Product Title

Testing System Project

LaTex Samurai

Julain Brackins Jonathan Dixon Hafiza Farzami

March 23, 2014

Contents

List of Figures

Mission

The mission statement for this project is to create a test suite designed to compile and run C++ projects with various test cases.

Document Preparation and Updates

Current Version [1.1.0]

Prepared By:
Hafiza Farzami

Revision History

Date	***Author***	***Version***	***Comments***
2/17/14	*Hafiza Farzami*	*1.0.0*	*Initial version*
3/23/14	*Hafiza Farzami*	*1.1.0*	*Edited version for Sprint 2*

1

Overview and concept of operations

This report covers the project overview, user stories, backlog, design and implementation, development environment, deployment, and documentation for the testing project.

1.1 Scope

This document covers the details of the project including its functionality, tools used, and the process that led to a solution.

1.2 Purpose

The purpose of this program is to run an entire directory of `.cpp` files with given test files, and grade them. There are certain test that are labeled as the critical tests; if a `.cpp` file does not pass one of the critical ones, there is no grade assigned. Otherwise, the percentage of passed test are calculated.

1.2.1 Traversing Subdirectories

Traversing subdirectories is one of the main components of this system. The program runs a `.cpp` file using test files, and the test files are stored in the same directory as student subdirectories.

1.2.2 Running the Program Using Test Cases

The software was designed in the Linux environment provided to the group by the university.

1.2.3 Generating Test Cases

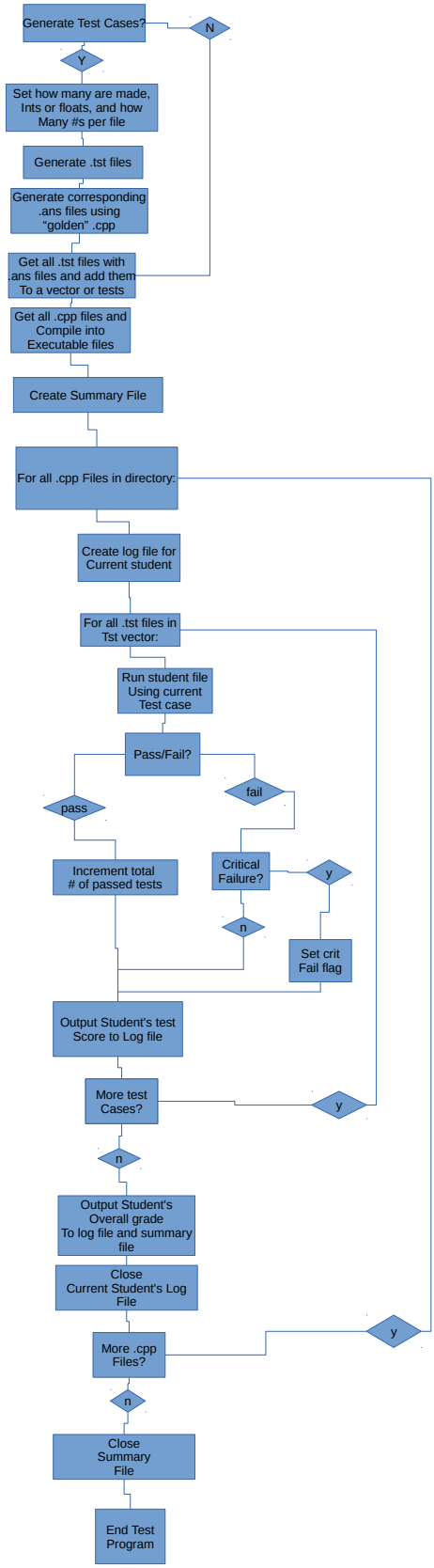
The software is capable of generating test cases if the user provides the constraints.

1.3 Systems Goals

The goal of this system is to make grading a less of a tedious task. This product grades an entire directory of `.cpp` files just by running the root directory. The product is built to test the `.cpp` file with all the given `.tst` test files in the current directory, and compare the results to the corresponding `.ans` files.

