College of Engineering

Senior Design Project Report

VOCODER - Voice Supported Programming

A project submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

(Software Engineering Track)

By

Binh An Pham (1003770)

M Rachel Van Pelt (726488)

Steven Tran (1013838)

Supervised by

Vinitha H Subburaj

Spring 2021

Table of Contents

1. List of Figures

2. List of Tables

3. Acknowledgment

Dr. Vinitha H Subburaj

Carnegie Mellon University

4. Abstract

In a world where software developers are required to type at the minimum 8 hours a day, tendonitis and carpal tunnel have become more common than ever. In an effort to battle this negative trend, our team of three proposed a solution to provide support for on-device and off-line programming using voice recognition technology. Studies and experiments done on this matter have shown that reducing the use of keyboards in combination with the usage of voice commands have either boosted the productivity of programmers or introduced programming to an even larger audience. This paper discusses the details needed to understand this emerging problem, presents the current state of the available solutions, proposes our solution to the problem, introduces the details on the process of developing the solution, and presents what more can be done in the future to improve the proposed solution.

5. Chapters

5.1 Introduction

In this project, our team will propose a solution to support Programming by Voice in an effort to battle the growing trend of programmers inflicted by carpal tunnel syndrome and tendonitis. The solution is a text editor called VOCODER and developed in Python using the tkinter library. The solution also supports on-device and offline voice recognition using the open source CMU Sphinx library. This paper will discuss the important aspects of the development process.

5.2 Background (Literature Review)

5.2.1 HyperCode: Voice aided programming

HyperCode is a voice-based programming framework that utilizes many speech technologies, including Dragon NaturallySpeaking (DNS), Dragonfly, and Natlink, to help programmers with Upper Limb Disorders (ULD) program and navigate inside IntelliJ IDEA, a commercial Java IDE. HyperCode was developed to achieve three goals: “[m]ap Java keywords and common code snippets with voice commands”, “[c]ontrolling IntelliJ IDEA interface using voice commands”, and “[r]educing the overall time needed for writing code” [5].

HyperCode’s strengths lie in its functionalities inside of IntelliJ IDEA, up to the extent of replacing the keyboard entirely. Another strength of HyperCode is its efficiency when used in combination with keyboard and mouse, up 19 seconds faster in a simple creation process of a Java object compared to just using keyboard and mouse.

However, HyperCode’s accuracy and time measurements methodology were never mentioned. Another disadvantage of HyperCode is its use of DNS which is based on Microsoft Speech API, making it unusable without an internet connection, not to mention data collection policy is also up to Microsoft to decide. In comparison to an on-device model, this framework is significantly less secure.

This framework's target users are very similar to ours being programmers with disorders caused by lengthy typing sessions. However, HyperVoice was designed to work specifically with IntelliJ IDEA and Java rather than Python in our case.

5.2.2 VoiceGrip: A Tool for Programming-by-Voice

VoiceGrip is a programming-by-voice tool that addresses a wide range of problems in other programming-by-voice systems at the time. One of which is how native was never meant to be spoken, making Speech Recognition (SR) systems at the time awkward to use. The proposed tool uses a unique technique of translating spoken pseudo-syntax into native code in the appropriate programming language, acting as a support system to the available SR systems.

The tool's biggest advantage is its capability of translating pseudo-syntax into native code making the interaction between the programmer and the editor more natural. The algorithm for this process is also clearly presented making it easy to replicate. The testing methodology and results are explained at length, giving us insight into the testing process. At the time of writing the paper, VoiceGrip was supported by multiple editors and commercial SR systems.

The system was initially developed back in 1998, so it is no longer supported by the developer. Furthermore, VoiceGrip is only a supporting tool that handles the translation of pseudo-syntax into native code, not a standalone SR system. The error rate of the translation algorithm is also higher in comparison to more modern systems.

VoiceGrip is a unique approach to the programming-by-voice problem, providing a supporting tool to the existing systems. This approach is different from our approach of trying to develop a standalone SR system, but it gives us an idea of what problems we should address. It also gives us a detailed test process that is highly applicable to our system [6].

5.2.3 Web based programming tool with speech recognition for visually impaired users

This research project "DICENS" addresses a unique programmers body, visually impared programmers, by creating a speech responsive, lightweight and web based programming tool. "DICENS" can also be used for new users trying to learn programming in a short period of time. This research project presents a gap in the field, where GUI programming tools are far more advanced than current iterations of voice programming tools.

DICENS system is unique in the way that it interacts with the users, the system takes in voice commands, feeds them through a semantic analysis engine, a code play supervision engine, and response back to the users in terms of Braille codes. This process helps eliminate the needs for the use of keyboard and mouse, hence incentivize the process of learning to program for visually impaired users. Another advantage of the system is its portability, being a web based application, the system reduces the hassle of setting it up in other environments.

This system utilizes Web Speech API (by Mozzila) to convert users voice input into text format so this process is not as secure as utilizing an on device, offline Speech Recognition engine. The results and testing methodology shown in this research project is not very compelling, 85% accuracy with around 50 words tested, the system also fails to give any positive results in a noisy environment. The response system in Braille codes is unique but limiting in many ways, and the computer-generated voice feedback is mentioned but never explained.

DICENS targets a different user base than our systems, being visually impaired users. The methodology for developing is also different than ours, waterfall methodology compared to prototyping methodology. This system gives us insights on how to design a responsive, user-friendly Speech Recognition system [7].

5.2.4 Designing For The Future With Voice Prototypes

This article addresses people's expectations for voice programming and gives insight on what a programmer should do when developing a system that uses voice programming. The article speculates that the future of software development will involve speech input in addition to the traditional forms of program input. It describes the design thought process including: research about the user, defining the necessary capabilities of the product, creating the user experience and determining the desired user perception of the product. It also addresses testing of the product and fine tuning for final release.

This article has some really good insight into some of the key points we are trying to address with our project, such as user interactions and finding ways to improve on the user experience. It has several good lists on what to be keeping in mind when developing our project. It is a good reference for where the technology is headed and what to prepare for when developing a technology that involves speech input.

This article is generalized for all types of voice input projects and does not really dive into any particular projects of reference other than the option of working with Adobe XD. In our case we are trying to avoid working with any licensed products such as Amazon or Google related apps, so this part of the article will not be of help to us.

Our project's goal is to help users eliminate keyboard input and rely on voice input instead. This article has helpful tips on how to better understand this user and design the project around the user's expectations and intent of use. We are also using prototyping in our methodology and this article has points to help with this [8].

5.2.5 Programming by voice, VocalProgramming

This document begins by addressing the need for voice programming by detailing the severity of carpal tunnel syndrome. It then goes into the current (as of January 2000 when the article was published) status of voice technology before describing their ideas of how to improve and design their own project. They speculate about the potential of their project and create a method for testing the effectiveness of their project.

The strengths of this report is the summary of technology studies that they had discovered up to that point and it gives good insight on the capabilities of voice recognition software. It does a good job explaining the motivation behind spending time on developing a system that can write programs using voice input. They address the concerns of what was limiting their ability to develop an ideal system including grammar used by the intended customer and the complexities involved in programming like nesting.

A drawback of this article is that it is quite old, especially since we are discussing voice driven technology that is just recently becoming a focus of potential in user interactions. Their approach involves purchasing Dragon NaturallySpeaking Professional Version and we are trying to avoid the purchase of commercial products or requiring the user to purchase a commercial product in order to use our system.

Our project shares the same motivation of their project in that we are trying to create a solution for users that suffer from carpal tunnel syndrome. We are in the process of researching existing projects much like they did in order to determine where improvements can be made on the existing solutions.

They have outlined a solution to address navigating to different structures within a program being edited or created which will be helpful in our design phase. They mention the applications of data entry which we may be able to use in conjunction with our idea of using a menu driven interface along with the traditional methods of voice input [9].

5.3 Project Proposal

Carpal tunnel syndrome (CTS) and tendonitis are conditions that all programmers are susceptible to if they don't take the necessary precautions against it. This problem could be alleviated by having the programmer being able to code simple statements without use of their hands, reducing the risk of developing CTS.

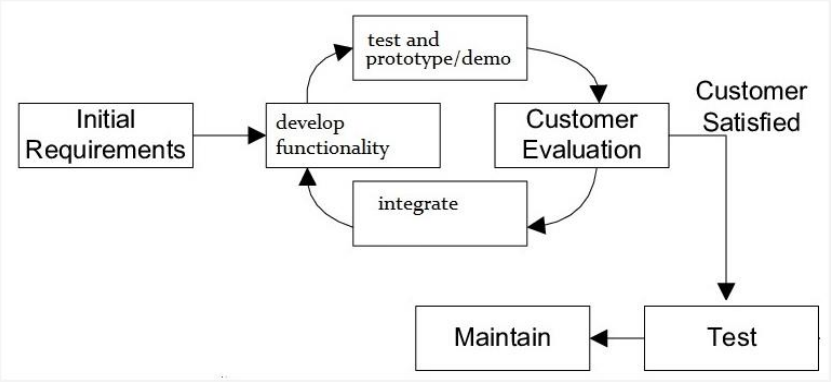
There are existing solutions to this problem that build on Dragonfly, a speech recognition framework for Python. [2] "Talon is currently the most promising project for hands-free coding. Lots of short words map to letters and syntax, which provides good efficiency, but can be hard to learn." [2] We believe this is a common technique that could be improved upon. It is important to not burden a user with learning another language. We should make the tool intuitive to use and in this case be more natural, mimicking human communication and replicate "the exact phrases that a target user uses when they speak with other people." [3]

The scope of this project will be "a system that supports all the functionalities. It will tackle a Python subset consisting of function and variable declarations and expressions (as many as possible). In terms of statements, only return, assignment, and declaration statements are required, although having for loops and if statements is useful. Dealing with lists is also not required. Although navigation is also very important, it is not required for this project. The reasoning is that problems such as repetitive strain injuries are more intensified by the act of writing code than by navigating through it using a keyboard or mouse." (1) It will take roughly 2 to 3 months to implement the solution.

Proposed methodology is to study current applications and trends in voice guided programming and find a way to improve and contribute to the existing community currently working on a solution. Analyzing guides on Human-Machine Grammar rules [4] will help us create a more user-friendly solution for the intended customer and hopefully an expanded audience using the technology. "It's evident that voice will be a natural way for the new generation of users to interact with technology" [3]. With the growing popularity of voice assistants, we hope to find many open-source utilities that we can use the best parts of and build on to meet the main purposes of this project.

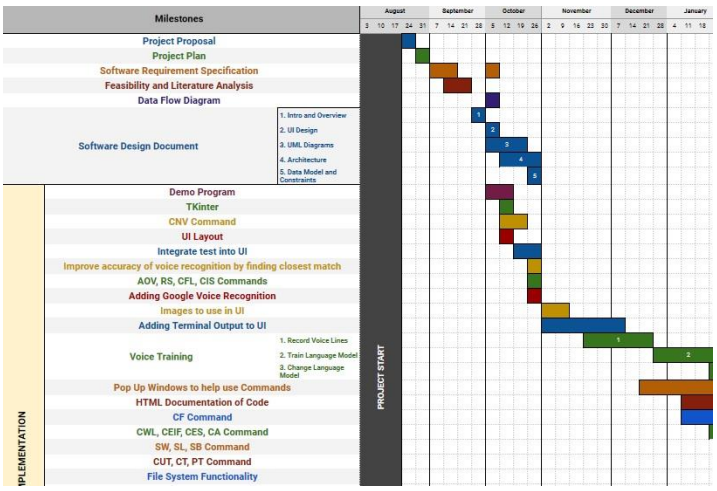
5.4 Project Plan

For our preferred SDLC model we are planning on using a hybrid of Prototyping and Agile.



For the agile part we have a heavy emphasis on involving the customer of our product since our project involves helping them directly as the sole user of the product. The aspect of making small incremental changes works well in conjunction with our idea of starting with minimal functionality and building on that as the project progresses. Therefore, there is no need for extensive design upfront. Making use of refactoring will be key in adding those changes to the overall project to keep the code organized and readable as well as easier to modify. We will be more attentive to individuals and interactions rather than processes and tools. Customer collaboration is a necessity in order to reach our goal of improving on current products and the ease of usability.

For the prototype part the desirable new solution is unknown (we have a sense of what is already done but know there is room for improvement) therefore evolutionary prototyping makes sense with a customer feedback loop. We want user needs met and to prevent overdoing it with unnecessary features.



We decided to use a project plan template in excel so we can use a spreadsheet to track the various functions we are working on, what has been finished/approved by the customer, when new features are added or taken out, etc. We will use dates and time spent on each aspect to see if we are on track to finish or to see if we are spending too much time on a feature. (completed template included in Appendix D)

The following tools enable us to keep up communication with our team members and also with our faculty member. These tools enable us to maintain our artifacts effectively and complete all the tasks on time proficiently.

Communication Tools: Discord, text message, email
Document Management Tools: Google Drive/Docs
SVC: Github
Changes to artifacts: Github logs and Google Drive/Docs logs

5.5 Feasibility Analysis (version: 4390.9.1)

5.5.1 Technical feasibility - Hardware

5.5.1.1 Things we will need:

5.5.1.1.1 Computers for programming and testing.

5.5.1.1.2 Microphone to be able to take voice input.

5.5.1.2 Hardware requirements:

5.5.1.2.1 Reliable computers: To program and run the app, we will need computers capable of running the various softwares for development

5.5.1.2.2 Accurate microphone: We need microphones that will take good quality input from our voices to use for development and testing.

