Discreet Approaches to AAC: A Teleprompter that assists people with difficulty in public speaking

Individual Project in MSci Computer Science

Author: Zihao You

Supervisor: Dr. Timothy Neate

Student ID: 1907318

April 2022

Abstract

Public speaking can always be quite challenging for a majority of people, especially for those with aphasia, in which their language and communication can be impaired from speaking and understanding all the way to reading and writing. They typically get stuck in specific parts of their presentation and struggle to proceed. This project establishes a tool that prompts the next word after the one users have lost said from their notes, via a mix of speech recognition and text-to-speech techniques. It as well gets evaluated in two separate ways, with Speech and Language Therapists (SLTs) at the early stage of development and usability testing against university students upon completion.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Zihao You

April 2022

Acknowledgements

I would like to say a massive thank you to my supervisor, Dr Timothy Neate, for his endless advice, patience and guidance, as well as my family for the encouragement and unconditional support throughout this project. I will forever remember these moments, which constitutes my best memories for this degree.

Contents

Chapter 1

Introduction

"It's a big concern...is [she] capable of doing it [living on her own]? How would she handle an emergency – those sorts of things are always a worry… The hardest thing to help her is to know when to not do things for her – when to let her do it on her own… I think the best advice you could give would be to remember that they have a problem with communication not with thinking or understanding. Don't treat them like children. And have unending patience and be prepared to do the hardest thing, which is to let them do things for themselves as much as possible[29]."

The above quote has come from an aphasic family member who is living alone[29]. It is not hard to recognise the fact that a majority of families are struggling to cope with the relationships with members with aphasia. Research from the Government of the United Kingdom suggests that the population suffering from communication impairment increased by nearly 70% between 2002 and 2012 (i.e. from 1.3 million all the way up to 2.2 million)[99]. Although no document has been found regarding this data afterwards, it is still highly possible that the trend above will continue, reaching approximately 3.74 million at the end of 2022 by estimation, which constitutes roughly 5.5% of the total UK population.

Hence, a tool to support those groups of people is essential, by helping them restore some of their ability to communicate. In the context of this project, a teleprompter has been proposed for those with difficulty in public speaking, assuming that they do not have any other forms of communication impairments. A pair of earphones connecting to the device is required for the users in order to be able to use this product, in which it successfully picks up the next word from when the presenter starts hesitating and can as well be used in other situations such as job interviews.

The concept of "people with aphasia" and "people with difficulty in public speaking" should be clarified before moving on to the details of this project. Specifically, people with glossophobia have to do with their fear of public speaking, with the likelihood of several physical symptoms involved such as sweating, increased heart rate and nausea, whereas people with aphasia refer to their own physical conditions, with the loss of abilities to produce language due to brain injury[22, 23, 46]. These two concepts will be further investigated and discussed, via interviews with speech and language therapists (SLTs) in Chapter Four. But for now, it is not too counter-intuitive that people with aphasia definitely have trouble speaking in public, whereas those with difficulty in public speaking do not necessarily have aphasia. This essentially implies that an inclusion relation exists between those two terms and the project itself will focus on a wider range of people, which is everyone with difficulty in public speaking.

1.1 Aims and Objectives

"I can not cut into a conversation because I am a slow thinker and I can not type fast enough to contribute to a conversation; that frustrates [me] the most. I don't know how to go back to that subject, so I just don't say anything[34]."

When developing a system, its usability should always be strongly emphasised. A review of the literature showed that quite a few people were struggling to properly use the Alternative and Augmentative Communication (AAC) devices. The quote above belongs to a person who found generating messages using his own AAC system time consuming, which in turn unable to let him fully engage in conversations, ending up with impatience and ignoration from his peers[34].

In fact, the acceptance of the public is as well crucial. It's likely that their family also become annoyed in addition to the AAC device users themselves, as this quote by a father of a 6 years old daughter who uses AAC shows:

"My daughter has been using devices for more than five years now and I am not sure that she can be said to be using it in 'the real world'. Although it is quite portable, she rarely uses it out of the house... she gets little to no encouragement from anyone other than professionals and immediate family[66]."

These quotes yield two aims of this project, named usability and public acceptance. To be specific, the system developed should let a majority of users find useful when they are delivering

their speech in front of an audience, and at the same time get recognised across a wider range of society. As a result, the entire project schedule will be split into two halves corresponding to the development of two separate prototype versions:

- **Initial Prototype:** a skeleton idea of the project consisting of the very basic functionalities will be shown to two SLTs via interviews, for the sake of making sure that the implementation is fallen onto the right track (i.e. the prototype built is indeed a teleprompter).
- **Final Prototype:** after getting some feedback from SLTs, a more sophisticated prototype will be built and tested against usability with randomly selected university students. Public acceptance, in this case, can be predicted via analysis of participants' experiences and feedback.

The primary contributions of this work consist of a combination of the use of speech recognition and text-to-speech techniques which takes the user's sound as input and the system's sound as the generated output. This will be further discussed in one of the later chapters.

1.2 Report Structure

The report is divided into eight chapters, begins by briefly introducing the project aims and objectives as well as comparing the concepts of aphasia and people with difficulty in public speaking. Following this introduction, Chapter Two highlights key relevant background information which involves some additional approaches to help people with aphasia, by providing a review of existing works.

The next four chapters constitute the main part of this project, in which Chapters Three and Four detail the requirements and specifications of the prototype and the project's scope are defined based on the ideation stage, as well as the technical implementation of the initial prototype and suggestions from two SLTs via interviews, followed by doing some optimisation within Chapter Five, carrying out usability testing against several university students plus evaluating the project's strengths and weaknesses which is placed under Chapter Six.

After spending another chapter discussing ethical and professional issues in the context of this project, the report ends with Chapter Eight including a final conclusion and some pinpoint of the areas in which the existing tool may be extended into future research.

Chapter 2

Background

This chapter of the report contains a brief tour of the method used to support people with communication impairments with two sub-categories, followed by three additional terminologies that are worth mentioning under this research context.

2.1 Augmentative and Alternative Communication

Augmentative and Alternative Communication (AAC) is the general term that seeks to investigate and compensate for temporary or permanent impairments, activity limitations, and participation restrictions in individuals with severe disorders of speech-language production and/or comprehension[20], which can be divided into two categories named aided and unaided communication[48].

2.1.1 Unaided AAC Communication

Unaided AAC Communication is adopted in a way that the user does not require any external devices or materials, two typical examples include manual signs and natural gestures (e.g., headshake for ”yes” and ”no”; waving to indicate ”hello” or ”goodbye”)[88].

Although unaided AAC involves no costs at all and it is available to the user at all times, this method of communication is constrained by only a relatively small amount of users which are completely familiar with the signs and gestures[74], plus there are still some words that are unable to be shown as a sign language, this is where aided AAC Communication has come into play.

2.1.2 Aided AAC Communication

Aided AAC Communication requires the user to carry tools external to them, consisting of low-tech and high-tech.

Low-tech devices cover everything that neither includes a battery nor needs to charge. For instance, photo boards and communication cards[83]. Due to the fact that low-tech devices are not "electricity-involved", it is impossible that the systems are able to generate any output to the user either visually via a screen or printer[83].

High-tech devices, however, refer to electronic aids such as computerised speech generating devices and iPad-based communication systems[67]. Some examples are discussed as follows:

Figure 2.1: Speech FlipBook App flipped by sounds[94].

Speech FlipBook (see Figure 2.1) is an app designed by Tactus Therapy for people who have apraxia, articulation or dysarthria. It enables users to customise their own word lists by selecting and combining various components of the word, and at the same time record their speech on screen for the sake of making comparisons to the target pronunciation as well as self-correction[94].

Figure 2.2: TD Snap App generated word sequence[97].

TD Snap (see Figure 2.2) is a flexible communication app developed by TobiiDynavox for people with speech and language impairments such as autism, cerebral palsy or aphasia. Each word has been categorised with an image such that users are able to generate complete sentences every time they click the button, and the output sound can be customised based on a range of available human-speaking voices from different parts of the world[97].

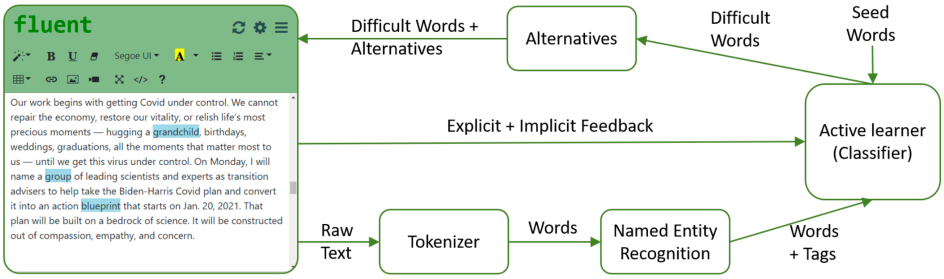

Figure 2.3: Workflow of Fluent[47].

Fluent (see Figure 2.3) is a machine-in-the-loop writing tool that assists people who stutter in producing scripts that they can speak fluently (minimize the number of stuttering events). Given a piece of literature, the tool can assist identify terms that a person may have difficulty pronouncing (trigger words)[47].

Unlike the tool above which explicitly requests user the words that they may have trouble pronouncing and tries to find a general pattern using the Active Learning technique[47], this project comes with a new modality by instead asking the last word of the user's speech before they get stuck, referring back to and searching through the user's note, then the cue word right next to the user's input will be generated. More details regarding design and implementation will be discussed in the next few chapters.

2.2 Abandonment

Once the user has started picking up the tool that supports them in communication, it cannot be denied that they are very unlikely to use just one tool throughout their entire life[54]. That is where the concept of abandonment has arrived, in which its percentage can become a metric when considering the effectiveness of the final product. Assuming that the changes in device type are also being treated as abandonment, a frequently cited study conducted by Philips and Zhao has shown that the abandonment rate can go up to 33% under high tech AAC devices[54, 75]. There are a few more reasons which can potentially cause the discontinuation of

the use of those tools. The designers may not take into full consideration the opinions and desires of those who use the devices[82, 90], or some people simply do not have enough motivation or effective training, or they even have trouble obtaining that specific equipment[75, 90].

For this prompter, it is reasonable to predict that elderly with difficulty in public speaking may have a relatively high abandonment rate of the device compared to youngsters, due to the fact that the listening skills decline with people's age.

2.3 Human-Computer Interaction and User-Centred Design

Human-Computer Interaction (HCI) is the main focus of this project, this section will begin with a definition taken from the Interaction Design book:

"Interaction design is about designing interactive digital products to support the way people communicate and interact in their everyday and working lives[86]."

Driving from that definition, the product should aim for three goals upon delivery, which are easy to learn, effective to use, and at the same time provide an enjoyable user experience[1]. The exact methodologies regarding interaction design will be discussed below:

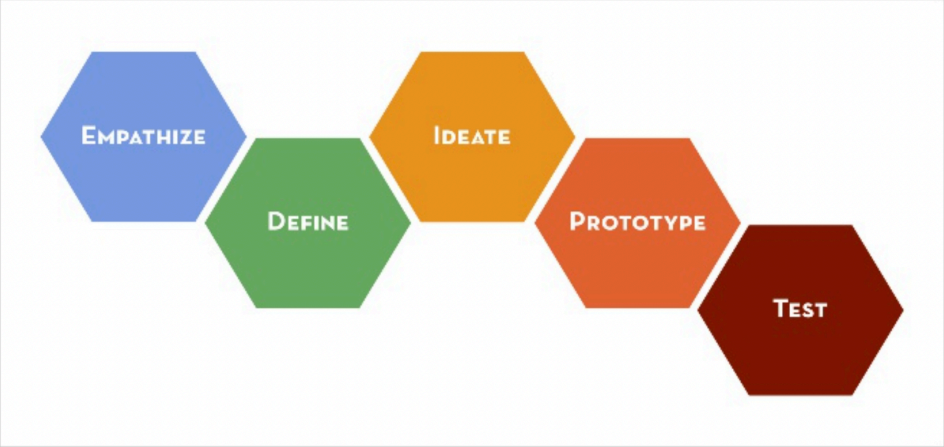

Figure 2.4: Steps in the Design Thinking Process. dschool.stanford.edu/resources[49]

- **Empathize:** observes users' preferences and looks for their needs.
- **Define:** relies on an understanding of people's requirements and preferences in order to gain insights into what their primary problem is that needs to be solved or what opportunity should be explored.