1.4 System Overview and Diagram

Here is a flow diagram showing the implementation process:


Generate Test Cases?
N
Y
Set how many are made, Ints or floats, and how Many #s per file
Generate .tst files
Generate corresponding .ans files using "golden" .cpp
Get all .tst files with .ans files and add them To a vector or tests
Get all .cpp files and Compile into Executable files
Create Summary File
For all .cpp Files in directory:
Create log file for Current student
For all .tst files in Tst vector:
Run student file Using current Test case
Pass/Fail?
fail
pass
Increment total # of passed tests
Critical Failure?
y
n
Set crit Fail flag
Output Student's test Score to Log file
More test Cases?
y
n
Output Student's Overall grade To log file and summary file
Close Current Student's Log File
More .cpp Files?
y
n
Close Summary File
End Test Program

2

Project Overview

2.1 Team Members and Roles

- Hafiza Farzami - Product Owner
- Julian Brackins - Scrum Master
- Jon Dixon - Technical Lead

2.2 Project Management Approach

The aproach taken to manage this project is `scrum`. The project is broken into tasks to be completed over two weeks sprint. The tasks are listed in `Spring Backlog` in trello. During the sprint, the team meets for ten minutes scrum meetings to explain their progress, next steps, and impediments.

2.3 Phase Overview

Once a team member starts a given task, then the task is moved from `Spring Backlog` to `In Progress`. A done task is then moved to `Ready for Testing` tab. After a completed task is test, then it is stamped as `Complete`. Then the member moves to the next task of higher priority.

3

User Stories, Backlog and Requirements

3.1 Overview

This section covers user stories, backlog and requirements for the system.

3.1.1 Scope

This document contains stakeholder information, initial user stories, requirements, proof of concept results, and various research task results.

3.1.2 Purpose of the System

The purpose of the product is to grade a `<filename>.cpp` file by running test files and comparing the results to answer files, and assigning percentage grade.

3.2 Stakeholder Information

This section would provide the basic description of all of the stakeholders for the project.

3.2.1 Customer or End User (Product Owner)

Hafiza Farzami is the product owner for this sprint, who is in contact with Dr. Logar regarding requirements, and with the scrum master and technical lead regarding the backlog.

3.2.2 Management or Instructor (Scrum Master)

Julian Brackins is the scrum master, who breaks the project into smaller tasks, and is in touch with both product owner and technical lead.

3.2.3 Developers –Testers

Jon Dixon is the technical lead for Sprint 2, and is in contact with both Brackins and Farzami regarding the requirements during scrum meetings and through trello notes. Due to the project size and limited number of people, the developers and testers are the same group of people.

3.3 Business Need

This product is essential for grading computer science programs focused on numerics. The user can save time and not have to do tedious, meticulous work while grading an entire class of students' programs.

3.4 Requirements and Design Constraints

This section covers the requirements and constraints in order to use this product.

3.4.1 System Requirements

This product runs on Linux machines.

3.4.2 Network Requirements

This software does not require internet connection.

3.4.3 Development Environment Requirements

There are not any development environment requirements.

3.4.4 Project Management Methodology

The method used to manage this project is `scrum`. The scrum master met with the product owner, and broke the tasks down to the technical lead. The team meets for ten minutes long scrum meetings to go over the progress, next steps, and impediments.

- Trello is used to keep track of the backlogs and sprint status
- Everyone has access to the Sprint and Product Backlogs
- This project will take three Sprints
- Each Sprint is two weeks long
- There are no restrictions on source control

3.5 User Stories

This section contains the user stories regarding functional requirements and how the team broke them down.

3.5.1 User Story

As user I want a testing system, in C++, so that given a directory of a class containing `<studentsNames>.cpp` files and a test files, it should grade them.

3.5.1.a User Story Breakdown #1

The program needs to be able to "crawl" through directories, be able to identify and run `<studentsNames>.cpp` files using the test files. In other words, the testing suite should be able to process the entire class.

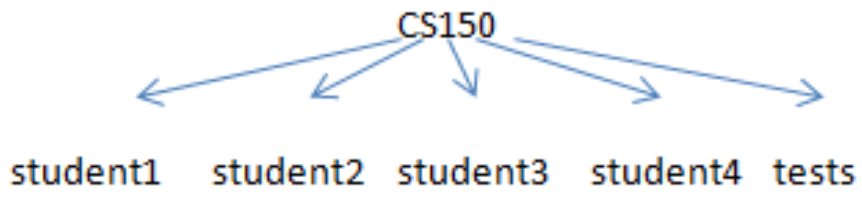


Figure 3.1: Directory Hiararchy Structure

3.5.1.b User Story Breakdown #2

The system needs to grade each individual student, store the result in a log file specific to a given student. The system also has to produce a summary file containing the list of grade percentages of all the students who passed necessary test cases, and the students who failed to pass an important critical case, with the word "FAILED" in the grade column.

3.5.1.c User Story Breakdown #3

The individual `<studentName>.log` files will be stored in that student's subdirectory. The summary `.log` file will be located in the root directory.

The summary file will have the following structure:

Student1 Name Percent grade `or` FAILED
Student2 Name Percent grade `or` FAILED
.
.
.
Studentn Name Percent grade `or` FAILED

3.5.1.d User Story Breakdown #4

The system should be able to generate test cases by asking the user about the conditions and constraints.