5.5.1.3 Is the project feasible from a hardware perspective?

5.5.1.3.1 As long as the computer can handle running the application and a text editor at the same time, it will be feasible. There is no risk since computers should be able to run those easily.

5.5.2 Technical feasibility - Software

5.5.2.1 Things we will need:

5.5.2.1.1 Access to an open-source voice recognition software

5.5.2.1.2 An environment to program the app

5.5.2.1.3 Once we have these ready, we will need to design and write the code for the project.

5.5.2.2 Software requirements:

5.5.2.2.1 To successfully develop this project, we will need basic knowledge of the software tools that are to be used. These tools include:

5.5.2.2.1.1 Environment to write the code - any text editor/IDE will be fine to use

5.5.2.2.1.2 Voice recognition software - method to receive the data the software gets from voice input

5.5.2.3 Is the project feasible from a software perspective?

5.5.2.3.1 We will need to learn and use certain skills to write the code.

5.5.2.3.1.1 How to get the text output from voice input using the voice recognition software

5.5.2.3.1.2 Creating a GUI for a text editor

5.5.2.3.1.3 Creating an algorithm to accurately follow user's command

5.5.3 Scheduling feasibility

5.5.3.1 Project must be completed by end of two semesters

5.5.3.1.1 We will need to finish all planning and design of project before end of first semester

5.5.3.1.2 Using a hybrid methodology of Agile and Prototyping and a template to match will help us break up the project into phases and appropriate deadlines

5.5.3.1.3 Using open-source software will save time and allow us to spend that saved time on refining and improving existing voice programming models

5.5.3.1.4 Weekly progress reports will show our progress and upcoming tasks

5.5.4 Financial feasibility

5.5.4.1 Costs of the project should be minimal

5.5.4.1.1 a microphone for voice input

5.5.4.1.2 open-source software is free; licensing would be the highest cost if the open-source is not available

5.5.5 Operational feasibility

5.5.5.1 Is the project feasible from an operational perspective?

5.5.5.1.1 This app will require users to be willing to learn a new set of commands using a new medium to input those commands. If the way we implement these commands are not intuitive and easy to learn, it will be hard to convince people to continue to use the program and meet the overall goal of making it easier to code.

5.5.6 Social and Ethical Considerations

5.5.6.1 Ethical factors

5.5.6.1.1 We will not be storing any of the voice input from the editing sessions, therefore avoiding ethical use of the data

5.5.6.1 Social factors

5.5.6.1.1 This project's target audience are programmers that have trouble with typing on their keyboard. We need to make sure to minimize the usage of the keyboard to appeal to this demographic.

5.5.6.1.2 Continued operation would include adding requested and common commands that would be useful to the users.

5.6 Software Requirements Specifications (SRS) (version 4390.10.1)

5.6.1 Overall Description

5.6.1.1 Definitions, acronyms, and abbreviations

5.6.1.1.1 **user**: The person, or persons, who operate or interact directly with the product [11]

5.6.1.1.2 **SRS**: Software Requirements Specifications

5.6.1.1.3 **Natural Language Interface**: (NLI) a user interface that allows people to interact using a human language, such as English, as opposed to a computer language, command line interface, or GUI [12]

5.6.1.1.4 **Menu Driven Interface**: (MDI) an interface consisting of a series of screens which are navigated by choosing options from lists, i.e. menus. ("Menu" is not used here to refer to pull down menus, but to lists of options on the screen that lead to other screens.) [12]

5.6.1.1.5 **Graphical User Interface**: (GUI) pronounced "GOOEY". A user interface that presents information graphically, typically with draggable windows, buttons, and icons, as opposed to a textual user interface, where information is presented on a text-based screen and commands are all typed [12]

5.6.1.2 Product Perspective

The project will put greater emphasis on the use of a natural language interface and minimize the use of a graphical user interface and command line interface. It should "work reasonably well without the need for users to provide input character-by-character" [10]. It could make use of menu driven interfaces in conjunction with the natural language interface. It could be "implemented either as a standalone editor or as a plugin to existing IDEs" [10]. It should "identify typical mistakes made by speech recognition systems when dealing with voice-based input in the context of programming" [10]. We also want the user interactions to be intuitive with the system and less of a burden on the user to learn a large vocabulary that is counter to a normal programmer environment.

5.6.1.3 User Characteristics

The main user is characterized as being a software developer suffering from "repetitive strain injuries such as carpal tunnel syndrome and tendonitis" [10]. "In addition, it can enable individuals with upper-body motor impairments, e.g, due to spinal cord injuries or strokes, to write code" [10]. The user is assumed to be a person with some software development knowledge including typical programming terminology.

5.6.1.4 Constraints

"The only constraint is that no commercial tools or services, e.g., Google's speech recognition service, must be required for the system to work." [10]. The project will be designed to work with Python as the user's programming language. Machine learning may be necessary to improve the quality of language recognition and make it "tolerant of imprecision and be able to handle unforeseen cases. The downside is that constructing the dataset to employ such an approach is non-trivial" [10].

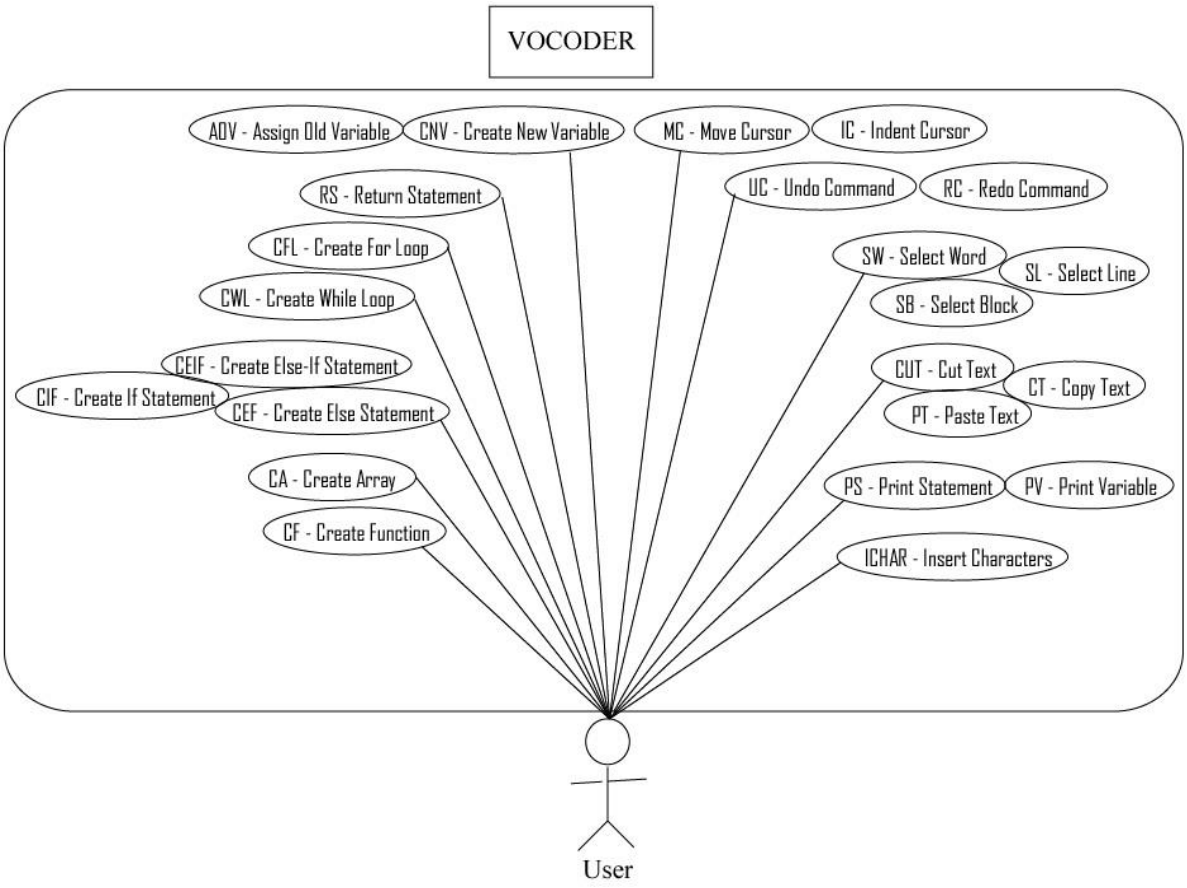
5.6.1.5 Assumptions and Dependencies

A language recognition system is a large part of this project and depends on us acquiring a usable product that already has a substantial base and would only need fine tuning on our part. The success of this project could be hindered by a sub-par language interface that requires more of our attention than initially allotted. If machine learning is used to enhance the language interface, it also would need to be an available resource that only needs some minor adjustments and not something that takes away from the overall goals of the project. Using the language interface in conjunction with other interfaces could reduce its weight on the project as a whole making it more manageable and feasible.

5.6.2 Specific Requirements

5.6.2.1 Use Cases

Overall use case diagram for the system and provided below are case descriptions for all the use cases shown



5.6.2.1.1 Use Case #1

Use Case ID:	CNV		
Use Case Name:	Create New Variable		
Priority:	High	**Trigger:**	
Date Created:	9/11/2020	**Last Revision Date**	10/6/2020
Actors:	User		
Description:	System initializes a new variable using a provided name, value(s), and operator(s).		
Preconditions:	If user wishes to assign a new value to a variable, that variable must already exist within scope		
Postconditions:	Variable name will be added to list of names in use		
Normal Flow:	1. User calls the *create new variable* command 2. System determines the variable name from voice input 3. System determines the value(s) from voice input 4. System determines if any operators were in voice input 5. System checks to see if desired variable name is in use 6. System outputs the statement assigning the value to the variable		
Alternative Flow	6a. System confirms that the desired variable name is in use 7a. System alerts the user that the name is taken and to say a new name to use 8a. System gets new name from voice input 9a. System checks to see if new variable name is in use Flow goes to either 6 or 6a		
Exceptions:			

5.6.2.1.2 Use Case #2

Use Case ID:	AOV		
Use Case Name:	Assign Old Variable		
Priority:	High	**Trigger:**	
Date Created:	10/4/2020	**Last Revision Date**	10/6/2020
Actors:	User		
Description:	System assigns a value to an already declared variable.		
Postconditions:	Value of variable is changed to new one		
Normal Flow:	1. User calls the *assign old variable* command 2. System determines the variable name from voice input 3. System determines the value(s) from voice input 4. System determines if any operators were in voice input 5. System checks to see if desired variable name is in use 6. System outputs the statement assigning the value to the variable		
Alternative Flow	6a. System confirms that the desired variable has not been initialized 7a. System alerts the user that the variable does not exist and to say the name again 8a. System gets name from voice input 9a. System checks to see if variable name is in use Flow goes to either 6 or 6a		

5.6.2.1.3 Use Case #3

Use Case ID:	RS		
Use Case Name:	Return Statement		
Priority:	High		
Date Created:	9/12/2020	**Last Revision Date**	10/6/2020
Actors:	User		
Description:	System will output a return statement		
Preconditions:	If user wishes to return a variable, variable must already exist within scope		
Postconditions:			
Normal Flow:	1. User calls the *return statement* command 2. System determines the desired return value from input 3. System outputs the statement returning the value		
Alternative Flow	3a. System determines that a variable is to be returned 4a. System looks up the variable name and makes sure it is in the scope and can be returned 5a. System outputs the statement returning the variable 5b. System alerts that it could not find variable in use or it is not in the scope 6b. System prompts the user to say the variable name again Flow goes to either 5a or 5b. 3c. System did not receive any value/variable to return 4c. System outputs the return command		

5.6.2.1.4 Use Case #4

Use Case ID:	CFL		
Use Case Name:	Create For Loop		
Priority:	Medium		
Date Created:	9/13/2020	**Last Revision Date**	10/6/2020
Actors:	User		
Description:	System will output a for loop using a specified looping variable and condition		
Preconditions:			
Postconditions:	A for loop is created and the cursor is placed in the line after the for loop, indented 4 spaces in.		
Normal Flow:	1. User calls the *for loop* command with a looping variable and condition 2. System gets looping variable from input 3. System checks if looping variable is already in use in scope 4. System gets looping condition from input 5. System outputs the for loop statement 6. System moves cursor to the next line and indents 4 spaces in		
Alternative Flow	4a. If variable is not already in scope, initializes a new variable with that name with value 0 Goes to 5		

5.6.2.1.5 Use Case #5

Use Case ID:	CWL		
Use Case Name:	Create While Loop		
Priority:	Medium		
Date Created:	10/4/2020	**Last Revision Date**	
Actors:	User		
Description:	System will output a while loop using a specified looping condition		
Postconditions:	A while loop is created and the cursor is placed in the line after the while loop, indented 4 spaces in.		
Normal Flow:	1. User calls the *while loop* command with a looping variable and condition 2. System gets looping condition from input 3. System outputs the while loop statement 4. System moves the cursor to the next line and indents 4 spaces in		

5.6.2.1.6 Use Case #6

Use Case ID:	CIF		
Use Case Name:	Create If Statement		
Priority:	Medium		
Date Created:	9/13/2020		
Actors:	User		
Description:	System will output an if statement		
Preconditions:			
Postconditions:	If statement is created and the cursor is moved to the next line and indented 4 spaces to the right.		
Normal Flow:	1. User calls the *if statement* command 2. System determines the condition from input 3. System outputs the if statement block 4. System moves the cursor to the next line and indents 4 spaces in		
Alternative Flow	3b. If a variable is used in the condition, system checks if it is valid 4b. If variable is not valid to use, system prompts user to say condition again Flow goes to 2		

5.6.2.1.7 Use Case #7

Use Case ID:	CEIF		
Use Case Name:	Create Else-If Statement		
Priority:	Medium		
Date Created:	10/4/2020		

Actors:	User
Description:	System will output an else-if statement
Postconditions:	Else-If statement is created and the cursor is moved to the next line and indented 4 spaces in.
Normal Flow:	1. User calls the *else-if statement* command 2. System checks if there is an if statement above the cursor already inside the scope 3. System determines the condition from input 4. System outputs the else-if statement block 5. System moves the cursor to the next line and indents 4 spaces in
Alternative Flow	3a. If there is no if statement, the system alerts the user of it and exits the command. 3b. If a variable is used in the condition, system checks if it is valid 4b. If variable is not valid to use, system prompts user to say condition again Flow goes to 3

5.6.2.1.8 Use Case #8

Use Case ID:	CEF		
Use Case Name:	Create Else Statement		
Priority:	Medium		
Date Created:	10/4/2020		
Actors:	User		
Description:	System will output an else statement		
Postconditions:	Else statement is created and the cursor is moved to the next line and indented 4 spaces in.		
Normal Flow:	1. User calls the *else statement* command 2. System checks if there is an if statement above the cursor already inside the scope 3. System outputs the else statement block 4. System moves the cursor to the next line and indents 4 spaces in		
Alternative Flow	3a. If there is no if statement, the system alerts the user of it and exits the command.		