- **Ideate:** produces solution ideas through a process in which judgement is postponed and both quantity and quality of possibilities are encouraged.
- **Prototype:** reduces down the outcomes of ideation to a rough, early solution to a specific problem.
- **Test:** learns what works and what doesn't, tweaks the basic prototype until it is ready to go into production and enterprise versions, and then scales up[49].

Although User-Centred Design (UCD) (see Figure 2.4) is commonly used when designing interactive systems, this project goes in a slightly different order of the design thinking process, by firstly start coming up with ideas based on the review of the existing literature, followed by implementing and developing an initial prototype, which is going to be displayed during interviews with two SLTs in order to not only gain feedback from them but also get a deeper understanding of the user needs. The prototype will then get refined and sophisticated before conducting "usability testing" against university students.

2.4 Usability

In HCI, usability is a key term that describes *"the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment"* as proposed by ISO, 1990[21, 28].

Based on this, every time an interactive system is being designed, there are six usability goals suggested by Nielsen that are worth considering, which are effectiveness, efficiency, safety, utility, learnability and memorability[86].

Although the first two goals in some sense can be considered synonyms, under usability goals' context effectiveness looks more on the product's side by questioning whether it is able to deliver everything in principle, while efficiency focuses on human's productivity by asking people whether they are able to carry out those tasks with the least amounts of energy required. One of the findings shows that some of the barriers to effective use of AAC systems include a lack of availability and accessibility of AAC systems, communication partners' lack of knowledge of AAC systems, an insufficient amount of therapy provided, the type of vocabulary in AAC systems, and other modes of communication available to AAC users[68].

Errors can occur at any point throughout the entire product lifecycle. When a faulty instruction (or instruction sequence or data) is triggered by an appropriate input pattern during execution, the fault becomes active and may cause one or more errors[33]. That's when

the safety goal appears, which entails safeguarding the user from potentially hazardous circumstances and scenarios[86]. Error-handling technique exists within this project as well, by thinking of a range of errors that can happen when the user is using the product, and then the solutions to each of those errors will be built for the sake of minimising the times when the entire program crashes, which will be further discussed under Chapter Three.

Utility refers to how well the product delivers the necessary functionality for people to perform what they need or desire to do[86]. This goal builds up upon effectiveness by not only the system possessing a functionality there that the user can work with, but also allowing people to use that functionality in ways that are actually useful for them.

Looking at the definitions of the last two goals, it is fair to argue that people who find a tool easy to use tend to also quickly pick up the skills they have learned the next time when they use it[86]. In fact, two existing findings have suggested that multiple people are likely to get involved when learning the AAC tools. Ideally, AAC users should be the ones who know the most about their own AAC systems, however many users have not yet attained this level of proficiency. Others including coworkers or supervisors who interact with AAC users must be familiar with the system, particularly when the user is still learning[60, 68]. Those two goals can as well be linked to one of the other concepts that have been covered earlier in this report, named Abandonment. With holidays and missed appointments factored in, an AAC user typically receives 40 hours of therapy per year. In comparison, it is estimated that approximately 200 hours of input are required for someone to learn English as a second language to the level of holding a basic conversation. Learning to use an alternative mode of communication, especially if one has a physical or learning disability, is a far more difficult task, yet there is comparatively little time allotted for it and far too few appropriately trained personnel[68].

In the mid-1980s Ben Shneiderman proposed eight usability heuristics that are frequently used at the design stage of the product, which its revised version in recent years is described as follows:

- Strive for consistency.
- Seek universal usability.
- Offer informative feedback.
- Design dialogs to yield closure.
- Prevent errors.

- Permit easy reversal of actions.
- Keep users in control.
- Reduce short-term memory load[86, 87].

With both Nielsen's six usability goals and Ben Shneiderman's eight usability heuristics coming up together, spotting patterns in between will become intuitive. For instance, those three heuristics "Prevent errors", "Permit easy reversal of actions" and "Keep users in control" can potentially be mapped to one of the six Nielsen's usability goals which are "safety"[2].

To further elaborate, both heuristics analysis and user testing should be employed at various phases of the product design to reap the most advantages, specifically heuristic analysis is applied early in the development process whereas user testing comes later[70, 91]. Although the feedback gained from the heuristic analysis may be utilised to develop a design standard for the remainder of the system, this project uses an alternative approach by instead obtaining feedback from SLTs via interviews, as gathering suggestions from some of the experts within this field helps project improvement better[91]. A standalone chapter will cover these in more detail later in the report.

Chapter 3

Design and Prototype

This chapter of the report begins with the ignition of design ideas via sketching, which elicits the project's requirements for both the user and the system, as well as its specifications and scope in order to successfully run the prototype. After that, a complete walkthrough of the initial prototype that will be shown to SLTs via interviews is provided, including two techniques borrowed from external libraries and the main algorithm adopted within this program, plus two possible situations when people use the system inappropriately.

3.1 Sketch and Ideate

The entire design of the project originated from a digital sketch on a tablet for the sake of more intuitively showcasing the ideas and creativity, consisting of two media entitled "Presenter's pre-made text" and "Presenter's speech", with bidirectional arrows in between correspond to two techniques been addressed within the following two sections.

Looking into every medium, the symbol 'X' as each single word and different text colours are used for readability concerns, with black representing the Presenter's note and blue representing the Presenter's speech. When the presenter gets stuck, the text is shown as red and what the system will do now is to look back to the persons' pre-made note, find the exact place where the text has stopped being uttered and then give some hints to the user by producing a word cue right next to the part where the presenter stops. The process will continue as the user goes along with their speech until the entire presentation has been finished.

The sketch displayed as a screenshot is attached below:

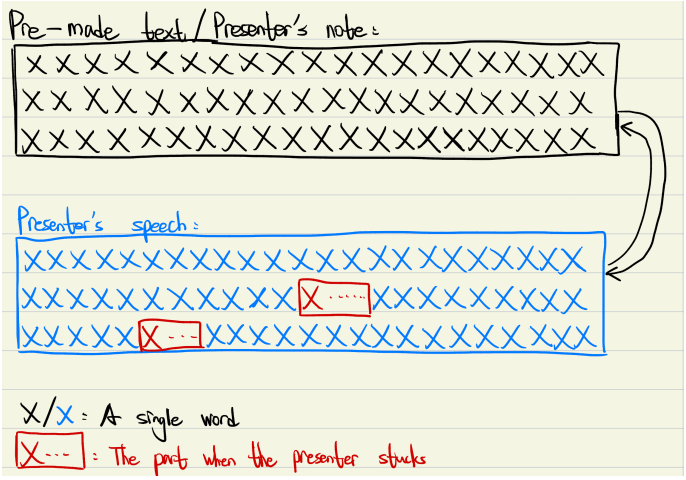

Figure 3.1: Sketch of the initial prototype implementation.

Going back to Chapter Two, a review of the current high-tech AAC devices reveals that none of the three (Speech FlipBook, TD Snap and Fluent) has adopted both speech recognition and text-to-speech techniques in one single execution. More precisely, the second tool TD Snap only uses text-to-speech, and the third tool Fluent detects human speech for model training purposes that enables it to make more accurate predictions. Regarding the first tool Speech FlipBook, although it allows users to record their speech for comparing to the actual word pronunciation this process is however done by human beings rather than the system itself. This is where the main idea of the project has come into play, which refers to a piece of software that not only compares the user's speech with the actual text but also generates sound cue words from the notes made.

3.2 Requirements

Once the brainstorming process has been finalised, it is time to map the ideas coming up from the sketch into a concrete list of descriptions of the services that a high-fidelity prototype should deliver as well as its constraints when operating, which are mainly referred to as user requirements and system requirements[86, 89].

3.2.1 User Requirements

User requirements refer to statements in natural language with diagrams that describe the services that the system is expected to offer to users as well as the constraints that it must function under[89]. In order for the user to smoothly perform all the actions when using the

system, the following preconditions must be enforced:

- A pair of earphones are connected to their device.
- A basic environment with Python 3.9.7 as well as external modules and libraries preinstalled within their machine.
- A stable internet connection.
- A piece of speech note premade and put that within the users_note global variable.
- A microphone turning on for their device.
- Run and terminate the program whenever it is required.
- Press the button on the main window whenever getting stuck on a word.

3.2.2 System Requirements

System requirements are more specific definitions of the functions, services, and operational constraints of a software system, consisting of two sub-categories named Functional and Non-Functional Requirements[89].

3.2.2.1 Functional Requirements

Functional requirements are assertions about the services the system should deliver, how the system should respond to certain inputs, and how the system should behave in specific scenarios[89]. For this project, the system must:

- Pop up the main window when the user presses the run button of the program.
- Allow the user to input (i.e. speak) words every time they get stuck.
- Record the user's input when the button at the bottom of the main window is pressed.
- Output (i.e. print and speak up) the next cue word after the user's input word.
- Remove punctuations and upper case letters for each visual output.
- Delete the previous output when the next output appears.
- Be able to handle four unexpected user events by printing out their corresponding error messages, for the cases when:
  - the user's input word is not found within their note.

  - the user runs out of time for producing their response.
  - the user inputs the last word of their notes.
  - the user inputs the last occurrence of a particular word within their notes.

- Terminate after the program execution (i.e. when the user closes the main window).

3.2.2.2 Non-Functional Requirements

Non-Functional requirements are limitations on the system's services or functions which involve timing constraints, limits on the development process, and constraints imposed by standards[89]. Under this project's context, it contains:

- The system should perform in a way that the user wants them to do.
- The system should wait for exactly 8 seconds when the user is speaking a word.
- The user interface of the system should be easy and unambiguous to use.
- The system should be maintainable such that new features can be easily added and the implementation is simple for any open source contributors to have a clear understanding.
- The prototype should be compatible with most operating systems including Windows, macOS and Linux.
- The entire design and implementation of the product must adhere to the British Computer Society Code of Conduct.

3.3 Specifications

Specifications can be further categorised into software and hardware specifications.

3.3.1 Software Specifications

The prototype itself is implemented in Python 3.9.7, consisting of external libraries for the sake of successfully implementing the mechanisms of speech recognition (i.e. for user's input) and text to speech (i.e. for system's output), which will be discussed in depth within the next few sections.

Additionally, two different Integrated Development Environments (IDEs) have been used across the implementation stage, with Jupyter Notebook mainly for breaking down the code

into small snippets and implementing or testing them separately, these at the end get merged and moved into another IDE named Visual Studio Code for submission purpose, as Jupyter Notebook by default generates .ipynb file rather than standard Python (i.e. .py) file.

3.3.2 Hardware Specifications

Due to an upgrade of the personal laptop, two prototypes were designed and implemented on separate machines, specifically MacBook Pro Quad-Core Intel Core i5 with 4 Cores CPU 16 GB Memory for the initial prototype and MacBook Pro Apple M1 Max Chip with 10 Cores CPU 32 GB Memory for the final prototype, in which the final prototype will be tested against various kinds of operating systems based on the participants selected.

Assuming that none of those users has a listening impairment, what they really need is a pair of earphones compatible with their own device for catching the system's sound output.

3.4 Project Scope

As mentioned within the first chapter, the target users are constrained by the speaking impairment only, in order to maintain a certain level of complexity for the proposed solution.

Although this product can be used in almost all cases when the user is performing a speech, it is not taken into account the situations when the users are singing or holding a concert. Since the concept of tones has kicked in the sensitivity of the speech recognition algorithm will get dramatically changed.

This project will not be tested against NHS patients due to the length of time taken to process ethical approval under the project's time constraint, but instead by reaching out to a few university students and conducting usability testing against them.

3.5 Speech Recognition

Speech recognition, in general, is defined as the transformation of identified words and phrases in spoken language into human-readable text[79]. The SpeechRecognition library is used under this project's context involving a series of widely known public speech recognition Application Programming Interfaces (APIs) such as Google Cloud Speech API, IBM Speech To Text and so on, with Google Speech Recognition being selected in particular for the sake of avoiding the additional use of API key[12, 79].

To properly use this library, the installation followed by import forms the first two steps, namely:

```
pip3 install SpeechRecognition pydub
```

```
import speech_recognition as sr
```

Listing 3.1: Speech Recognition import.

Since the app is designed in a way that users are able to speak up to their own device, reading directly from the microphone seems the most feasible approach in comparison with reading from an audio file, with additional two components named portaudio and pyaudio being installed for macOS, the operating system that is used throughout the entire project development:

```
brew install portaudio
pip3 install pyaudio
```

After those side-works mentioned above, the system now is in a good position of doing the actual speech-to-text conversion. The following lines of code are included within this prototype:

```
r = sr.Recognizer()

with sr.Microphone() as source:
    audio_data = r.record(source, duration = 5)
    print('Recognizing...')
    text = r.recognize_google(audio_data)
```

Listing 3.2: Speech-to-Text conversion.

