a driver and the students submit a class file that the professor's code uses when the assignment is being graded. The grading printout includes a pass/fail evaluation for each test and can also return compiler errors and a few other anticipated types of coding errors.

The database class allows the users to store and interact with their information in a database. The database stores information about administrators, assignments, courses, professors, students, student grades and student courses. This amalgam includes data on usernames and passwords, testing results, and assignment specifications among other things. The interface is built to restrict access to information that a user is not allowed to see; for example, professors can view their students' grades, but they are not allowed to view another professor's student information.

III. USER INTERFACE

The user interface was programmed in a Visual Studio 2012 C# web application. It was made available on the University of Michigan – Dearborn network in early July. It will eventually be made available outside the network so Oakland University students can access the program. The start page displays the CaseSense logo and the option to go to one of three different login screens for each type of user (and each login screen goes to a screen specific to the type of user). Each of these is built with a login control provided by the visual studio toolbox, and programmed to check for the user's name and password as stored in the database. If the entered data does not match up, they do not get transferred to the next screen. The logo was designed by Marjorie Gaber and is displayed below.

CaseSense

The students are transferred to their submission screen where they choose a course, assignment, source code language type, and source code file to upload. The assignment dropdown responds to the selection of the course dropdown, by displaying all the assignments in the selected course [3]. The language type options are C++, C#, Java, C++ Class, C# Class, and Java Class. The code is set up so it knows if you submitted the matching type of file or not. Then for the response screen, any compiler errors are printed in red, and otherwise normal feedback is printed in black.

The professors are transferred by default to their technical work screen, where they create courses, add students, add students to a course, add an assignment to a course, and specify an assignment's name, description, submission time period, number of submissions allowed for students, the number of points it is worth, and upload input, answer, hint, and driver files. Functions for removing courses, assignments and students have also been added.

The other professor screen is where grades can be viewed; this has a listing of the professor's courses and assignments for those courses; when both have been chosen from, a list for all the student assignment grades in the course is generated. When a student's grade record is picked from the list, three boxes below are filled with data: one text section displays the student's source code for the assignment, another one has the student's output, and the last box right next to it has the assignment's expected output. Lastly, the professor can choose to run this student's last submitted source code and see the grading responses.

The administrators are transferred to their own work space where they can add and delete professors, add and delete administrators, and update passwords. This may be reworked later so students and professors can update their own passwords.

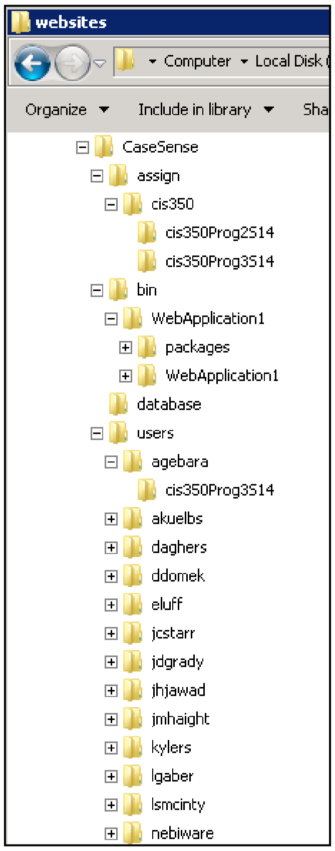
IV. FILE CLASS

The file class fileStuff was programmed in a Visual Studio 2012 C# class file, and was tested during production with a driver. FileStuff's construction was first focused on getting C++ compilers for the windows command line to compile then run programs with input files and redirect its output to files using batch files. The compilers that were studied for this task were the Visual Studio CL compiler and the open source MinGW g++ compiler [4]. The conclusion of the research was the g++ compiler worked better for the software's needs because it could compile C++ 11 files without any difficulty and without too many dramatic changes to the server's environment variables [5]. The C# and Java compilers were comparatively easier to handle after the C++ compiler trouble had been sorted out, and were installed for use shortly after.

The next task entailed translating the work from batch files to C# code. System.Diagnostics.Process and System.Diagnostics.ProcessStartInfo enabled interaction between the program and the command line by sending commands in strings with the StandardInput.WriteLine function [6]. The running of the program code with multiple input files was managed via Task kills, where each file was given one to two seconds to compile and one second to run a single test and produce output; if it didn't complete its job in this amount of time, then the run would be cut short so the job could stay time efficient. A problem that is still unresolved arose from this: it was noted that if a test completed and produced output in under a second, the allotted time had to be used up before the task was killed and the next one could run. This meant a loss in efficiency by a great deal; but until it has been resolved, estimation time for grading has been added into the assignment description display for the users. In order to provide the compiler responses in the grading printout, compiler errors were stored (and in some cases extracted) [7] from the command line responses via the StandardError.ReadToEnd function. Once the tests were done, the executable files had to be removed so they can be freshly created from the most recently submitted code.

