TalkBox Testing Document

Version 1.0

EECS 2311 Submission: Group 12

Mohammed Nasiful Haque
Amaan Vania
Tony Ly

Purpose

The purpose of this document is to give a summary of the testing procedures the TalkBox Application went through to verify the correctness of the software and how it ensures that the features that are mentioned in the Requirements document work as expected. It is also going to discuss the testing coverage of said testing in regards to both the TalkBox simulator and the TalkBox configuration app.

Test Plan

Front End Testing:

The functionality of the user experience interfacing with the software needs to be tested to make sure that the user doesn't get subjected to glitches and irritating bugs that ruin the user experience. In this software package, there are two sets of GUI. One of them is for the TalkBox simulator and the other one is the TalkBox Config App and each of them needed a different approach to testing as they each did completely different things with just a few similarities between them.

Back End Testing:

In addition to testing a functional user experience. Testing the correctness of class methods with respect to their class access permissions will validate the quality of the software and ensure a smooth experience for both the users of the TalkBox Application and also the Config Application. The Back-End mostly involved testing the functionality of both the applications and so all the features and the tasks the application was capable of, had to be tested to ensure that the application performed, as promised in the Requirements document.

Test Scenarios

Manual Based Testing:

This testing involves interfacing with the software through the user interface (UI). The UI is implemented using *JavaFX* and therefore is platform independent. This testing was performed through actual usage of the software and covered usage scenarios the user would find

themselves in as per the requirements document. This was done repeatedly with multiple users and their feedback was then used to further improve the Graphical User Interfaces of both the simulator and the config app.

Test Cases

The test cases/scenarios were divided between the main applications of the project. Since each of the apps had their own functionality and responsibilities, the approach for the test cases/scenarios were just as different

TalkBox Configuration App:

- The number of buttons selected in the ‘New Project’ screen should be greater than 0
  When the user selects to make a new project and is now prompted to use the slider to then choose how many buttons the user would like to initialize in the Config App. If the user selects ‘0’, the config app prompts with an error prompt saying that you aren’t allowed to initialize zero buttons. This test case was derived because if you were initializing 0 buttons, the Config app will be in an empty state where the main purpose of the application will be revoked and the app will then need to be restarted to be able to choose again. Therefore an error prompt saying that the user needs to at least select one button. To also ensure the user doesn’t select zero buttons by mistake, we make sure the slider initially starts from 1.

- No empty fields are allowed when the user ‘Edits’ a button in the Config App
  When the user is done selecting the number of buttons they will be met with the configuration app. One of the things that the config app allows to do is configure Audio Buttons with the appropriate title, audio and image. If any of those fields are left empty before clicking on the submit button, it results in the entire configuration app crashing. To prevent this from happening we prompt the user with another error pane notifying the user that there are empty fields in the Editor window. This test case was derived because incomplete audio buttons would be created and that would result in errors such as unwanted crashes when the config app tries to communicate and transfer these audio buttons over to the TalkBox simulator. This test was good enough for the Edit button because the only way the user could change the state of the configuration app (Edit already initialized buttons) was by filling in the fields in the Edit window correctly.

- Audio Buttons that haven't been initialized are handled properly:
  The configuration app allows the user to select the maximum number of buttons they want to configure for usage in the TalkBox simulator. In doing so, the user can pick and choose whichever buttons they want to edit. But there are two cases that can be problematic.
    - When the user fills less no. of audio buttons than they have selected to initialize:
      For e.g, the user selects that they want to initialize 12 buttons from the selector screen but doesn't end up filling all of them.
      The Config app handles this error by just ignoring the empty audio buttons and just adding the buttons that have already filled and then opening them in the simulator. This approach doesn't get in the way of the user and they can appropriately test whatever buttons they have already filled in the TalkBox simulator.
    - When the user leaves unedited (not filled) audio buttons in the middle of two other already filled audio buttons:
      This occurs when a user decides to leave gaps in between the Audio buttons in the config app. The config app opens an error pane to warn the user that there can't be any gaps in between the audio buttons in the config app. Otherwise, it results in transferring unedited audio buttons for the talkbox simulator to build.

  The handling of these two scenarios ensure that the information that the TalkBox simulator gets is correct and so the Audio buttons the simulator builds is the same no. of audio buttons the user actually filled and edited in the Talkbox configuration app; There aren't any buttons that go missing when the configuration app communicates with the simulator and vice versa.

