Assignment Grader

Senior Design Final Documentation

Kernel_Panic

Ben Sherman Anthony Morast James Tillma

February 19, 2014

Contents

List of Figures

List of Algorithms

Mission

The mission for this project is to develop an assignments grader for programs. The target is CSC150 (or introductory level) assignments. This grader should be easy to use, intuitive, and bug free.

Document Preparation and Updates

Current Version [1.0.0]

Prepared By:
Anthony Morast
Ben Sherman
James Tillma

Revision History

Date	***Author***	***Version***	***Comments***
2/2/12	*All*	*1.0.0*	*Initial version. Combined parts of existing documentation for first full version of document*

1

Overview and concept of operations

The purpose of this document is to outline the project. That is, it's purpose, the niche it is meant to fill, and it's future.

1.1 Scope

This document covers the details of the project. It describes how it works, what tools it uses, and what has been done so far.

1.2 Purpose

The purpose is to make grading easier for professors and teaching assistants (in particular at the introductory programming level).

1.2.1 Major System Component #1

Recursive File Searcher

1.2.2 Major System Component #2

Iterative Executable Tester

1.3 Systems Goals

The primary goal of the project is to aid in the grading of programs. Without a tool to do this, it can be very time-consuming to grade programs that are all meant to be "the same." This is the problem that Assignment Grader is mean to solve.

1.4 System Overview and Diagram

These are the main system components. For clarity, the student's code file has been included in this diagram even though it is an outside resource. See Figure **??** below.

1.5 Technologies Overview

Linux (Fedora 19, Ubuntu 13.1) Operating System: Used for basic runtime needs. G+ compiler: used to compile and run both our code and student code. For further information on these two system features, consult Linux documentation.

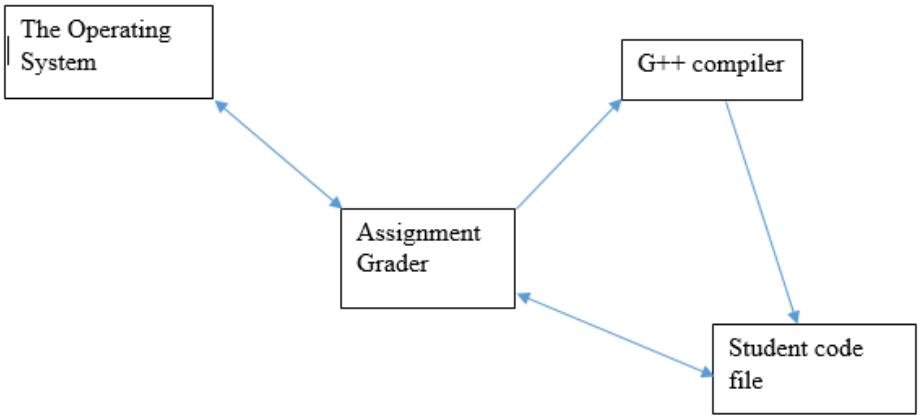

Figure 1.1: System Diagram

2

Project Overview

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

2.1 Team Members and Roles

Describe who was involved and what role(s) were played.

2.2 Project Management Approach

This section will provide an explanation of the basic approach to managing the project. Typically, this would detail how the project will be managed through a given Agile methodology. The sprint length (i.e. 2 weeks) and product backlog ownership and location (ex. Trello) are examples of what will be discussed. An overview of the system used to track sprint tasks, bug or trouble tickets, and user stories would be warranted.

2.3 Phase Overview

If the system will be implemented in phases, describe those phases/sub-phases (design, implementation, testing, delivery) and the various milestones in this section. This section should also contain a correlation between the phases of development and the associated versioning of the system, i.e. major version, minor version, revision.

2.4 Terminology and Acronyms

Provide a list of terms used in the document that warrant definition. Consider industry or domain specific terms and acronyms as well as system specific.

3

User Stories, Backlog and Requirements

3.1 Overview

This section contains an overview of the purpose of this document and the system developed by Kernal_Panic, important information for stakeholders, the niche the project is meant to fill, the requirements of the project, and some introductory user stories meant to be satisfied.

