Automatic Program Tester

Software Engineering Documentation

Lounge Against the Machine

Joe Manke Alex Wulff Adam Meaney

March 22, 2014

Contents

List of Figures

List of Algorithms

Mission

To create a program capable of automatically grading programs written for lower-level programming courses such as CSC150 and CSC250. Given the name of a source code file, it will compile the code and find all test cases (notated by a ".tst" extension) in the parent and any child directories of where this program's executable is located. The target program will be run against each test case and be recorded as a pass or fail based on the difference between expected and actual output. After all the tests are executed, the results, including statistics, will be written to a log file.

Document Preparation and Updates

Current Version [1.0.0]

Prepared By:
Joe Manke
Alex Wulff
Adam Meaney

Revision History

Date	***Author***	***Version***	***Comments***
2/18/14	*Joe Manke*	*1.0.0*	*Initial version*
3/21/14	*Alex Wulff*	*1.1.0*	*Updated Documentation*

1

Overview and concept of operations

The overview should take the form of an executive summary. Give the reader a feel for the purpose of the document, what is contained in the document, and an idea of the purpose for the system or product.

1.1 Scope

This document covers the design and implementation of the program.

1.2 Purpose

The purpose of this program is to assign pass/fail grades to simple programs run against a number of test cases.

1.2.1 Compilation

In order to test a program, it must be compiled into an executable.

1.2.2 Directory Crawl

This component finds all of the test cases.

1.2.3 Test Execution

The target program will be executed against a number of test cases.

1.2.4 Result Logging

Results of the tests must be recorded for the user to read.

1.3 Systems Goals

The goal of this system is to perform automated testing and grading for C++ programs.

1.4 System Overview and Diagram

The program begins by finding all test files. Then, the target program is compiled. Next, all of the test cases are executed. Finally, the results are written to the log file.

1.5 Technologies Overview

Developed in Linux in C++, using the g++ compiler.
Documentation created using TexWorks and MikTex.
Flowchart created using Gliffy.

1.6 Terminology and Acronyms

See Table 1.1

GPL	General public licenced software such as GNU\Linux environments and tools
Agile Methodology	An approach to project management as it relates to software engineering `agilemanifesto.org`
C++ STL	Standard Template Libraries: classes and methods used in common C++.

Table 1.1: Defining Some Important Terms

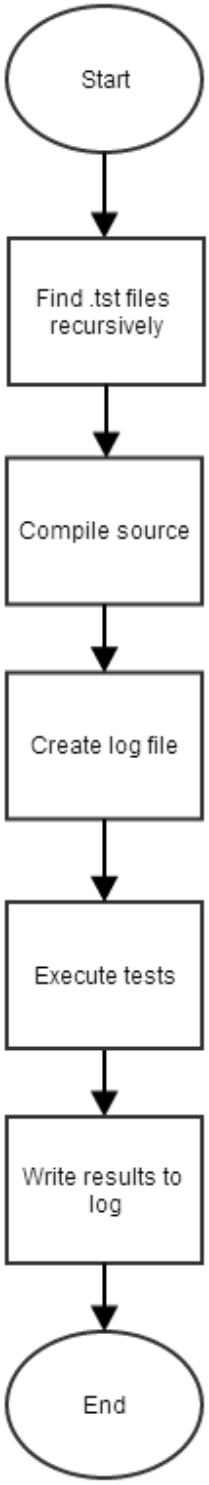

Figure 1.1: Program flowchart

2

Project Overview

This section provides some housekeeping type of information with regard to the team, project, etc.

2.1 Team Members and Roles

Joe Manke - Scrum Master
Alex Wulff - Technical Lead
Adam Meaney - Product Owner

2.2 Project Management Approach

This project was completed in 2 one-week sprints, managed on Trello.

Source control was done through GitHub.

Sprint 1 consisted of establishing our source control repository and implementing the functions necessary prior to actual test execution. This included directory traversal to find .tst files, compiling the source code, and creating the log file.

Sprint 2 consisted of combination of these functions and execution of tests.