- The File Choosers can only open files of specific types:
  Every single action that opens the FileChooser in this application makes the user intentionally limit the types of files they can choose. This can be seen whether the user wants to open or edit an existing .tbc file from the welcome screen, the Jfilechooser only pick .tbc files. This was done so that the application need not make any further checks as to what types of files the user uploaded and can get on processing the file knowing sure that the file that user uploaded is of the type the application can definitely work with. Otherwise, the configuration app would crash because the file they would be serializing and deserializing wouldn't be the type of file that can be serialized/deserialized. This approach was good enough because the user uploads a file of the correct type as ensured by the Filechooser.

- Testing the Drag & Drop feature:
  One of the features that were implemented in the Config App was the ability to Drag and Drop Images into the Edit buttons that get generated. We needed to ensure the drag and drop feature actually worked. There were a few things that needed to be considered. We had to check if the drag and drop action updated the text field label of the audio button that the image was dragged on to. This is crucial because dragging an image on to an

audio button should only update that specific audio button but all the remaining Audio button states need to be remain unchanged. We tested it by generating different number of buttons on the GUI and then dragging and dropping images on to the intended audio buttons and opening up the Edit window of that specific button to see if the path of the image was successfully updated and then checked if it carried over correctly to the TalkBox simulator app when the Run button was pressed. Also checked if the file was properly saved when using this feature and if the .tbc file that was saved could be opened up the TalkBox simulator.

- Testing the Record Audio feature:
  Another noteworthy feature of the TalkBox config app was allowing the user to record their own audio and then saving it on their machine for use for the config app. This feature needed to be tested because otherwise there was no way of knowing if the recording function actually worked. The way it was tested was, assuming the user successfully followed the steps of recording the audio (Clicking the record button, speaking into it, having a microphone installed and then saving the file). We tested this with various recorded audio files with different lengths using an external music player (outside of the TalkBox environment) to ensure that the files that were recorded worked the way the Requirements document intended.
  Also this feature gives a warning to the user if they don't have a microphone installed. The way it was tested was by disabling the microphone feature and our own machines and then opening up the Configuration Application to see if the Record Audio feature was able to notice that there wasn't any microphone available for the application to use. As expected, the Recording feature gives you a warning saying that it didn't find any devices to record audio on.

TalkBox Simulator:

The TalkBox simulator is mostly dependent on what the Configuration app feeds it. So the test scenarios and cases are fewer in number.

- All the buttons on the welcome screen work as expected:
  The buttons that first greet the user in the welcome screen of the TalkBox app had to be checked to ensure that there weren't any dead links connected to the buttons (or the buttons being dead themselves). There are a total of four events that can happen in the simulator welcome screen. All the buttons below were tested to work consistently on multiple platforms across various devices to ensure a positive user experience.
  1) Creating a new .tbc file
  2) Open an existing .tbc file
  3) Help
  4) GitHub link

- The connection between the TalkBox Simulator and the Configuration App:
  The TalkBox simulator depends on everything the Configuration app sends. This is important to test because this connection is extremely important for the Simulator to be able to simulate what the config app wants to send. The way this was tested was that we had to check if the Save button in the configuration app actually saves the file in the path that the user selects. So then upon opening the simulator, the user will able to access that .tbc file by clicking on the 'Open an existing talk file' button to select it. To ensure that all of the above was carried out properly, the number of buttons that were opened in the simulator is to be the same as the number of buttons the user filled out in the config app. This ensures that the connection between the two applications was successful and the serialization object that is serializing and deserializing the files is working as expected.

Testing Coverage

Testing the coverage of the code is important because it lets the developers know how much of their code is being covered with the test cases/scenarios they made. It gives them an idea of how much they need to test their code and how much their code is run in an exhaustive usage of their software. Low code coverage indicates that more testing is necessary while high code coverage gives little information about testing quality The applications have an average coverage of 76% and it is a little higher than the average of 70% in traditional software systems which is surely a good sign due to the fact that we were able to reach 75% of the codebase through our test cases and test scenarios resulting in an above average testing coverage.

The config app when it is run gets a test coverage of 78% and it uses all the classes in making sure it works as intended. Depending on the number of error panes the user runs into this percentage can vary as this coverage percentage was arrived by going through the maximum possible number of errors that the user can run into before actually using the software as intended.

The simulator app, on the other hand, gets to a testing coverage of 74% of the entire application. This is almost very similar to the config app due to the fact that these two applications work hand in hand in making the user experience possible. They exchange data through the Serialization object in order to ensure that there is a separation of concern between the two applications.

The highest coverage classes are obviously the classes that deal with building the GUI as that is where the majority of the code resides. The other helper classes get their fair share of the coverage depending on what the user wants to achieve within the application