The final step in assembling the file class was determining how to grade the student submissions. The two major differences among submitted files were if they were to be regular driver files (with the option of having a class, struct, or some other kind of data structure included in the code along with it) or class files with the driver pre-submitted by the professor. This difference was important because it decided naming conventions. If it was a class file alone, then upon submission the file needed to be named the assignment's name, and the professor's driver would already be submitted and have to be named the assignment name with the addition of "_driver" on to the end of the name. If it was a regular driver file, then it could be submitted by the student and be named anything and it would automatically be changed to the convention of the student's username.

When the program is grading the student output against the professor's answers, they are expected to be nearly identical with the exception of new lines and tabs (which are removed for their analysis) [8]. The grading responses that a student may see are passes, fails with hint, fails with no hint, compiler error and no testing, loop error suspected in program, loop error suspected in program output, no answer file found, and no answer or output file found. These specific responses are detected and suggest educated guesses based on the results of the file comparison, file sizes, run time, and compile time to make an executable. The files needed to run this program are stored in professor and student directories in the case sense folder, as pictured below.

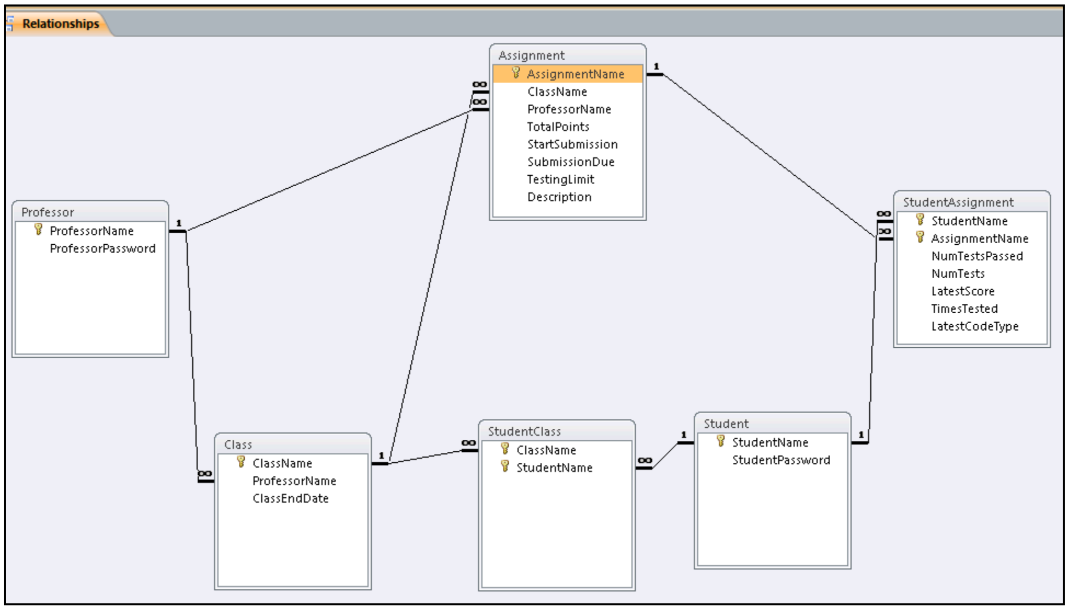


```
⊟ CaseSense
  ⊟ assign
    ⊟ cis350
        cis350Prog2S14
        cis350Prog3S14
  ⊟ bin
    ⊟ WebApplication1
      ⊞ packages
      ⊞ WebApplication1
    database
  ⊟ users
    ⊟ agebara
        cis350Prog3S14
    ⊞ akuelbs
    ⊞ daghers
    ⊞ ddomek
    ⊞ eluff
    ⊞ jcstarr
    ⊞ jdgrady
    ⊞ jhjawad
    ⊞ jmhaight
    ⊞ kylers
    ⊞ lgaber
    ⊞ lsmcinty
    ⊞ nebiware
```

The CaseSense folder has four main directories and is currently stored at C:\inetpub\wwwroot\CaseSense. The assign directory has a subdirectory of courses, and the courses have a subdirectory of assignments where all the necessary professor-submitted files are stored for grading. The bin directory has a subdirectory containing the source code for the user interface which includes the file class and database class structures. The database directory has a subdirectory containing the Microsoft Access database file that stores the software's data. Lastly, the users directory has a subdirectory of student username folders, and the student username folders have a subdirectory of assignments where all the necessary student-submitted files are stored for compilation, running, and grading.

V. DATABASE CLASS

The database class databaseStuff was programmed in a Visual Studio 2012 C# class file, tested during production with a driver and a copy of the Microsoft Access database. The database contains seven tables: the Administrator table, Assignment, Class (course), Professor, Student, StudentAssignment and StudentClass (course) tables. The only tables that might need more detailed explanation beyond their names are StudentAssignment (that holds student grade records for an assignment) and StudentClass (stores a student's username next to the name of a course that they're in). The relationship diagram containing primary keys for each table and a listing of each table's relationship with the others has been pictured below.



