# Program Tester Sprint 2

**Software Engineering Spring 2014 Documentation**

Software Engineering Adventure Line

Erik Hattervig Andrew Koc Jonathan Tomes

April 11, 2014

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Mission

To improve the unimprovable.

# Document Preparation and Updates

Current Version [2.0.0]

*Prepared By:*
*Erik Hattervig*
*Andrew Koc*
*Jonathan Tomes*

***Revision History***

| ***Date*** | ***Author*** | ***Version*** | ***Comments*** |
|---|---|---|---|
| ***2/19/14*** | *Samuel Carroll* | *1.0.0* | *Initial version* |
| ***3/23/14*** | *Jonathan Tomesl* | *2.0.0* | *Sprint 2 Version* |
| | | | |
| | | | |
| | | | |
| | | | |

# 1

# Overview and concept of operations

This document will look at the Software Engineering Adventure Line team second sprint for building a program tester. Looking at the team members and their roles, the project management that we used, the sprint retrospective, any terminology or acronyms that we use. Next we will look at the user stories, backlog and requirements of the program. Then we will look at the design and implementation of the program focusing on major pieces of the code. The next section will look at system and unit testing of our program, followed by development environment, and release, setup and deployment of our program. We will finish this document with a look at a user documentation (including a user guide, installation guide and programmer manual) class index and class documnetation.

## 1.1 Scope

This document will cover the second sprint of our program tester built for Dr. Logar's software engineering class in Spring 2014 .

## 1.2 Purpose

The purpose of this program is to compile, run and test the simple programs created by others.

The program willl also offer to generate additonal random tests.

### 1.2.1 Normal Run

The programs it tests are supposed to be guarenteed to compile and run correctly. It will then search through a root directory, find each studen'ts subdirectory, compile their code into the root directory, open log files for the studen and the class as a whole. Student logs will go into the student's directory and the class log will be in the root. The student logs will have the results of each test and a final score. The final score will be based on the precent of the tests passed. Unless the program fails a critcal test. If a student fails a test labeled as "crit_(something).tst", that student will be immediately marked as "FAILED".

### 1.2.2 Generate Tests

The program will ask for a number tests to generate, the number of inputs for each test, and the data type for the tests. The program will then generate a "GeneratedTests" directory in the test directory of the root folder. At which point, the program will generate the number of tests specified. If the program has previously generated tests, it will then remove the past generated tests and create new ones.

## 1.3 Systems Goals

1) Find student programs to compile.

2) Find the tests and use them to test the found programs.

3) Generate new random test cases.

## 1.4 System Overview and Diagram

The program will be started on the command line. You may specify a starting directory or not, if not the program will assume that it is running in the directory it is supposed to be searching in. It will then display a simple menu of Run, which tells the program to run normaly, searching for student programs and testing them. Or Generate, which will tell the program to generate tests. It will ask the user for the number of test cases, number of inputs, and the data type to use. See Figure 1.1.

