Matthew O'Brien

Term Project: *The Task Manager*

Test Plan Document

Table of Contents

1 Introduction

This document will show an in depth test schedule for the project Task Manager. This document will serve as an explanation for the testing the project will go through before it is considered finished, as well as the expected results each testing portion requires to show it meets the specifications. The document will break the project down into pieces and describe the testing for each portion of the project, as well as a testing outline for each part coming together for the final complete project.

The project itself is designed to be a simple program that can help manage the task for a group of individuals. There are different different categories that the user can interact with in order to describe and move tasks depending on what they require, going from To do, In Progress, and Complete. There is also a security feature so that there can be multiple users on the program, managing their own tasks, and an external storage feature. All of the testing details for this program will be contained in this document.

1.1 Purpose and Scope

This document contains the information for the testing of the project Task Manager. It contains all of the detailed information of each individual test, how it is carried out, and the results for each of the tests. There will also be an expected result for each test to show that it is up to the requirements of the project. The overall testing schedule and plan for this project will also be contained within this document.

1.2 Target Audience

The intended target audience of this document is the coders and implementer of this project. This document is meant to be used during the coding of the project, to allow for the proper testing of each part of the project to ensure it is up to the required standards. As well as follow the schedule and test plan of the project itself.

1.3 Terms and Definitions

Any of the terms or definitions that are used within this document will be explicitly explained or given throughout the document as they are needed.

2 Test Plan Description

This is the overall plan for the testing of the Task Manager. Here a brief outline or guide will be given to explain the tests, the schedule of the testing, and what portions of the project were not tested, as well as why they were not tested.

2.1 Scope of Testing

The scope of the testing for the project will be both the larger interactions that the user will have with the project itself, as well as how each different list of the project will interact with one another. Ideally, the testing will show that each piece of the project, will interact with each other in the intended way, as well as, showing that the project itself, is easy and simple to use for the the client. The security features of the project will also be tested to ensure they are working properly, and only allow access to the intended user. The final portion of testing for the project will be to show that the information, or tasks, changed when the user interacts with the project, is stored externally and loaded correctly when the user enters the proper username.

There will be no testing for the limitation of how many tasks can be stored. This will be done off the assumption that no individual will have so many tasks as to cause an issue with the external storage limitation. For a larger scale project, some testing to see how many tests could be done would be helpful, but since this is meant for a smaller scale with only a few individuals, no limitations will be tested.

2.2 Testing Schedule

The general testing schedule testing for this project can be broken down into roughly six different pieces. Three of which will be more or less the same, because it is the testing of each individual list itself. This will be the first portion of the testing schedule. It will be the testing of each individual list for the To do, In Progress, and Done lists of the Task Manager. This testing portion will ensure that each list, individually, is working as intended, and are all slightly different specialized objects of the original List class.

Once each list is shown to be working correctly as an individual, the next portion of testing comes from the integration of each list together. This portion of testing is to ensure that the lists are able to interact with one another and move tasks around as per the requirements of the project. This will be the bulk of the testing because it is the actual framework of what the project does for the user.

The next portion of the testing will move toward the external save and load for each list for the different users. Since all of the lists were already tested and shown to be working properly before moving toward this portion of testing, the next step will be to ensure that the external save and load is working correctly. All of the lists for each individual user should be able to externally save and load all of the previous and current tasks they have entered. It will be important during this portion to be sure that the tasks are saved to the correct user, and other users aren't altered or changed when another user is using the program.

The final portion of testing will be for the GUI of the project. This portion of testing is just to ensure that the GUI is working properly, and all interactions with the user will use the correct methods when entered into the GUI. This testing portion is purely a cosmetic and to show proper interaction with the interface and the framework of the project.

2.3 Release Criteria

The minimum fault tolerance for this project is only within the scope of the outer portion of the project itself. The smaller components of each list need to work properly for this project to meet the requirements. Because of this, all interactions from list to list, as well as externally loading and saving the tasks, need to be completely tested and working properly. When each list is up to the standards of the requirements, then the project will have met the release criteria. The fault tolerance for the actual interaction with the user has more room for error. Allowing for some small faults when interacting with the GUI, will not cause any major problems with the program itself. For example, when having to move a task from one list to another, the requirement that the string be typed exactly as it is seen, is not ideal and would be considered a fault, but the project itself will remain unharmed.

3 Unit Testing

The different units that will be test are broken down into three different pieces. The first unit will be the List Testing portion. This unit testing will be about the functionality of the list themselves, and ensure each one will be working properly. This unit will be explained in more detail below since it covers all three different lists. The next unit will be the external save and load for the lists. Once the first unit test is complete, and is working properly, the external save/load will be the next test for these lists. This will ensure that all of the information of each individual user will be saved or loaded properly for each list of tasks they have, or have altered. The final unit of testing is the GUI interface. This is the cosmetic portion of the project. This will ensure that the interface is feeding the correct strings and information to the proper methods. The purpose of this unit testing is just ensure all information given into the GUI are heading to the right list, or method. As well as that the GUI itself is working correctly. Also to test the cosmetic portion of the project, and make sure the interface easy to understand.

3.1 List Testing

The purpose of this unit testing is to ensure that each individual list is working properly. The actual integration of each list will be done at a later time, this is only to show that the lists, individually, work as they are intended. Since this will be the testing for the List class that all other lists are derived from, each method will be fully tested with a set of strings. In order for this unit to meet the minimum criteria for release, the methods for adding, and removing from the list need to be fully met without any issues. These two methods will be used later on in testing to show the integration of each list. And the final testing scope will need to show the full list of tasks.