2.3 Phase Overview

2.3.1 Design

The design phase consisted of determining specifications from the client, Dr. Logar. With the specifications determined, we devised a flowchart.

2.3.2 Implementation

The implementation phase was to write the code to fit the specifications of the program.
This phase was split over two sprints, as described in section 2.2 above.

2.3.3 Testing

After the code was written, it was tested to ensure proper functionality.

3

User Stories, Backlog and Requirements

3.1 Overview

This section contains the overview for defined user stories, initial backlog and requirements as gathered from stakeholder meetings with Adam Meaney the Product owner of Lounge Against The Machine.

3.1.1 Scope

This section is intended to provide proof of concept data and function as a check-sheet for completion of task.

3.1.2 Purpose of Development

The purpose of this application is to assist in various users (likely professors, TA's and other staff at SDSMT) to perform automated testing on simplistic programs as submitted as source code to be compiled in a standard Linux environment with current GCC.

3.2 Stakeholder Information

The main stakeholder for this program is Dr. Logar, but other stakeholders may include other computer science professors and TAs.

3.2.1 Customer or End User (Product Owner)

In his role as Product Owner, Adam Meaney was the liason between our team and Dr. Logar with the assistance of Alex Wulff.

3.2.2 Management or Instructor (Scrum Master)

As Scrum Master, Joe Manke assigned tasks and managed the Trello board. He also wrote documentation with the assistance of Alex Wulff.

3.2.3 Investors

The investors of this project will be both those that use this program for automated testing and future teams who will modify this program to suit further defined features. Should this program fail to be properly coded or documented, successive teams will likely not be successful in making timely adjustments and additions. As such attention to detail is paramount.

3.2.4 Developers –Testers

Alex Wulff was the Technical Lead, and shared development duties with Adam. All three members of the team participated in code review and testing.

3.3 Business Need

The program must be able to compile and execute another program given only the name of the source code file.

The program must be able to find all test cases in the parent directory where the program is executed, and all of its child directories.

The program must be able to determine success or failure of individual tests and record results.

This program meets all of these needs. Program compilation and execution are handled using system calls. Test cases are found through a recursive directory crawl. Test success is determined by performing a diff between expected and actual output and recorded in a log file.

3.4 Requirements and Design Constraints

3.4.1 System/Development Environment Requirements

The program must be written in C++ and is required to work on the MCS department's Linux boxes, using Fedora. This necessitated that the program be developed and tested in Linux.

3.4.2 Project Management Methodology

- Trello will be used to keep track of the backlogs and sprint status.
- The development team and the primary stakeholder (Dr. Logar) will have access to the Sprint and Product Backlogs.
- The number of sprints and sprint length were left to the development team. We decided upon two one-week sprints.
- GitHub is the recommended source control.

3.5 User Stories

3.5.1 User Story #1

As a user, I should be able to compile and execute a program by providing the source code.

3.5.2 User Story #2

As a user, I should be able to locate test cases in child directories.

3.5.3 User Story #3

As a user, I should be able to see individual and aggregate results of test cases.

3.5.4 User Story #4

As a user, I should be able to run tests against a program multiple times without overwriting previous test results in the log.

3.5.5 Research or Proof of Concept Results

As testing environments are relatively commonplace in test-driven development and Agile, research is taken praesumo presumo. Additional information regarding test cases and test suites can be found though IBM at `http://pic.dhe.ibm.com/infocenter/clmhelp/v4r0/index.jsp?topic=%2Fcom.ibm.rational.test.qm.doc%2Ftopics%2Fc_testcase_overview.html`

3.5.6 Supporting Material

This document might contain references or supporting material which should be documented and discussed either here if approprite or more often in the appendices at the end. This material may have been provided by the stakeholders or it may be material garnered from research tasks.

4

Design and Implementation

This section is used to describe the design details for each of the major components in the system. This section is not brief and requires the necessary detail that can be used by the reader to truly understand the architecture and implementation details without having to dig into the code. Sample algorithm:

4.1 Compilation

Please use the included make file to update and rebuild your executable.

