

---

# ***TALKBOT***

## **TEST PLAN**

---

Version 2.0

03/30/2019

## VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Kai Xu	02/10/2019	Yash Desai	02/12/2019	Test Plan for MIDTERM SUBMISSION
2.0	Kai Xu	03/30/2019	Yash Desai	04/03/2019	Test Plan for FINAL SUBMISSION

# 1 INTRODUCTION

## 1.1 PURPOSE OF THE TEST PLAN DOCUMENT

The purpose of this document is to plan the tests to be conducted. The documentation will track the test cases that have been implemented and test coverage. Explain the process and cause of generating test cases. The documentation will also discuss the test coverage and explain why it is not possible to reach 100% or how to reach 100%.

# 2 UNIT TESTING

## 2.1 TEST METHOD / ISSUES

We used Junit to design unit testing for each class. We try to test each method in each class as much as possible. Based on the feedback of midterm submission, we redesigned our project, so the final submission was separated the interface, controller and data model. Our test coverage increased from 72.1% to 84.1%. Test coverage still does not reach 100%, because some private methods we can't test, and some interfaces can only be tested at runtime.

## 2.2 ITEMS TO BE TESTED

Test Case	Test Class
TBCVisualFrameTest	TBC_Interface: VisalFrame, PanelPreText, PanelMain, PanelHeading
TalkBoxUITest	Interface: TalkBoxUI, MainPanel, HeadingPanel, BodyPanel, PreTextPanel
TalkBoxSimulatorTest	TalkBoxSimulator
RecorderFrameTest	RecorderFrame
DeviceTest	Device
ConfiguratorTest	Configurator
ConfigPanelBodyTest	Config_src.PanelBody
ConfigCustomBtnTest	Config_src.CustomBtn
ButtonInterfaceTest	ButtonInterface
AudioObjectTest	AudioObject
ConfigVisualFrameTest	Config_src.VisualFrame

## 2.3 TEST APPROACH(S)

- **TBCVisualFrameTest**
  - *Describe:*  
Tested all the classes in TBC\_Interface package.

- *How:*  
We first assign a value to the variable with set method and then compare this value with the value obtained by get method. If the values are equal, the method is correct.
- *Why:*

Those are UI program for these TBClog APP. In order to ensure that the program works properly, we need to pass the test. In the Exception part of the program, we are also testing when testing APP. These can tell the user the operation error, if it cannot run normally, the user will not know why the APP does not achieve the desired effect.

- **TalkBoxUITest**

- *Describe:*  
Tested all the methods in TalkBoxUI TalkBoxUI, MainPanel, HeadingPanel, BodyPanel, PreTextPanel class.
- *How:*  
We first assign a value to the variable with set method and then compare this value with the value obtained by get method. If the values are equal, the method is correct.
- *Why:*  
Those are UI program for these Simulator APP. In order to ensure that the program works properly, we need to pass the test. In the Exception part of the program, we are also testing when testing APP. These can tell the user the operation error if it cannot run normally, the user will not know why the APP does not achieve the desired effect.

- **TalkBoxSimulatorTest**

- *Describe:*  
Tested the loading .tbc file.
- *How:*  
Compare .tbc file data with read data.
- *Why:*  
This is the entry point of the Simulator APP, the user will select the tbc file, if it cannot be read correctly, the entire Simulator APP UI loading will be wrong.

- **RecorderFrameTest**

- *Describe:*  
Tested actionPerformed() method in RecorderFrame class.
- *How:*  
Automatically click each button and see if the corresponding button function is executed.
- *Why:*  
If the button does not correspond to the function, some app functions will not be executed, so we must make sure it is correct.

- **FileManagerTest**

- *Describe:*

Test whether `setDefault()`, `setup()` can read the corresponding `.tbc` file

- *How:*  
Compare `.tbc` file data with reading data.
- *Why:*  
If the correct `tbc` data is not read, the app cannot run the correct UI.

- **EditBtnTest**

- *Describe:*  
Tested the all methods in `EditBtn` class.
- *How:*  
Constructor: Use the constructor to create a new object and then compare the variables in the object with the parameters we used when creating.  
Get method: Compare the original value with the return value of the get method.  
`actionPerformed()`: Automatically click each button and see if the corresponding button function is executed.
- *Why:*  
This is the interface to modify the button. We want to test get methods, because we must make sure that the read button information is correct, otherwise the wrong button will be modified. If the button does not correspond to the function, some app functions will not be executed, so we must make sure it is correct. This is why we also want to test `actionPerformed()`.

- **DeviceTest**

- *Describe:*  
Tested `turnButtonON()` and `turnButtonOff()` methods in `Device` class.
- *How:*  
Whether the variables become the corresponding value after the method is called.
- *Why:*  
This is a helper class for many classes. If the return value is incorrect, some functions will have incorrect results.

- **ConfiguratorTest**

- *Describe:*  
Tested all the methods in `Configurator` class.
- *How:*  
We first assign a value to the variable with set method and then compare this value with the value obtained by get method. If the values are equal, the method is correct.
- *Why:*  
This class saves the data read by `tbc` into variables. If the save process has an error, the app will not work properly. For example, the button cannot be displayed and the sound cannot be played.