5.6.2.1.9 Use Case #9

Use Case ID:	CA		
Use Case Name:	Create Array		
Priority:	Medium		
Date Created:	10/4/2020		
Actors:	User		
Description:	System will create an empty 1D array with a specified name		
Preconditions:	Desired name is not already in use		
Postconditions:	List name will be added to list of names being used		
Normal Flow:	1. User calls the *create array* command 2. System gets array name from voice input 3. System outputs statement creating an empty 1D array with the specified name		
Alternative Flow:	3a. If array name is already taken, system prompts user to say another name to use instead - Goes to 2		

5.6.2.1.10 Use Case #10

Use Case ID:	MC		
Use Case Name:	Move Cursor		
Priority:	Low		
Date Created:	9/13/2020		

Actors:	User
Description:	System will move cursor to a specified position
Preconditions:	
Postconditions:	Cursor will be in a new position
Normal Flow:	1. User calls *move cursor* command 2. System gets direction to move from input 3. System moves the cursor

5.6.2.1.11 Use Case #11

Use Case ID:	CUT		
Use Case Name:	Cut Text		
Priority:	Low		
Date Created:	3/01/2021		
Actors:	User		
Description:	Deletes currently selected text from text window and adds it to the clipboard		
Precondition:	Text has been highlighted using a select command		
Postconditions:	Cursor will be at place where text was deleted, text is on the clipboard		
Normal Flow:	1. User calls *cut text* command 2. System deletes text from text window 3. System leaves the cursor at place of deleted text 4. System adds deleted text to the clipboard		

5.6.2.1.12 Use Case #12

Use Case ID:	UC		
Use Case Name:	Undo Command		
Priority:	Low		
Date Created:	9/13/2020		
Actors:	User		
Description:	System will execute the editor undo command		
Preconditions:	An action that can be undone has already happened		
Postconditions:	Text file state goes back by one		
Normal Flow:	1. User calls *undo* command 2. System executes the undo command 3. System relays to user what was undone		

5.6.2.1.13 Use Case #13

Use Case ID:	RC		
Use Case Name:	Redo Command		
Priority:	Low		
Date Created:	9/13/2020		
Actors:	User		
Description:	System will execute the editor redo command		
Preconditions:	Undo was previously done and an action can be redone		
Postconditions:	Text file state goes forward by one		
Normal Flow:	1. User calls *redo* command 2. System executes the redo command 3. System relays to user what was redone		

5.6.2.1.14 Use Case #14

Use Case ID:	SW		
Use Case Name:	Select Word		
Priority:	Low		
Date Created:	10/4/2020		
Actors:	User		
Description:	System will select the current word that the cursor is on		
Postconditions:	The word that the cursor is on is selected		
Normal Flow:	1. User calls the *select word* command 2. System selects the current word that the cursor is on		

5.6.2.1.15 Use Case #15

Use Case ID:	SL		
Use Case Name:	Select Line		
Priority:	Low		
Date Created:	10/4/2020		
Actors:	User		
Description:	System will select the current line that the cursor in on		
Postconditions:	The line that the cursor is on is selected		
Normal Flow:	1. User calls the *select line* command 2. System selects the current line that the cursor is on		

5.6.2.1.16 Use Case #16

Use Case ID:	SB		
Use Case Name:	Select Block		
Priority:	Low		
Date Created:	10/4/2020		
Actors:	User		
Description:	System will select the current line and every adjacent line without an empty line that is on the same indentation level		
Postconditions:	A block of code is selected		
Normal Flow:	1. User calls the *select block* command 2. System finds the current block that the cursor is on		

5.6.2.1.17 Use Case #17

Use Case ID:	CT		
Use Case Name:	Copy Text		
Priority:	Low		
Date Created:	9/13/2020	**Last Revision Date**	10/4/2020
Actors:	User		
Description:	Copies currently selected text into clipboard		
Preconditions:	Text has been highlighted using select command		
Postconditions:	Selected text has been added to clipboard		
Normal Flow:	1. User issues *copy text* command 2. System puts selected text into clipboard		

5.6.2.1.18 Use Case #18

Use Case ID:	PT		
Use Case Name:	Paste Text		
Priority:	Low		
Date Created:	9/13/2020	**Last Revision Date**	10/4/2020

Actors:	User
Description:	Content in the clipboard is pasted at the cursor's position
Preconditions:	There is text in the clipboard that can be pasted
Postconditions:	Text is pasted from the clipboard to the cursor's position
Normal Flow:	1. User issues *paste text* command 2. System pastes from clipboard at cursor's position

5.6.2.1.19 Use Case #19

Use Case ID:	PS		
Use Case Name:	Print Statement		
Priority:	High		
Date Created:	1/29/2021	**Last Revision Date**	1/29/2021
Actors:	User		
Description:	when command is called a print statement is created in the text editor		
Postconditions:	a Python print statement of the form: print(“spoken words”) is created in the text editor window and the cursor is moved to the next line		
Normal Flow:	1. User issues *print statement* command 2. user is prompted for desired statement to print 3. system asks for verification 4. user verifies with “yes” 5. system creates a Python print statement in the text window		
Alternate Flow:	4a. user verifies with “no” 4b. system asks for statement again, goes to 3		

5.6.2.1.20 Use Case #20

Use Case ID:	PV		
Use Case Name:	Print Variable		
Priority:	High		
Date Created:	1/29/2021	**Last Revision Date**	1/29/2021

Actors:	User
Description:	when command is called a print variable statement is created in the text editor
Postconditions:	a Python print statement of the form: print(variable) is created in the text editor window and the cursor is moved to the next line
Normal Flow:	1. user issues *print variable* command 2. user is prompted for desired variable to print 3. system asks for verification 4. user verifies with “yes” 5. system creates a Python print statement in the text window
Alternate Flow:	4a. user verifies with “no” 4b. system asks for variable again, goes to 3

5.6.2.1.21 Use Case #21

Use Case ID:	CF		
Use Case Name:	Create Function		
Priority:	High		
Date Created:	1/29/2021	**Last Revision Date**	1/29/2021
Actors:	User		
Description:	when command is called a function statement is created in the text editor		
Postconditions:	a Python function statement of the form: def func(): is created in the text editor window and the cursor is moved to the next line with indent		
Normal Flow:	1. User issues *create function* command 2. user is prompted for desired name of function 3. system asks for verification 4. user verifies with “yes” 5. system asks for number of arguments, verification 6. system asks for names of arguments, verification 7. system creates function in text window		
Alternate Flow:	4a. user verifies with “no” 4b. system asks for name again, goes to 3 5b. system asks for number again 6b. system asks for names again		

5.6.2.1.22 Use Case #22

Use Case ID:	IC		
Use Case Name:	Indent Cursor		
Priority:	Low		
Date Created:	4/01/2021	**Last Revision Date**	04/01/2021
Actors:	User		
Description:	when command is called the cursor is indented in the text editor		
Postconditions:	the cursor is indented 4 spaces in the text editor		
Normal Flow:	1. User issues *indent cursor* command 2. System places a string of 4 spaces at current cursor location		

5.6.2.1.23 Use Case #23

Use Case ID:	ICHAR		
Use Case Name:	Insert Characters		
Priority:	Low		
Date Created:	4/01/2021	**Last Revision Date**	04/01/2021
Actors:	User		
Description:	when command is called the system will insert spoken string of characters at current cursor location in the text editor		
Postconditions:	new string of character(s) has been added in the text editor		
Normal Flow:	1. User issues *insert characters* command 2. user is prompted for desired character(s) 3. system asks for verification 4. user verifies with “yes” 5. system inserts characters at cursor location		
Alternate Flow:	4a. user verifies with “no” 4b. system asks for string again, goes to 3		

5.6.2.2 Functional Requirements

5.6.2.2.1 The system must receive input from the user using a Natural Language Interface
5.6.2.2.2 The system must receive input from the user using a Menu Driven Interface
5.6.2.2.3 The system could receive input from the user using a Graphical User Interface
5.6.2.2.4 The system must allow the user to create statements
- 5.6.2.2.4.1 statements that give a value to a variable
- 5.6.2.2.4.2 statements that return a value
- 5.6.2.2.4.3 if statements including if, else if, else

5.6.2.2.5 The system should allow the user to create a loop
- 5.6.2.2.5.1 for loop
- 5.6.2.2.5.2 while loop

5.6.2.2.6 The system should be able to navigate through the program
- 5.6.2.2.6.1 with a cursor position
- 5.6.2.2.6.2 with a word position

5.6.2.2.7 The system should handle creating arrays
5.6.2.2.8 The system could execute an undo command
5.6.2.2.9 The system could execute a redo command
5.6.2.2.10 The system could handle clipboard functions
- 5.6.2.2.10.1 select text
  - 5.6.2.2.10.1.1 select a word
  - 5.6.2.2.10.1.2 select a line of text
  - 5.6.2.2.10.1.3 select a block of text
- 5.6.2.2.10.2 copy text
- 5.6.2.2.10.3 paste text

5.6.2.3 Non-Functional Requirements

5.6.2.3.1 Performance
- 5.6.2.3.1.1 Startup time
  The system should be ready for editing in less than 10 seconds from startup.
- 5.6.2.3.1.2 Response time
  The system should take less than 500ms to display the result from any voice commands.

5.6.2.3.2 Reliability
The system should maintain accuracy of at least 80% when responding to user's voice commands

5.6.2.3.3 Availability
- 5.6.2.3.3.1 Voice Input Device Connection
  The system requires a channel of communication to a voice input device.
- 5.6.2.3.3.2 System Availability
  The system should be available for the user without requiring an internet connection to work.

5.6.2.3.4 Security
The system should not share or upload any recorded audio from the editing sessions.

5.6.2.3.5 Maintainability
- 5.6.2.3.5.1 System Testability
  The system should be designed in a way that individual functions can be tested separately from the main system and each of those functions can be tested as they are implemented without affecting the whole system.

- 5.6.2.3.5.2 System Modifiability

The system should be designed in a way that when new features are added there is minimal duplication of code that would make modifying the code cumbersome. The code will not be lacking in documentation to fully understand how the code works and sufficient refactoring will be achieved to allow for seamless modifications.

