Amanda Ryman

Term Project: *Tudu Task Management Software*

Test Plan Document

Table of Contents

1 Introduction

This document will outline the test plan for the Tudu software product, including expectations, parameters, metrics and methods for testing program quality.

1.1 Purpose and Scope

This document aims to provide a detailed methodology for assessing the extent to which the requirements for the Tudu software have been met. This document also serves as a proposal to the client so that project direction can be affirmed and tailored to the client's needs.

1.2 Target Audience

This document is provided as a reference document for the current and future development team, and for the client commissioning the development of this program.

1.3 Terms and Definitions

Authentication: before a user gains access to data, it must be established that their credentials (username and password) match the data being accessed.

GUI: Graphical User Interface, the visual component of the program including panes, windows, text fields, and clickable buttons.

I/O: input and output, referring to the reading and writing of information that is external to the program itself.

Swing: a Java toolkit which allows programmers to easily code and design graphical components such as windows, panels, lists, scroll panes, clickable buttons, toggle buttons, text fields, customized menus, etc.

Task type: tasks are categorized as "to do", "in progress" or "done"

2 Test Plan Description

This section will provide a brief overview of limitations and considerations for development of the design for the Tudu software. The plan will aspire to be as thorough and realistic as possible, with diligent screening for any operational faults. For the modestly object-oriented design of the Tudu software, “Sandwich Integration” was chosen for design, development and testing. The integration between artifacts will be discussed more in depth in section 4, and in section 3 we will discuss all units of the program in terms of their differing qualities and the test approach best-suited to each.

2.1 Scope of Testing

The aspects of the program which ought to be tested can be broken into two different realms: data and display. Data consists of two primary entities: user credentials and raw task data. The former serves as a portal to the latter and integration between the two must be rigorously scrutinized for integrity. The display portion of testing will be more abstract, ensuring that the correct data is displayed to the user in a clear, intuitive way that facilitates ease of use. Testing to specification will be the primary methodology for ensuring program correctness in regard to data. Test cases have been selected and boundary value analysis will be used wherever possible to maximize efficiency of testing and probability of finding faults. Functional testing will be useful for analyzing the logical artifacts of the code and ensuring that the flow of control executes as it ought to. Complete path coverage will not be attempted as the integration of a GUI vastly complicates the breadth and type of data to be tested.

2.2 Testing Schedule

All units are to be tested individually as they are built. As units are linked together, they are tested rigorously using the “Sandwich” technique outlined in section 4. Following the modification of a unit, testing must be redone both for the individual unit and for its integration with other units.

To date, individual unit testing and integration testing has completed and run successfully. Final modifications will be tested no later than June 1st for final product release June 4th.

2.3 Release Criteria

While task management is not a life-or-death matter, fault minimization is nonetheless absolutely critical in building a product that invokes trust, recognition and preference in the user base. The most severe and least-tolerated faults will be those regarding data: the user must trust that their data is theirs and theirs alone, and nothing will frustrate users more than the mishandling, deletion and corruption of their hard work. User credentials must be secure and their data must be scrupulously tied to their account. All care must be taken to ensure that during reading and writing there are no possible ways that the software might overwrite, bungle or otherwise lose the user's data.

More moderate faults might include incorrect data being displayed, incorrectly categorizing an item, or displaying an item which has been deleted. These faults are also intolerable.

The least egregious faults will be the most difficult to detect and primarily correspond to graphical display concerns. These may include improper display of an item that is very long, strange display behaviour when a user resizes the window, or other nuances regarding the look and feel of the GUI. Being the least vital, these will be the last errors to be addressed, but an intuitive and responsive GUI is nonetheless integral to a complete and satisfying user experience, and so every measure will be taken to ensure that the GUI unit is accurate and fault-free.

3 Unit Testing

The program will be analyzed on the basis of its individual units which are somewhat synonymous with the major classes. Various test cases will be considered for each unit with all possible inputs, outputs and outcomes assessed. The integration of these various units with one another will be discussed in section 4.