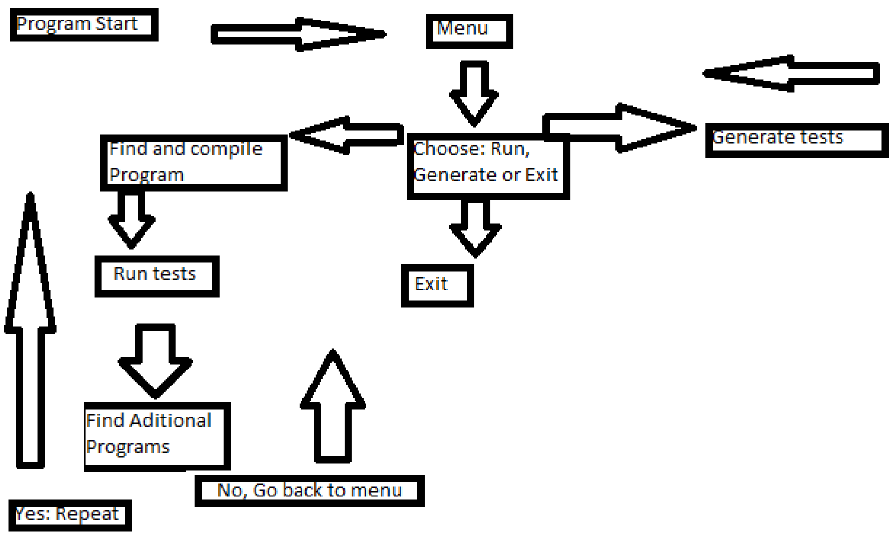

Figure 1.1: System Diagram

## 1.5 Technologies Overview

The main programing langue used was C++, and was compiled with g++ on a linux environment.

# 2

# Project Overview

This section provides information regarding the team and their roles, how the project was managed, and any relatively unkown terminology or acronyms.

## 2.1 Team Members and Roles

The team members were Erik Hattervig, the Product Owner; Andrew Koc, Tech Lead; and Jonathan Tomes, Scrum Master.

## 2.2 Project Management Approach

The program was created through the Scrum Agile Approach. The sprint length for this project was 2 weeks. We began with a meeting to decide the user needs and split the program accordingly. Each of us would code different parts of the program and then we would all test and re-code as needed. placed on Trello to help design break points to split up the program between team members.

The code was stored, backed up, and shared through git hub. The back log and ownership was tracked through Trello. The user stories were condensed and

## 2.3 Phase Overview

Sprint 1 (Done by the team We Can't Follow Directions):

This program started in a gathering information phase, the first thing we had to do as a team was generate several questions to ask the customer (Dr. Logar) about the software that she wanted. The next phase was asking if it could be done, as most of the code had been written at one point or another for our team, we decided that it was a program that could be written. The next phase was breaking the program into smaller milestones, we decided that the program would need to compile source code, search for test cases, run compiled code with test case, output test case results to a file, compare that output file to an expected output file, and then write to a file if the two files match or not, if they do we write the text case then passed to a log file, if they don't we write the text case and failed to the log file. Once we are done traversing the directory we write the total number of passed, failed and percentages of each.

Sprint 2( Done by the team The Software Engineering Adventure Line):

First we began with an information gathering session by Dr. Logar and the product owners. This is where we found out the goals of the program and the desired results. After this we were assigned a team's work from the previous sprint, and evaluted it. Unfortunately we had to do a lot of revising and creating new functions because the old cold was not very modular. So we divided up the work of reworking the code and did our parts. After that we gathered and came up with how to meet the new requirements and worked out what we needed to code, ask the user for, and finished coding the program.

## 2.4 Terminology and Acronyms

none

# 3

# User Stories, Backlog and Requirements

## 3.1 Overview

This section will look at the userstories of the development of the program, the requirements of the program, the proof o fconcept results, and research task results. It will also look at the reason for developing this software.

The userstories Compile source code from tester. Run compiled code from tester. Compare two files from tester to see if they are different. Traverse a directory looking for test cases. Write program output to a file. Write to human readable log file.

Below: list, describe, and define the requirements in this chapter. There could be any number of subsections to help provide the necessary level of detail.

### 3.1.1 Scope

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

### 3.1.2 Purpose of the System

To test a set of basic computer programs written in the C++ language so a grade can quickly be assigned to a class of students. The system will also be able to generate random test cases from user input.

## 3.2 Stakeholder Information

The person most interested in this project is Dr. Logar (the end user) who is looking to quickly, and easily, grade programs that are turned in by her CSC 150 students.

### 3.2.1 Customer or End User (Product Owner)

The product owner on the first sprint was Samuel Carroll, he created a list with his teammates and on a day, selected by Dr. Logar, met with her and the other teams' product owner to determine exatly what Dr. Logar wanted. Samuel was also the team member most involved in keeping the Trello board up to date.