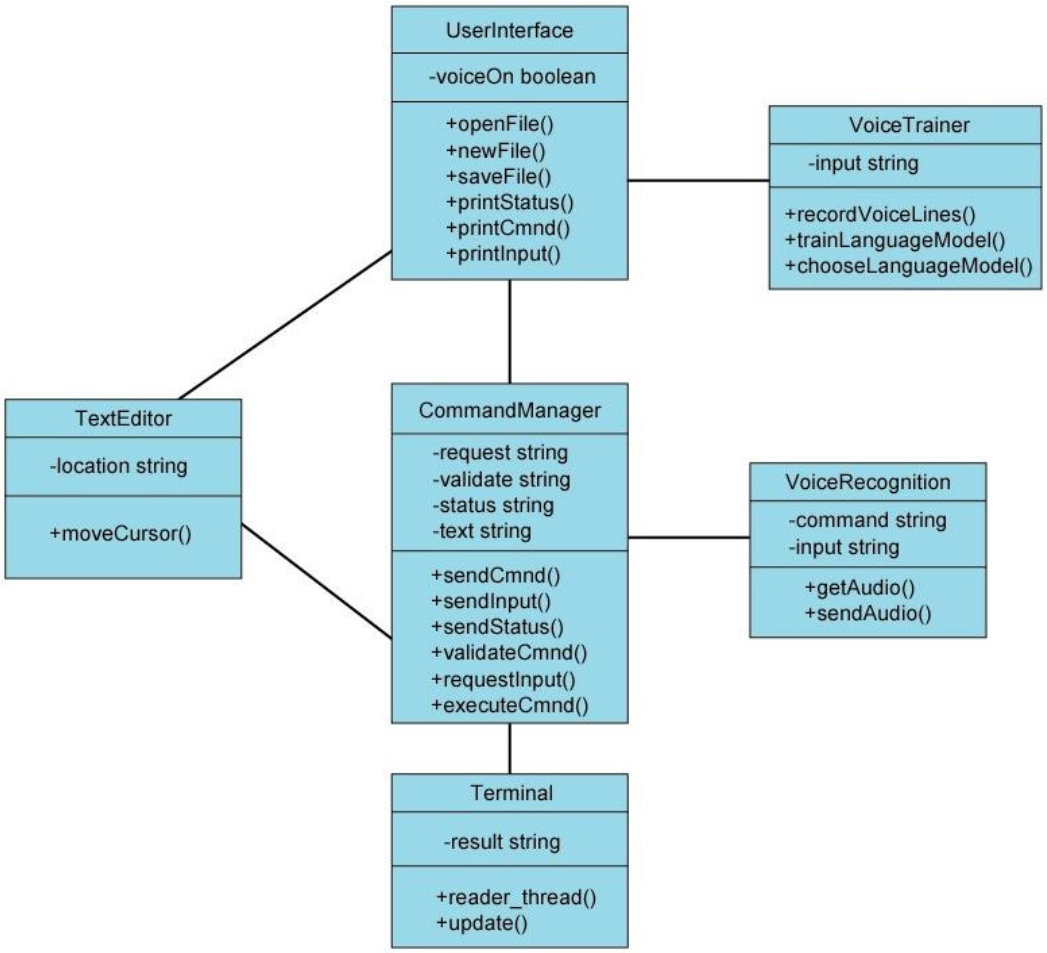
5.7 Software Design (SDD) (version 4391.4.1)

5.7.1 Technologies Used

Language: Python; Tools: CMU Sphinx, GitHub, Google Drive, Adobe Illustrator, Diagrams.net

5.7.2 UML Design Diagrams

5.7.2.1 Class Diagram

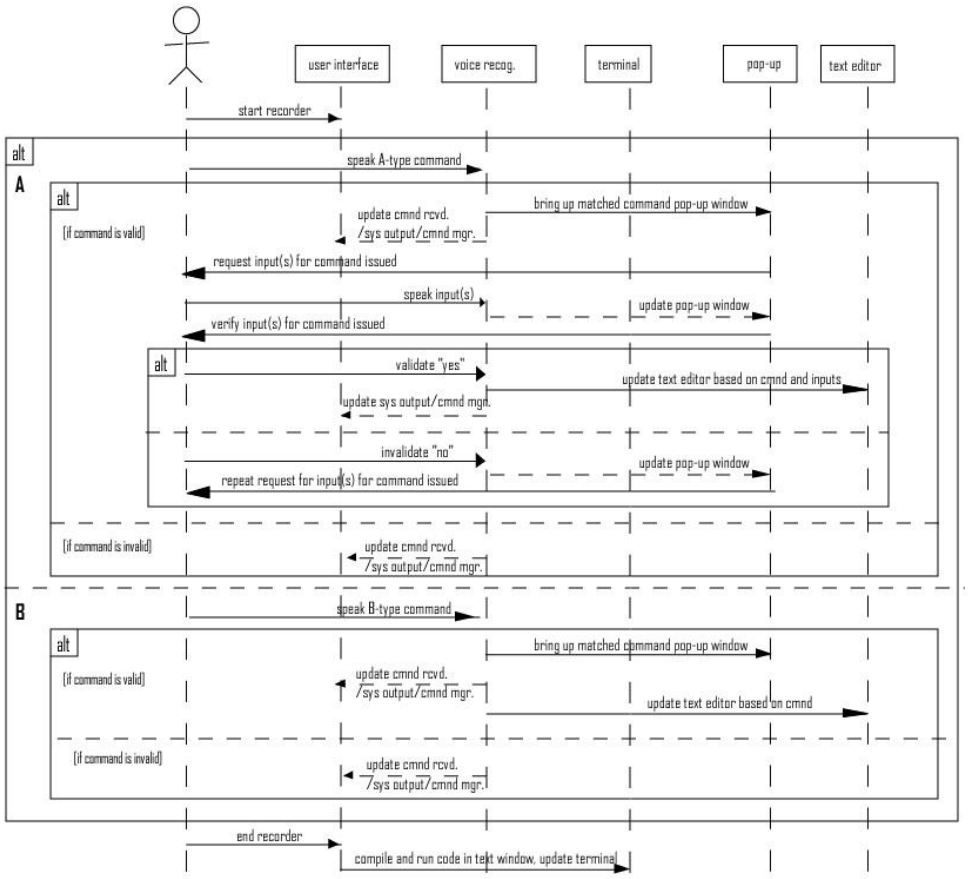


UserInterface interacts with the TextEditor to open, save and start a new file for editing. UserInterface and TextEditor interact with CommandManager when receiving output. UserInterface reacts with VoiceTrainer for improving voice recognition system. CommandManager receives input from VoiceRecognition. Terminal receives input from the CommandManager.

5.7.2.2 Sequence Diagram

A-type command:
CNV - Create New Variable
CA - Create Array
CEF - Create Else Statement
CEIF - Create Else-If Statement
CIF - Create If Statement
CWL - Create While Loop
CFL - Create For Loop
CF - Create Function
RS - Return Statement
AOV - Assign Old Variable
ICHAR - Insert Characters
PS - Print Statement
PV - Print Variable
SB - Select Block
SL - Select Line
SW - Select Word
MC - Move Cursor

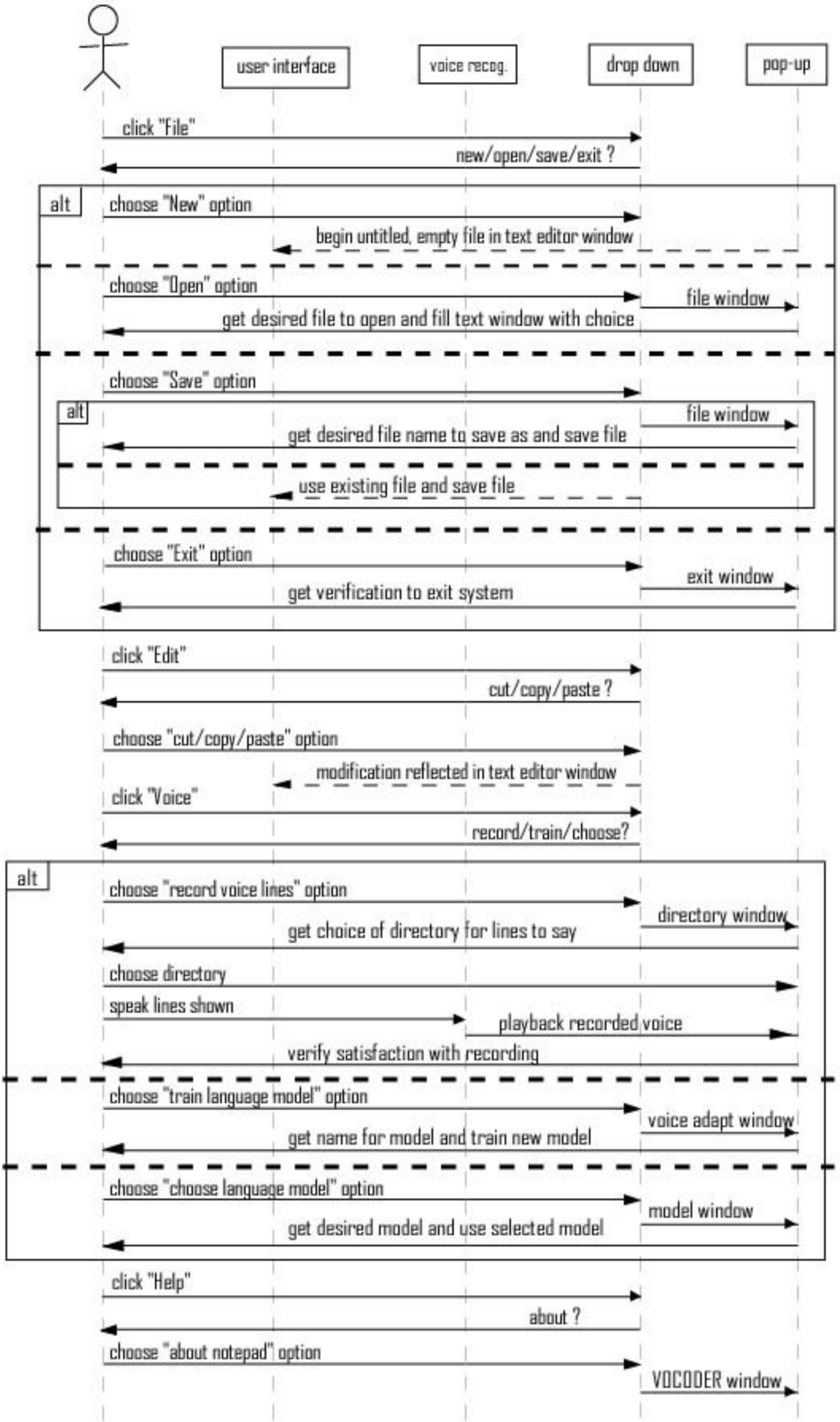
B-type command:
CT - Copy Text
CUT - Cut Text
PT - Paste Text
IC - Indent Cursor
RC - Redo Command
UC - Undo Command



A-type commands involve an extra step of creating inputs such as the user creating an if statement. That would require specifying the conditions of the if statement, like: if (x > y): s = "x is less than y" B-type commands do not involve more steps beyond simply executing the command given with a location or a begin and end position (cursor position)

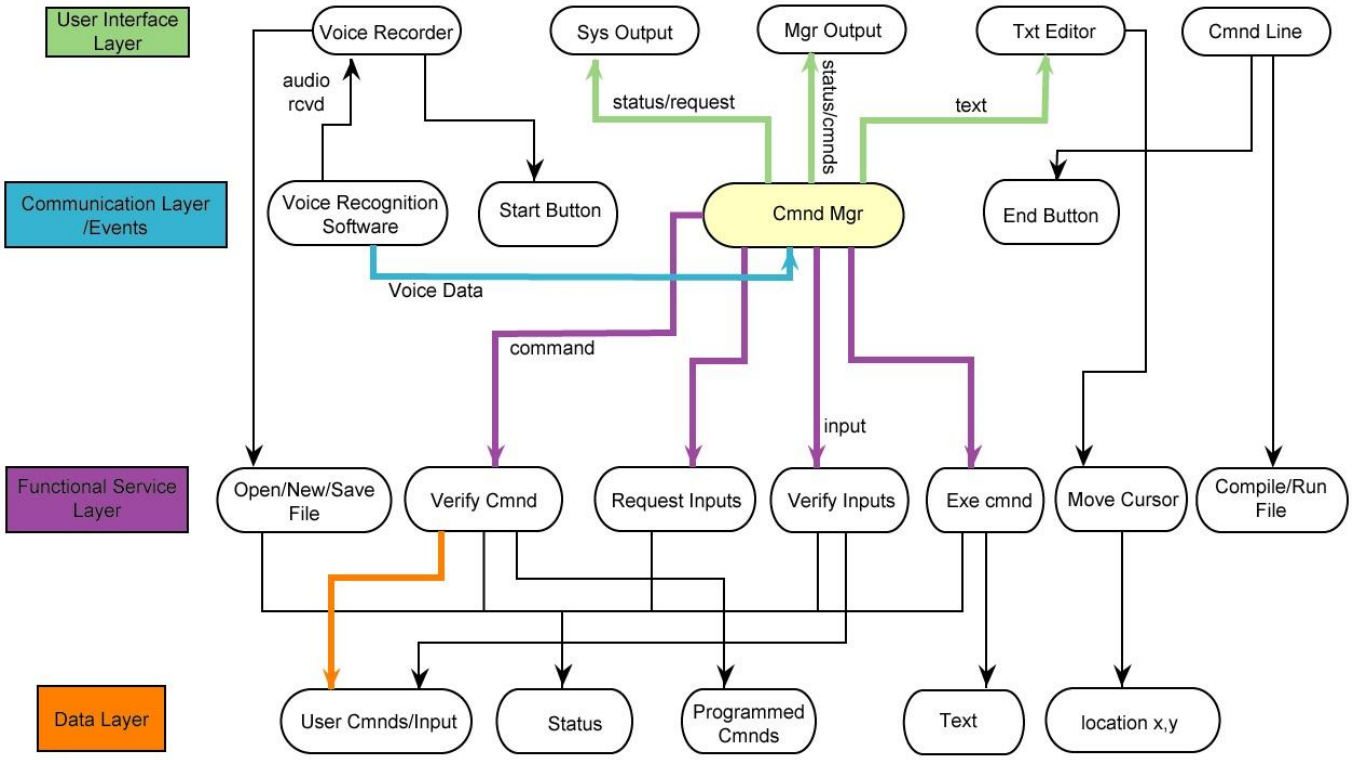
*additions of A-Type commands: Print Statement, Print Variable, Create Function 1/29/2021; Insert Characters 3/01/21. additions of B-Type commands: Indent Cursor 3/01/21

*addition of File System and Voice Training System 2/01/21; diagram continued on next page


user interface
voice recog.
drop down
pop-up
click "File"
new/open/save/exit ?
alt
choose "New" option
begin untitled, empty file in text editor window
choose "Open" option
file window
get desired file to open and fill text window with choice
choose "Save" option
alt
file window
get desired file name to save as and save file
use existing file and save file
choose "Exit" option
exit window
get verification to exit system
click "Edit"
cut/copy/paste ?
choose "cut/copy/paste" option
modification reflected in text editor window
click "Voice"
record/train/choose?
alt
choose "record voice lines" option
directory window
get choice of directory for lines to say
choose directory
speak lines shown
playback recorded voice
verify satisfaction with recording
choose "train language model" option
voice adapt window
get name for model and train new model
choose "choose language model" option
model window
get desired model and use selected model
click "Help"
about ?
choose "about notepad" option
VOCODER window

5.7.3 Software Architecture

5.7.3.1 Architecture Diagram



The User Interface Layer is composed of the visual aspects that the user directly interacts with or reads system generated messages from. The next layer is the Communication Layer and Events Layer. These are the components that are generating output or receiving input from the User Interface Layer. The Functional Service Layer is the main part of the system that is executing the necessary actions based on the commands and inputs received in the Communication Layer. Lastly we have the data layer that contains user generated data and pre-programmed data that will be used to circle back to the User Interface for communicating to the User everything that the system has processed.