The speech recognizer is firstly being initialised by creating a speech recognition object, and Microphone() object is also being created in order to read the audio (i.e. user's input) from the device's microphone. A duration variable is then used and set to 5 inside the record() function to stop the recording after 5 seconds. The audio data is eventually get uploaded to Google to get the converted text[79].

In fact, according to the tutorial website, there is an additional print statement at the end regarding the recognised text, which has been omitted when adapted to the prototype as speech detection really is an intermediate step of the whole program[79]. However, it still cannot be fully neglected, especially when comes to the process of code debugging.

3.6 Text-to-Speech

Speech synthesis (i.e. text-to-speech), as the name suggests, does the reverse of what is been mentioned in the previous section. It forms one of the other main parts of the algorithm by converting human-readable text into human-like speech audio[80]. Again, there are lots of APIs and engines that offer this, the one which has been used in this prototype is Google Text-to-Speech.

As before, installing and importing required modules is a strict prerequisite of using this functionality, they are:

```
pip3 install gTTS pyttsx3 playsound
```

```
1 import gtts
2 from playsound import playsound
```

Listing 3.3: Text-to-Speech import.

As one of the user requirements in previous sections suggests that a stable internet connection is a precondition for users to successfully perform all the tasks required, online text-to-speech, in this case, has outweighed offline text-to-speech, in which the offline library pyttsx3 is not essential under this project's context. The Google Text-to-Speech is a Python library abbreviated as gTTS that interfaces with Google Translate's text-to-speech API[80].

After that, the actual text-to-speech conversion will be a 3-step process, below is a specific example extracted from the prototype implementation:

```
1 tts2 = gtts.gTTS('Sorry, try again')
2 tts2.save("error.mp3")
3 playsound("error.mp3")
```

Listing 3.4: Text-to-Speech conversion.

In the first step, a gTTS object is instantiated as an interface to Google Translate's Text-to-Speech API containing the text passed into it. Once this line has been executed, the text has been sent to the API and the actual audio speech has already been retrieved from there. The next two things the program will do is to save this audio to a file that is going to show up in the current directory and play it using the preinstalled module named playsound[80]. The final output in this case, therefore, is "Sorry, try again" as this is what the system has been told to say.

3.7 Language Availabilities

Although English is the only language used throughout the entire project development and testing, it is still worth mentioning the various language availabilities for those two techniques above very briefly.

In speech recognition, an additional language parameter is passed to the recognize_google() function[79]. For instance, to recognise French speech, it can be modified as:

```
text = r.recognize_google(audio_data, language = "fr")
```

Listing 3.5: French speech recognition.

Here, "fr" refers to the language code for French, which as well applies to the text-to-speech scenario by passing the lang parameter to gTTS object[39, 80], which is:

```
tts = gtts.gTTS('Merci', lang = "fr")
tts.save("thank you in French.mp3")
playsound("thank you in French.mp3")
```

Listing 3.6: French text-to-speech.

Additionally, if a dialect has to be specified, such as Canadian French, then the language code can be extended as "fr-ca" for both speech detection and text-to-speech situations[39].

3.8 Core Functionalities

The next two sections will give a complete walkthrough of the initial prototype implementation based on the actual code written.

The entire program involves a single function named prompter(), which is called at the last line of code to allow users to play the run button when they get stuck in a speech. To avoid endless execution, 5 user attempts are set via a for-loop with the main algorithm placed inside:

```
for users_try in range(5):
    # code for the main algorithm...
```

Listing 3.7: Total user attempts.

The user's premade note is assigned to the users_note variable as a multiline string in triple quotes [14] and converted to a list of strings delimited by white space characters using the split() method, which is incredibly useful at the later stage of the process when making word comparisons.

To accept the user's inputs, an input() function is used which explicitly asks users to type some words via a displayed message, this gets stored as a list of strings, being looped and compared with the text in the same data structure, which can be checked in two cases:

- If the word entered by the user can be found within the text, the system returns a cue word right next to the input.
- If the entered word cannot be found inside the text, an error message will show up.

This can be hardcoded as:

```
word_list = input('Forgot what you wanna say next? Type something: ').split()

for word in word_list:
    if word in note_list:
        note_list = note_list[note_list.index(word) + 1:]
        next_word = note_list[0]
        print(next_word)
        # sound output...
    else:
        print('Sorry, try again')
        # sound output...
```

Listing 3.8: Two main situations.

Inside the if-statement, slicing notation is used for grabbing all the words after the user's input word, and from there retrieving only the first word that is assigned to the next_word variable displayed as the system's output, both in text and sound format[31, 51].

Besides speech-to-text and text-to-speech import, another module called string is imported that removes punctuations and upper case letters inside the user's pre-made note for the sake of cleaning up the system's output, which is written as:

```
import string

note_list = [''.join(letter.lower() for letter in word
             if letter not in string.punctuation) for word in note_list]
```

Listing 3.9: Punctuations & Upper-case letters removal.

A widely-used syntax for manipulating list elements called list comprehension is adopted here. It cannot be denied the difficulties of reading this line of code when first seeing it, hence the best way for making explanations will be from inside to outside:

- string.punctuation refers to a pre-initialised string constant containing all possible punctuation marks in the English language.
- The inner part of the code checks whether each character inside a single word appears within the string.punctuation constant and only returns those characters that are not in string.punctuation. They are then joined altogether with an empty string via the enclosed join() method which gives users a complete word without any punctuation signs and upper case letters.
- The outer part runs the inner list comprehension for each word inside the list of notes, in which the results returned will again be stored into the note_list variable before displaying the final output onto the screen[37].

3.9 Error and Exception Handling

When something in the program goes wrong, the execution normally stops immediately and reports an error message at the terminal. However, the users will certainly feel overwhelmed when they have to manually terminate and restart the program every time it crashes, implying that only implementing the system's desired output given the user's input is deemed insufficient. Ideally, the whole process is still supposed to continue even though there is an error occurs somewhere in the program, in other words, the errors will be handled by the Python interpreter that prevents the system from being halted[76].

For this prototype, two cases when the users either do not give any response or input the last word of the text is handled, which can be written in a try...except statement as:

```
try:
    # code which may raise an exception...
except sr.UnknownValueError:
    # print and speak out the error message...
except IndexError:
    # print and speak out the error message...
```

Listing 3.10: Handling two exceptions.

It can be intuitive that all the implementations that do with requesting user's input and producing an output from the system go inside the try block. The execution will touch the except clause only if an error appears, specifically UnknownValueError for no inputs and IndexError for entering the last word of the note, which still allows the program to continue running as normal.

Chapter 4

Interview

This chapter of the report assesses the initial prototype discussed in the previous chapter via interviews with 2 SLTs, with transcripts being analysed and extracted as quotes, followed by two thematic analysis metrics named Codes and Empathy Maps, which are beneficial for having a better understanding of user's needs, drawing out insights from feedback as well as seeking further project improvements.

Notice that the term "coding" defined in Thematic Analysis for this chapter should not be confused with the general means of writing programs in order for the computers to understand, which is being mentioned for the rest of the chapters. As suggested by Virginia Braun, coding refers to "a process of identifying aspects of the data that relate to your research question", categorised and summarised as a word or brief phrase[3, 27].

4.1 Interview Guide

Interviews can be treated as a "conversation with a purpose" consisting of three categories named Structured Interviews, Semi-structured Interviews and Open(guided) Interviews[4, 55]. The final type is the one that has been adopted under this project's context as the answers to the questions that will be asked are mostly non-deterministic, inferring that the data generated from the interviewee will be more complex and give a deeper understanding of the conversation topic[4].

Hence coming up with an interview guide becomes essential for successfully conducting the entire interview, in which it is not fully pre-structured but in fact, building some suggested questions of how the interviewee might be asked, preventing from getting stuck, and at the

same time covering the purpose during the conversation.

When drafting the interview guide, inspiration has been taken from the 'Anatomy of an Interview' part in the Ethnography Fieldguide from the Institute of Design at Stanford[95], with 7 main categories listed below:

- **Intro to Yourself & Into to Project** explain the reason and purpose of carrying out this project/interview.
- **Build Rapport** gets to know more about interviewees themselves and creates a nice atmosphere.
- **Evoke Stories** gets stories and explicit examples from SLTs of how people with trouble speaking in public do things that usually bring most of the evidence, as well as the things needed for project enhancements.
- **Explore Emotions** further drives the design and the need for improvements based on the user's emotions.
- **Follow-up and Question Statements & Thanks and Wrap-up** fill out the little gap by asking more questions that are interesting or worth mentioning.

The trajectory reveals the depth of questions asked as the interview proceeds, starting up slowly for the sake of building the conversation up, creating some relationships and sharing exact context with the person that is being interviewed, then the questions get deeper and deeper when the interviewees get really engaged and emotional, and finally wrap it up nicely so that everyone is happy about the entire process[95].

On a more formal note, the interview is split into two parts, with Part A belongs to requesting the explicit verbal consent of the SLTs to be interviewed. As the interview will be analysed based on the recording and transcription, asking their permission of recording the audio and video cannot be neglected at all.

Below is the final interview guide being used during 2 separate interviews with SLTs, conducted via a Microsoft Teams Video Call that lasted roughly 30 minutes for each:

Table 4.1: Interview Part A - Consent Taking

Category	Question	Remark
Intro to Yourself	Thank you for coming to meet me today.	Create a good atmosphere for the entire interview.
	I am <name>, a current 3rd-year undergraduate student from King's College London doing Computer Science.	Introduce yourself and briefly describe your professional occupation.
Intro to Project	This interview is to help me gain a deeper insight into my goal to develop a teleprompter to help people with difficulties in public speaking. It is one of the tasks for my undergraduate individual project.	Inform the interviewee of the aim of the interview.
Consent	Would you mind me recording the whole interview process?	Ask for the interviewee's permission in a clear and explicit manner.

Table 4.2: Interview Part B - Actual Interview

Category	Question	Remark
Build Rapport	What is your name and your age?	To ensure the interviewee fits the group of interest.
	What made you become a speech and language therapist?	
	What course did you study at university?	
	Is it related to the job you are doing at the moment?	
	How were the first few weeks of you becoming a speech and language therapist?	
	What's the daily life of you as a speech and language therapist?	
Evoke Stories	How often do you meet people who have trouble speaking in public?	Get a rough estimate of the proportion of those groups of people in our daily life.
	Can you reflect on their experience in general compared to people without difficulties in public speaking?	Explore what really makes them different.
	From those kinds of people you have met so far, how their behaviour differs between elderly and youngsters?	Gain a deeper insight into this behaviour by comparing different age groups.
	In addition to having difficulties in public speaking, what other obstacles did they have in their life when you were talking/interacting with them?	Knows what other types of challenges they are facing at this moment.
	Can you think of a specific person you met and when was it?	Let them give a specific example of people with difficulties in public speaking.
Explore Emotions	Walk me through your feeling about meeting those people unable to speak normally in front of a crowd.	Compare and contrast the feelings of people themselves having trouble speaking in public as well as the speech and language therapist.
	How did they feel when they realised that they have difficulties speaking in public?	
	What they would normally do when they got stuck during their speech?	Get to know their reaction to those kinds of occasions.
	How did you feel in this situation?	Reiterate the emotion as a speech and language therapist when they physically witness that situation.
	How did other people feel when you were listening to their speech in a large hall?	Further explore emotions by connecting this to larger amounts of people.
Follow-up and Question Statements	What do you think are the biggest struggles of their life for people with difficulties in public speaking?	Understand their needs by getting familiarised with their major obstacles.
	Do they actively seek help?	Figure out if the tool I am developing is beneficial to them.
	Is it fine if I show you a prototype of the tool I am developing right now which can potentially help those people?	Get permission to show them the project work I have done so far.
	To what extent does my project help those people who have trouble speaking in public?	Let them evaluate my work done.
	What improvements could you make to let my project become more realistic?	Explicitly asking them for feedback.
	In addition to my teleprompter's proposal, can you think of any other tools which can also help those groups of people?	Seek their additional/alternative approaches to assisting those groups of people.
Thanks and Wrap-Up	Much appreciated for such a valuable response. I learnt a lot and it will be really beneficial to my project.	Leave them a good aftertaste of the interview.