3.1 Manager Class

This unit is responsible for the overarching control flow of the program and contains logic artifacts which guide the instantiation of the program. Test cases are high-level with minutia of data left to units of greater refinement – the Manager class is concerned with the successful meshing of the individual units. This includes the provision that, should one portion of the program return an exception or fail, the rest of the program is carried out with as much integrity as possible. The Manager class does not take input or return output, but rather moderates the input and output of other classes – rather, its success or failure is measured by the flow and collaboration between the other artifacts of the program.

3.1.1 Authenticate User Credentials

The Manager class must validate that the information being accessed belongs to the user. From its scope, the Manager class merely needs to ensure that the Account Management class has authenticated the user's credentials before loading and displaying the corresponding data. There should be no return from the Account Management unit which results in another user's data being displayed. If a user is not successfully authenticated in Account Management, the Manager class will abort the session without accessing any user data.

3.1.2 Load User Data

After the user has been authenticated, the Manager class calls upon the I/O class to load the corresponding file. Again, the Manager class merely needs to ensure that the I/O class has successfully completed its task before proceeding. If loading user data is unsuccessful, the user will have the option of continuing with any data that was successfully read or aborting the program. The manager class will execute this decision.

3.1.3 Initialize GUI

Following successful user authentication and data load, the user interface will be loaded. Closing events (saving user data) are set at this time, and control is handed to the MainFrame class.

3.1.4 Save User Data

Upon close of the session, the Manager calls upon a method which saves the user's data. Success of this action is absolutely paramount to the overall product quality. The success of Manager will be proven if this method is successfully called regardless of any instability or faults in the MainFrame class over the course of the session.

3.2 Account Management Class

This unit is responsible for creating new user accounts, authenticating the user's credentials, and reporting user status back to the Manager class. This unit takes two strings, `userName` and `password` as arguments and returns various types of feedback depending on authentication success.

3.2.1 Authenticate User Credentials

The sole purpose of the Account Management class is to authenticate the identity of the present user of the program. Other functionality may later be built in, such as a broader user profile with address, birthday, connections with other social media accounts, and so forth, but for now, this unit has a singular purpose. This unit should be able to handle long strings and other incorrect input that may be a part of a buffer overflow attack.

3.2.1.1 Username and password are authenticated

If the arguments `username` and `password` are compared against one another and match database entries, the User Credentials class returns a `File` object to the Manager unit which is used to load the user's data.

3.2.1.2 Username is unknown

Prompt the user to re-enter credentials or create a new account. If user successfully authenticates their `username` and `password`, return a `File` object to the Manager unit. Otherwise return failure to the Manager unit -- user may then try to log on with a different username or quit the program.

3.2.1.3 Password does not match username

Prompt the user to re-enter the `password` for a limited number of tries. Lock account and require reset after the number of tries per `username` is exceeded. Return failure to Manager -- user may then try to log on with a different username or quit the program.

3.3 I/O Class

This Unit is responsible for reading and writing the user data at the start of each session. It builds a `list` object with Task objects and returns it to the Manager class. At the conclusion of the session, the `list` is written to the external file. Success is measured in terms of accuracy and ability to read the user data. All attempts to read and write are nested in try/catch blocks to contain faults and provide documentation for troubleshooting.

3.3.1 Load a file

The I/O class searches for the specified file to load. Task data is read off as strings one field at a time, and then converted to the correct format if necessary (dates, for example). The file is closed and the `list` is then returned to the Manager.

3.3.1.1 File is successfully loaded

If all is functioning correctly, the I/O class will read in the properly formatted data and form the `list` without any faults.

3.3.1.2 No file is detected

If no file is detected (as would be the case for a new user), an initialized list is returned to Manager so new items may be added with the same considerations as with a list with items in it already. The user file will be created upon close.

3.3.1.3 The file associated is empty

If an empty file is detected (as would be the case with an established user who deleted all their data last session), an initialized list is returned to Manager so new items may be added with the same considerations as with a list with items in it already. The user file will be updated upon close.

3.3.1.4 The file is corrupted

If a file is corrupted (the lines of data are not in order or are not in the proper format to be read) the catch block will display the stack trace in the program log. The program will query the user whether to continue with any data that has been successfully read, overwriting any corrupted data, or whether to abort the program and leave the external file as-is.

3.3.2 Save to file

The I/O class creates a BufferedWriter stream with the selected user file. Task class methods are used to write objects to the external file one field at a time, and converting to the correct format if necessary (dates to strings, for example). The file is closed and success is then returned to the Manager.