5.7.3.2 Component level design

5.7.3.2.1 User Interface Layer

5.7.3.2.1.1 Voice Recorder - responsible for receiving user inputs of audio and mouse clicks, will open file if started and close file upon stopping, relays to user what the Voice Recognition Software recorded

5.7.3.2.1.2 System Output - relays to user status of system and incoming requests from the command Manager

5.7.3.2.1.3 Manager Output - relays to user status of Command Manager and inputs that it has received for current command it is processing

5.7.3.2.1.4 Text Editor - shows user what has been programmed so far and is responsible for cursor movement, receives text from the Command Manager

5.7.3.2.1.5 Command Line - a place to compile/run file created by user

5.7.3.2.2 Communication/Events Layer

5.7.3.2.2.1 Voice Recognition Software - sends received audio to the Voice Recorder and the processed audio to the Command Manager for verification

5.7.3.2.2.2 Start Button - user clicks button to start recording

5.7.3.2.2.3 Command Manager - receives voice commands and inputs from the Voice Recognition Software, relays the status/requests/commands to the User Interface, verifies commands, requests/verifies inputs, executes completed commands to the Text Editor, communicates with the Syntax Processor to relay errors to the user if applicable

5.7.3.2.2.2 End Button - user clicks button to compile and run code in text editor window

5.7.3.2.3 Functional Service Layer

5.7.3.2.3.1 Open/New/Save File - receives notification of drop down menu activated to open/close file for the text editor window in the User Interface

5.7.3.2.3.2 Verify Command - receives commands from Command Manager and compares to known pre-programmed commands to determine validity, picks up status statements for User Interface output

5.7.3.2.3.3 Request Inputs - Command Manager picks up status and request statements for user interface

5.7.3.2.3.4 Verify Inputs - same as 4.2.3.2 but for requested inputs, does not need to make comparisons as inputs will be unique but checked with Syntax Processor

5.7.3.2.3.5 Execute Command - the Command Manager will execute the verified command with inputs and make changes to the Text Editor, picks up status statements for User Interface output

5.7.3.2.3.6 Move Cursor - retrieves location values and moves cursor within the Text Editor for applicable commands

5.7.3.2.3.7 Compile/Run File - process initiated by End Button

5.7.3.2.4 Data Layer

5.7.3.2.4.1 User Commands/Input - collected data of spoken commands and inputs

5.7.3.2.4.2 Status - programmed and generated statements for output

5.7.3.2.4.3 Programmed Commands - database of commands for verification

5.7.3.2.4.4 Text - assembled text from user Commands and Inputs used to create file in the Text Editor

5.7.3.2.4.5 Location X,Y - database of cursor locations used for moving the cursor and executing undo/redo, copy/paste, select commands

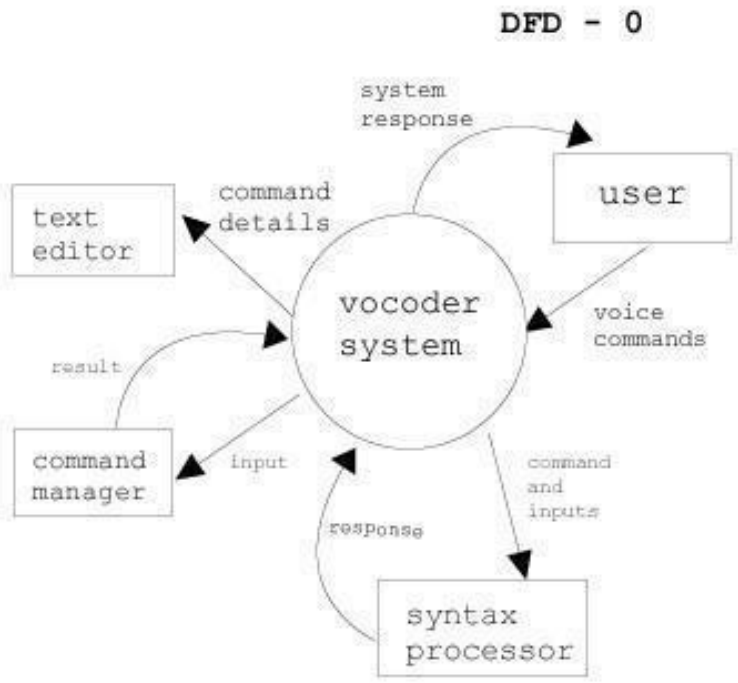
5.7.3.3 Architectural Alternatives

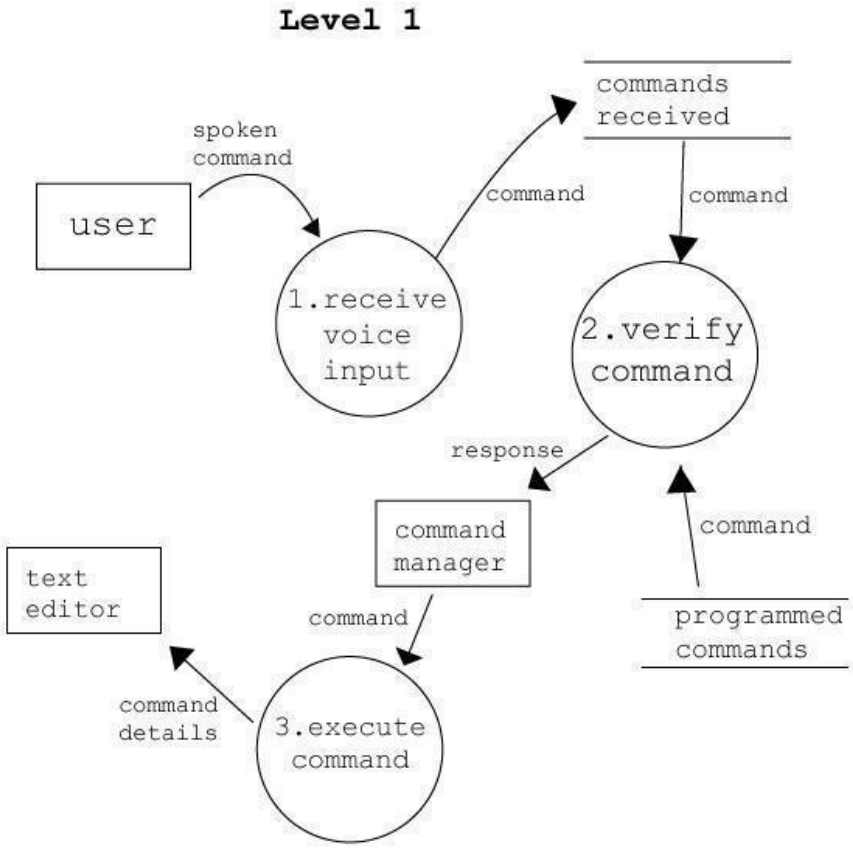
We briefly discussed the possibility of using a microkernel architecture but decided we would not be using the plug-in feature that this architecture uses.

5.7.3.4 Design Rationale

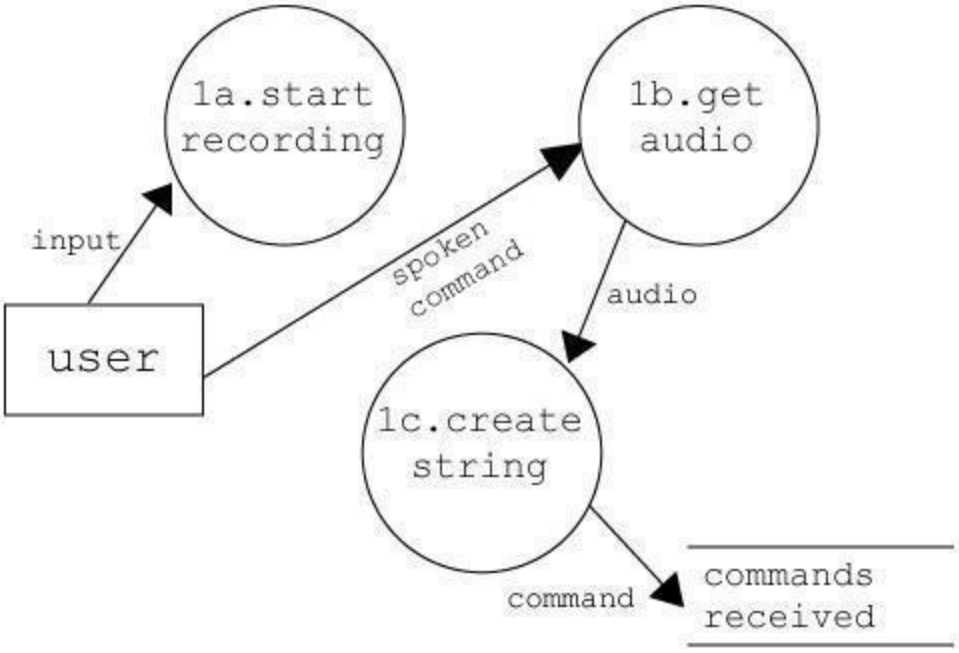
After studying several architecture types we decided that a layered architecture describes our project best. The layered architecture allows us to take advantage of it being “maintainable, testable, easy to assign separate roles and easy to update and enhance layers separately” [13]. According to techbeacon.com it is also considered best for “teams with inexperienced developers who don’t understand other architectures yet” and “applications requiring strict maintainability and testability standards” [13]. We also feel that the event-driven architecture best describes the voice recognition segment of our project, because we will have a command manager that receives all voice input and delegates the correct response and action from the system depending on what command was issued. As the project progresses we will be adding more functionality that addresses more commands and being able to efficiently and easily scale the project to accommodate these is a nice advantage of this architecture type [13].