4.2 Interview Analysis

The interviews overall yield a wealth of information, with SLTs' responses ranging from a broad overview of people with aphasia/difficulties in public speaking all the way down to some specific pinpoints of where the project can potentially be refined, which brings many important things to attention and in turn boosts a series of thematic analysis discussed in the next few sections.

A complete list of interview data and its analysis can be found inside Miro (password: HCIprj2022.):

`https://miro.com/app/board/uXjVOAp5gCI=/?invite_link_id=939947094099`

4.2.1 Quote Selection

After downloading the transcription, a closer look at interviewees' responses to each question has been carried out. Quotes are then selected based upon the comparison of the same questions answered by both SLTs, with a few of them being selected below and split into 4 clusters, including the motivations behind why they have been chosen.

*"First of all, it's very difficult for them to express themselves so they cannot find the word, they need to know what they want to say, but they cannot find the words to say them. So if you take them to speak publicly, they will find this very difficult to express themselves, and especially if this is very spontaneous, this would be very difficult for them. Maybe it will be easier if you had put them to prepare their speech, but also because they know that they will face difficulty this distressed them more so sometimes this makes them stuck more, I would say, and so it's mostly difficult to express myself to get their words across. It's difficult to find the words they want to say, so it will take a lot of time to find the words."***(SLT1)**

*"So with aphasia, that language and communication can be impaired across speaking and understanding what people say to you and reading and writing as well. And some people struggle with numbers too."***(SLT2)**

The first group of statements highlights some possible challenges people with aphasia/difficulties in public speaking might face, which are chosen as it provides an indication that the project should consider as many difficulties they encounter as possible, including but not limited to helping users find a specific cue word, letting them know its exact pronunciation as well as making it explicit within the premade text.

*"There are people without aphasia who still have problems with public speaking, and I think there are even people with no language disorders who have a problem with public speaking because some people just find it difficult to present and speak in front of the audience."***(SLT1)**

*"Aphasia is caused usually by a brain injury, most commonly a stroke, and so it's an acquired communication impairment that happens to people when they've had some kind of brain injury, and it affects particular areas of the brain which control our ability to form and produce some language, and to understand it, whereas people who have difficulty with public speaking, you know that could be anybody, any member of the public who doesn't like talking in front of an audience, for example, or speaking in the classroom, or something like that. So that's not an impairment, that's just a personality trait."***(SLT2)**

The second group of statements identifies the key difference between people with aphasia and people with difficulties in public speaking. As mentioned briefly within the first chapter, these two terminologies get explained and further clarified by two experts within the field of project research, which as well serves as a verification of the claim drawn from that earlier chapter. Those two quotes are selected as a reinforcement of the project's target user group, not only those who suffer from communication impairments but also those with a fear of speaking publicly.

*"I think the function of prediction is the best plan because it can give them a clue to go on. So I think from those it's very good. The fact that it says the next quarter predicts the next word and also for some people who will be very helpful to write things down as you have there. And so yeah, overall, I think it's a good kind of model."***(SLT1)**

*"I think it would probably be helpful so it finds the next word after the one that you've lost said from a text."***(SLT2)**

The third group of statements briefly generalises what has been done well for the prototype mentioned in the previous chapter, which are selected as a quick check of whether the implementation so far is doing the right thing, and at the same time a huge motivation of the willingness of bringing this project into a next higher level.

"Because people with aphasia specifically, can sometimes have dysarthria, which is something like when you speak when your muscles are weak and it's not very clear what you're saying. I think sometimes for people who have aphasia, it's very important for them to get their message across rather than say the word 100% correctly. So maybe if the system is very

*sensitive, I'm thinking that maybe if they keep saying it wrong, maybe this comes so it should have a kind of flexibility in terms of how it records the word because maybe they will have difficulty saying the word 100%. But this will be clear to the audience. So if the system says, say it again, say it again, say it again, maybe this could confuse them because they will not be able to say that 100% accurately. It's like it doesn't need to be very sensitive in times of because if someone says something with this kind of clear but it's not 100% accurate, and the system keeps asking them to say that right, maybe this can disrupt their speaking."***(SLT1)***

*"I suspect that 5 seconds might be a bit too short to hear the prompt. I think people might get more flustered by hearing their prompt rather than helped, perhaps."***(SLT2)**

*"It might be worth user testing it with people at some point and asking them how they experienced that by asking them so maybe read out a passage of text and then pretend to hesitate and see. See how that happens. See how that goes I think but yeah, I think it could be a kind of a mixed blessing to have that prompt."***(SLT2)**

*"I think user testing is the best way of finding out all these things so doing some kind of collaborative work with somebody who might use something like that in the future and asking them how they feel about using it and giving it a try and telling you which bits they like and which bits that need to tweak. That's what we usually do with some of the tools that we use for assessments and things like that."***(SLT2)**

The last group of statements constitutes one of the most important parts of the entire interview process, which offers clear hints regarding the later stages of the product such as the system's 2nd iteration and testing. The quotes extracted above pinpoint four areas in which the current prototype can be improved:

- The first quote focuses on the system's sensitivity indicating it is not the case that users will utter a word 100% correct all the time, which potentially brings them annoyance if the system fails to detect some of the words that the user is struggling pronouncing accurately.
- The second quote criticises the time given for users to produce a response, which ideally is supposed to be extended.
- Although the last two quotes both refer to user testing, the third quote is oriented toward the general group of users, inferring that the constraints of the project's target

user group can be relaxed by assuming that everyone is likely to encounter a certain degree of difficulties during their public speech, whereas the fourth quote introduces a new technique called codesign that enables designers and end-users to collaborate on design solutions[101]. This involves finding out a specific group of people who might rely on these kinds of tools in their daily life and therefore it is unlikely to be one of the project's next steps, as mentioned within Chapter Three under the "Project Scope" section.

4.2.2 Code Generation

Once the similar pattern of the quotes for each cluster are spotted, it is time to agglomerate them further into a single word or phrase. When looking through two interviewees' responses and making comparisons between them, some discrepancy exists, specifically, there are questions that haven't been answered by SLT1 but this is not the case for SLT2, and vice versa. That's where creating separate codebooks becomes essential. In this case, questions that have responses from both SLTs are moved into one codebook, plus two additional ones that include responses exclusively from either one of the SLTs. For instance, two SLTs both address the potential challenges people with aphasia/difficulty in public speaking might face, how are they looking for help and critically evaluates the current prototype version. However, only the first SLT compares people with aphasia of elderly to youngsters generalised as "Age Group Comparison", while the second SLT specifically comes up with her own way of dealing with people who struggle delivering their speech publicly coded as "Solution".

Encapsulating all those quotes into phrases provides a much deeper understanding of the interview topic itself and what potentially the project can be brought further, a screenshot of the codebook exclusively for SLT2 and an extract of the coding examples are displayed below:

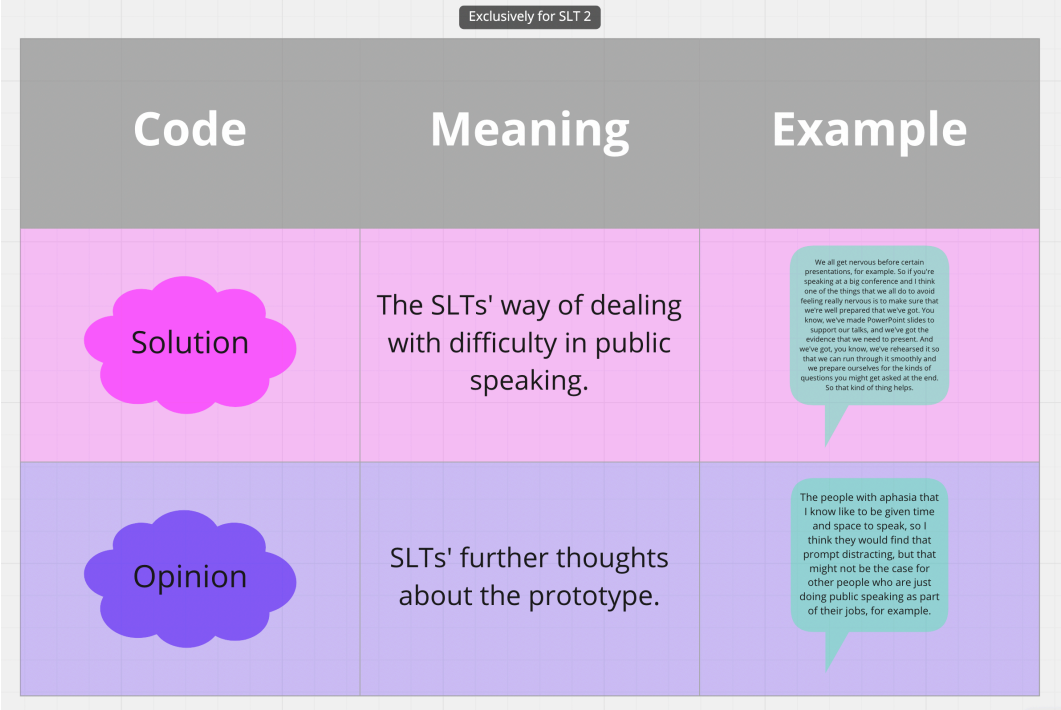

Figure 4.1: Code Book exclusively for SLT2.

Figure 4.2: Codings extract for SLT2.

4.2.3 Empathy Maps

Here comes another important phase of thematic analysis which helps pull out insights by start synthesising the quotes and codes generated from the previous sections[25]. Again, the general structure of the empathy map has been adapted from the Design Project Guide in the Institute of Design at Stanford[25], with 4 main quadrants outlined below:

- **SAYS** contains quotes that exactly come out of the SLTs' mouths, in particular describing the struggles people with aphasia/difficulty in public speaking might encounter and some online assisting tools.
- **DOES** contains quotes of SLTs' actions and behaviours such as the jobs they normally do on a daily basis (i.e. meeting people with aphasia/difficulty in public speaking).
- **THINKS** contains quotes of SLTs' thoughts, beliefs, or opinions that are more related to these interviews' context, specifically giving feedback to the state of the art of system as well as making a clear differentiation between the groups of people with aphasia and those who have trouble speaking in public.
- **FEELS** contains quotes of SLTs' emotions. In other words, their own feeling of interacting with people with aphasia/difficulty in public speaking, especially at the beginning of their professional career[25].

As a result, an empathy map involving all quotes and codes for each SLT is produced, in which a broader pattern of each quote has been discovered, spreading diagonally with one quadrant (i.e. SAYS) mainly addresses a wide range of challenges users face while its opposite (i.e. THINKS) holds the detailed project's feedback. An extract of the empathy map for SLT2 is shown below:

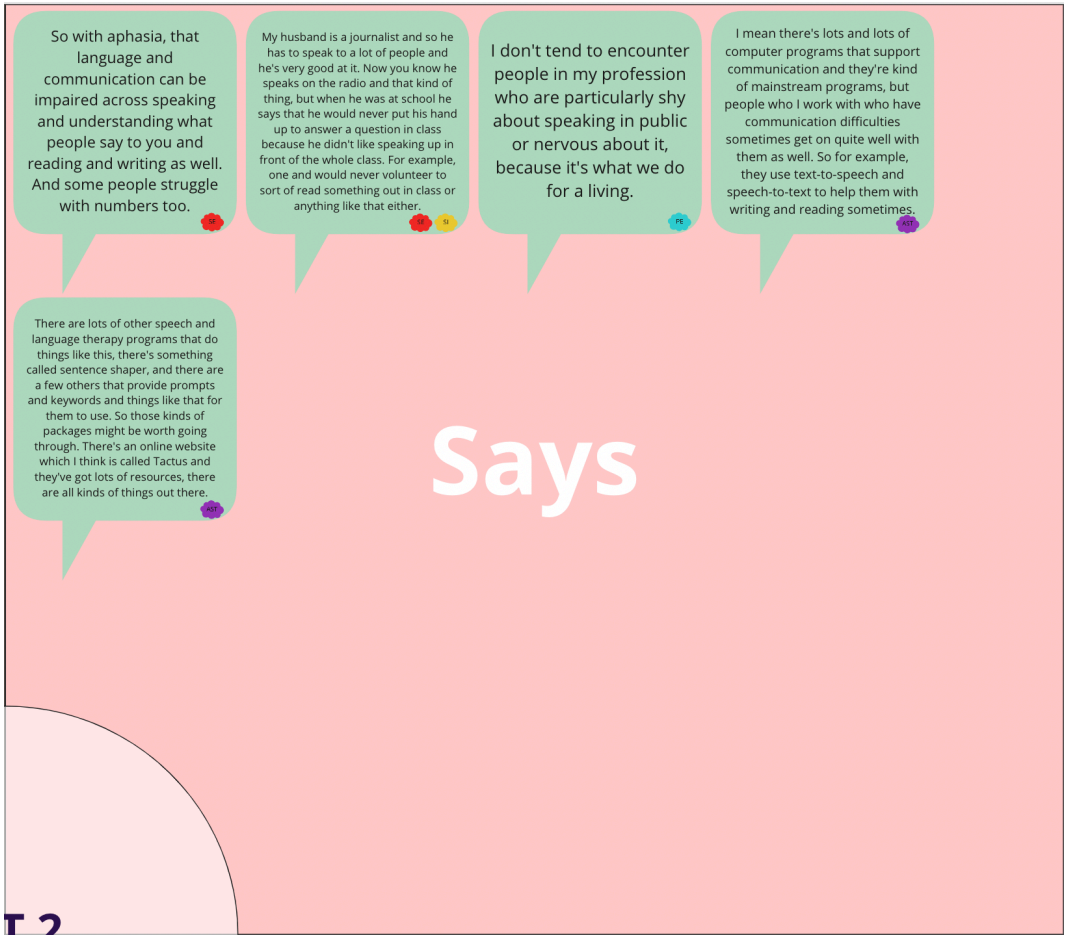

Figure 4.3: Empathy Map extract for SLT2.

