Vocoder – Voice Programming
Binh An Pham, M Rachel Van Pelt, Steven Tran
Testing Document

I. Introduction

This document serves as the plan for testing all software artifacts as well as the reporting of test results. This document provides the test plan and test procedures for carrying out various levels of testing on the Vocoder product and to ensure the product runs without errors and to meet the customer requirements at an accredited level.

II. Test Item

- application.py
- voice_recognition.py
- compiler.py

III. Intended Audience

- Developers
- Testers
- Customers

IV. Scope

In this testing document, all the core functionalities of the system will be tested.

V. Definitions and acronyms

- test commands: {1.create array, 2.create else statement, 3.create else-if statement, 4.create if statement, 5.create while loop, 6.create for loop, 7.return statement, 8.assign old variable, 9.create new variable,10.copy text, 11.select block, 12.select line, 13.select word, 14.cut text, 15.move cursor, 16.paste text, 17.redo command, 18.undo command, 19.print statement, 20.print variable, 21.create function, 22. indent cursor, 23. insert characters}
- tex, tex2, tex3, tex4: references to information being displayed in the GUI windows; tex = text editor window, tex2 = command(s) received, tex3 = command manager, tex4 = system output
- GUI outputs - command manager window will display the processes that the system completed, system output window will display the status of the system processes, command(s) received window will be updated to show the user what commands have been called, text editor window will display users intended code

VI. Environmental needs

Hardware

- Laptop or Desktop PC
- Microphone

Software

- Python 3.8
- pocketsphinx
- IDE with Python support

V. Features to be tested

All the requirements outlined in the requirement definition document will be tested as new features.

VI. Features not to be tested

Following features shall not be tested because they are either in executable or uncontrollable.

[we do not have anything at this time]

VII. Approach

Unit test. Developers are responsible for unit testing. The implementation of each module and individual component will be verified separately. List all the units implemented as separate units and discuss how are you planning to test them?

The user interface includes: start button, end button, recording indicator light, commands received text box, system output text box, command manager output box, text editor box, command line box, drop down menus. First level functions include: getVoiceInput(), phraseMatch(), test_compiler(), getClosestString(), text2int(), showSet(), confirm(), listen(), chooseLanguageModel(), recordVoiceLines(), and trainLanguageModel(). Second level functions include: createNewVariable(), assignOldVariable(), returnStatement(), createForLoop(), getCondition(), createWhileLoop(), createIfStatement(), createElseIfStatement(), createElseStatement(), createArray(), moveCursor(), selectWord(), selectLine(), selectBlock(), copyText(), pasteText(), cutText(), printStatement(), printVariable(), createDef(), getSymbols(), insertChars(), changeLanguageModel(), recordingVoice(), getPrevLine(), getNextLine(), playWavFile(), recWavFile(), checkNameButton(), and trainModelButton(). We will be using the equivalence partition technique to test each function for Pass/Fail. The second level functions rely on the first level functions and thus the first level functions are regarded as priority 1 for testing before moving on to the second level functions. The user interface units will be monitored visually for errors and used for indications of error in the first and second level functions in addition to the terminal outputs from running the program.

Integration test. After the unit test is passed above the defined quality threshold, testers will execute the integration test cases. After all the modules are integrated, it's crucial to test the product as a black-box. List the end-to-end scenarios that will be tested to ensure the communication functionality.

We referred back to our Sequence Diagram and followed each possible path as a potential process to test. For example: [Start recorder, speak command "return statement", verify command, speak command inputs, verify inputs, end recording] would be a complete set of actions to test for Pass/Fail. Upon completion of the Unit tests listed in the Acceptance Test Procedure below, we have concluded that the integration test has passed all intended scenarios for the system.

Positive and negative testing design technique. This approach will be combined with the unit test and the integration test. Test cases are designed in sunny day scenarios, which ensure that all functional requirements are satisfied. What's more, rainy day test cases will also be covered to show how the system reacts with invalid operations. List the positive and negative test cases that you will use with your system.

Positive cases included the planned flow of user interactions listed in our requirements document. Negative cases for our particular project included a user not speaking after clicking the start button, a user speaking commands and inputs not included in the list of valid commands, or a user clicking outside of the

user interface.

System test. System testing has a particular purpose: to compare the system or program to its original objectives. Describe how you will perform this test with your system.

We referred back to our design documentation as a checklist against the system to verify if the system was built correctly and if the correct system was built for the proposed need of the user. After reviewing the Project Proposal and Software Requirements Specifications, we feel we have accomplished both of these, therefore passing the System test.

Acceptance test. Acceptance testing is the process of comparing the program to its initial requirements and the current needs of its end users.