5.7.4 Data Flow Diagram

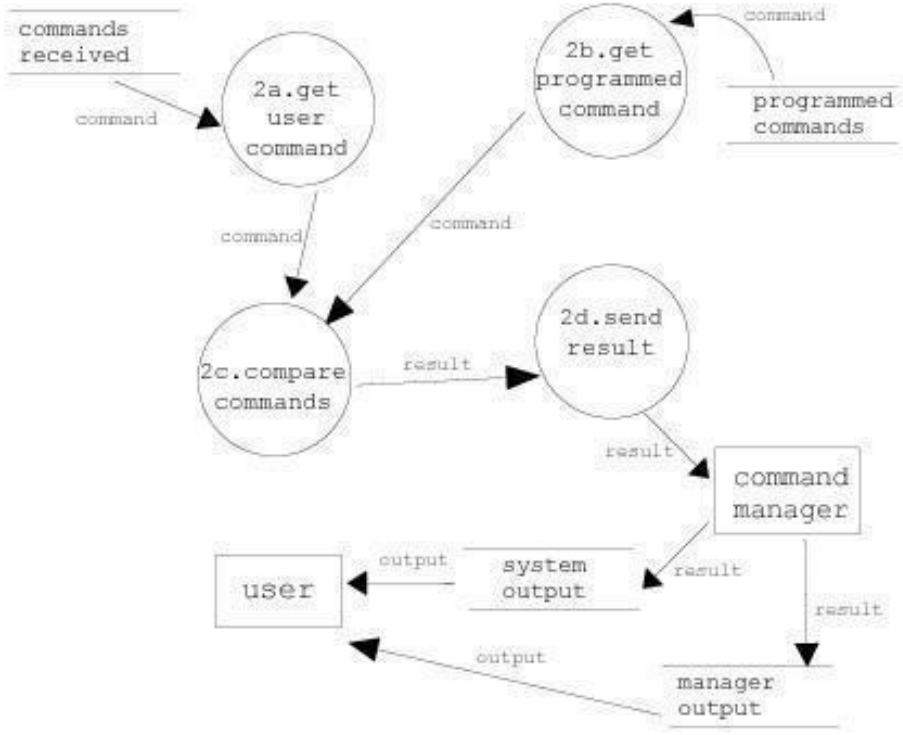


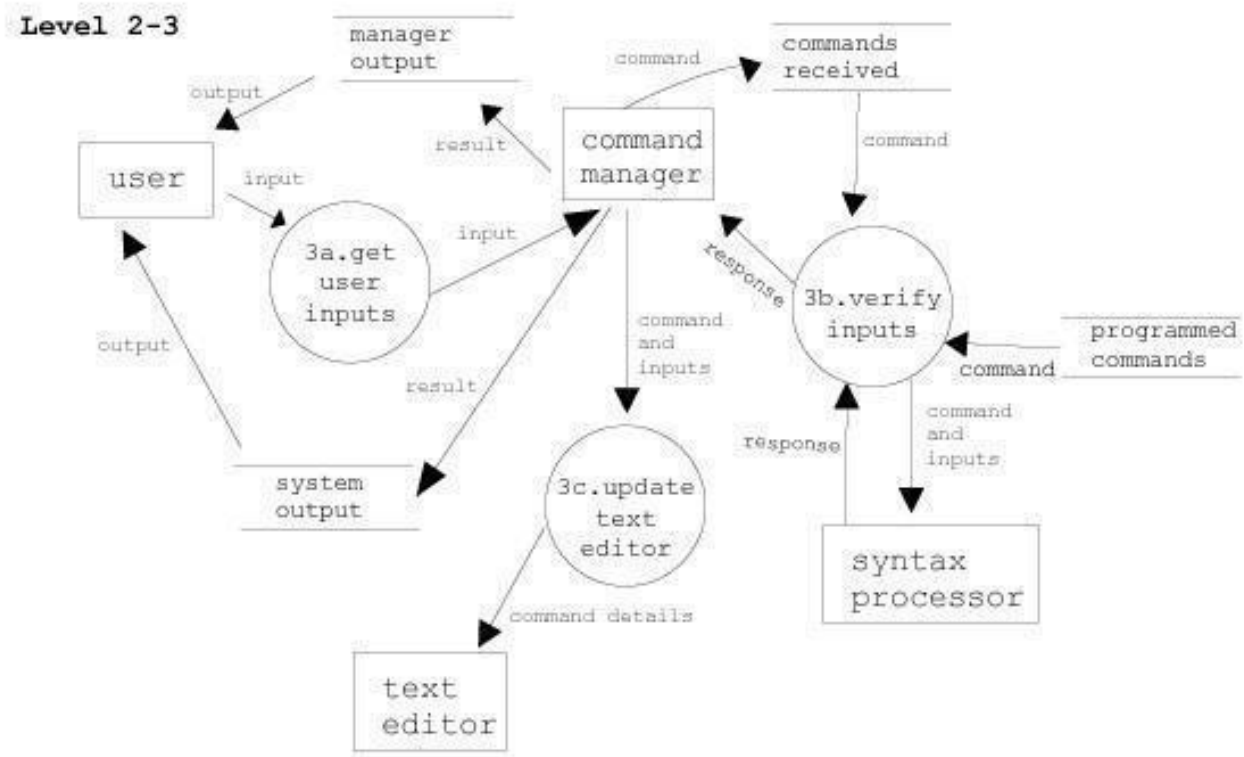


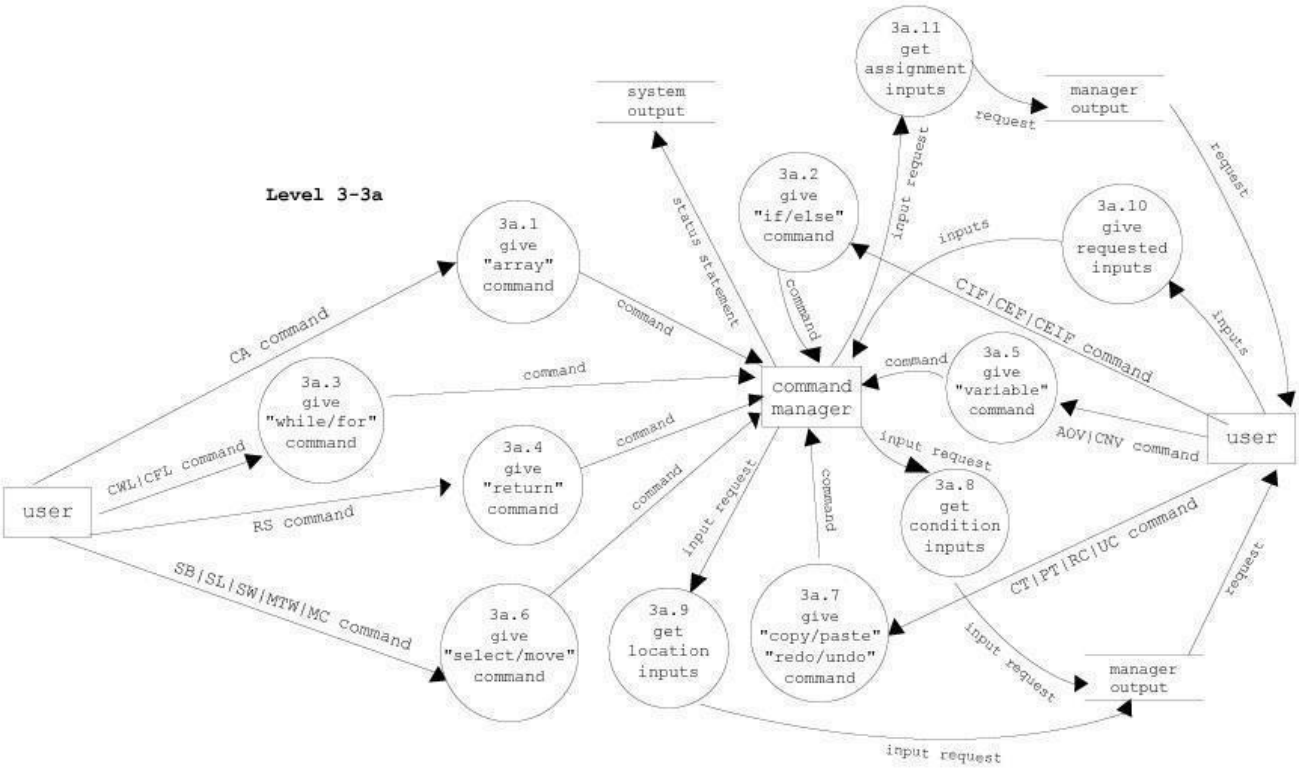
Level 2-1



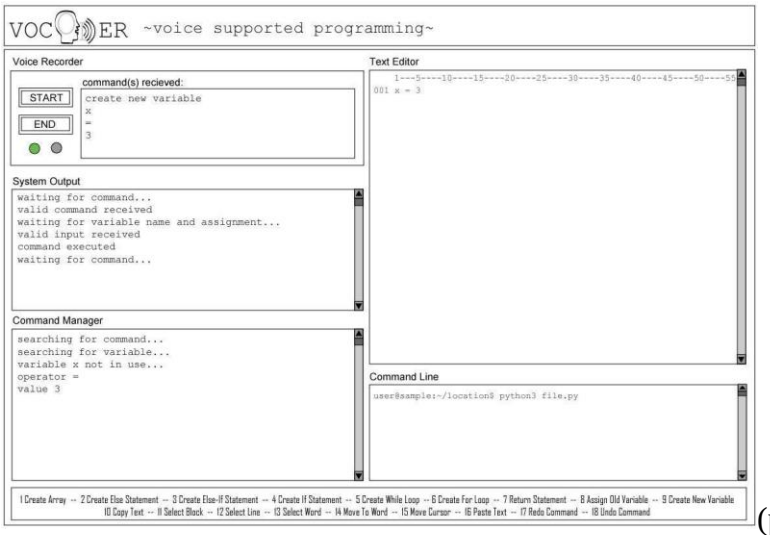
Level 2-2



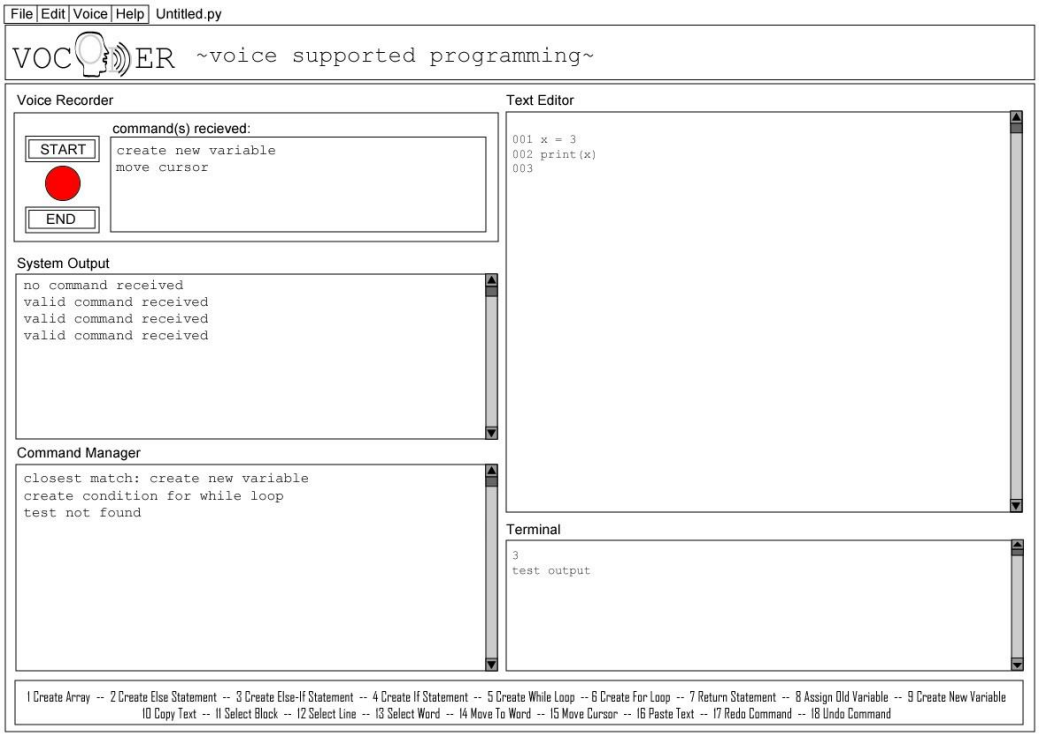

Level 2-3
manager output
command
commands received
output
result
command manager
command
user
input
3a.get user inputs
input
response
3b.verify inputs
programmed commands
output
command and inputs
result
command
response
command and inputs
system output
3c.update text editor
syntax processor
command details
text editor


Level 3-3a
3a.11 get assignment inputs
system output
request
manager output
request
3a.2 give "if/else" command
status statement
input request
inputs
3a.10 give requested inputs
3a.1 give "array" command
command
command
CIF|CEF|CEIF command
inputs
CA command
command
command
3a.5 give "variable" command
command manager
3a.3 give "while/for" command
command
input request
AOV|CNV command
user
CWL|CFL command
3a.4 give "return" command
command
3a.8 get condition inputs
user
RS command
command
input request
command
CT|PT|RC|UC command
request
SB|SL|SW|MTW|MC command
3a.6 give "select/move" command
3a.9 get location inputs
3a.7 give "copy/paste" "redo/undo" command
input request
manager output
input request

5.7.5 User Interface Design



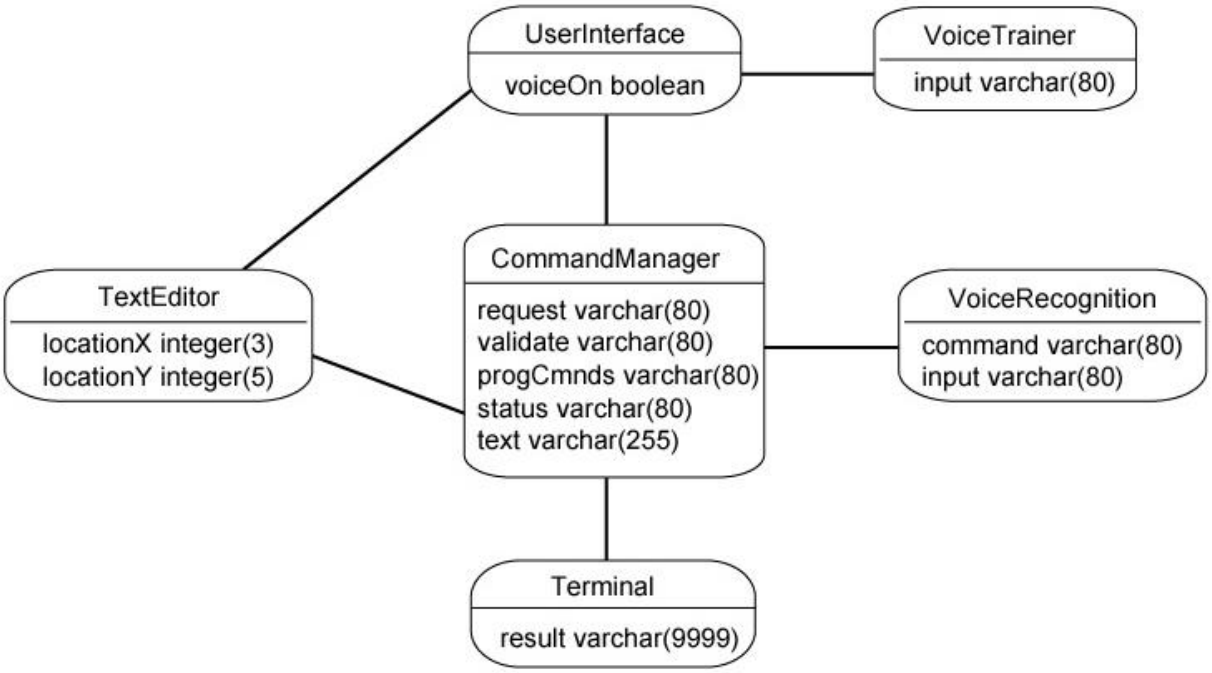
(proposed version)