The product owner on the second sprint was Erik Hattervig, he gathered the nessary requirments of the second sprint from Dr Logar, relaied the information to his teammates, and set up the product backlog on the Trello board.

### 3.2.2 Management or Instructor (Scrum Master)

The scrum master for the first sprint was Colter Assman, his duties were to ensure that the project stayed on track and if any team member ran into some issues he would help them get back on track. Colter also was responsible for the running of the daily scrum.

The scrum master for the second sprint was Jonathon Tomes, he was in charge of managing the scheduling for the team members, creating spring schedules, and moving tasks from the product backlog to the sprint backlog. He also lead the scrum meetings.

### 3.2.3 Investors

There were no investors for our first sprint.

### 3.2.4 Developers –Testers

Shaun Gruenig was the biggest tester for our program. As the team technical lead he kept us updated on if the project was running as we expected it to, and would often debug the issues our code had.

For the second sprint, all three members of the team tested each other's code and gave feed back on bugs and code quality via Github.

## 3.3 Business Need

Currently many computer science teachers have to write each test case out by hand. This is a very time consuming endeavor (especially considering how many students each one has), so this program would enable them to write a test cases which will then be input to our program. Quickly and accurately giving grades to students.

## 3.4 Requirements and Design Constraints

The requirements was that our tester run in the Linux environment. We also needed this program to be ready to send out by the time the first CSC 150 program was due, therefore we only had about two weeks to write and implement this code.

For sprint two we were limited to a set of features that were given to us by Dr. Logar.

### 3.4.1 System Requirements

The program must be able to run on a Linux machine, using the GNU operating system. therefore the code must able to compile using the GNU compiler. This means all of our code must be executable on Linux machines. Of course we may have had to write this program for another system, but the Linux environment is the nicest one for us to use.

### 3.4.2 Network Requirements

No network requirements were needed.

### 3.4.3 Development Environment Requirements

Linux/GNU system should be able to run our tester.

### 3.4.4 Project Management Methodology

The stakeholders had several requests on how the project was implemented. Including what to use to keep track of backlogs and sprint status, which parties had access to the sprint and produt backlogs, how many sprints will be used for this project, and restrictions on the source control.

- Trello was used to keep track of the backlogs and sprint status
- All parties will have access to the Sprint and Product Backlogs
- Three sprints will encompass this project

- The sprints will vary in length a little bit but be about 2-3 weeks in general
- Github was used for source control on the second sprint

## 3.5 User Stories

This section contains information about the user stories (what the program must be able to do, and what the user should have to do).

### 3.5.1 Compile and Run Source Code

The program must be able to compile and run source code found in the directory

For sprint two we now must be able to compile and run test on an entire class directory.

### 3.5.2 Write Pass/Fail and Percentages to Log File

This program must be able to write output to a log file and to keep track of the total number passed cases and the total number of failed cases.

For sprint two we need to be able to keep track of both individual records as well as the whole class.

### 3.5.3 Compare Output with expected output

The program must be able to compare the output that we get after running a test case to the output that we expect to get from the test case. The expected output will be found when searching the directory.

### 3.5.4 Searching/Traversing the Directory

The program must be able to search through all the files and sub-directories of the directory that we are currently in.

For sprint two this is used in the test directory.

### 3.5.5 Invoking the Program

The user must be able to run our program by typing ‘test ¡directory¿’.

## 3.6 Research or Proof of Concept Results

Most of the code had been written by our team before. We knew how to run the system function in C++ to invoke a system command. We had built a directory crawler in an earlier class (though in Windows so some modification had to take place). All in all starting the program we knew we could complete it.

For sprint two much of the same concepts applied for the new features that were added such as test case generation.

## 3.7 Supporting Material

In the man pages for the diff function it shows us that it returns one of three values and the case those values are returned, a zero if there is no difference between the two files, a one if there is a difference between the two files, or a two if something went wrong (doesn’t happen often)

# 4

# Design and Implementation

This section will describe the design details for each of the major components in the system.

