

---

# Product Title

---

## Testing System Project

LaTeX Samurai

Julain Brackins

Jonathan Dixon

Hafiza Farzami

March 23, 2014

---

# Contents

---

<b>Mission</b>	<b>v</b>
<b>Document Preparation and Updates</b>	<b>vi</b>
<b>1 Overview and concept of operations</b>	<b>1</b>
1.1 Scope . . . . .	1
1.2 Purpose . . . . .	1
1.2.1 Traversing Subdirectories . . . . .	1
1.2.2 Running the Program Using Test Cases . . . . .	1
1.3 Systems Goals . . . . .	1
1.4 System Overview and Diagram . . . . .	1
<b>2 Project Overview</b>	<b>3</b>
2.1 Team Members and Roles . . . . .	3
2.2 Project Management Approach . . . . .	3
2.3 Phase Overview . . . . .	3
<b>3 User Stories, Backlog and Requirements</b>	<b>4</b>
3.1 Overview . . . . .	4
3.1.1 Scope . . . . .	4
3.1.2 Purpose of the System . . . . .	4
3.2 Stakeholder Information . . . . .	4
3.2.1 Customer or End User (Product Owner) . . . . .	4
3.2.2 Management or Instructor (Scrum Master) . . . . .	4
3.2.3 Developers –Testers . . . . .	4
3.3 Business Need . . . . .	4
3.4 Requirements and Design Constraints . . . . .	5
3.4.1 System Requirements . . . . .	5
3.4.2 Network Requirements . . . . .	5
3.4.3 Development Environment Requirements . . . . .	5
3.4.4 Project Management Methodology . . . . .	5
3.5 User Stories . . . . .	5
3.5.1 User Story . . . . .	5
<b>4 Design and Implementation</b>	<b>7</b>
4.1 Traversing Subdirectories . . . . .	7
4.1.1 Technologies Used . . . . .	7
4.1.2 Design Details . . . . .	8
4.2 Running the Program Using Test Cases . . . . .	8
4.2.1 Technologies Used . . . . .	8
4.2.2 Design Details . . . . .	8

<b>5</b>	<b>System and Unit Testing</b>	<b>9</b>
5.1	Overview . . . . .	9
5.2	Dependencies . . . . .	9
5.3	Test Setup and Execution . . . . .	9
<b>6</b>	<b>Development Environment</b>	<b>10</b>
6.1	Development IDE and Tools . . . . .	10
6.2	Source Control . . . . .	10
6.3	Dependencies . . . . .	10
6.4	Build Environment . . . . .	10
<b>7</b>	<b>Release – Setup – Deployment</b>	<b>11</b>
7.1	Deployment Information and Dependencies . . . . .	11
7.2	Setup Information . . . . .	11
7.3	System Versioning Information . . . . .	11
<b>8</b>	<b>User Documentation</b>	<b>12</b>
8.1	User Guide . . . . .	12
	<b>Supporting Materials</b>	<b>13</b>
	<b>Sprint Reports</b>	<b>14</b>
8.1	Sprint Report . . . . .	14

---

## List of Figures

---

3.1 Directory Hierarchy Structure . . . . .	5
---	---

---

## 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*

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

# 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 the `.cpp` files.

#### 1.2.2 Running the Program Using Test Cases

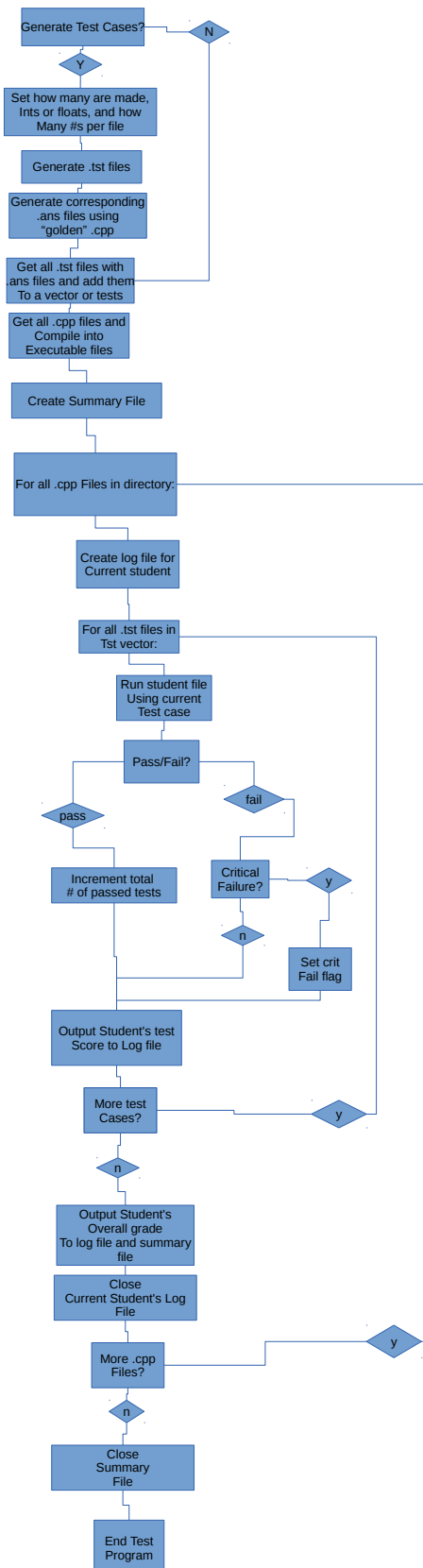
The software was designed in the Linux environment provided to the group by the university.