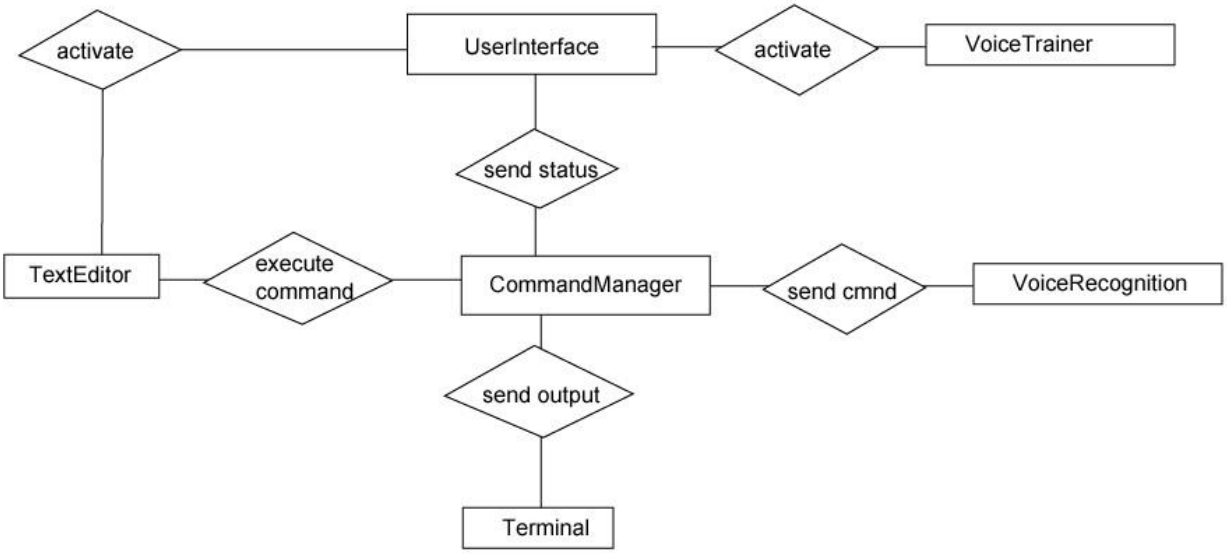


(final version)

Individual boxes for each process including the voice recorder, text editor, command manager, terminal and system output that interacts with the user to let them know what input or command the system is needing next. The voice recorder has an indicator to let the user know if it has started recording input or if the current recording session has ended. The command manager will be an indicator of what the system is processing before making changes to the text editor. The text editor includes a line number to help the user navigate by voice for the commands needing an x/y location. Beneath the text editor is a window to display the output for compiling and running the users code. A box at the bottom includes all possible commands as a quick guide reference to make it easier for the user to issue commands.

5.7.6 Data Model





5.7.7 Design constraints

The only constraint we are facing with our system is that “no commercial tools or services, e.g., Google’s speech recognition service, must be required for the system to work.” [10] This requires us to utilize open source voice recognition, such as CMU Sphinx. We will need to incorporate machine learning to increase the accuracy of the recognition in order to be comparable to a commercial tool. The choice of using pre-programmed commands for comparison to the spoken commands is also influenced by the accuracy of a non-commercial voice recognition tool.

5.8 Implementation

5.8.1 Presentation

https://docs.google.com/presentation/d/1gqG-PXa3qJPbnYf9UYf0whSLxYRlOvIylOaSF8PMnkk/edit#slide=id.gd3864351ec_0_1223

5.8.2 Core Tech Stacks

5.8.2.1 Python - Tkinter

5.8.2.2 CMU-Sphinx

5.8.2.3 Google Speech API

5.8.2.4 Sphinx-doc

5.8.3 Project Repository

https://github.com/anbinhpham1107/VOCODER

5.8.4 Programming Languages/Libraries

Language: Python

Libraries:

- tkinter
- speech_recognition
- PIL
- threading
- queue
- os
- sys
- glob
- shutil
- sounddevice
- scipy.io.wavfile.write
- pydub.AudioSegment
- pydub.playback.play
- screeninfo.get_monitors
- re
- fuzzywuzzy
- vosk.SetLogLevel
- compiler

5.8.5 Code Snippets

```
def saveFile(self):
    """Handling save file option for the menu"""
    if self.file == None:
        # Save as new file
        self.file = asksaveasfilename(initialfile='Untitled.txt',
                                      defaultextension=".txt",
                                      filetypes=[("All Files","*.*"),
                                          ("Text Documents","*.txt")])

        if self.file == "":
            self.file = None

        else:
            # Try to save the file
            file = open(self.file,"w")
            file.write(self.txt_editor_field.get(1.0,tk.END))
            file.close()
            # Change the window title
            self.root.title(os.path.basename(self.file) + " - VOCODER")

    else:
        file = open(self.file,"w")
        file.write(self.txt_editor_field.get(1.0,tk.END))
        file.close()
```

```
def getVoiceInput():
    """returns a string of the recorded voice input"""
    r = sr.Recognizer()
    with sr.Microphone() as source:
        audio = r.listen(source)

        # if useGoogleFlag has been assigned true, use google voice recognition
        if useGoogleFlag:
            audioToText = r.recognize_google(audio)
        else:
            path = resource_path("VoiceTraining/Profiles/en-US")
            print("inside getVoiceInput(): path = " + path)
            audioToText = r.recognize_sphinx(audio, language = path)

    return audioToText.lower()
```

```
def moveCursor(tex3, tex, prompt):
    """moves cursor in text editor window to user provided location"""
    correctLine = False
    while not correctLine:
        prompt.insert(tk.END,"State line number.\n")
        vInput = getVoiceInput()
        line = vInput
        prompt.insert(tk.END,"line number: " + line + "\nIs this correct? (Yes/No)")
        if confirm(prompt): correctLine = True

    correctColumn = False
    while not correctColumn:
        prompt.insert(tk.END,"State column number.\n")
        vInput = getVoiceInput()
        column = vInput
        prompt.insert(tk.END,"column number: " + column + "\nIs this correct? (Yes/No)")
        if confirm(prompt): correctColumn = True

    pos = line + '.' + column
    tex.tag_remove(tk.SEL, "1.0", tk.END)
    tex.mark_set(tk.INSERT, pos)
    tex3.insert(tk.END,"moved cursor to: " + pos + "\n")
```

5.8.6 Pseudocode/Algorithm

```
getVoiceInput():
    r = speech recognizer
    while there is audio to use as input
        audio = formatted input audio
        audioToText = speech to text from audio using r
    return audioToText

phraseMatch(audioInputText):
    closestString = command string best matched to audioToText
    if closest string == “command 1”
        stringP = command1()
    elif closest string == “command 2”
        stringP = command2()
    …
    return stringP
```

```
createNewVariable(...):
        while not the correct name:
            prompt the user for name of variable
            confirm with user if inputted name is correct
            if not, continue()

            check if name is already in use
            if yes, inform user and confirm if user still wants to use the name
            if not, continue()

        while not the correct expression:
            prompt the user for expression of variable assignment
            replace symbols with their symbolic version
            confirm with user if inputted expression is correct
            if not, continue()

        string = variable name + " = " + expression
        return string
```

5.8.7 Code Metrics

Using Lizard code complexity analyzer [16], we are able to assess where we stand in meeting our goals mentioned in 5.6.2.3.5 Maintainability. We currently have all of the use cases implemented out of the original 18 proposed in addition to 5 others added over time. We have a few high numbers in the complexity column for the file voice_recognition.py that will need to be addressed in order to achieve our goal of refactoring and making clean code that is modifiable.

File Type .py | Token Count 5526 | NLOC 507

application.py

Function Name	NLOC	Complexity	Token #	Parameter #
__init__	3	1	34	
attach	2	1	13	
redraw	10	3	96	
__init__	124	4	1926	
showAbout	2	1	16	
openFile	12	2	110	
newFile	4	1	33	
saveFile	17	3	146	
cut	2	1	14	
copy	2	1	14	
paste	2	1	14	
checkNameButton	11	3	91	
trainModelButton	53	4	278	
trainLanguageModel	21	2	376	
getPrevLine	14	2	103	
getNextLine	14	2	110	
playWavFile	8	2	67	
recWavFile	9	1	83	
recordingVoice	40	4	566	
recordVoiceLines	9	2	150	
featureNotImplemented	2	1	16	
changeLanguageModel	27	5	174	
chooseLanguageModel	10	2	205	
text_queue	3	1	42	
image_resizer	6	1	60	
update_text	10	1	92	
listen_for_result	16	4	147	
change_indicator	3	1	41	
on_closing	4	2	29	
onScrollPress	2	1	23	
onScrollRelease	2	1	23	
onPressDelay	2	1	23	
get	2	1	25	
insert	2	1	25	
delete	2	1	25	
index	2	1	25	
redraw	2	1	13	
main	9	4	82	

http://www.lizard.ws/#

File Type .py | Token Count 4898 | NLOC 704

voice_recognition.py

Function Name	NLOC	Complexity	Token #	Parameter #
getVoiceInput	9	2	62	
phraseMatch	127	26	749	
test_compiler	2	1	15	
getClosestString	21	5	130	
text2int	23	9	227	
showSet	5	2	43	
confirm	9	2	72	
createNewVariable	60	17	454	
assignOldVariable	51	16	396	
returnStatement	37	12	288	
createForLoop	20	5	150	
getCondition	35	12	318	
createWhileLoop	3	1	24	
createIfStatement	3	1	24	
createElseIfStatement	3	1	24	
createElseStatement	3	1	13	
createArray	38	9	232	
moveCursor	19	5	151	
selectWord	40	9	296	
selectLine	16	3	126	
selectBlock	24	5	184	
copyText	8	1	77	
pasteText	6	1	57	
cutText	8	1	71	
printStatement	13	3	79	
printVariable	14	3	89	
createDef	14	3	89	
getSymbols	12	2	68	
insertChars	14	3	104	
listen	19	4	106	

File Type .py | Token Count 326 | NLOC 59

compiler.py

Function Name	NLOC	Complexity	Token #	Parameter #
iter_except	6	3	21	
__init__	11	1	129	
reader_thread	7	3	46	
update	8	3	55	
quit	3	1	19	
main	2	1	15	

5.8.8 Voice Training

Currently, the main source we use for accurate results is the Google Speech-to-Text voice recognition system, but since we had the constraint of not being able to use any commercial service for the system to work, we also included CMU Sphinx. Sphinx came with a basic language model that it would use when recognizing speech but that only gave us an average accuracy of about 56.7%. The ones behind CMU Sphinx had a way of creating a more personalized language model, so we incorporated it into the system to allow the user to adapt their own language model for use when using Sphinx as the voice recognizer.

By using the personalized language models, we were able to increase our average voice recognition accuracy up to 84.2%. For now, we have the option for the user to choose between using Google or one of the Sphinx language models to use for voice recognition so that the system can also work when offline.

5.8.9 Testing

5.8.9.1 Technologies/Programming Languages/Libraries

- Pocketsphinx and Google Voice Recognition (for precision, but not required)
- IDE with Python support
- Python 3.8
- Hardware (Laptop or Desktop PC and Microphone)

5.8.9.2 Testing Report

5.8.9.2.1 **Introduction**

This section serves as the plan for testing all software artifacts as well as the reporting of test results. This section 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.

5.8.9.2.2 **Items Tested**

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

5.8.9.2.3 **Intended Audience**

- developers
- testers
- customers

5.8.9.2.3 **Scope**

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

5.8.9.2.4 **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

5.8.9.2.5 **Features to be Tested**
All the requirements outlined in the requirement definition document will be tested as new features.

5.8.9.2.5 **Approach**

Unit test. Developers are responsible for unit testing. The implementation of each module and individual components 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.

5.8.9.2.5 **Acceptance Test Procedure**
(The following is a portion of the Acceptance Test Procedure, please see the Appendix C for the full testing results)

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

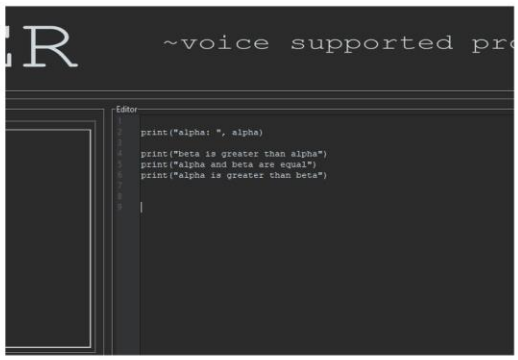
5.8.9.2.6 **Testing Summary**

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 section, we feel more confident in saying that we achieved that goal.

