Test Plan Document

for

Group Maker

Prepared by:

Anthony Phimmasone

Denzel Saraka

James Donahue

Igor Asipenka

CPSC 430

03/28/19

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide the testing team with reasons behind the testing methodologies chosen, the necessary functionality that is required to test, and the general format of what completed work products after testing should appear. Additionally, this testing plan contains several factors that are integral to delivering a successful product on time: priorities, scheduling, personal approaches, and requirements. This document is intended to serve as a plan for the implementation team and the client as well.

1.2 Scope

The purpose of this document is to provide a detailed schedule and plan of how the testing team will examine the Group Maker software. This testing plan analyzes testing strategies, provides the testing team with a guideline of what to examine in the software, and how that team should report their findings.

1.3 References

Software Requirements Specification for Group Maker
Group Maker Requirements Document
Group Maker Project Plan Document

1.4 Overview of the Remainder of the Document

The remainder of the document is organized in the following way:

- Section 2 [Project Description]: This section presents a general overview of the product, introduces the client and users, explains the project constraints, and provides priorities for the functional requirements.
- Section 3 [Test Plan]: This section provides the breakdown of the proposed planning implementation, includes what tests should be performed by the testing team, and describes how the tests should be recorded.

- Section 4 [Test Procedure]: This section includes a chart which describes a detailed test procedure including test tactics and test cases for the software.

2. Project Description

This section gives an overview of the product, introduces the client, explains who the users are, explains what the constraints of the project are, and assigns priorities to the functional requirements.

2.1 System Overview

The product will be a Python-based software that will create groups based on a CSV file. The product will import the CSV file, parse through it, and ultimately group students together based on the condition that the user chooses. The user will be able to sort groups based on preferred lists, black lists, personality, and gender. The user will also be able to set the number of groups once they start running the program. The program will display the groups to the user after they are created. It will also create an output CSV that will contain all the groups that have been run by the program.

2.2 Program Walkthrough

When the software runs it will first prompt the user for the input CSV file. Once the user enters the CSV file, the program will ask the user a question related to how the user wants the groups to be created. This question is How would you like to have your groups created? By the number of students per group[N]? Or By how many groups will be made[G]? The user will then enter either N or G and answer a follow-up question based on their answer.

After finishing the preliminary phase, the user will be asked if they want to complete random groups. If they answer yes, random groups will be created. If they answer no, more questions will appear that deal with the parameters and settings that can be applied to the group maker. After going through all the questions, the groups will be made based on your answers. All groups will be displayed on the screen and will be written to a CSV file that is stored in the same folder as the software.

2.3 Client Characteristics

The client is Dr. Anewalt. She is a member of the University of Mary Washington's Computer Science department. She requested this project because she wanted an easier way to create groups for her computer science classes. She wanted a system that could make these groups based off certain conditions, such as a preferred list.

2.4 User Characteristics

The intended users of this project are all the professors in the UMW Computer Science department. All the professors have similar needs and wants as the client when it comes to creating groups for their classes. This will allow the product to be useful for the entire Computer Science department. Students and other departments of Mary Washington will not have access.

2.5 Assumptions

We can assume that all the users are Computer Science professors at the University of Mary Washington. We can also assume that the professors will be providing the input CSV and that they have experience using the command line.

2.6 Functional Requirements

The user will interact with the software through the command line. The user will first give the program a CSV file to get input from, then select the options they want to use, and then make any desired changes to the generated groups before the output is saved to a CSV file.

Requirement 1: As a user, I want to be able to pass a CSV file created from a Google Forms survey, so that the software will have all students that I want to put into groups. (Priority: 1)

The software must be able to read an input CSV file created from data gathered through a Google Forms survey, passed in as an argument.

Requirement 2: As a user, I want the CSV file in the same directory as the software so that the software will have access to the CSV file. (Priority: 1)

The user must run software within the same directory as the CSV file.

Requirement 3.1: As a user, I want the software to write a CSV file that contains all groups generated by a single run of the software, with each student's name and each group sorted by numbers. (Priority: 1)

The software must be able to write an output CSV file with sorted groups numbered and containing the names of the students in each group.

Requirement 3.2: As a user, I want the software to write a CSV file that contains multiple possible sets of groups instead of just one group set. (Priority: 1)

The software must be able to write an output CSV file with multiple sets of groups. To elaborate, the software will generate one CSV file that will contain multiple groups. For example, one CSV file may contain four groups.