- **ConfigPanelBodyTest**

- *Describe:*  
Tested the getGonfig method in Config\_src.PanelBodyTest class.
- *How:*  
Compare the data of the config obtained by the method with the data of config.tbc.
- *Why:*  
This is part of the Configurator APP UI, we tested it for the UI to display properly.
- **ConfigCustomBtnTest**
  - *Describe:*  
Tested all the methods in Config\_src.CustomButton class.
  - *How:*  
Use the constructor to create a new object and then compare the variables in the object with the parameters we used when creating.
  - *Why:*  
This is part of the Configurator APP UI, we tested it for the UI to display properly.
- **ButtonInterfaceTest**
  - *Describe:*  
Tested the all methods in ButtonInterface class.
  - *How:*  
We first assign a value to the variable with set method and then compare this value with the value obtained by get method. If the values are equal, the method is correct
  - *Why:*  
This class holds information about whether the button should be displayed. If an error occurs, the UI will display the wrong button.
- **AudioObjectTest**
  - *Describe:*  
Tested the all methods in AudioObject class.
  - *How:*  
Input a name for the sound to see if the correct sound file is playing.
  - *Why:*  
This class determines whether the sound can be played correctly, so we have to test its correctness.
- **ConfigVisualFrameTest**
  - *Describe:*  
Tested the constructor and whether it can throw an exception normally. Automatically detects if a GUI can be generated when the constructor is called.
  - *How:*

Call the constructor and then use getGonfig method to get the data compared to the stored config.tbc.

- Why:

This is part of the Configurator APP UI, we tested it for the UI to display properly.

### 3 GUI TESTING

#### 3.1 TESTING PURPOSES

After unit testing, some UI functions can only be tested at runtime, so we use a manual test to test these functions. Unit tests separate each class, but GUI tests we combine them for testing.

#### 3.2 ITEMS TO BE TESTED







Item to Test	Test Description
Change audio set	Click the audio set button and the audio button will be converted to the corresponding button.
Play Audio	Click the button to play the corresponding sound.
Edit audio Button	Click the button to change the button name, icon and audio.
Add audio set	Click the button to add a new sound set.
Display log	Observe whether the log reads the log file, and whether the app records the log.
Play Audio	Click the record button to record sound.
Drip and Drop	Drip and drop image to "Drip And Drop Here" to add image.

#### 3.3 TEST APPROACH(S)


Run three APPs and test each button. Record the effect of each button and compare it to the expected effect. Running a program like a user is the most intuitive GUI test method, so we can test which features are working as expected. If each button has been tested, we can know whether our program is working as expected. We need to test the various button sequences so that we can guarantee that the program won't crash due to a specific button sequence.

### 4 TEST COVERAGE

## 4.1 GRAPH

EECS2311		84.1 %	5,710	1,080	6,790
> Config_src		77.5 %	2,262	657	2,919
> Test		96.2 %	1,849	74	1,923
> src		81.2 %	1,241	287	1,528
> TBC_src		88.4 %	358	47	405
> Helper_Methods		0.0 %	0	15	15

### COVERAGE

Runs: 42/42	Errors: 0
	
<ul style="list-style-type: none"><li>TBConfiguratorTest [Runner: JUnit 5] (0.000 s)</li><li>AudioObjectTest [Runner: JUnit 5] (0.080 s)</li><li>ButtonInterfaceTest [Runner: JUnit 5] (0.020 s)</li><li>ConfigCustomBtnTest [Runner: JUnit 5] (0.815 s)</li><li>ConfigPanelBodyTest [Runner: JUnit 5] (2.570 s)</li><li>ConfiguratorTest [Runner: JUnit 5] (0.040 s)</li><li>ConfigVisualFrameTest [Runner: JUnit 5] (1.580 s)</li><li>DeviceTest [Runner: JUnit 5] (0.480 s)</li><li>EditBtnTest [Runner: JUnit 5] (17.315 s)</li><li>FileManagerTest [Runner: JUnit 5] (0.072 s)</li><li>RecorderFrameTest [Runner: JUnit 5] (4.531 s)</li><li>TalkBoxSimulatorTest [Runner: JUnit 5] (0.549 s)</li><li>TalkBoxUITest [Runner: JUnit 5] (0.241 s)</li><li>TBCVisualFrameTest [Runner: JUnit 5] (0.354 s)</li></ul>	

### JUNIT TEST

## 4.2 DISCUSSION

From the calculation data of EclEmma, our test cases cover 84.1% of the code. Compared with the midterm submission, our test coverage increased from 72.1% to 84.1%. Although not reaching 100%, but it has reached the average level of code coverage. The main reason for the increase in coverage is that we have separated the interface, controller and data model. There are two factors that make the test not reach higher coverage. First reason, many private variables and methods are defined in the program. Test cases can't access them, so we can't test them. These can't be tested even if we manually test them. The second reason, when we designed the GUI, we didn't leave an interface for the JUnit test. We can't activate GUI components in test cases, so many UI functions can't be tested. Fortunately, we can test these features manually. In the next version, we need to reduce the private methods and add a get method for each private variable.