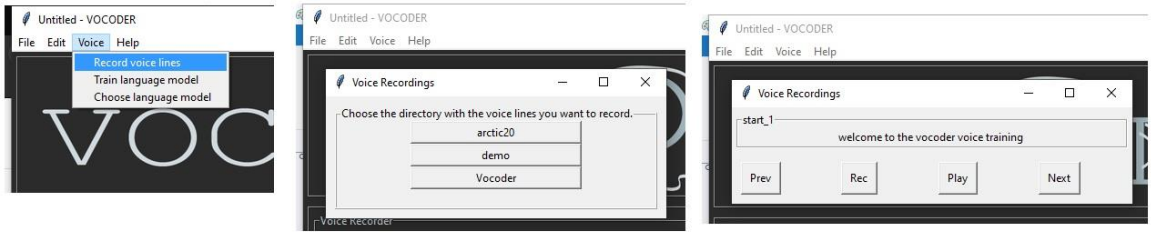
6. Results

User chooses"File" and "Open" in drop down menu

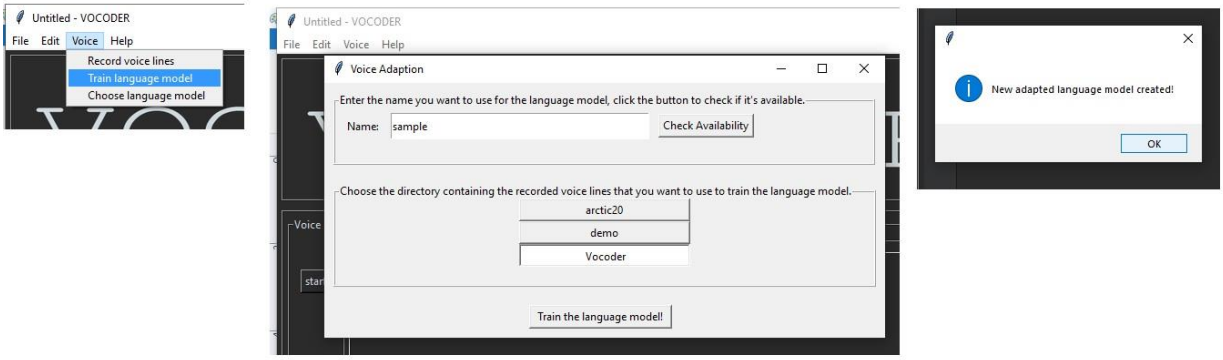
System brings up standard window for user to choose a desired file to open



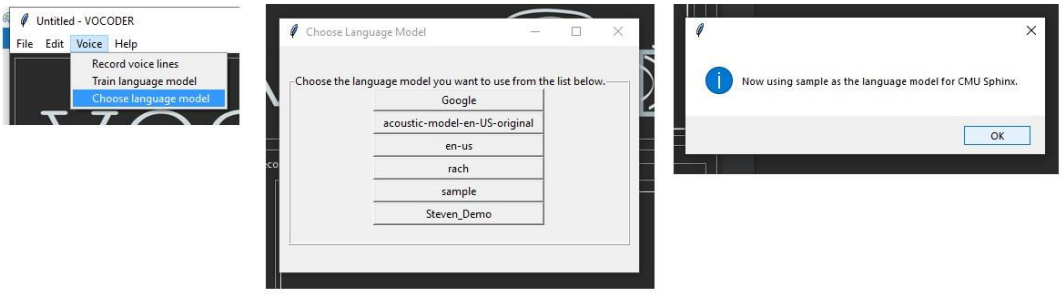
System displays content of selected file in the text editor window and file is ready for editing



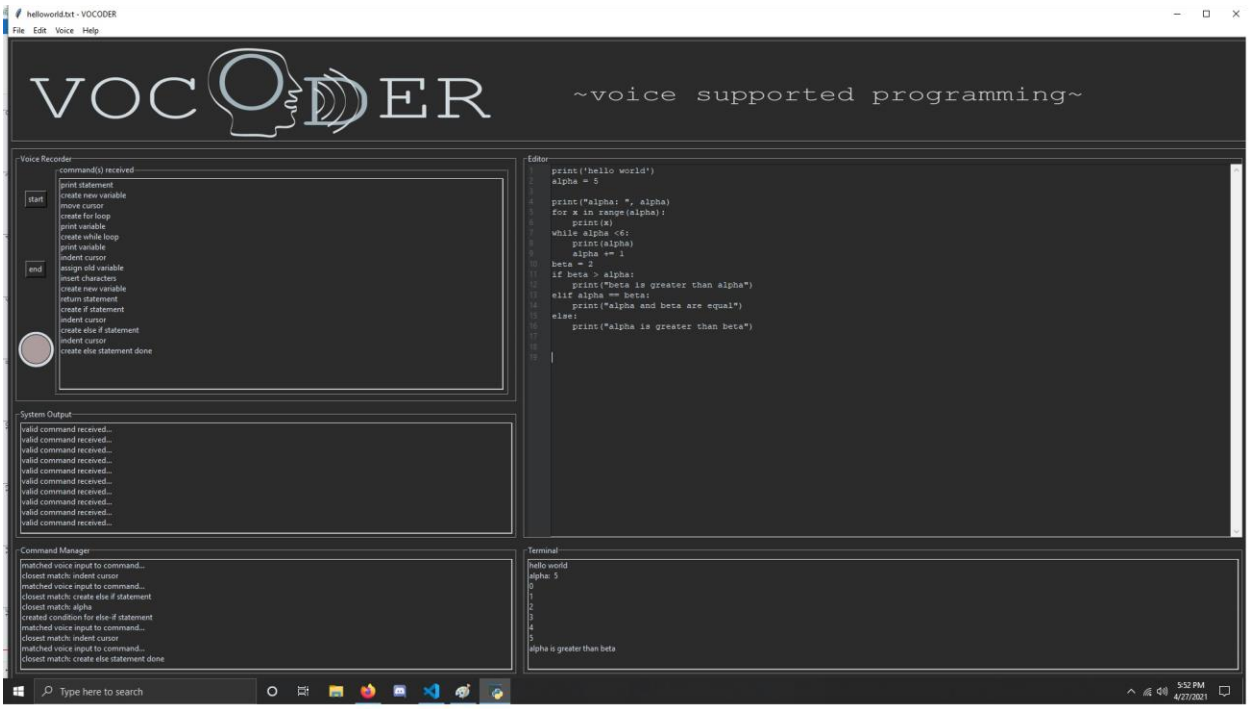
User chooses "Voice" and "Record Voice Lines".The system displays options for lines to be read. Based on lines chose, the user then records themselves saying each line with options of going to the previous line, next line, and playback of current line.



User chooses "Voice" and "Train Language Model". The system displays options for naming the model, choosing the recorded voice lines and a button to begin the training of the new model



User chooses "Voice" and "Choose Language Model". The system displays options for a created model to choose from and notifies the user of which one was chosen.



An example of the GUI after the user has issued several commands. The command(s) received window has been updated after each successful command being called. The system output window and command manager window have been updated after the system executes the commands. The text editor window shows the current state of the Python file being edited by the user and the terminal shows the output after the user clicks the "End" button to see the results of compiling and running the code they are working on.

7. Summary and Future Work

7.1 Summary

With many people deciding to choose working with software as their careers, there's a reasonable chance for them to develop repetitive strain injuries and will need assistance to perform their jobs. There are limited options available to help those in need and even then, the transition to those options can be a slow and painful process. Our goal was to create a user-friendly system that improves on what is currently available. We have achieved several milestones with our project including meeting all of the initial requirements, staying close to the original design specifications, improving the accuracy of the stand alone voice recognition software and completing exhaustive testing. We have also created extra commands and functionality along the way as we became more familiar with the process of coding in Python and thinking about what a typical user would benefit most from. We hope that with this project, Vocoder can be a preferred option to use for those needing help.

7.2 Future Work

- Create a working executable to run the program
- Continue improvements of accuracy of voice recognition output
- Address the possibility of coding in other languages beyond Python
- Expansion of the command capabilities
- Long term maintenance including a User Manual, HTML documentation and bug reports posted through Github

8. References

[1] https://docs.google.com/document/d/1YUF9WVc5fsXha1LMcNvFgt8X6Wb1dKXg23Je33HJ2hs/edit
[2] https://blog.logrocket.com/programming-by-voice-in-2019-3e1855f5add9/
[3] https://www.smashingmagazine.com/2019/05/future-design-voice-prototypes/
[4] http://redstartsystems.com/human-machine-grammar-the-rules
[5] Maloku, Rinor S, & Pllana, Besart Xh. (2016). HyperCode: Voice aided programming. IFAC PapersOnLine, 49(29), 263-268.
[6] Desilets, A. (2001). VoiceGrip: A Tool for Programming-by-Voice. International Journal of Speech Technology, 4(2), 103-116.
[7] Lunuwilage, Kaveendra, Abeysekara, Sameera, Witharama, Lahiru, Mendis, Shamini, & Thelijjagoda, Samantha. (2017). Web based programming tool with speech recognition for visually impaired users. 1-6.
[8] Babich, N. (2019, May 02). Designing For The Future With Voice Prototypes. Retrieved September 19, 2020, from https://www.smashingmagazine.com/2019/05/future-design-voice-prototypes/
[9] Arnold, S. C., Mark, L., & Goldthwaite, J. (2000, January). Programming by Voice, VocalProgramming. Retrieved September 20, 2020, from https://www.researchgate.net/publication/221652444_Programming_by_voice_VocalProgramming
[10] VBP: Support for Voice-Based Programming, SCORE 2021, https://conf.researchr.org/home/icse-2021/score-2021#vbp-support-for-voice-based-programming
[11] IEEE Recommended Practice for Software Requirements Specifications,Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc., https://ieeexplore.ieee.org/Xplore/home.jsp
[12] Foraker Labs (2015). Www.usabilityfirst.com. Retrieved October 06, 2020, from https://www.usabilityfirst.com/glossary/natural-language-interface/index.html
[13] Wayner, P. (2020, July 14). How to choose the right software architecture: The top 5 patterns. Retrieved October 18, 2020, from https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice
[14] Computer Programmers : Occupational Outlook Handbook. (2020, September 01). Retrieved October 23, 2020, from https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm
[15] Tendinitis. (2017, December 14). Retrieved October 23, 2020, from https://www.mayoclinic.org/diseases-conditions/tendinitis/symptoms-causes/syc-20378243
[16] http://www.lizard.ws/#

9. Contribution Table and Teamwork

Team Member	Percentage
Binh An Pham	33%
Steven Tran	33%
M Rachel Van Pelt	34%

Appendix A: Glossary

CTS: Carpal tunnel syndrome

programmer: Computer programmers write and test code that allows computer applications and software programs to function properly. [14]

SDD: Software Design Document

SRS: Software Requirements Specifications

tendonitis: inflammation or irritation of a tendon — the thick fibrous cords that attach muscle to bone. The condition causes pain and tenderness just outside a joint. While tendinitis can occur in any of your tendons, it's most common around your shoulders, elbows, wrists, knees and heels. [15]

user: The person, or people, who operate or interact directly with the product [11]

Natural Language Interface: (NLI) a user interface that allows people to interact using a human language, such as English, as opposed to a computer language, command line interface, or GUI [12]

Menu Driven Interface: (MDI) an interface consisting of a series of screens which are navigated by choosing options from lists, i.e. menus. (“Menu” is not used here to refer to pull down menus, but to lists of options on the screen that lead to other screens.) [12]

Graphical User Interface: (GUI) pronounced “GOOEY”. A user interface that presents information graphically, typically with draggable windows, buttons, and icons, as opposed to a textual user interface, where information is presented on a text-based screen and commands are all typed [12]

Appendix B: keyboard symbols reference for use in Python code

_ : underscore
: hashtag
* : asterisk
( ) : open parenthesis and closed parenthesis
- : hyphen
= = : equal to
! = : not equal to
[ ] : open bracket and closed bracket
{ } : curly brace and closed curly brace
\ / : forward and backslash
| : pipe or vertical bar
< > : open angle bracket
: ; : colon and semi-colon
, . ? : comma, period, question mark
“ “ : quotation mark
‘ : single quote

Appendix C: 5.8.9.2.5 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: Returns the numerical form of that number --- If input string contains a non-valid word: Returns an error	pass
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	a = 1 my_var = 1	popup window guides user for inputs needed, GUI output updated after popup window is closed	pass

			initial value to variable			
AOV	2	assignOldVariable()	say “assign old variable” after clicking the start button. say name of variable, confirm yes/no to confirmation, give new value to variable, verify	a = 2 a = b + 2 c = 3	popup window guides user for inputs needed, GUI output updated after popup window is closed, if no match, create new variable, warn user if using unassigned variables in equation	pass
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	n/a	clipboard data is pasted at cursor location	pass

			having a selection copied			
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 added to the clipboard	pass
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	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	pass

		line, a record button, and play button			user, and allows the user to click on the previous, record, play, and next buttons	
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 "check availability" button	any string input	displays a message box informing the user if it is a pre existing directory	pass
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

Appendix D: Project Management Excel Spreadsheet