3.1.1 Scope

This particular document covers the entirety of the information required and discovered for the first stage of development.

3.1.2 Purpose of the System

The purpose is to aid professors and teaching assistants with grading.

3.2 Stakeholder Information

This section covers stakeholder information.

3.2.1 Customer or End User (Product Owner)

Ben Sherman is the product owner. The end-user (in this specific case) is Dr. Logar. The target end-user group is all professors and teaching assistants. Dr. Logar will interact via monitoring Trello and GitHub. Ben Sherman will interact more directly through scrum meetings and other miscellaneous project discussions and tasks.

3.2.2 Management or Instructor (Scrum Master)

James Tillma is the Scrum Master. He will interact by conducting scrum meetings and all other official discussions and task assignments. As Technical Lead, Anthony Morast will also play a role in assignments, timeline estimation, and other management tasks discussed at Scrum Meetings.

3.2.3 Investors

At this point there are no investors, besides those listed above who have donated, in the form of time, to the project.

3.2.4 Developers –Testers

In this case, the developers are the testers and each of the three members of Kernel_Panic are participants in both groups. The main manager of the product backlog (as it currently exists) is James Tillma because we do not know about the future of the product. When more becomes known, and the product backlog and sprint backlog gain more separation, management duties will fall to the current Project Mananger.

3.3 Business Need

The business need is that professors and teaching assistants currently spend vast amount of times grading programs that have fairly short and stable runtimes. This lends itself well to a basic automation system that require little to know user intervention.

3.4 Requirements and Design Constraints

This section details requirement for usage of the system.

3.4.1 System Requirements

The system requirements are mainly that it must be a Linux platform with the g++ compiler. The SDSMT distribution of Fedora 19 meets this requirement.

3.4.2 Network Requirements

This does not use an networking (unless the student program does) and therefore does not have any network requirements.

3.4.3 Development Environment Requirements

The development requirements are that it will be done in C++ and for a Linux environment. Assignment Grader is not expected to be cross-platform in any way for the first stage.

3.4.4 Project Management Methodology

Below are some questions and answers reguarding the project management methodology.

- What system will be used to keep track of the backlogs and sprint status?
  - Primarily Trello
- Will all parties have access to the Sprint and Product Backlogs?
  - All parties involved with it's development and initial release
- How many Sprints will encompass this particular project?
  - At this point that is unknown, however there will be at least 2
- How long are the Sprint Cycles?
  - Two weeks
- Are there restrictions on source control?
  - GitHub was recommended and that is what Kernel_Panic has used

3.5 User Stories

The following are the user stories thus far.

3.5.1 User Story #1

: As a professor I want to be able to automatically grade a student's source code so that I can save time.

3.5.1.a User Story #1 Breakdown

The highlight of this story is "automatically". The professor does not want to have to interact with the grading tool after it is run.

3.5.2 User Story #2

As a professor I want to be able to provide test cases with answers so that I can grade different program assignments.

3.5.2.a User Story #2 Breakdown

This story refers to assembling test cases and running the student code on those test cases

3.5.3 User Story #3

As a professor I want to be able to regrade a students source code without loosing the data from a previous grading so that I can see what changes upgrading test cases have.

3.5.3.a User Story #3 Breakdown

This story refers to saving data. Data from an old runtime should not be removed by a new runtime. Data should be appended to a neatly formatted file for viewing.

3.6 Research or Proof of Concept Results

There was a very small amount of research that needed to be done. This was limited to looking into redirected I/O on a Linux platform and compiling a code file from an external running program.

3.7 Supporting Material

There is no supporting material at this time.

4

Design and Implementation

This section is used to describe the design details for each of the major components in the system.

Algorithm 1 Find and all case file pairs

4.1 Major Component: Recursive File Search

Require: Working Directory Path
  get folders
  while all items in folder have not been checked **do**
    if item in directory is a folder **then**
      add folder path to a list as a string
    end if
  end while
  while list of found folder paths not empty **do**
    remove folder path from list and call this algorithm on the folders path.
  end while
  check files
  while all items in folder have not been checked **do**
    if item is a file AND files extension is ".tst" OR ".ans" **then**
      if ".tst" extension **then**
        add found file's path to answer list for later use
      end if
      if ".ans" extension **then**
        add found file's path to test list for later use
      end if
    end if
  end while

4.1.1 Technologies Used

Linux (Fedora 19, Ubuntu 13.1) Operating System: Used for basic runtime needs. G+ compiler: used to compile and run both our code and student code. For further information on these two system features, consult Linux documentation.