4.1.1 Technologies Used

- Uses the stdlib.h library to make a system call.
- C++ vectors

4.1.2 Phase Overview

This component has been created for the initial deployment of V1.x

4.1.3 Design Details

- Find the core name of the file name (trim extension)
- Construct command: g++ -o corename filename -g
- Make system call

4.2 Directory Crawl

4.2.1 Technologies Used

- Uses the dirent.h library.
- Uses C++ STL vectors

4.2.2 Component Overview

This component uses C++ STL vectors to contain the list of test files (denoted by a '.tst' extension) needed to execute against a specified source file. These are found by making recursive directory searches (see figure 4.1)

4.2.3 Phase Overview

This component has been created for the initial deployment of V1.x

4.2.4 Design Details

```
#include<dirent.h>
#include <string>

void ParseDirectory(string root)
{
    string temp;

    DIR *dir = opendir(root.c_str()); // open the current directory
    struct dirent *entry;

    if (!dir)
    {
        // not a directory
        return;
    }

    while (entry = readdir(dir)) // notice the single '='
    {
        temp = entry->d_name;
        if ( temp != "." )
        {
            if ( temp != ".." )
            {
                if ( temp[temp.size() - 1] != '~' )
                {
                    int length = temp.length();
                    if ( length > 4 && temp[length-1] == 't' && temp[length-2]
                        == 's' && temp[length-3] == 't' && temp[length-4] == '.' )
                    {
                        TESTVECTOR.push_back(root+'/'+temp);
                    }
                    else
                    {
                        ParseDirectory(root+'/'+temp);
                    }
                }
            }
        }
    }

    closedir(dir);

}
```

Figure 4.1, Directory crawling for '.tst' filesS

4.3 Test Execution

4.3.1 Technologies Used

Uses the stdlib.h library to make system calls.
Uses a vector to hold test cases and their results.

4.3.2 Phase Overview

This component has been created for the initial deployment of V1.x

4.3.3 Design Details

For each test case:

- Get input and output file names
- Execute program against test case
- Diff output file and expected answer file
- Determine pass/fail, add to vector

4.4 Result Logging

4.4.1 Technologies Used

Uses the ctime library to generate test run timestamps.
Uses a vector to store test results.

4.4.2 Design Details

```
#include <ctime>
#include <vector>

void WriteLog(string prog)
{
    // make name / date log.
    time_t now;
    time(&now);
    string currTime = ctime(&now);
    string name = prog + " " + currTime.substr(0,currTime.length() - 2) + ".log";
    ofstream log(name.c_str());

    for (int i = 0; i < TESTVECTOR.size(); i+=1)
    {
        log << TESTVECTOR[i] << endl; // flush buffer as long strings
    }

    // now push stats
    log << "             |" << endl;
    log << "Bottom Line V" << endl;
    log << "-------------------------------------------------------------------------------

    log << "Correct: " << CORRECTTESTS << "\nFailed: " << TESTVECTOR.size() - CORRECTTESTS
    log << "Success Rate: " << CORRECTTESTS/ TESTVECTOR.size() * 100 << "%" << endl;
}
```

5

System and Unit Testing

This section describes the approach taken with regard to system and unit testing.

5.1 Overview

Provides a brief overview of the testing approach, testing frameworks, and general how testing is/will be done to provide a measure of success for the system.

5.2 Dependencies

The manufature of this software was designed to presume that all tests will be located at either the current directory level of execution or at any contained, non-hidden or write protected directories. It furter assumes that there are only valid pairings of tests and answer files to validate the tested program's responses and will labeled respectively with '.tst' or '.ans'. Finally, this program is only intended to be run on program source files that both compile and do not break execution (Exception, Segmentation Fault, et cetera).

5.3 Test Setup and Execution

The initial tests were constructed by our customer to fascilitate the design and development process. All of these such tests were acompanied by accurate answer files. However, as a test development system, additional test had to be made locally to test for failures outside of the scope of the customer as limits needed to be verified against values and data types. All tests that caused any exception or segmentation fault were deemed unsatisfactory and were removed from further testing.

These tests were logged through out the execution process and comprise the final log summary that is titled with a time stamp and the name of the program being tested. A '.log' extension was also appended to illustrate that the file is infact log data from execution.