4

Design and Implementation

This section describes the design details for the overall system as well as individual major components. As a user, you will have a good understanding of the implemetation details without having to look into the code. Here is the

- Ask if the user if the program needs to generate test cases
- If the user's answer is yes:
  - Get the requirements for test cases
  - Generate `.tst` files and corresponding `.ans` files using `golden.cpp`
- (If no) get all the `.tst` files and add them to a vector
- Get all `.cpp` files and compile them into executable files
- Create summary file
- For each `.cpp` files in the directory:
  - Create `.log` file for current student
  - For all `.tst` file in the vector:
    - Run student file using current test case
    - If pass the increment the number of passed tests, and output the score to student's `.log` file
    - If fail, check if it is a critical test, if so, the student has failed, else output the score to the `.log` file
- Check if the user wants more cases
- If yes, then restart from the beginning
- If no:
  - Output student's overall grade to summary file
  - Close current student's `.log` file
- Check if there are more `.cpp` files to be processed
- If yes, then repeat the previous steps
- If no:
  - Close summary file
  - End test program

4.1 Traversing Subdirectories

4.1.1 Technologies Used

The dirent.h library is used for traversing subdirectories.

4.1.2 Design Details - Adding Test Cases to a Vector

```
void drct_recur (char * buffer)
{
    get_folders ( buffer );
    get_files (  buffer );
}

void get_folders( char * buffer )
{
    DIR *a_folder;
    struct dirent *dir_handle;
    vector<string> paths;
    bool subdir = false;
    int attrib;
    char path[ PATH_MAX ] = "";
    strcat( path, "~" );
    strcat( path, buffer );
    a_folder = opendir( buffer );
    if ( a_folder == NULL ) // call directory
    {
        return;
    }

    dir_handle = readdir( a_folder );
    if ( dir_handle == NULL )
    {
        return;
    }
    strcpy( path, buffer );
    // search for and step into folders
    do
    {
        attrib = ( int )dir_handle->d_type;
        if ( attrib == 4 && strcmp( dir_handle->d_name, "." ) != 0
             && strcmp( dir_handle->d_name, ".." ) != 0)
        {
            // set to true when we find and step into a folder
            subdir = true;
            char name[ PATH_MAX ];
            strcpy( name, dir_handle->d_name );
            strcat( path, "/" );
            strcat( path, dir_handle->d_name );
            paths.push_back( path );
        }
        if ( subdir )
        {
            strcat( path, "/.." );
```

```
            getcwd( path, sizeof( path ));
        }
        // reset test variable that determines
        // if we found and processed a folder
        subdir = false;
        // reset path
        strcpy( path, buffer );
    } while ( ( dir_handle = readdir( a_folder ) )!= NULL );

    while( !paths.empty() )
    {
        string temp =  paths.back();
        paths.pop_back();
        char pth[ PATH_MAX ];
        strcpy( pth, temp.c_str() );
        drct_recur( pth );
    }

    closedir( a_folder );
}

void get_files( char * buffer )
{
    DIR *a_file;
    struct dirent *dir_handle;
    a_file = opendir( buffer );
    dir_handle = readdir( a_file );
    string ext = dir_handle->d_name;
    string path;
    string name = dir_handle->d_name;

    if ( dir_handle != NULL )
    {
        do // search for files with "tst" extension
        {
            path = buffer;
            name = dir_handle->d_name;
            /*Check to see if the file has an extension BUT special case
              so that test.cpp file is not added to the compiled file stack.*/
            if( name.find_last_of( "." ) != string::npos && name != "test.cpp" )
                ext = name.substr( name.find_last_of( "." ) );
            else
                ext = "";
            if ( 8 == ( int )dir_handle->d_type
                && ( ext == ".tst"  || ext == ".ans" || ext == ".cpp" ) )
            {
                path += ( "/" + name );
                if( ext == ".tst" )
                   tstLocations.push_back( path );
                else if( ext == ".ans" )
              ansLocations.push_back( path );
                else if(ext == ".cpp")
              cppLocations.push_back( path );
            }
```

```
        }
        while ( ( dir_handle = readdir( a_file ) ) != NULL );
    }

    closedir( a_file );
}
```

4.2 Running the Program Using Test Cases

4.2.1 Technologies Used

The software was designed in the Linux Environment provided to the group by the University.