### 1.3 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:





## 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 approach 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.

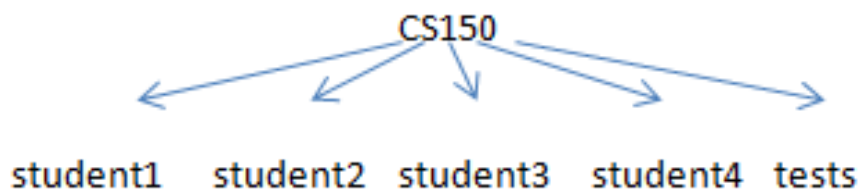


Figure 3.1: Directory Hierarchy Structure

### 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 implementation details without having to look into the code. Here is the

- Create a queue of every subdirectory in program folder
- Change to directory where program is located
- Compile program
- While subdirectory queue is not empty:
- Dequeue first subdirectory in queue
- Change into that subdirectory
- Create a queue of every .tst file in current directory
- While test case queue is not empty:
- Dequeue first test case in queue
- Run program using that test case
- Count whether the program passed or failed test case
- Change back to home directory (where program is located)
- Create a queue of every .tst file in home directory
- While test case queue is not empty:
- Dequeue first test case in queue
- Run program using that test case
- Count whether the program passed or failed test case
- Write log file containing percentage of tests passed and final grade

## 4.1 Traversing Subdirectories

### 4.1.1 Technologies Used

The dirent.h library is used for traversing subdirectories.

### 4.1.2 Design Details

```
bool change_dir(string dir_name)
{
    string path;
    if(chdir(dir_name.c_str()) == 0)
    {
        path = get_pathname();
        return true;
    }
    return false;
}

bool is_dir(string dir)
{
    struct stat file_info;
    stat(dir.c_str(), &file_info);
    if ( S_ISDIR(file_info.st_mode) )
        return true;
    else
        return false;
}
```

## 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

```
int run_file(string cpp_file, string test_case) //case_num
{
    //create .out file name
    string case_out(case_name(test_case, "out"));

    //set up piping buffers
    string buffer1("");
    string buffer2(" &>/dev/null < ");
    string buffer3(" > ");

    // "try using | "
    //construct run command, then send to system
    //./<filename> &> /dev/null < case_x.tst > case_x.out
    buffer1 += cpp_file + buffer2 + test_case + buffer3 + case_out;
    system(buffer1.c_str());

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

## 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.

---

## User Documentation

---

This section contains the user guide, user documentation.

### 8.1 User Guide

This product is to make it easy for you to grade a computational computer program written in C++ language. In order to benefit from this product, the `test.cpp` file and Makefile for the testing program must be in the same directory. Located in the same directory as the `test.cpp` file should be a folder containing a `.cpp` file to be tested. The directory and `.cpp` names should match. In the `.cpp` file's directory, you should have:

- The `.cpp` file to be tested
- Test files (with `.tst` extensions)
- Corresponding solution files (with `.ans` extensions)

As a user, all you need to do is type `grade <filename>.cpp` in the terminal. It is assumed that you are using this product on **Windows** or **Linux** operating system. The program will run through all the test cases and compare the results from the `.cpp` file with the answer in `.ans` file. If the answers were similar, the `.cpp` file will get 100% credit for that particular case, else the percentage will be zero. The program will run and return you a `.log` file containing the grade for a given `.cpp` file.

---

## 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 functioning 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.