VIII. Acceptance test procedure

TEST ID [Use case derived from]	Priority	Feature Description	Test Case Description	Input	Expected Output	Actual Output
GVI	1	getVoiceInput() - convert spoken command into a string, save in variable `audioToText` and return that string	speak a command after clicking start button	(see commands 1-18 in part V above) such as "create variable"	string of spoken command saved in variable - `audioToText`	pass
PM	1	phraseMatch() - calls the getClosestString function to find the best matching function to the input	speak a command after clicking start button	Test Commands 1-18 from part V such as "create variable"	related function is called based on translated command	pass
GCS	1	getClosestString() - takes in string and a list as input and finds the item in the string that best matches the input string and returns it	speak a command after clicking start button	Any input string, list to find closest match from ---------- invalid command would be "test something"	If input list has a match, returns the element that best matches the input string ---------- All other commands, Print out that "no matching phrase was found"	pass ---------- pass
T2I	1	text2int() - takes in a string as input and returns a string containing the numerical form of the input string	speak a command and give input using numbers	spoken word such as "zero" or "one hundred twenty four"	If input string contains only the word form of a valid number:	pass

					Returns the numerical form of that number --- If input string contains a non-valid word: Returns an error	
SS	1	showSet() - displays a list of variables that are currently stored in setOfVariableNames	Call the showSet function when prompted for a command	No input needed	Returns a list of the variables currently saved in setOfVariableNames to the system output window	pass
C	1	confirm() - calls getVoiceInput and prompts the user to reply "yes" or "no" to confirm the data collected	Call any command that takes in data	"yes/no"	If closest match is "yes": Proceed with function If closest match is "no": Retake data and confirm with user again	pass
L	1	listen()		Microphone input	If a valid voice input is recognized, pass with no errors, else throw invalid voice input	pass
GS	1	getSymbols()	issue "insert characters" command and test strings that include symbols ()[]{},.\<	spoken string	converts spoken keyboard symbols from word string to the symbol desired	pass
GC	1	getCondition()	issue "while loop","if/elseif" command and test strings that include x<y, x==6, etc	spoken string	converts spoken conditions for a loop from word string to the symbols desired	pass
CNV	2	createNewVariable()	say "create new variable" after clicking the start button. say name of variable, confirm yes/no to prompted questions, give initial value to variable	a = 1 my_var = 1	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
AOV	2	assignOldVariable()	say "assign old variable" after clicking the start	a = 2 a = b + 2 c = 3	popup window guides user for inputs needed,	pass

			button. say name of variable, confirm yes/no to confirmation, give new value to variable, verify		GUI output updated after popup window is closed, if no match, create new variable, warn user if using unassigned variables in equation	
RS	2	returnStatement()	say “return statement” after clicking the start button. say none or expression, verify	a none 3 a+3	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CFL	2	createForLoop()	Say “create for loop” after clicking the start button, give the number of loops, looping variable name, and then verify	for x in range(5):	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CIS	2	createIfStatement()	Say “create if statement” after clicking the start button, give the expression to be tested, verify	a<b a<=b a!= a==b	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CMDW	3	commands received window on the GUI	coincides with issuing commands	Any voice input	a history of user issued commands	pass
TEXW	3	text editor window on the GUI	“ ”	Any text input from keyboard or voice input from user	results of completed commands and edits made by the user	pass
SYSW	3	system output window on the GUI	“ ”	activated by issuing commands	history of system status’	pass
MGRW	3	command manager window on the GUI	“ ”	activated when user issues a command	history of system processes	pass
TERW	3	terminal window on the GUI		Text in the text editor window	Return the standard Python output for the code created in the text editor window	pass

RI	4	recording indicator	click start/end	left mouse button	turn red when start button is clicked, turn gray when end button is clicked	pass
PS	2	print statement	call “print statement”, give statement, verify	spoken word(s) of desired statement	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
PV	2	print variable	call “print variable”, give variable, verify	spoken word(s) of desired variable	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CF	2	create function	call “create function” after clicking the start button, give the name of function, verify	spoken word(s) of desired function name	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CWL	2	create while loop	call “create while loop”after clicking the start button, give the expression to be tested, verify	a<b a<=b a!= a==b	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
CEIF	2	create Else-If statement	Say “create else if statement” after clicking the start button, give the expression to be tested, verify	a<b a<=b a!= a==b	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
STB	1	start button	click start button	left mouse button	indicator turns red, system listens for user input	pass
ENB	1	end button	click end button	left mouse button	indicator turns gray, if not already, terminal shows output of users executed code	pass

CES	2	create Else statement	Say “create else statement done” after clicking the start button	n/a	GUI output updated	pass
CA	2	create Array	call “create array”after clicking the start button, give name and values of new array, verify	cars = ["Ford", "Volvo", "BMW"]	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
MC	2	move cursor	call “move cursor”after clicking the start button, give values of new location, verify	1,0 4,4	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass
SW	2	select word	call “select word”after clicking the start button, give values of selection begin and end, verify	2,2 and 2,5	popup window guides user for inputs needed, GUI updated after popup window is closed	pass
SL	2	select line	call “select line”after clicking the start button, give line number, verify	2	popup window guides user for inputs needed, GUI updated after popup window is closed	pass
SB	2	select block	call “select block”after clicking the start button, give line number of begin and end, verify	3 and 4	popup window guides user for inputs needed, GUI updated after popup window is closed	pass
CT	2	copy text	call “copy text”after clicking the start button and having a selection made	n/a	selected text is added to the clipboard	pass
PT	2	paste text	call “paste text” after clicking the start button and having a selection copied	n/a	clipboard data is pasted at cursor location	pass
CUT	2	cut text	call “cut text” after clicking the start button and having a selection made	n/a	selected text is removed from the text editor window and	pass