## 4.1 Student Directory Crawl

### 4.1.1 Technologies Used

This uses the C++ standard library namely the dirent.h library.

### 4.1.2 Component Overview

This function creates a log file for the entire class in the root. Then it searches for subdirectories other than the Test directory, which contains only the .tst and .ans files. When it finds an applicable subdirectory it changes in, creates a log file for that student, and compiles the .cpp file found within. Then it runs the test directory crawl on the compiled program. After it has returned from the test crawl it does the final log write for the student and writes to the class log as well.

## 4.2 Compile a Program

### 4.2.1 Technologies Used

This uses the C++ standard library, system calls, and the g++ compiler.

### 4.2.2 Component Overview

This program recives a string which determines the name of the executable to be produced. Then from the current directory finds and compiles the first .cpp file found.

## 4.3 Test Directory Crawl

### 4.3.1 Technologies Used

This uses the C++ standard library namely the dirent.h library.

### 4.3.2 Component Overview

This function recusivly searches the test directory and upon finding a .tst file will run the function to test the executable against it. When the function finds a subdirectory it calls itself on that subdirectory, allowing it to fully search for all of the test files.

## 4.4 Run Test Case

### 4.4.1 Technologies Used

This uses the C++ standard library and system calls to execute the program.

### 4.4.2 PComponent Overview

This function takes the name of the .tst file and generates the name of the .ans and .out files. The .out file being created in the same directory as the one the .cpp file was found in, and the .ans in the same one as the .tst file was found in. After the names are generated, using the system command the executable is run with the .tst file used as input and the output being piped to the .out file. Once it has completed the RunDiff function is called to determine if the program executed properly. The log file and record are updated accordingly.

## 4.5 Run Difference Function

### 4.5.1 Technologies Used

This uses the C++, and system call function, as well as the diff function in Linux/GNU

### 4.5.2 Component Overview

This function will create a function that will run the diff function in Linux/GNU via the system call. First however it must create the string that will invoke the call, and take in the file names. Then it will pass inform others if the test case passed or failed so they can handle the information accourdingly. Also this doesn't print anything to the screen if there is a difference between the two files.

### 4.5.3 Phase Overview

This function can create the command string that will compare two files. This function will call the diff function. This function will not output anything to the screen if there is a difference between the two files. This function informs if there is, or is not, a difference between the two files.

## 4.6 Student Log Write

### 4.6.1 Technologies Used

This uses the C++ standard library, namely the fstream library.

### 4.6.2 Component Overview

This function takes an ofstream, the name of the test, and the success or failure of that test, and writes the formated results to the ofstream.

## 4.7 Final Log Write

### 4.7.1 Technologies Used

This uses the C++ standard library, namely the fstream library.

### 4.7.2 Component Overview

This function takes an ofstream, the name of the student, and a record of the pass and failure of all of the test cases. This function writes the percent of the tests passed, assuming that no critical tests were failed. If a critical test was failed then only "FAILED" is written, rather than the percentage. This function is used to write the last line of each student log file, as well as each line in the class log file.

# 5

# System and Unit Testing

There was no strict testing format for this project.

## 5.1 Overview

Each programmer tested their code before pushing it up to the git repository. After compatible parts were pushed up, they were combined and tested, to ensure they worked correctly together.

## 5.2 Dependencies

All of the tests depend on the g++ compiler, and linux platform. The code needs to compile a C++ file inorder to run and test that file.

## 5.3 Test Setup and Execution

The test cases were provided by Dr. Logar. These test cases were used to test the program. Then after we were satisfied with the programs performance on the test case, the file of the test cases was changed to ensure it still ran, more or less, the same, differences in the order the test cases were found being unimportant. Several different test sets were used to test the program.

# 6

# Development Environment

This section will provide all the neccessary information to run, test and develop our source code.