Requirement 3.3: As a user, I want the software to export the CSV file into the same directory as where the original file was, or the directory of my choosing for my convenience. (Priority: 2)

The software must be able to write the output CSV file into the local storage of the user's machine, into a directory of their choosing.

Requirement 4: As a user, I want to be prompted to choose available options from the menu so that I can create the most dynamic groups I can. (Priority: 1)

The software must prompt the user for specific parameters for group making use including but not limited to: group size, number of group sets, randomization, student "blacklists", group generation by similarities or differences, and whether students should be included in more than one group.

Requirement 5.1: As a user, I want to be able to use the software through the command line as it is straightforward and easy to use. (Priority: 2)

The software must be utilized through the command line.

Requirement 5.2: As a user, I want the software to display the prompts for my input through the command line. (Priority: 1)

The software must display menu options through the command line.

Requirement 5.3: As a user, I want to specify the size and/or number of groups to create. (Priority: 4)

The software's menu will display the following questions in this order through the command line: "How many groups will be made?", "How many sets of groups will be made?". If there will be more than one set of groups, software will ask "Will there be repeated members within groups?".

Requirement 6.1: As a user, I want to be able to look at the generated sets of groups before they are written to make sure that everything looks correct. (Priority: 1)

The software will allow the generated sets of groups to be shown to user before being written to an output CSV file.

Requirement 6.2: As a user, I want to be able to change the groups if I spot a problem with them before the groups are finalized. (Priority: 4)

The software will allow the user to make changes to the groups after being shown to the user, but before being written to an output CSV file.

Requirement 7: As a user, I want to receive a clear, informative error message when I enter invalid input into the software, or when something goes wrong with the software. (Priority: 1)

The software must display an error message when not given the correct information.

Requirement 8: As a user, I want to be able to create groups formed from a metric with similar or different values. (Priority: 1)

For each metric the user uses to create groups, the software will ask if the groups should be formed by using similar or different values of that metric. If there are not enough people with similar/different values the program will prompt the user that this group making process is not possible.

2.5 General Constraints

Non-functional Requirement 1: As a stakeholder, the budget set for this software project is $0.00 because that is what is left in the budget for the semester. (Priority: 1)
The budget for the product is $0.00.

3. Test Plan

3.0 Testing Overview

An important part of software development is testing that the developed code is valid and functional. Software testing should check that the results from the test match the user's expectations. Testing is an important tool to utilize in order to identify errors and bugs in the software that may cause inconsistent results. It is necessary for developers to minimize the impact of errors and bugs so that the software can work more efficiently. Additionally, testing can be used to examine what can be changed and improved within the development process. Code enhancement leads to more user-friendly interfaces, higher efficiency, and fewer bugs.

To ensure that the code the software development team delivers meets the specifications that the client expects, it is necessary to use validation and verification techniques. Validation involves meeting with the client to discuss the software being developed and using client recommendations to mold the software to conform to their needs. The main goal of the implementation team is to satisfy the client and the end-user by meeting the client's requirements and goals. Verification focuses on the software meeting all of the specified requirements from the client.

For our project we considered many different testing strategies. For each strategy, we examined the pros and cons of each method in reference to the Group Maker software to determine the advantages and disadvantages in implementing each various test. The testing strategies the team considered include: usability testing, unit testing, integration testing, continuous integration testing, and test-driven development. It was not necessary to review other testing methods (security testing, system testing, and system integration testing) as they were not applicable to the software. The team implemented functional tests to ensure that the proper functionalities were operating properly, such as reading from a CSV and writing output to a CSV

within the directory of the users choosing. Below are the types of testing that our group considered in making our decision on which testing strategy to use for our software and an explanation of advantages to using that particular testing strategy, as well as disadvantages to using that testing strategy for our software.

Usability Testing: This type of testing involves how the user interacts with the user interface. Usability testing allows for the development team to understand how the user interacts with the user interface. By utilizing this method, the implementation team can receive feedback from the testing team about how intuitive the command line interface is for users. Furthermore, usability testing can be used to cement a simple menu that will not need to be adjusted in later milestones. In order to reduce personal bias, a testing team that is unrelated to the development team will need to be contracted to test the user interface.