					added to the clipboard	
UC	2	undo command	call “undo” after clicking the start button	n/a	last user action in the text edit window is reversed	pass
RC	2	redo command	call “redo” after clicking the start button	n/a	if “undo” was the last user action, the user action is restored	pass
IC	2	indent cursor	call “indent” after clicking the start button	n/a	4 spaces are added and the cursor is moved forward in the text edit window	pass
ICHAR	2	insert character(s)	call “insert characters” after clicking the start button	spoken character(s) including +, =, {}, [ ], < >, etc	spoken character(s) of desired string are added at cursor location	pass
CLM	3	chooseLanguageModel() allows the user to select from a list of given choices to use for the language model	click the “voice” menu button then click on “choose language model”	n/a	displays a menu with available language models for the user to choose from	pass
CHLM	3	changeLanguageModel() - changes the language model to the choice that the user selected	while in the menu for chooseLanguageModel(), click on one of the options available	n/a	copies the files from the chosen language model directory over to the directory being used for audio parsing	pass
RVL	3	recordVoiceLines() - gives the user a choice to choose which training model to use to record voice lines	click the “voice” menu button then click on “record voice lines”	n/a	displays a list of available training models for the user to choose from	pass
RV	3	recordingVoice() - takes the user’s choice and displays the relevant text for the user to say and has buttons for getting the previous and next line, a record button, and play button	while in the menu for recordVoiceLines(), click on one of the options available	n/a	displays a window that pulls text from the needed file in the training model directory for the user, and allows the user to click on the previous, record, play, and next buttons	pass

GPL	3	getPrevLine() - gets the data for the previous line from the transcription file and displays it on the window	while in the menu for recordingVoice(), click on the "previous" button	n/a	retrieves the text for the previous line and displays it in the relevant window, if the displayed line is already the first one in the list, a message pops up saying so	pass
GNL	3	getNextLine() - gets the data for the next line from the transcription file and displays it on the window	while in the menu for recordingVoice(), click on the "next" button	n/a	retrieves the text for the nextline and displays it in the relevant window, if the displayed line is already the last one in the list, a message pops up saying so	pass
PWF	3	playWavFile() - retrieves the relevant wav file that is being displayed and plays it to the audio output	while in the menu for recordingVoice(), click on the "play" button	n/a	plays the audio from the wav file displayed in the window, if there is no wav file available, a message pops up saying so	pass
RWF	3	recWavFile() - records the user's audio input for a predefined amount of time and saves it at the relevant path	while in the menu for recordingVoice(), click on the "record" button	n/a	records the user's audio input and saves it to the correct location, if there is a file with the same name already there, it will overwrite it	pass
TLM	3	trainLanguageModel() - allows the user to create a new language model with a custom name and a choice from which training model to adapt it from	click on the "voice" menu button then click on "train language model"	n/a	displays a window that contains a text field for a language model name and a list of available training models	pass
CNB	3	checkNameButton() - checks to see if a directory already exists within the AcousticModels directory	while in the menu for trainLanguageModel(), click on the	any string input	displays a message box informing the user if it is a pre existing directory	pass

			“check availability” button			
TMB	3	trainModelButton() - takes in the user’s choice for name and training model to create a new language model	while in the menu for trainLanguageModel(), click on the “train the language model!” button	any string input, training model choice	trains a language model using the chosen training model	pass
DDM	4	drop down menus	click on File, Edit Voice or Help to see drop down options and click on the desired one	left mouse button	activates the process that was clicked	pass

Where:

- Test ID is a unique identifier for the test case. The unique identifier should relate back to the particular requirement the test case is verifying.
- Feature Description should clearly describe the feature that is used to run the test case.
- Test Case Description should clearly document the steps that need to be done in order to run the test case.
- Input should identify the set of valid inputs to be used to run the test case
- Expected results is a statement of what should happen when the test case is run.
- Actual results are an indication of whether the test case is currently passing or failing when it is run. The actual results could be recorded simply as “Pass” or “Fail.” However, it is also helpful to describe what happened in cases where a test case fails.

Ultimately, your customer should agree to the test case. When test cases are written so specifically, often requirements understanding is enhanced.

IX. Item pass/fail criteria

- The result works as what specified in output => Pass
- The system doesn't work or not the same as output specification => Fail

X. Summary

Using the equivalence partition technique we have been able to complete the testing of our code and throughout the testing process we made the necessary adjustments to the implemented functions and added more that we discovered would benefit the user. We continued to make updates to this testing document as more functions were implemented or as more changes occured in the code. It was our goal to make this system user friendly by regularly inspecting the system for errors and using all previously written documentation in the creation phase to make sure we stayed in line with the specifications. After analyzing the system using the various methods discussed in this paper, we feel more confident in saying that we achieved that goal.