6

Development Environment

This section describes to any user how to set up a development environment for modification and extension of existing code. This information can also be used for developing a "release" version of Lounge Against The Machine's code with which to execute tests.

6.1 Development IDE and Tools

Any current Linux distrobution with a gcc compiler atleast at version three should build and run the program. Editing the program can be done with a text editor such as Gedit, VIM, or Emacs.

6.2 Source Control

Source control was done through GitHub both on windows and linux environments while leveraging git for the actual control mechanism. SVN is also supported through GitHub, however for ease and understaning of previous tools, git was sellected.

6.3 Dependencies

There are no specific dependencies with this development system aside from the development environment, namely a Linux with gcc version three or higher.

6.4 Build Environment

Program is compiled using a makefile, but can be compiled using "g++ -o grade grade.C".

6.5 Development Machine Setup

All members of Lounge Against The Machine leveraged their own computational systems from raspberry pi's to workstations to verify cross architecture compatability. No further setup was made.

7

Release – Setup – Deployment

The purpose of this section is to provide any particulars to the setup of a release build of Lounge Against The Machine's software.

7.1 Deployment Information and Dependencies

- Validate your linux installation has a current install of gcc (yum -install gcc / apt-get install gcc / emerge install gcc / pacman -S gcc)
- (Optional / Troubleshooting) validate linux headers are installed (above package manager with the arguments: linux-headers-$(uname -r))

7.2 Setup Information

Copy grade.C and the Makefile into the parent directory containing your target program's source code and test cases. Run make to compile the program. Then run the program with ./grade sourcefile

7.3 System Versioning Information

Whenever a working section of this project was complet and before the initial testing was complete, a version of 0.X where 'X' was a commit id from GitHub would be established. After further revision and at release the versioning was changed to V1.X. All further revisions are incouraged to continue to commit code and follow this versioning style.

8

User Documentation

This section contains the basis for any end user documentation for the system and covers the basic steps for setup and use of the system.

8.1 User Guide

Key to propper execution of the automated tester Grade by Lounge Against The Machine is the specification of a program's source file to be tested and the placement of the test files.

The run command from a command line if compiled with the included make file should be: ./grade <target to compile and test.cpp>

NOTE

The source code to test need not be in the directory path that execution occurs at.

All test files must be located at the current directory for execution (the '.' in the above './') or in any subsequent directory. For example the /home/user1 location is acceptable for executing a test, as long as all files reside in /home/user1 or /home/user1/tests or /home/user1/anything but a test/. Test files located out of path such as /temp/ will not be found.

Lounge Against The Machine and it's affiliates will not be liable (not be held responisble) for impropper use of this test suite.

8.1.1 Test Files

Files that contain the test cases are required to have '.tst' as an extension. The filename proceeding the extension will be the name of that test in the log file. These should be constructed to be the desired input to the program.

8.1.2 Answer Files

For each test case file a corresponding file is expected with the desired "True" output. This file must have the same name as the test case with an '.ans' extension.

8.1.3 Results

The results will be placed in a file named as the specified executed source, but with a .log extension and a date / time stamp in the directory of execution.

8.2 Installation Guide

All instalation is handled by meeting the requisites for release or development environment and execution of the make file or "g++ -o grade grade.C"

8.3 Programmer Manual

All instalation is handled by meeting the requisites for release or development environment and execution of the make file or "g++ -o grade grade.C"

9

Class Index

Not Applicable

Acknowledgement

Thanks to Dr. Logar for outlining requirements for the program and providing sample documentation.

Supporting Materials

Not Currently Applicable

Supporting Materials

Sprint Reports

9.1 Sprint Report #1

Sprint 1 consisted of program design, establishing source control, and implementing the major features. All of these tasks were accomplished on time.

9.2 Sprint Report #2

Sprint 2 consisted of integrating the major components, testing, and documentation.

Industrial Experience

9.3 Resumes

9.4 Industrial Experience Reports

9.4.1 Name1

9.4.2 Name2

9.4.3 Name3