Chapter 5

System's 2nd Iteration

This chapter of the report draws from the interview and feedback given by 2 SLTs in the previous chapter, with another prototype delivered be the final, more sophisticated version of the project that builds up upon the initial idea but potentially some further extensions have been added.

5.1 Initial Prototype Review

To simulate how it will look like when it becomes a real application, an executable file has been created via Pyinstaller[96], specifically in the following two main steps:

```
pip install pyinstaller
pyinstaller -onefile Initial_Prototype.py
```

The pyinstaller package has firstly been installed. After navigating to the directory where it contains that Python script file, the actual executable file will be created when running the second command[96]. In this way, installing extra libraries mentioned in Chapter Three will no longer be necessary for users. A short demo of the initial prototype's output discussed within Chapter Three is displayed as a screenshot below:

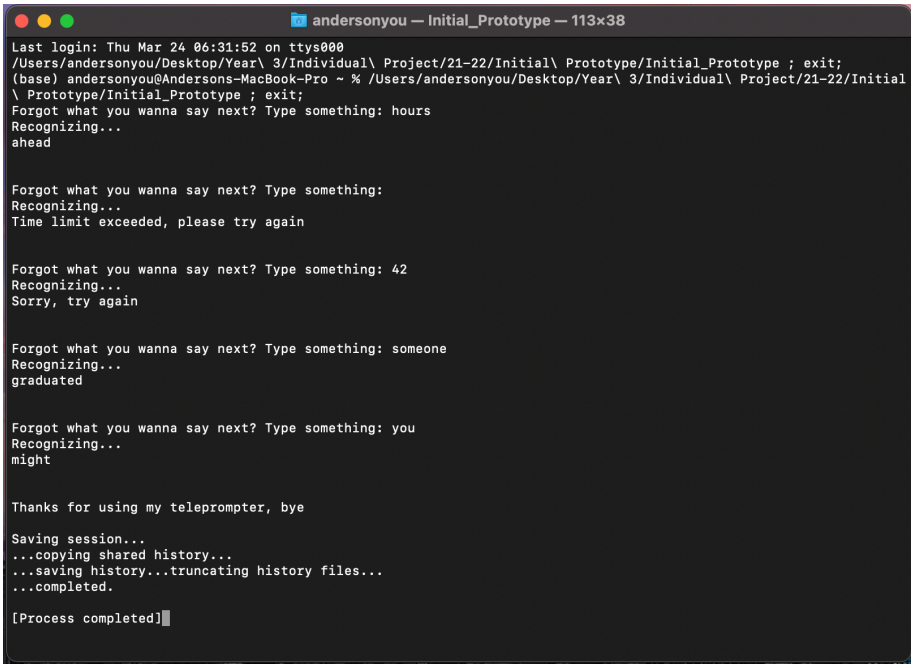

Figure 5.1: A short demo of the initial prototype.

It is impossible not to criticise its readability in the first instance. People without any technical background will find it particularly challenging when being asked to read texts straight from the terminal, forcing them to hold the hand-written text by hand or put them in some other places on the laptop while using this software. Hence, placing the user's note and the system's output together in a new window can potentially be one of the next steps of this project.

Additionally, allowing users to both type and speak to the system when requesting cue words can sometimes be confusing. The insight has been drawn from observing the interview quotes of one of the SLTs *("That could be useful because some people are really helped by typing or by words, so I think this can be useful for some people. Some other people are not good at typing, so they will find it difficult to type.")*, which intends to simplify the entire process by removing the "typing" mechanism when they are interacting with the system.

5.2 Frontend Addition: Graphical User Interface

Graphical User Interface(GUI) refers to the first thing that comes into the user's view after they open an application, which makes the overall flow more smoothly and improves the user experience[30, 53]. A commonly-used Python GUI Framework named Tkinter has been adopted for this project after comparing and contrasting it to others.

5.2.1 Introduction to Tkinter and Root Widget

To have a proper start with using Tkinter, importing everything (i.e. all classes and functions that interact with the Tk toolkit) from that module will be essential[63], which can be done by:

```
from tkinter import *
```

Listing 5.1: Tkinter import.

As everything displayed within Tkinter is a widget, so does the remainder of this section. The main window that holds all the user's text and system's output has been built via the following few lines of instructions:

```
root = Tk()
root.title("PROMPT ME OUT!")
root.iconbitmap("/Users/andersonyou/Desktop/Year 3/Individual Project/21-22/Final
    Prototype/teleprompter.ico")
root.geometry("800x600")

root.mainloop()
```

Listing 5.2: Tkinter root widget.

The widget above must be created before any other widget appears, named root widget[63], with its title "PROMPT ME OUT!" and default size 800x600 (i.e. the size displayed when the user runs the program for the first time). The last line corresponds to an event loop that enables the program to stay there until the window is closed by the user[63].

The third line of code's feasibility, however, is something that is still debatable in recent days. To be specific, a teleprompter icon has been created and shown on the left of the program's title when the user opens the application. However, depending on what operating system people are using, for example, in macOS the icon fails to display correctly within the title bar as illustrated below, whereas this is not the case for Windows users[52].

Figure 5.2: Icon of the final prototype.

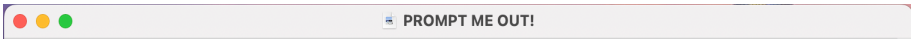

Figure 5.3: Title bar of the final prototype.

5.2.2 Full-screen Scrollbar

Normally, a majority of the user's notes being used as speech tend to be relatively long and will not fit onto a single screen of a regular-sized laptop, which is where the importance of the scrollbar begins to shine. However, in this case, the scrollbar is required to run the whole length of the Tkinter app instead of using it just for listboxes.

The implementation below in fact refers to a general method of adding a full-screen scrollbar to the main window in Tkinter that can as well be adapted and used in many other applications, the credit here will be given to John Elder who has made and delivered this insightful video on Youtube[41]. To begin with, an additional ttk widget has been imported, which is the widget the scrollbar belongs to:

```
1  from tkinter import ttk
2
3  main_frame = Frame(root)
4  main_frame.pack(fill = BOTH, expand = 1)
5
6  my_canvas = Canvas(main_frame)
7  my_canvas.pack(side = LEFT, fill = BOTH, expand = 1)
8
9  my_scrollbar = ttk.Scrollbar(main_frame, orient = VERTICAL, command = my_canvas.yview
       )
10 my_scrollbar.pack(side = RIGHT, fill = Y)
11
12 my_canvas.configure(yscrollcommand = my_scrollbar.set)
13 my_canvas.bind('<Configure>', lambda e: my_canvas.configure(scrollregion = my_canvas.
       bbox("all")))
14
```

```
second_frame = Frame(my_canvas)

my_canvas.create_window((0,0), window = second_frame, anchor = "nw")
```

Listing 5.3: Tkinter full-screen scrollbar.

It may be too scary to read at the beginning, a breakdown into a 6-step process is therefore being applied:

- **Step one** creates a main frame that holds the root widget (i.e. fills the entire size of the application).
- **Step two** creates a canvas that is placed within the main frame, starting from the left-hand side and expanding out to fill the whole frame.
- **Step three** creates a scrollbar that is added to the right side of the main frame with vertical orientation, in which its scrolling action will instead be placed within the canvas.
- **Step four** configures the canvas for the sake of customising the scroll action (i.e. passing in the area that the user wants to scroll via a lambda function).
- **Step five** creates another frame named second_frame inside the canvas.
- **Step six** creates a window positioned top right (i.e. (0,0) coordinate and "nw" anchor) within the canvas and place the frame created in the previous step into the window[41].

Now the rest of the widgets will all be situated inside second_frame in order for the text and the output to be scrollable[41], which is what the next section is going to discuss.

5.2.3 Other widgets and Font

Obviously, there are two main types of widgets that haven't yet been covered, one for the Label widget that is used as the user's note, system's output and empty line space, the other refers to the Button widget.

As far as coherence's concern, the code attached below does not appear in the exact same order as the actual program but instead adheres to the order displayed inside the Tkinter window from top to bottom (i.e. the user's note is on the top, followed by a button and a line space, with its output being placed at the bottom), as the screenshot is below shown:

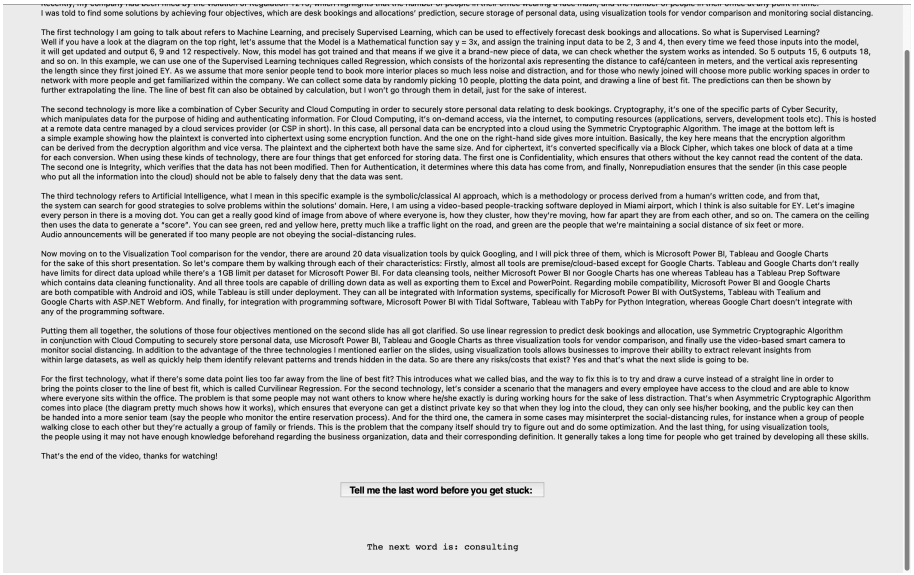

Figure 5.4: A short demo of the final prototype.

In this prototype, a substantial longer text has been selected as the user's note, which is roughly equivalent to a 10 minutes speech, again been assigned to the users_note variable. To successfully get displayed inside a Tkinter window, the following instructions will be passed to the terminal:

```
global note_label

note_label = Label(second_frame, text = users_note, justify = "left")
note_label.grid(row = 0, column = 0, pady = 15)
```

Listing 5.4: Tkinter note label widget.

In order to strive for a high degree of cohesion (i.e. each class and method ideally does one single task) and reduce the overall function's length[18], the users_note and its label widget have been moved outside the prompter() function, which is defined as a global variable. A 2-step process is then been applied, creating that widget with left alignment[85], followed by placing it onto the window (i.e. the second_frame as mentioned before) with some vertical padding from the border.

When constructing button and output widgets, various font types (i.e. Helvetica for the button text and Courier for the output text) have been selected for the sake of making the key information more explicit to the users[36, 98]. Thus an extra tkinter.font module will be imported before heading to the creation process:

```
import tkinter.font as font

```

```
buttonFont = font.Font(family = 'Helvetica', size = 16, weight = 'bold')
btn = Button(second_frame, text = "Tell me the last word before you stuck: ", font =
    buttonFont, command = main)
btn.grid(row = 1, column = 0)
```

Listing 5.5: Tkinter button widget.