3.1.1 Add to the list

This test case is for the addition of a task to a list. This will be tested within the List Class that all other lists will be derived from, and this method will be used for all other lists. The method will take in a string, and add it to whichever list is being altered.

This will work for all list, but the integration of this will be explained later on. An expected input for this test will be any string, and it will be passed into the node class contained within the deserved list. If nothing is entered, the expected output will be an error message stating a task can not be empty. The main testing for this portion is the ensure the string entered by the user is passed to a new node, which is held by the appropriate list. As well as, show an error message if the user attempts to enter an empty task.

3.1.2 Remove from the list

This test will show that the user is able to remove a task from a list. The test will take in a string, that matches one of the tasks already entered, and it will be removed from that list. The expected input will be the matching string of a list, and output will be the list without that task on it. If the user enters an input that does not match a task within the list, the expected output will be an error message stating that no task matches that description. This method will be used for later integration as well. The empty string input will have an expected out similar to an incorrect string. There will be an error message stating that the input must match a string already within the list in order for something to be removed.

3.2 External Save/Load

This unit test will be a set of possible loads and saves to an external file for the project. This is meant to show that all of the data that the user has altered will be externally stored for later, and can be accessed again when they user wants to manage their tasks. The first portion of this test is for the loading of an external file to all of the lists. The expected input for this test will be an external file of information for the lists. The output will be all of the lists properly loaded into their corresponding positions. It would help to give each list a fake set of tasks, “To Do Task 1”, for example, and make sure it ends up in the To Do list for the output. Doing this multiple times for each list will ensure that each list gains the right information. After which, the saving externally will be tested. There really aren't any inputs or outputs for this portion. Once the load has been fully tested, the save can be tested since the lists will have a set of tasks to save externally. Upon saving the list externally, check the external file to make sure that the

lists have been properly saved. This can also be done by altering a list using the previously tested add or remove methods, exiting the program, re-entering the program, and displaying the lists again. The expected output will be the altered list. This portion does not really have much for an input to test, only that the previously tested functions are working properly and that they are saving and loading correctly. There should not be any error messages for this portion.

3.3 GUI Interface

The final unit test for this project is the GUI interface. This unit test will be for all of the possible GUI interfaces that the user must interact with while using this project. Each difficult interface will use the same basic principle of sending a string to the appropriate method. Whether that be sending the string of a task to the add task method, or remove task. This will also be done with the security feature of matching the username and password together to access a given set of lists. For example, the interface will allow the user to type in a string for Add Task, and the interface will pass the string to that method. And since it was previously tested, it should add the task to the list. The username and password interface will allow for a string to be entered into both categories, and an expected output would be access to the appropriate set of tasks. If it is incorrect, the expected output will be an error message stating incorrect information. For adding and removing a task, the expected input is again a set of strings, and output would be these strings passed to the appropriate methods. These were already tested, so the primary focus of this testing is to ensure that each interface will send the string entered to the correct part of the project.

4 Integration Testing

The purpose of this portion of the document is to explain and show the appropriately working integration of each unit of the project between each other. This will explain the expected interaction between the pieces of the project, and what would be considered meeting the criteria for the client. Many portions of this testing requires that the unit testing has already been completed and is working properly. Without the previous testing, this portion will not work.

4.1 List Connections

This integration is to test the functionality and connections between the three different lists of the project. Since each list has the general add and remove functionality from the base class, they will be used in specific ways to alter each list as described by the design document. The first connection that will be tested is moving a task from the To Do list, to the In Progress list. The input for this will be a string that matches an item on the To Do List. This string will be copied, removed from the To Do list, and passed into the add method for the In Progress list. Since all of these methods were already tested and ensured they work properly, no output for the inner workings is needed. But the final output for the list is expected to be the To Do list without the matching string, and the In Progress list with the new task string. If a string that is not in the first list is entered, the unit testing should have covered that, and an error message will be shown. This will work the same for moving a task from In Progress to Done as well. The unit test for removing a task should cover the final Remove of the Done List, since it will completely remove the task from the list. Again, if the unit testing was done correctly, this portion will not be needed. The key part of this is to test and show that tasks of one list can be moved easily to another using the two functions already tested previously.

4.2 GUI Interface

The GUI interface integration testing is very similar to the unit testing portion previously described. This portion is just to ensure that each GUI interface will actually interact with the correct part of the project it is trying to access. If the user is trying to alter the To Do list, the GUI interface will pass strings to that class, and the same for the other two lists. There should also be a connect interaction from the interface of the username and password to the stored strings from the external file. Though these were previously tested in the unit testing phase. The key portion of this is to ensure that each interface input string goes to the correct method that will take in that string. If all previous testing was done correctly, there should not be any issues after that part. There won't be any output for this portion of testing. As it is only showing that the interface interacts with the right part of the project.

4.3 External Load/Save

The external load/save portion of this integration testing is for the information stored in the list. A test file should be created showing a fake list of usernames, passwords, and tasks. It is suggested to name the tasks “To Do Task 1”, and so on for each list as previously stated in the unit testing. This portion of testing will ensure that when the user chooses a given account to access that the external information is loaded properly. Again, assuming the previous testing was completed, all of the information generated into the lists should match the test file that the implementer created for test purposes. Now that the file as been successfully loaded, using previously tested functionality of the project, adding or removing tasks from all three lists should be the input. The expected output for these tests will be the external file. Once changes have been made to all three lists within the program, upon exiting the project the external file should be appropriately changed as well. The check this, load the external file outside the project to see if the changes were made, and again load the project, and display the three lists. The changes made previously should reflect on both the file and the program lists. This should ensure that the external load/save functionality was successfully integrated into the project.