## 6.1 Development IDE and Tools

The code was written in simple text editors, and requires no special IDE's or tools. All the code was tested on a Linux machine using the g++ compiler. The debug tool gdb was used to debug the code when necessary.

## 6.2 Source Control

We used github for source control. The repository can be found at this url:
https://github.com/TheSoftwareEngineeringAdventureLine/ProgramTesterStage2.git

## 6.3 Dependencies

This program is dependent on the C++ Standard Library as well as the g++ compiler on a Linux system.

## 6.4 Build Environment

The execuatble is built by the g++ compiler. You can either compile it manually, using the command:

```
g++ -o tester ProgramTester.cpp
```

Or by using the following make file:

```
# compiler
CC = g++

# compiler options
CFLAGS = -c -Wall

all: tester

tester: tester.o
   $(CC) -lm tester.o -o ProgramTester

tester.o: ProgramTester.cpp
   $(CC) $(CFLAGS) ProgramTester.cpp

clean:
```

```
rm -rf *o tester
```

## 6.5 Development Machine Setup

For the Development Machine set up any Linux machine with the GNU compiler should work.

# 7

# Release – Setup – Deployment

The source code will be provided to Dr. Logar using email or the Submit It! page of the SDSMT Math and Computer Science webpage.

## 7.1 Deployment Information and Dependencies

Will need g++ and a linux like environment to run our program.

## 7.2 Setup Information

By navigating to the directory and running make, the program will be installed.

## 7.3 System Versioning Information

The system is in 2.0.0 stage, or second release.

# 8

# User Documentation

## 8.1 User Guide

To use our program simply run it on the command line. You may specify a starting directory as a command line argument, or the program will assume to run in the directory it already is in. Next a menu will apear and ask for one of the options.

Run will let the program run normaly, it will search through the directory and find student directories containing the student source code. It will compile the code to the root directory. It will then search through the test directory in the root to find test casses to run on the student programs. It will log the result of each test to a student log file located in the student directory, and the final result will be repeated in an overal class log located in the starting directory. If the student fails a test labeled "crit_(something).tst" the student will immediately fail and no more testing will be done.

Generate will then prompt for a number of test cases, the number of inputs per test, and finally a data type to use for each test. It will then create a sub-directory called "Generated" in the test sub-directory of the root where the program was prompted to go or started in. If it had previously created tests, it will clear the directory and create a new one and all new test cases.

Exit will exit the program.

## 8.2 Installation Guide

1) Open the terminal and navigate to the directory containing our program source code and the makefile.

2) Run make.

## 8.3 Programmer Manual

# 9

# Class Index

## 9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Acknowledgement

Thanks to Dr. Logar, for her help, knowledge and guidance.

## Supporting Materials

There are, for this sprint, no supporting materials.

Still no supporing materials for sprint 2

# Sprint Reports

## 9.1 Sprint Report #1

The testing function can traverse a directory looking for source code. Compile that code, then look for test cases and run the compiled code with the given test case outputting the result in a new document. It can then compare that document with the expected results document and write to the log file if it passed or failed. Then it will continue looking for test cases until all the test cases in the root folder have been found. The tester will then write the total number of test cases passed, total failed and the percentage passed and failed.

## 9.2 Sprint Report #2

Had to redo a lot of the code. Most of it wasn't split into seperate functions and some of it was misdocumented. After getting around that, we got it to travese the root directory, finding the student directories and testing them against the tests located in the test directory. It can generate random tests (the number of and data types specified by the user). Introduced a simple menu system to make it easier to generate then run tests. It will log the results of the testing each student into a student log located in their directory, and to a class log located in the root directory.

## 9.3 Sprint Report #3

# Industrial Experience

## 9.4 Resumes

## 9.5 Industrial Experience Reports

### 9.5.1 Colter Assman

### 9.5.2 Samuel Carroll

### 9.5.3 Shaun Greunig