Each box represents a data table in the database, and the name of the box is the name of the database. The listing of words in the boxes is a listing of the columns in the order they appear in each table. The key symbol next to some columns indicates a primary key, or a column whose data is unique and cannot have duplicates and serves as the primary reference for sorting the data. The lines drawn between boxes are the relationships the tables have with each other; the type of relationship that is being used here is called a one-to-many. One-to-many relationships mean that one type of data point can be represented many times in another table. For example, the data type ProfessorName, which represents a professor's username, can be listed many times in the Assignment table because a professor can create more than one assignment to be graded. In the StudentClass table, both a ClassName and StudentName can appear multiple times in the table; but because they both make up the table's primary key, no combination of the two can appear more than once in the table.

In order for the databaseStuff class to work with the Access database, a System.Data.OleDb.OleDbConnection has to be opened before any commands can be read [9]. The main interaction with the database was done through query operations with the OleDbCommand type, through the insert, update, select, and delete commands. The insert command is used for adding new records of any type to the table, like a new student or an assignment but can't be used to update data. Update commands update data in a record that already exists, like passwords or data not previously available for the record like a student's grade on an assignment they haven't submitted yet [10]. When a user wants to get information stored in the table, the select command is used to return strings with all the data to be displayed [11, 12, 13, and 14]. Lastly, the delete command is used to remove individual records or everything connected to a single record with a primary key. The last thing that was done was to use the fileStuff class RunOperation and runClassOperation to allow the professor to run the last version of a student's program and see the printout.

VI. CONCLUSIONS AND FUTURE WORK

The software that we developed will fulfill the basic needs of the researchers conducting the project by providing them with an interface that utilizes unit and black box testing for grading and record storage for user information. In the future, the software should be reviewed for improvements and an expansion of its functions, with the end goal of being published publicly on the web, to be used as a useful grading tool for computer science scholars. Security and product testing would be an essential part of this goal, and another team of programmers is being assembled to continue development.

REFERENCES

[1] Edwards, S. H. ,Perez-Quinones, M. A. *Experiences using test-driven development with an automated grader*, Journal of Computer Sciences in Colleges, 22(3):44-50, January 2007.

[2] Edwards, Stephen H. *Using software to move students from trial and error to rejection and action*, Proc 35th SIGCSE Tech Symp. Computer Science Education, ACM 271-275, 2004.

[3] M. Smith. (2012). *ASP.Net dropdown always return first value on button click event* [Online]. Available at: http://stackoverflow.com/questions/13331160/asp-net-dropdown-always-return-first-value-on-button-click-event

[4] J. LeComte. (2007). *Getting started* [Online]. Available at: http://www.mingw.org/wiki/Getting_Started

[5] O. N. (2012). *Compiling C++11 with g++* [Online]. Available at: http://stackoverflow.com/questions/10363646/compiling-c11-with-g

[6] D. D. Chavan. (2011). *How to send series of commands to a command window process* [Online]. Available at: http://stackoverflow.com/questions/4788863/how-to-send-series-of-commands-to-a-command-window-process

[7] E. J. (2012). *Extract part of a string between point a and b* [Online]. Available at: http://stackoverflow.com/questions/9505400/extract-part-of-a-string-between-point-a-and-b

[8] K., J. Brunscheon. (2013). *How to remove new line characters from a string* [Online]. Available at: http://stackoverflow.com/questions/4140723/how-to-remove-new-line-characters-from-a-string

[9] M. Nasir. (2013). *How to read/write data from/to MS Access database using C# winform* (Edition 5 of 7) [Video]. Available at: http://www.youtube.com/watch?v=Jp9v0FAx2io&t=573

[10] W3Schools. *The SQL update statement* [Online]. Available at: http://www.w3schools.com/sql/sql_update.asp

[11] MSDN. (2014). *OleDbDataReader Class* [Online Reference]. Available at: http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader%28v=vs.110%29.aspx

[12] MSDN. (2014). *How to: create and execute an SQL statement that returns rows* [Online Reference]. Available at: http://msdn.microsoft.com/en-us/library/fksx3b4f%28v=vs.110%29.aspx

[13] MSDN. (2014). *Retrieving data using a DataReader* [Online Reference]. Available at: http://msdn.microsoft.com/en-us/library/haa3afyz%28v=vs.110%29.aspx

[14] MSDN. (2014). *How to: create and execute an SQL statement that returns a single value* [Online Reference]. Available at: http://msdn.microsoft.com/en-us/library/eeb84awz%28v=vs.110%29.aspx

---

(University of Michigan – Dearborn, 4901 Evergreen Road, Dearborn MI 48128-1491)
Date: August 30, 2014.
Key words and phrases: computer science, education, tools.