4.2.2 Design Details - Running Files and Comparing to Test Case

```
int run_file(string cpp_file, string test_case) //case_num
{
    string case_out(case_name(test_case, "out"));
    //set up piping buffers
    string buffer1("");
    string buffer2(" &>/dev/null < ");
    string buffer3(" > ");

    buffer1 += remove_extension(cpp_file);
    buffer1 += buffer2;
    buffer1 += test_case;
    buffer1 += buffer3;
    buffer1 += case_out;

    system(buffer1.c_str());

    //0 = Fail, 1 = Pass
    return result_compare(test_case);
}

int result_compare(string test_file)
{
    int length;
    ifstream fin;

    string case_out(case_name(test_file, "out"));
    string case_ans(case_name(test_file, "ans"));
    string case_tmp(case_name(test_file, "tmp"));   //create temp file

    //perform diff command
    string buffer("diff ");
    buffer += case_out + " " + case_ans + " > " + case_tmp;
    system(buffer.c_str());
    fin.open(case_tmp.c_str(), ios::binary);    //open file
    fin.seekg(0, ios::end);                     //cursor at EOF
    length = fin.tellg();                       //find cursor position
    fin.close();
    buffer = "rm " + case_tmp;
```

```
    system(buffer.c_str());
    buffer = "rm " + case_out;
    system(buffer.c_str());
    if ( length == 0 ) //File is empty, no diff between .ans and .tmp
        return 1;
    else
        return 0;
}
```

4.3 Generating Test Cases

4.3.1 Design Details - Test Case Generation

```
void genTstCases()
{
   bool validnumToGen = false, validfilesToGen = false;
   char likeToGen;
   char typeToGen;
   int numToGen;
   int filesToGen;

   //seed random number generator
   srand(time(0));

   //kind of a pretty looking menu
   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::" << endl;
   cout << ":          TEST CASE GENERATION PHASE           :" << endl;
   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::" << endl;

   //get user input
   cout << "Would you like to generate test cases? (y/n): ";
   cin >> likeToGen;

   //clear anything after the initial character
   cin.clear();
   cin.ignore( 500, '\n' );


   likeToGen = tolower( likeToGen );

   //check for validity
   while( likeToGen != 'y' && likeToGen != 'n' )
   {
      cout << "*** Please enter a valid response. (y/n): ";
      cin >> likeToGen;
   }

   if( likeToGen == 'n' )
      return;

   //find out if they want ints or floats
   cout << "Would you like to generate ints or floats? (i/f): ";
   cin >> typeToGen;
```

```
//clear after the first letter
cin.clear();
cin.ignore( 500, '\n' );

typeToGen = tolower( typeToGen );

//check for validity
while( typeToGen != 'i' && typeToGen != 'f' )
{
   cout << "*** Please enter a valid response. (i/f): " << endl;
   cin >> typeToGen;
}

//get how many files should be generated
while( !validfilesToGen )
{
   cout << "How many files would you like to generate? (1-100): ";
   cin >> filesToGen;

   //make sure they enter an int
   while( cin.fail() )
   {
      //if a char was entered, this clears input, gets a new one
      cin.clear();
      cin.ignore( 500, '\n' );
      cout << "*** Enter a number, please (1-100): ";
      cin >> filesToGen;
   }

   //check for a valid number of files
   if( filesToGen > 100 || filesToGen < 1 )
   {
      cout << "*** Please enter a valid number of files. (1-100)" << endl;
   }
   else
      validfilesToGen = true;
}

//get how many numbers to generate
while( !validnumToGen )
{
   cout << "How many numbers would you like to generate? (0-200): ";
   cin >> numToGen;

   //make sure they enter an int
   while( cin.fail() )
   {
      //if a char was entered, clear input, get new input
      cin.clear();
      cin.ignore( 500, '\n' );
      cout << "*** Enter a number, please (1-200): ";
      cin >> numToGen;
   }
```

```
      //make sure number is in the correct range
      if( numToGen > 200 || numToGen < 1 )
      {
         cout << "*** Please enter a valid number. (1-200)" << endl;
      }
      else
         validnumToGen = true;
   }

   //now, it is important to create these files in the 'tests' subdirectory
   //if the directory is successfully created, move into it
   if( !system("mkdir tests &>/dev/null") )
   {
      cout << "\"tests\" directory created!" << endl;
      chdir( "tests" );
   }
   //if the directory already exists, move into it, and clear the old test cases
   else
   {
      cout << "\"tests\" directory exists!" << endl;
      chdir( "tests" );
      system( "rm -f *" );
   }

   //loop once for each file to be made
   for( int i = 1; i <= filesToGen; i++ )
   {
      stringstream temp;
      ofstream fout;

      //assemble the filename
      temp << "test" << i << ".tst";

      //open the file
      fout.open(temp.str().c_str());

      //generate all of the random numbers
      for( int j = 0; j < numToGen; j++ )
      {
         if( typeToGen == 'f' )
            fout << ((float)rand()/(float)(RAND_MAX)) * 2000 << endl;
         else
            fout << rand() % 2000 << endl;
      }

      //close file output
      fout.close();
   }

   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::" << endl;
   cout << ":       TEST CASE GENERATION SUCCESSFUL          :" << endl;
   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::" << endl;

   //move back to the parent directory
   chdir( ".." );
```

```
   //make all of the answers to the test cases
   make_ans( filesToGen );
}

void make_ans( int filesToGen )
{
   char buffer[PATH_MAX];
   DIR *a_file;
   struct dirent *dir_handle;
   string ext;
   string path;
   string name;
   string command;

   //get cwd into a character buffer
   getcwd( buffer, sizeof(buffer) );
   a_file = opendir( buffer );
   dir_handle = readdir( a_file );
   ext = dir_handle->d_name;
   name = dir_handle->d_name;

   //make sure directory handle is not null
   if( dir_handle != NULL )
   {
      do //search for .cpp files
      {
         //set the file path equal to the buffer, and name to the directory name
         path = buffer;
         name = dir_handle->d_name;

         //here, we exclude test.cpp, because we don't want to compile our own code
         if( name.find_last_of(".") != string::npos && name != "test.cpp" )
            ext = name.substr( name.find_last_of(".") );
         else
            ext = "";

         //make sure we're looking at a cpp file
         if( 8 == (int)dir_handle->d_type && ext == ".cpp" )
         {
            //finish constructing the path
            path += ( "/" + name );

            //assemble the command to compile the .cpp
            command = "g++ ";
            command += path;
            command += " -o key";
            system( command.c_str() );
         }
      }while( ( dir_handle = readdir( a_file ) ) != NULL );
   }

   //for each file we have generated
   for( int i = 1; i <= filesToGen; i++ )
   {
```

```
        //run the correct program, piping from the input file to a .ans file
        stringstream runcommand;
        runcommand << "./key < tests/test" << i << ".tst > tests/test" << i << ".ans";

        system( runcommand.str().c_str() );
    }
    system("rm -f key");

}
```