Continuous Integration Testing: Continuous integration focuses on testing multiple units of code within a timeframe (often a milestone) to expose errors and fix bugs within the software. Contrastingly to integration testing, CIT utilizes a version control system to push code to the master branch in order to have the most updated version of working code accessible at all times. As milestones are completed, the additional code will be continually added to the build of the sprint. The version control system the team is using, GitHub, has an overarching history of all code changes, so in the event of a mistaken push, the team can retrieve the old version of the code and fix the software. Compared to other methods the implementation team can easily test code functionality using this method. Similar to integration testing, continuous integration is very time consuming. Continuous integration would take the same amount of time as integration testing. Therefore, the team will primarily use continuous integration as the main testing strategy, due to the inclusion of version control which is extremely beneficial in team-based software development.

After evaluating all the possible testing strategies, the team came to the conclusion that using continuous integration along with usability testing was the best strategy for our project. This will allow the team to focus on creating verifiable and valid code that is consistently updated. The usability testing will be performed by a third-party testing group and will inform

the development team of necessary alterations to the menu that need to be added to maintain an intuitive command line menu.

3.1 Testing Strategy

As previously mentioned in subsection 3.0, we will be using a combination of usability testing and continuous integration.

The testing team will need to assist the development team with the continuous integration testing strategy by validating that the input CSV file is read in correctly, and the groups are properly outputted to a CSV file. Furthermore, the team will need to test that the groups are randomly generated each time, and that no students are paired with the students they included in their blacklist.

Additionally, the testing team will serve as a third-party test group to implement proper usability testing procedure. This will involve analyzing the command line interface menu and ensuring that each option is easy to understand and efficiently formatted. The inputs must also be tested to ensure that improper options are correctly handled, and accessibility is not hindered.

3.2 Testing Resources and Staffing

3.2.1 Testing Environment

The Testing Team will need to install Python3 in order to run the software. Additionally, the team may need to install the "Pandas" library which the program utilized to analyze the input CSV. This can be accomplished with Anaconda, a cross-platform (Linux, Mac OS X, Windows) Python distribution, or any other desired method. The program can be found on GitHub, and can be cloned or directly downloaded by using the following hyperlink: https://github.com/Anthony-Phimmasone/CPSC-430-Group-Maker

3.2.2 Testing Team Responsibilities

The Testing Team will be responsible for testing the user interface, basic random group creation, and the blacklist functionality. The team will need to use the sample input CSV file and test that the program functions as intended. The groups will need to be randomized and not include blacklisted students with their pairs. Additionally, the user interface should be intuitive

and easy to use, so it should be closely examined as well. The testing will be similar to a black box testing environment; only the core functionalities previously assigned will need to be tested, so written code will not need to be reviewed.

3.2.3 Integration Team Responsibilities.

The Integration Team will be responsible for providing the Group Maker software to the testing team. Furthermore, the software will need to implement a working user interface, random group generation, and blacklist features before the testing period.

3.3 Test Work Products

Due to the high-level functionality being tested by the Testing Team, the work products should be generated following a similar format to:

Test Name:

What functionality was tested:

Result of the test: Pass/Fail/Inconclusive

Expected Output:

Actual Output:

How to replicate the result if it was failed:

The records of bugs and tests should be documented in a spreadsheet or document where it is easy to read and understand the results of each test. The work products produced as an outcome of each test will be documented in the aforementioned manner and provided to the implementation team in order to discuss possible courses of action to remedy the inconsistencies in the software.

3.4 Test Record Keeping

3.4.1 Checklist

The checklist will be used to mark what the software is doing as the program runs. This form provides a simple way for the testing team to record what is supposed to occur in real time.

Group Maker Checklist

- [ ] Prompts user for filename
- [ ] Prompts user how they want groups to be made (By number of students per group or by how many groups will be made)
- [ ] Prompts if user wants completely random groups
- [ ] Prompts if user would like to use blacklists
- [ ] No groups contains someone from blacklists
- [ ] Displays groups on terminal
- [ ] All groups are random
- [ ] Creates output CSV with proper groups

3.4.2 List of Inputs

The testing team will be provided with a form that allows them to record all the inputs that they used. By filling out the form the team will be able to replicate tests if bugs or errors arise.

Prompt	Input Used
"Enter a filename (include the .csv): "	
"How would you like to have your groups created? By number of students per group[N]? Or By how many groups will be made[G]"	
"Would you like the groups to be completely randomized? [Y/N]"	
"Would you like to use the blacklist? [Y/N]"	

3.4.3 List of Output CSV's

This form will be used to store the name of each output CSV generated by the software. This will allow the testing team to easily see which CSV corresponds with which test.