3.3.2.1 Data is successfully saved

If all is functioning correctly, the I/O class will write out the properly formatted data and save to the file without any faults.

3.3.2.2 Unable to save to file

If there is a problem with writing to the file, the catch block will print the stack trace in the program log. A message is displayed to the user warning them of the fault and they may opt to either quit anyway or cancel the program quit.

3.3.2.3 There is no data to be written

It must be ensured that, when a session that began with data closes with no data, the old (deleted) data is overwritten with a blank file.

3.4 MainFrame Class

This unit is responsible for initializing the GUI, including formatting and moderating the data that is displayed to the user. The MainFrame class also takes user input and selections and sends them back to the Manager class, updating the display accordingly. Success and failure here is somewhat less quantifiable than for other units, but still measurable: the key question is “is the correct data displayed at the correct time?”

3.4.1 Build display elements

Ensure that the proper elements are displayed on the GUI. Success will be measured on the visibility and clearness of each element. This is a visual-based application -- if the user cannot see the data they’re working with or if the data is unclear, the program has failed. To verify correctness of the display, the statements made for each test case below will be tested and observed to be either true or false.

3.4.2 Display possible types of task

The GUI must display the three different types of tasks (“to do”, “in progress” and “done”) for the user to select.

3.4.2.1 No type is selected

If no type of task is selected by the user, a list of all tasks will be displayed.

3.4.2.2 One type is selected

If a type of task is selected, the list of all tasks is replaced by a list of only tasks that are of the selected type.

3.4.2.2.1 No task is selected

The panes displaying task details are blank. This should be the case upon initialization, before anything is selected, and after a selected item is deleted.

3.4.2.2.2 A task is selected

The panes for displaying task details are updated with the selected task’s details.

3.4.2.3 There are no tasks to display

If no tasks are available for display, the task list pane and task details panes are blank.

3.4.3 Adding a new task

The graphical apparatus for adding a new task (`titleField`, `descriptionField` and `addButton`) are visible at all times. If a user adds a task and no task type is selected on the main panel, it will be added to the already-displayed list of all tasks. New tasks are “to do” by default, so if the “to do” type tasks are being displayed, the new task should be added to the list of “to do” tasks, but it should not be visible in the “in progress” or “completed” lists.

3.4.4 Deleting a selected task

When a task is selected, clicking the “delete” button will remove it from all lists. The deleted task’s details should also disappear from the detail panels.

4 Integration Testing

Even if each individual unit is functioning properly, it must be ensured that no emergent faults are caused due to the interactions of those units with one another. As mentioned, “Sandwich Integration” will ensure that our low-level operational artifacts, such as the raw task data, are properly developed and tested from the bottom up, while our logical artifacts regarding control flow are sound from a theoretical level, and accurately moderate the interactions of the other units as they are developed from the top down.

4.1 Data structure management

Information in the Tudu software is stored as data objects in a list. Specifically, the DefaultListModel library was used. Each object in the list is a Task object. This is our unit of smallest refinement, and it must be ensured that the Task object integrates with every other element up to the display. To facilitate displaying list objects using Java Swing, DefaultListModel was used in tandem with jList, which makes our Task objects easily displayable in Swing visual containers. Use of the DefaultListModel also ensures that our data structure is sound, with simple built-in methods for Remove and Add.

Black-box testing within the Manager unit ensures that sending in various types of data (long strings, numbers, blank strings) all result in stable, reliable list behavior.

4.2 The Graphical Interface

We must also make sure that our data is properly integrated with and expressed in the user interface. To ensure that our raw Task objects are displayed in a comprehensible and attractive way in the high-level interface, a custom TaskCellRenderer was used to display the title, description and timestamp details of each item of the list.

4.3 OOP and Flow Control

To successfully partition different units of the program and isolate errors, the I/O class and User Account class were developed and tested separate from the Manager class. In this way, the Manager class can focus on control flow while other specialized classes do the actual processing and data management, returning either success or failure to the Manager. Using this approach in tandem with a straightforward and Swing-compatible data structure, integration of the individual units is facilitated while maintaining the appropriate unit boundaries.

Ryman