4.1.2 Component Overview

This component automatically traverse directory where program is being run from and look at that directory level and all levels below it for test and answer test case file pairs.

4.1.3 Phase Overview

1. Enter root directory.
2. Look for test case files at root directory level and deeper.
3. Store list of test case files for later use.

4.1.4 Data Flow Diagram

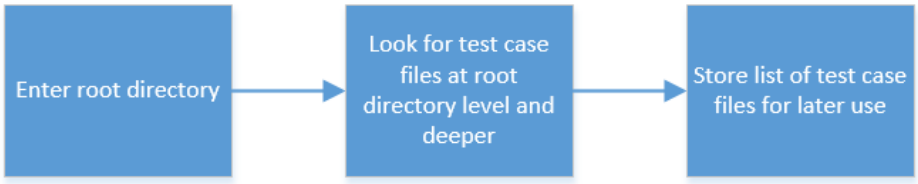

Figure 4.1: Recursive File Search Data Flow Diagram

4.1.5 Design Details

This component uses the C++ library *dirent.h* to access and traverse the root directory and its children. The root directory being the directory level from which the product is used. The algorithm looks for folders and then recursively calls its self on the path of found folders. This continues until no more children folders are found. Once this base case has been achieved the algorithm works its way out of the recursion by checking the folder it is in for files that are test cases and returns to the next higher directory level.

The algorithm finds pairs of test case files, namely [filename].tst and [filename].ans. Once the algorithm is completed a check is done to ensure that each ".tst" file has a matching ".ans" file.

4.2 Major Component: Iterative Executable Tester

Algorithm 2 Compile and test student programs

Require: valid and compile-able .cpp source code
  Compile user specified source code
  for each test case **do**
    input test case (".tst") file to students program
    compare students output to test case answer (".ans")
    if students output differs from test case answer **then**
      append student program failed to "[$source_code$].*log*" file
    else
      append student program passed to "[$source_code$].*log*" file
    end if
  end for

4.2.1 Technologies Used

Linux (Fedora 19, Ubuntu 13.1) Operating System: Used for basic runtime needs. G+ compiler: used to compile and run both our code and student code. For further information on these two system features, consult Linux documentation.

4.2.2 Component Overview

This component is responsible for iteratively testing a student source code against multiple test cases.

4.2.3 Phase Overview

1. Compile source code specified by the user.
2. Redirect input and output for student source code executable.
3. Input a test case into student source code executable.
4. Compare output of student source code executable to solution.
5. Record whether the student passed the test case or failed the test case.