5

System and Unit Testing

Testing was first done on small parts of the program, and as we added more, we tested larger parts of the program.

5.1 Overview

In general, we began testing on a lower level by testing our capabilities for rerouting input and output. Once we managed that, we moved on to comparing output files to the test case files. Once Dr. Logar released the test cases, we created a function to confirm directory traversals. Then, when we combined the two, our program functioned correctly and we were able to add the finishing touches.

5.2 Dependencies

No dependencies.

5.3 Test Setup and Execution

In earlier phases of the sprint, a simple "Hello World!" .cpp file was created to test the usage of system commands to compile and run a program within our testing software.

6

Development Environment

The basic purpose for this section is to give a developer all of the necessary information to setup their development environment to run, test, and/or develop.

6.1 Development IDE and Tools

Code was implemented and modified through the use of gedit in Linux and the text editor available on the git website.

6.2 Source Control

Git was used for source control. It was set up on GitHub, and can be accessed via the command
git connect https://github.com/jbrackins/CSC470.

6.3 Dependencies

An internet connection to access the Git repository will be necessary.

6.4 Build Environment

The program should be built using the g++ compiler on linux.

7

Release – Setup – Deployment

7.1 Deployment Information and Dependencies

There are no specific dependencies for this program.

7.2 Setup Information

The program is built with the g++ compiler on Linux.

7.3 System Versioning Information

The program will be versioned depending on the current Agile sprint.

8

User Documentation

This section contains the user guide, user documentation.

8.1 User Guide

This product is to make grading an easy, not-so-meticulous task. Here is the structure you need to have your `.cpp` files:

- The root directory
- Individual subdirectories for each student (Use students' names as directory names)
- A tests' subdirectory

This software will create individual log files for each student in the appropriate student's subdirectory. It will also create a summary file in the root directory containing all the students' names with their earned grades. If they have not passed a critical test, they will get `FAILED` for their grade entity.

Acknowledgement

Figure 3.1 is taken from Dr. Logar's sprint 2 FAQ sheet.

Supporting Materials

There are no supporting materials to go along with the product for this sprint. Supporting material will be added if and when needed.

Sprint Reports

8.1 Sprint Report

This is the second sprint for the testing suite system. The resulting product is fucntioning well and meets the user requirements. There are no bugs related to user stories, and it performs the required functionality specified by the user successfully.

The scrum roles was changed in this sprint from the previous one, so we all experience being in different roles. We will rotate roles again in the next sprint for the same reason.