The first line of code creates an object of type Font from tkinter.font.Font() class that customises the font type of button widget[84, 98]. This then gets passed into the button widget and assigned to the font variable. The rest of the implementation again follows the general 2-step process as mentioned previously when making the label widget, except for the fact that the variable command has been assigned to a function called main(), which is a helper function that runs the function prompter() and will be covered in more details later in this chapter.

An additional line space has been put between the button and the output text as a one-liner:

```
Label(second_frame, text = "\n").grid(row = 2, column = 0)
```

Listing 5.6: Tkinter empty line space label widget.

Now move on to the final label widget that displays the output when the user requests a cue word. The aim here is to set all the outputs in one single line via updating the results every time the button is pressed, which can be achieved by:

```
output_label = Label(second_frame)

    global output_label

    output_label.destroy()
    output_label = Label(second_frame, text = "Sorry, try again", font = ("Courier",
    15))
    output_label.grid(row = 3, column = 0, pady = 30)
```

Listing 5.7: Tkinter output label widget.

The appearance of indentation and scope undoubtedly adds more trickiness to this implementation, indicating that the variable output_label must be defined and created globally outside the prompter() function before it gets destroyed[42], which causes the system to delete the previous output when a new word appears. It then creates a new label and places it onto the screen exactly as before.

5.2.4 Positioning Contents

One last note of this project's frontend refers to its relative position. As Figure 5.4 depicts that all the contents stay roughly at the centre of the Tkinter window, this can be configured as:

```
second_frame.columnconfigure(0, weight = 1)
second_frame.rowconfigure(0, weight = 1)
```

Listing 5.8: Tkinter centering contents.

The second argument weight = 1 causes the row and column to expand and fill the entire window equally[43, 72].

5.3 Backend Modification

Now it is time to have a closer look at what is actually doing behind the scene (i.e. how the system will react every time the user presses the button), which is what this section mainly talks about.

Recalling the aim of "remove the typing mechanism" for this prototype, which causes another problem after this action has been taken. In the previous prototype, the users are forced to input 5 times no matter how many times they get stuck on their notes in order for them to reach the end of the interaction loop. This seems less awkward when they are still allowed to type the input. Once this has been disabled, however, the users have to wait for a definite period of time until the entire program terminates even when they do not need to use the prompter at this moment, with hearing the "exceeding time limit" warning in the meantime if they keep staying silent.

As the software is designed in a way that the entire program gets run once for every button press, this ideally should produce only one output at a time rather than 5 outputs in one go. However, considering now the case that the main for-loop within the initial prototype has been removed, the next cue word gets successfully retrieved but the system always utters the word after the first occurrence if the user's input word appears multiple times inside the text, which is again slightly troublesome.

Besides output failure, there are some dead codes in which the declared variables never going to be used in the entire program[13], such as the users_try inside the for-loop header and the text when doing the speech-to-text conversion. In this case, both variables are being reported as not accessed and dimmed out in Visual Studio Code IDE, which is indeed a bad practice in writing programs as it may not exactly change and update with the project designs,

even if the program still runs[65].

Here's where tweaking the algorithm becomes necessary, with the aim of "3 ones" (i.e. "one button press, one user's input, one system's generated result") and at the same time keeping track of the occurrences of a specific word if it appears more than once within the user's note.

5.3.1 "defaultdict" and Main Algorithm

In addition to the list of strings used in the initial prototype that manipulates the user's note, another data structure called defaultdict has been used specifically for handling the occurrences of the same word in the text, which belongs to a subclass of the dictionary class, with almost the same properties except for the fact that defaultdict returns back a default value when the key does not exist rather than raise a KeyError exception[15]. This can be found and used from the collections module by:

```
from collections import defaultdict
```

Listing 5.9: Defaultdict import.

There are two functions in total for this prototype. The first function prompter() holds the core functionalities and the second function main() acts as a helper that runs the first function and handles an exception when the users run out of time for producing the responses as discussed in Chapter Three. The default value of the defaultdict data type has been set within the first function's parameter as:

```
def prompter(current_index = defaultdict(int)):
    # code for the main algorithm...

def main():
    try:
        prompter()
    except sr.UnknownValueError:
        # speak out and display the error message onto the Tkinter window...
        return
```

Listing 5.10: Two functions of the final prototype.

In this case, the int class is passed as a default_factory argument with 0 as its default value[15, 57], meaning that this value assigns to the current_index variable only once when the function prompter() is defined, rather than each time the function is called[56]. Therefore, its key represents the user's input word and the value corresponds to the next word of the previous user's input.

Now move on to the main body of the algorithm inside the prompter() function, this time only the changes made since the previous prototype version will be discussed:

```
# code for string conversion, punctuation/upper case letters' removal and recording user's input...

try:
    next_word_index = note_list.index(word, current_index[word]) + 1
except ValueError:
    if word in note_list:
        last_occurrence = f'There are no more occurrences of {word} in the text, please try again'
        # speak out and display the above error message onto the Tkinter window...
    else:
        # speak out and display the error message (i.e. "Sorry, try again") onto the Tkinter window...
    return

current_index[word] = next_word_index

if next_word_index == len(note_list):
    # speak out and display the error message (i.e. "This is the last word of the text, please try again") onto the Tkinter window...
    return

next_word = note_list[next_word_index]
# speak out and display the next word generated above onto the Tkinter window...
```

Listing 5.11: Main algorithm of the final prototype.

It is not counterintuitive to spot that the occurrence check is all done within the try…except statement. Specifically, the note_list.index() method grabs the next first occurrence of the input word's index starting from the next word of the previous user's input defined as current_index[word], this then gets incremented as the user goes along their notes consisting of two possible outcomes both classified as ValueError[78]:

- If the user reaches the last occurrence of a specific word in the text, an error message will show up that reminds them there are no more occurrences of this word for the rest of the passage.
- If the word user uttered cannot be found within the text, the system will ask them to try again as before.

The index of the next word/next first occurrence of the input word will therefore get stored into the current word's index if none of those two exceptions above is triggered.

Instead of handling the case of the user reaching the end of the text as an exception, this has been modified as a conditional-statement check by comparing the next word's index with the length of the user's note as a list of strings, the system will again report an error message if they are equal.

The program will at the end generate the next word/next first occurrence of the input word by accessing the index of the note_list variable.

5.3.2 User's Response Time

The final tweak of this prototype draws from a specific point one of the SLTs has mentioned during the interview *("I suspect that 5 seconds might be a bit too short to hear the prompt. I think people might get more flustered by hearing their prompt rather than helped, perhaps.")*, by tunning the duration variable's value inside the record() function as:

```
audio_data = r.record(source, duration = 8)
```

Listing 5.12: User's response time of the final prototype.

Since people with non-fluent aphasia or severe mixed aphasia tend to struggle to speak at a normal pace[93], the modification greatly benefits this group of users in particular by allowing them more time to react to the system.

Chapter 6

Testing and Evaluation

This chapter of the report verifies the project requirements outlined in Chapter Three by undergoing a usability study with the final prototype version, followed by a brief introduction of the concept of software maintenance as well as how this can be applied to the project's context.

6.1 Study Protocol

As usability enhancement forms one of the two aims of this project described within Chapter One, so does the aim of this user study[38], via inviting people to a constrained environment in order to get an in-depth understanding of how they will be using the prototype developed[5]. Although a horizontal prototyping approach (i.e. most of the features and functionalities are partially built) has been carried out throughout the entire implementation phase[6, 86], it is sufficient to provide users with a broader picture of what the product may look like once this has been deployed to the public.

The sections below give a complete tour of the usability testing divided by several key components[5], with the "Think Aloud" technique being adopted throughout for the sake of better capturing what is going on in a user's head[86], which potentially drives the project's evaluation and possible future improvements. More details regarding participants' responses and analysis of the study can be found within the Miro board.

6.1.1 Testing Objectives

As Nielsen Norman Group suggests testing objectives are defined in a way that allows user behavioural data to be gathered for conducting qualitative and quantitative analysis purposes at a later stage[62], the following list is constructed which showcases five main aspects that the system will be tested against:

- Users are able to understand how the button works, specifically when exactly they need to press it.
- Users are able to spot the relationship between the system's output and their input words.
- Users are able to realise the consequence of producing various anomalous inputs, such as not saying anything, uttering a word that cannot be found inside the text and so on.
- Users are able to easily locate where exactly they are inside the text when they get stuck.
- Users overall find the application useful when they get stuck during their speech.

6.1.2 Participants, Testing Environment and Methodology

Participants in a total of four at this stage are recruited in the most accessible manner possible, with everyone being a current student at King's College London and having connections with all of them beforehand, which provides a quicker response when reaching out to them via email, WhatsApp and other social media platforms, attaching the project's brief introduction as well as the consent of audio and video recording upon a message.

The study is again taking place in Microsoft Teams, for the purpose of using the record and transcription to evaluate each participant's performance with a range of testing metrics. They will be given a short tutorial at the beginning that involves the objectives of this usability testing and the main features of this application. Since Pyinstaller only creates dependencies rather than fully standalone executables[100], it may bring unnecessary trouble to both participants and developers by guiding them to undergo multiple additional installations on their machine, which is not quite feasible given the project's time constraint. Instead, participants will be given control of the screen where the software has been developed and perform tasks there straight away, with a maximum completion time of 30 minutes being set in order not to defeat the purpose of improving the project's overall usability.

To reiterate the "Think Aloud" technique being selected for this study, participants being prompted to keep taking amplifies their reactions and thoughts[5], which can be used as a gauge

for their understanding and acceptance of this prototype.

6.1.3 Testing Guide

With all those testing groundworks being set up, participants will be asked to carry out the following steps in a controlled experiment fashion:

- Wear a pair of earphones and connect them to your device.
- Open the Unix executable file named "Final_Prototype".
- Respond to the questions:
  - What do you think the main text is about?
  - Have you noticed the scrollbar on the right side of the main window?
  - Do you recognise that the thing at the bottom is a clickable button?
- Assume that you are the presenter at this moment, pick a specific paragraph and read them out loud.
- Now pretend that you are getting stuck, press the button and speak out the last word before you stuck.
- Respond to the questions:
  - The system's output is not displayed on the screen when the button is pressed for the first time, would you be able to figure this out?
  - What is the relationship between the system's output and the word you just entered?
- Repeat the word that you have uttered last time.
- Respond to the question:
  - How does the system's output this time compare to the previous time with the same word being detected?
- Respond to the follow-up questions:
  - If the word you have entered cannot be found within the entire text, what do you think the system's output should be this time?
  - Would you be able to verify by trying this yourself?

  - Does this match your guess before?
  - Are you happy about the way the system responds to?
  - What if you don't say anything at all when you press the button?
  - And what about you reach the last word/occurrence of the text?
  - Would you be able to attempt those situations this time by yourself?

- Close the application.
- Respond to the final two questions:
  - Based on your overall experience, to what extent do you think the system can help when you get stuck in a real-world situation?
  - Are there any improvements that you can think of in order to let this project become more realistic?

6.1.4 Testing Metrics

Based on the tasks and questions participants need to react to in the previous section, it deduces that the data gathered from the study can be categorised as quantitative and qualitative[5], with the former interacts with numbers or anything that can be readily turned into numbers, whereas the latter focuses more on real-time data, such as participants' quotes, feelings and the description of tasks that they have undertaken in this case[86].

Below is a list of examples of data that can be collected by their specific category:

- Quantitative:
  - The time spent for participants to spot the exact type of main text.
  - The time spent for participants to recognise the scrollbar on the right end.
  - The time spent for participants to recognise the button at the bottom.
  - The time spent for participants to fix the "not displaying output" problem when they press the button the first time.
  - The number of attempts participants have been made to get the expected cue word when they get stuck.
  - The time spent for participants to spot the occurrence check being done by the system.

  - The time spent for participants to come up with what the system is likely to respond to for the case of words does not exist inside the text.
  - The number of attempts participants have been made to get the expected warning message for entering words that cannot be found within the text.
  - The time spent for participants to come up with what the system is likely to respond to for the case of staying silent when pressing the button.
  - The time spent for participants to come up with what the system is likely to respond to for the case of reaching the last word/occurrence of the text.
  - The number of attempts participants have been made to get the expected warning message for no input.
  - The number of attempts participants have been made to get the expected warning message for reaching the end of the text/last occurrence of a specific word.
  - The number of tasks participants have completed.