4.2.4 Data Flow Diagram

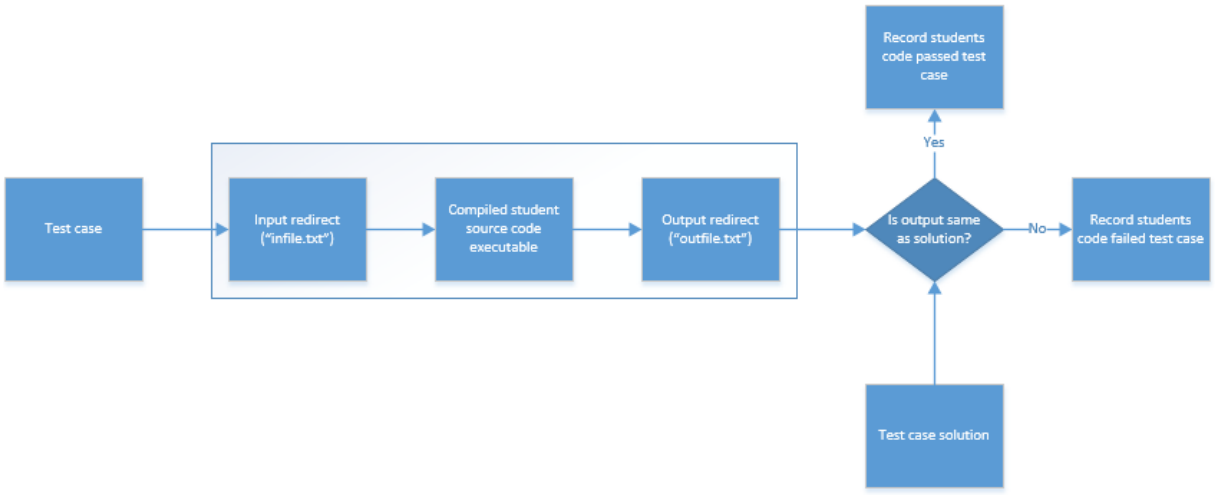

Figure 4.2: Iterative Executable Tester Data Flow Diagram

The process illustrated in the figure above is repeated for all pairs of test cases found.

4.2.5 Design Details

This component implements *popen* from *stdlib.h*. This is used instead of a system call because it allows pipe-lining of the input and output for the system calls and effectively silences the student source code executable output to the command line. This provides a cleaner experience for the user.

When it is determined that a students source code either passed or failed a test case, it is recorded to [studentsourcecode].log. According to the customers requirements this file is not overwritten, instead the results of running the product are appended to the log file.

5

System and Unit Testing

5.1 Overview

Basic tests were run on this code. Each section was individually tested. That is, the function to compile and execute the source code was tested before it was integrated, as was the recurssive directory search.

5.2 Dependencies

There weren't many dependencies when testing the functions of this program.

The user needs only to ensure the c++ source code will compile and execute without any major bugs, such as infinite loops.

5.3 Test Setup and Execution

For the end product we ran the test cases supplied by Dr. Logar for this project.

When testing other portions of the code other test cases were developed. For example, when writing the code to open, compile, and execute c++ source code some c++ programs were written and the output was analyzed to ensure the program ran correctly. When testing the code to walk through directories arbitrary directories were used to ensure the code would walk through the entire directory,

6

Development Environment

6.1 Development IDE and Tools

A Linux environment was used with generic text editors. Most compiling, testing, and debugging was done in the Linux Terminal. QT was used for some debugging and editing the code.

6.2 Source Control

The source control system used was GIT. We used GitHub to create the repository and add each team member as a collaborator. A developer needs to be added to the repository as collaborator in order to edit the source code.

6.3 Dependencies

Dependencies for this project are Linux, g++, and QT. The way these were used is mentioned above.

7

Release – Setup – Deployment

7.1 Deployment Information and Dependencies

As some installs of Linux may not contain the gcc-g++ compiler a user may need to install these packages. Other packages used are not absolutely necessary for the deployment of this preject.

8

User Documentation

8.1 User Guide

The intent of this program is to compile and execute c++ source code found in a directory. Test cases are found by this program, test cases should have .tst as the file extension. Answer files will also need to be availible and should have the .ans extension. The program is executed and the test and answer cases are compared in order to grade a Computer Science student.

The end user will need to have Linux installed on their machine. In Linux the user will also need the g++ compiler. The user will need to have valid c++ source code file names, .tst file names, and .ans file names. A valid directory is needed so that the program will produce results.

8.2 Installation Guide

To run this program the user needs to compile the source code in the Linux Terminal.

The program should be compiled via: g++ grade.cpp -o progname

If the -o switch is not specified a.out will be created as an executable and may cause conflict when running the program.

Acknowledgement

Special thanks to Dr. Logar for being our test user.

Sprint Reports

8.1 Sprint Report #1

At this point there is only one sprint. So far, our product seems to be functioning. There are no known bugs in our project related to Assignment Grader.

However, the usablility of our product the end of this first stage is not overwhelmingly positive. It meets each of the requirements outlined by the user stories. However, it does do "good practice" error checking (discussed previously in the "Phase Overview" section).

For a first release version (or Alpha version to be more realistic), this meets everything that it is required to meet. However, it has a long ways to go in the future before it would be considered a full release.

In the interest of the future of the product, the members of Kernel_Panic will remain in their current positions for the next sprint.