Test Number	Output CSV
1	
2	
3	
4	
5	

3.5 Test Schedule

The testing process will occur during the first week of April. The tests will begin on April 2nd and finish on April 4th. These two testing days will both be for the software's usability testing and will all be done by the testing team.

April 2nd:

The testing team will begin testing the software on this day. They will be testing the reading and writing functions of the software. The goal of the tests will be to see if the software reads in a CSV file and write to an output CSV file correctly.

April 4th:

On this day, the testing team will be testing all the group generation functions of the program. They will be testing if the program can create random groups and if it can create groups based on a blacklist.

4. Test Procedure

The test team will be responsible for ensuring that the main functionality of the program works as described. They will do so by running the program several times with different sets of data provided by the implementation team and examining the behavior and output of the program on each run. A listing of requirements to be tested and the expected results can be seen below:

Continuous Integration Testing:

Test Number	Related to Requirement	Subsystem	Purpose	Test Case Data	Expected Results
1	1 and 2	Read CSV	To test correct input of a CSV	Sample CSV files provided by implementation team	Data in CSV loaded into program and displayed on screen.
2	3.1	Write CSV	To test creating an output CSV file	Data loaded in Test 1	Output CSV created with groups generated by program.
3	5.3	Group Generation	To test that group sizes are correctly set	Data loaded in Test 1	Groups created with the correct size.
4	General Requirement	Group Generation	To test that groups are randomized each time	Data loaded in Test 1	Groups are generated randomly each time.
5	4	Group Generation	To test that blacklisted students are not in groups with their pairs.	Data loaded in Test 1	Blacklisted students not included a group together.

Usability Testing:

Test Number	Related to Requirement	Subsystem	Purpose	Test Case Data	Expected Results
1	1 and 2	Read CSV	To test that the user can correctly input a CSV file	Sample CSV files provided by implementation team. Please input a filename: FILE.CSV	Command is easy to understand, and the user knows how to input a file by name to the software.
2	5.3	Group Generation	To test that the user can select how they would like their groups to be created.	By number of students per group [N] Or By how many groups [G]:	The user is able to understand how to select whether their groups should be created by number or groups.
3	4	Group Generation	To test that the user can use the blacklist.	Would you like to use the blacklist? [Y/N]:	The option is easily understood, and the user understands how to use the blacklist.

5. Appendices

5.1 Glossary of terms related to your project

CSV - CSV stands for “Comma Separated Values”. CSV files are a file format that stores data, such as a spreadsheet or database. CSV files can be imported and exported to and from a program, such as Microsoft Excel.

Blacklist - A list of names of people that someone does not want to work with. This could be for many reasons, such as people who do not get along with one another. People are put on the blacklist are those that one may think of as unacceptable and wants to avoid for any given reason.

Preferred List - A list of names of people that one wants to work with. The people that someone puts on this list are those that they get along well with and may regard to them as colleagues or friends. Those put on the preferred list are people that are trustworthy.

Random Generation - A function that produces a random number. The number will be truly random meaning that it is chosen without method or conscious decisions. The random generation will pick a number finite number starting from one through a specific given number needed, for example, twenty. This random generation will be used to randomize groups.

Pandas - A Python library for working with data manipulation and analysis. This library includes many functionalities but is mainly used for data structures and operations used for manipulating numerical tables. In our software, Pandas will be used for working with our CSV file.

DataFrames - A Python data structure that stores data in tables separated by rows (records) and columns (fields). A DataFrame can be easily viewed and manipulated by a user. In our software, DataFrames will be used to work with the parameters given by the user.

Command Prompt - A command prompt is a command line interpreter that is used to execute entered commands. For example, reading and writing to the file.

GUI - A GUI is a Graphical User Interface. A GUI is a type of interface that allows the user to interact with visual buttons, words, pictures, and images. A graphical user interface is used in most software rather than a text-based interface, this is because it is easier to read and comprehend.

5.2 Author Information

Document each group member's contributions to the document.

Anthony Phimmasone - aphimmas@mail.umw.edu

- Section 1, 2, 3.0, 3.1, 4, 5

Igor Asipenka - iasipenk@mail.umw.edu

- Section 1, 2, 3.0, 3.1, 3.2, 3.3, 5

Denzel Saraka - dsaraka@mail.umw.edu

- Section 1, 2, 3.1, 3.4, 3.5, 5

James Donahue - cdonahue@mail.umw.edu

- Section 1, 2, 4, 5