- Qualitative:
  - How do the participants interpret the main text?
  - What the participants will do when they do not see the output for the first time they press the button?
  - What the participants will do when they do not get the desired output from the system?
  - How do the participants feel when their verification matches their guess?
  - What are the participants' thoughts with regards to their overall experience of using this prototype?
  - What suggestions participants will make for fine-tuning the product?

6.2 Study Analysis

After collecting data from all participants, it is time to look back at all the system requirements listed in Chapter Three one more time but from a slightly different angle, specifically verifying each one of them via checking its validity. It cannot be denied the appearance of bias due to the lack of users involved in this usability test, but still, the essence of stepping through all the requirements and undergoing evaluations is not supposed to be neglected. Below is where a full

list of requirements' verification categorised as functional and non-functional has been placed. An alternative analysis approach that focuses more on each question being asked during the user study, as well as its testing metrics can be found within the Miro board.

6.2.1 Functional Requirements Verification

This section mainly checks against the system's expected behaviour under a series of user inputs, detailed as follows:

- **ID:** 1

  Requirement: Pop up the main window when the user presses the run button of the program.

  Validity: Although all 4 participants were being asked to run the software within a single executable file, this is still a valid requirement as "opening the executable file" essentially is an equivalent action to "pressing the run button of the actual program", the window has all successfully being showed up a few seconds after the executable file has been opened by the participants.

- **ID:** 2

  Requirement: Allow the user to input (i.e. speak) words every time they get stuck.

  Validity: Valid as none of the 4 participants has experienced any system crash immediately after the button was pressed.

- **ID:** 3

  Requirement: Record the user's input when the button at the bottom of the main window is pressed.

  Validity: Valid as the record action will certainly process if no program crash has been reported right after the button press, as verified in the previous requirement.

- **ID:** 4

  Requirement: Output (i.e. print and speak up) the next cue word after the user's input word.

  Validity: The four usability tests have in fact conducted in various locations, with the first two being held in the college library whereas the last two inside the student accommodation for the sake of inducing the network speed as an additional constraint throughout the entire testing process, which results in a restart of the laptop due to the system being stuck at generating outputs for an infinite amount of time after the button

press (depicted as a screenshot below). However, this is still a valid requirement as the system still eventually produced the output except for taking longer than usual.

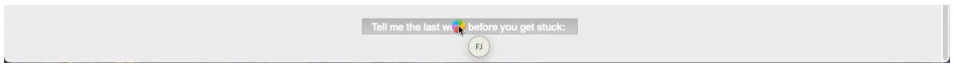

Figure 6.1: System being stuck at producing outputs.

- **ID:** 5

  Requirement: Remove punctuations and upper case letters for each visual output.

  Validity: Valid as it worked for all the participant's input words that require this action. For instance, the first participant has uttered "Algorithm" which is a word at the end of a sentence, leading to an output "the" that is at the beginning of the next sentence instead of "The" as shown inside the original text.

- **ID:** 6

  Requirement: Delete the previous output when the next output appears.

  Validity: Valid as participants were all unable to view the program's previous outputs, plus they would see a different output every time they speak something to the system.

- **ID:** 7

  Requirement: Be able to handle four unexpected user events by printing out their corresponding error messages, for the cases when the user's input word is not found within their note, the user runs out of time for producing their response, the user inputs the last word of their notes, as well as the user inputs the last occurrence of a particular word within their notes.

  Validity: Valid due to the fact that every participant has spent less than two attempts getting those expected warning messages according to the quantitative data being shown inside the Miro board. The "last occurrence check" case has been set as a bonus task due to the strict time constraint (i.e. 30 minutes) of this user study, with only the third participant having successfully triggered that warning message within one single attempt.

- **ID:** 8

  Requirement: Terminate after the program execution (i.e. when the user closes the main window).

  Validity: Valid as the main window has all immediately disappeared when the close button of the application has been pressed by every participant, without leaving any warning message being reported within the terminal, as what the screenshot below shows:

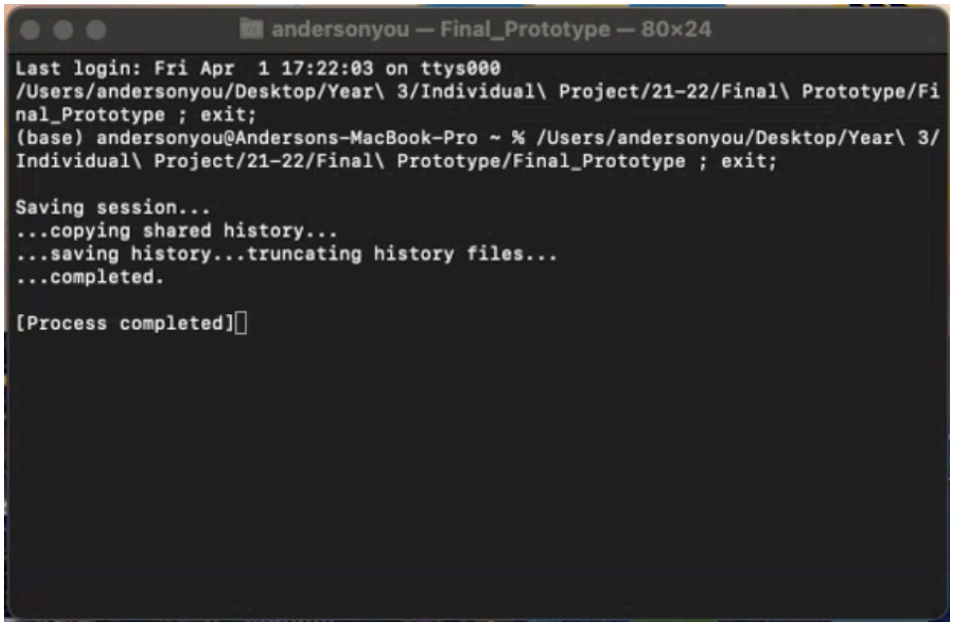

Figure 6.2: Terminal upon the program's completion of running.

6.2.2 Non-Functional Requirements Verification

This section mainly checks against the software's limitations and constraints imposed across development, detailed as follows:

- **ID:** 1

  Requirement: The system should perform in a way that the user wants them to do.

  Validity: According to the testing Q&A table constructed inside the Miro board, there are a few questions asking about the participant's feelings with regards to the system's response. A majority of positive user feelings infer this requirement is valid.

- **ID:** 2

  Requirement: The system should wait for exactly 8 seconds when the user is speaking a word.

  Validity: This can be a valid requirement if the network speed constraint is neglected, due to the fact that in the first two experiments, a restart of the laptop deduces a significant increase in the participant's waiting time.

- **ID:** 3

  Requirement: The user interface of the system should be easy and unambiguous to use.

  Validity: Throughout the entire testing process, a few issues were raised by participants regarding the graphical user interface developed for this system, especially the "not

displaying output" problem when the button is pressed for the very first time, with an average time spent of 56 seconds to figure out implies the annoyance participants' found about this limitation Tkinter gives. Apart from that, three-quarters of the participants have mentioned the additional "user note highlight" mechanism inside their feedback due to struggle to navigate back to the main text and search for a particular word. Therefore this requirement overall is considered to be invalid at this project's stage.

- **ID:** 4

  Requirement: The system should be maintainable such that new features can be easily added and the implementation is simple for any open source contributors to have a clear understanding.

  Validity: Valid as the software itself possesses a majority of the basic functionalities (i.e. detecting user's input words, generating corresponding cue words etc.) that a modern teleprompter gives. New features can be inspired and synthesised based on the feedback each participant gave during the usability testing phase. For implementation concerns, the readability of the program has been enhanced by writing detailed source code comments, with the Numpydoc docstring format being used for each function written[32], as well as including the original source from which the code has been taken and adapted. Notice that the terminology "maintainability" introduced here should not be confused with another term "maintenance" discussed in the next section, with the latter focusing more on "software post-delivery" activities[19].

- **ID:** 5

  Requirement: The prototype should be compatible with most operating systems including Windows, macOS and Linux.

  Validity: As a reiteration of one of the points mentioned earlier in this chapter, only one operating system has been involved throughout the entire user study which is macOS. Hence, there is insufficient evidence to prove the system's compatibility across a wider range of operating systems at the current stage of the project, leading to an "Invalid" status being assigned to this requirement.

- **ID:** 6

  Requirement: The entire design and implementation of the product must adhere to the British Computer Society Code of Conduct.

  Validity: Valid as not only the items listed inside the British Computer Society Code

of Conduct have been thoroughly considered across the project's development but also some other professional issues including cyber security, project management as well as intellectual property rights, with an in-depth discussion for each being carried out inside the next chapter.

6.3 Software Maintenance

As Lehman suggested that "There is no such thing as a 'finished' computer program"[50, 58], the project at this stage is in a good position of thinking about what its software will go through after being deployed to the public. Hence triggered the essence of having a brief tour of another new concept "Software Maintenance" via three aspects as well as its usage under this project's context, with its definition inside IEEE Standard 1219 [IEEE93] as:

"The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment[19, 40]."

6.3.1 Software Maintenance Activity

To accomplish the maintenance goals addressed within the above definition, a broad range of changes to the software product may be required[50]. In recent years several authors have tried to categorise these modifications, yielding a taxonomy of corrective, adaptive, perfective, and preventive changes outlined below[16, 17, 50, 59, 61]:

- **Corrective change** refers to a change brought about by software flaws.
- **Adaptive change** is motivated by the requirement to adjust changes in the software system's environment.
- **Perfective change** corresponds to adjustments made to a system's current requirements.
- **Preventive change** solves issues caused by the deteriorating structure of a software system[50].

It turns out that those four activities can easily be differentiated by inspection. However, this may not be the case when applying them in practice, with its inextricably linked property being depicted as follows[50]:

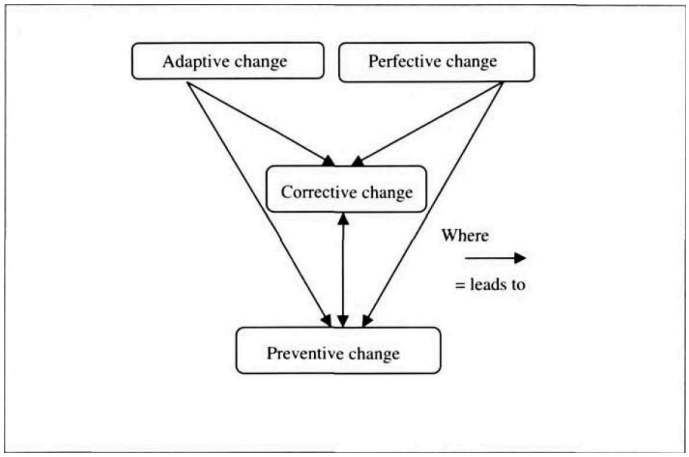

Figure 6.3: Potential relationship between software changes[50].

6.3.2 Reverse Engineering

Reverse engineering by definition gains a high-level overview of a program starting all the way from the bottom of the abstraction level in an ascending fashion before it is being modified (as shown inside the diagram below)[50, 81]. As far as the project time constraints are concerned, only the implementation redocumentation has been carried out during the system's 2nd iteration, with the GUI being added plus some tweaks to the main algorithm.

The usability testing conducted within the first half of this chapter certainly drives further iterations of the system, therefore those recoveries with regards to a higher abstraction level such as design and specification are likely to be ignited, which are generalised as "Future Project Work" in the final chapter.

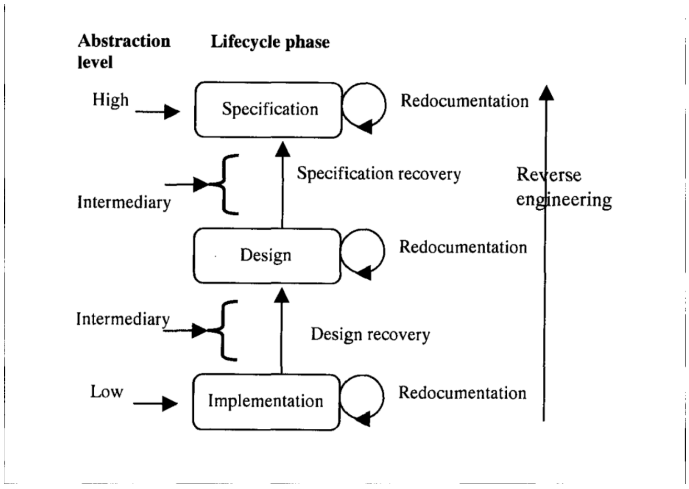

Figure 6.4: Levels of abstraction in a software system[50].

6.3.3 Software Maintenance Cost

The expenses of software maintenance vary based on the software's specific situation, in which the maintenance cost normally is directly proportional to the software's age as technology and programming languages evolve with time[92]. Additionally, it is also worth investigating the trend of this cost in the past five decades as built by Floris and Harald[35, 44]. The diagram below depicts the maintenance costs have almost doubled in 2009 compared to the beginning of the 1970s, with a possibility of gradually approaching 100% in the future.

The future iterations of this project therefore should ideally be carried out regularly, for the sake of avoiding the possible "generation gap" between the software's development team and the maintenance team after a long period of maintenance inactivity[35, 44].

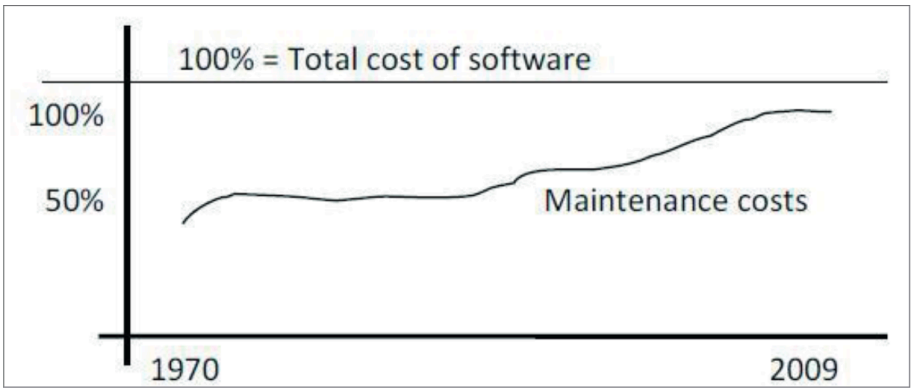

Figure 6.5: Development of software maintenance costs as a percentage of the total cost[35, 44].

Chapter 7

Professional and Ethical Issues

This chapter of the report takes a step further by linking the individual's behaviour of carrying out a project at the university's undergraduate level to the workplace, and at the same time following ethical guidelines and codes of conduct[7].

7.1 BCS Code of Conduct and Code of Good Practice

As a member of the British Computer Society (BCS) since the beginning of the university who stays updated on their current affairs, the rules stated inside the BCS Code of Conduct and Code of Good Practice have been closely followed throughout the entire software development, although not many of them are actually relevant to the project given its current scope[24, 77].

7.2 Cyber Security

Cyber Security addresses all elements of protecting individuals, corporations, and critical infrastructures against risks posed by their usage of computers and the internet[8]. Awareness of 5 Cyber Security Design Principes has been shown across the project implementation, with 2 specific security problems that may stem from technical causes[8].

7.2.1 Miro Board Access and Password Setting

When controlling access to the Miro Board designed for the interview and usability testing purposes, a "read-only access" has been set for the link that prevents guests from maliciously altering the contents inside. To further strengthen its security, a password has been created with a mix of upper, lower case letters and special characters so that users will be prompted

to enter the password after clicking the invitation link, with an example screenshot attached below:

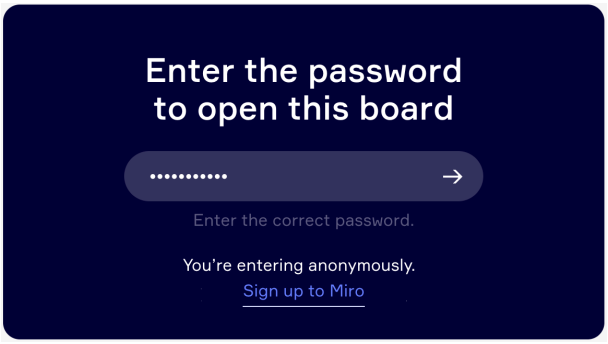

Figure 7.1: A window for entering the password of Miro Board.

7.2.2 Pyinsaller Executable Warning Message

As the "Pyinstaller does not property create executables" issue mentioned in the previous chapter, the operating system (i.e. macOS in this case) will mark the executable as "unregister with Apple by an identified developer" status and throw a warning message upon download as the screenshot displayed below, which stops malware being accidentally installed onto the machine[64].

Here's where guiding users in going through all installation steps become essential, with a user guide located within the Appendix section of the report and an additional README file being provided.

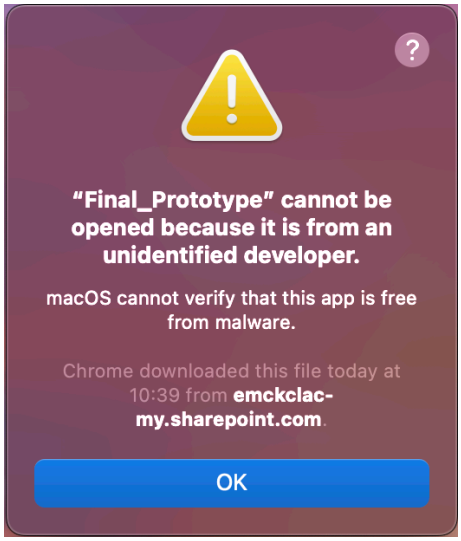

Figure 7.2: A warning message when opening Pyinstaller executable from the download.

7.3 Project Management

As stated by Association for Project Management (APM) that project management uses procedures, techniques, skills, knowledge, and experience to accomplish specified project goals[9, 45], most aspects of the project management process have been considered when creating and developing projects, especially scope management during initiating, as well as schedule management during planning[9].

7.3.1 Scope Management

Scope management involves mainly requirements management and solutions development[9], with listing out what the product is needed after generating ideas from the sketch, followed by the interview that can be deemed the process of looking for the optimal solution of satisfying requirements. The requirements themselves are as well verifiable through usability testing.

7.3.2 Schedule Management

As soon as the project title has been assigned, a Gantt chart has been built following the general project timetable that highlights internal deadlines for each phase of the project. Looking back on it shows that there is one discrepancy in which the co-design process has been replaced by interviewing with SLTs due to the project's time constraint.

Below displays an extract of the Gantt chart for a specific week, a full chart across both semesters can be found via the following link:

`https://docs.google.com/spreadsheets/d/`

`12uaUK-SgteSNX1x6RxYaK3z3Q7OnANf0YBpL1LINqMk/edit#gid=0`

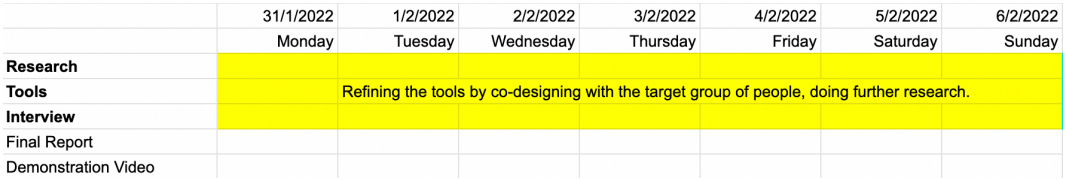

Figure 7.3: Gantt chart extract.

7.4 GDPR and Intellectual Property Rights

General Data Protection Regulation (GDPR) ensures that people can trust you to handle their personal information fairly and ethically[10], which amplifies the necessity of going through

ethical approval for both the interview and usability testing stages of the project. Specifically, the transcripts obtained from those two processes are stored solely inside KCL OneDrive, as stated when filling out the application form.

As far as the software's concerned, its intellectual property rights are highly contentious and all too easy to be misled by information obtained on the internet since it does not identify specifically to which jurisdiction it is referring or to what moment in time[10, 26]. These have been carefully considered when carrying out the project and the report, via citing all the original sources that have taken the inspiration or directly borrowed from, including importing Google Speech Recognition and Text to Speech API, as well as adapting a generic method of implementing a full-screen scrollbar inside Tkinter window for the sake of avoiding harming the author's rights of personality[41, 73, 79, 80].

Chapter 8

Conclusion and Future Work

The project itself lies at the intersection of two fields named Human-Computer Interaction (HCI) and Natural Language Processing (NLP), with the former playing a more important role in the current state of the project whereas the latter is likely to become dominant with respect to the system's further iterations. This is due to the fact that the project at this stage only strives for usability and public acceptance referred to as two aims highlighted in the first chapter.

> *"A project is usually deemed to be a success if it achieves the objectives according to their acceptance criteria, within an agreed timescale and budget[9, 45]."*

When comes to judging the project's success, a quote defined by APM has been considered. Looking back to the requirement's verification section, all functional requirements and two-thirds of non-functional requirements being satisfied deduce that the project overall is successful, despite some reported user interface limitations and system compatibility have not yet been addressed under the project's time constraint.

The remainder of this chapter dedicates to a few aspects that may be taken into account for the project's future iterations according to the feedback given by participants during the user study, for the sake of slowly transitioning this project to be NLP oriented (i.e. to improve the model's flexibility by generating more appropriate outputs based upon the user's interaction to the system), as well as how this project can be applied and adapted to some other scenarios.

8.1 Interaction Loop Improvement and Performance Metrics

*"I guess it can clearly understand where I am in the sentences in the paragraph that suggests me the word that I need to do to use later. Uh, maybe if it could suggest me not just one word like it was 'it's' maybe if it could suggest me some sentence it would be a bit more beneficial while I'm speaking because if I say cryptography, 'it's' will not be very helpful for me I guess, because it's just one word."***(Participant 3)**

*"What if the word 'Cryptography' has appeared in the earlier paragraph and the user doesn't need help there?"***(Participant 3)**

The above two statements pinpoint the exact flexibilities the system can give, with the first quote suggesting that the model can generate a brief phrase or even a complete sentence rather than a single word and the second one criticising the current "occurrence check" mechanism, by which the system should ideally know what occurrence exactly the user does struggle with it and what does not, rather than starting all the way back from the first occurrence every time the button is pressed by the user.

These are deemed to be the most challenging part of the entire project since the various presenting skills of the user have triggered the essence of the project being trained by a wide range of data sets (i.e. human language in this case) in order for the model to produce more flexible and accurate outputs.

It may also be feasible to compare the output user expects with what the system actually generates via a confusion matrix displayed as a screenshot below[11, 69]. The percentage of users correctly making predictions of the system when generating expected outputs can be computed as[11]:

$$Precision = TP/(TP + FP) \tag{8.1}$$

Similarly, the percentage of users correctly predicting the system when generating unexpected outputs can be calculated as[11]:

$$Precision = TN/(TN + FN) \tag{8.2}$$

		Estimate	
		0	1
Truth	0	TN	FP
	1	FN	TP

Figure 8.1: Confusion matrix for a binary classification problem, with TN being True Negative, FN being False Negative, FP being False Positive and TP being True Positive[69].

Although any real model will produce at least some FNs and FPs, the project still aims for a relatively high number of TP and TN and low numbers of FN and FP in the future[11].

*"Maybe giving some suggestions of like words that can replace something. For example, you've been using the word software like 10 times or like so software is not actually a good example. But let's say uh you've used the word example 20 times that clearly isn't good for a presentation to include, like to be repetitive and those kinds of things. And maybe if you reach some limit of uhm occurrences of a word the software can for example suggest you uh a word to use next time, for example, just not to be repetitive, I don't know if you catch what I mean."***(Participant 4)**

This quote selected refers to another tweak that can be made with regards to the interaction loop, via offering users alternative words that they can use for the case of using a single word/phrase repetitively, which not only helps improve the user's presentation skills but also gives the audience a better experience.

8.2 Frontend Improvement

*"Maybe one thing it could do is as a user interface it could highlight whether the user got stuck so it can also find the word in the text easier because I struggled to find the word in the text for example, it could do like a highlight."***(Participant 2)**

*"Maybe if it would navigate to that sentence in the text automatically, it would be quite useful without saying it out loud, because this way I would be able to not confuse listeners if they hear what the software is telling and I'll be able to just go back in the text, it will be highlighted for example, and I'll be able to continue reading it."***(Participant 3)**

"Maybe also when you're speaking or when you're presenting it like not only, uh highlights the output, but when you're talking like, it highlights the text where you're speaking, it's gonna be

*visually better for like for example, the people watching the presentation. For example, when I'm presenting you saw how I was reading the text, right? While I'm reading the text it goes through the text. I don't know if you know about Spotify, for example, when you listen to music and how to like while you're listening to music, they're like showing the lyrics in, they're going through the lyrics while the song is going on. That might be very useful for a presentation scenario coz then you not only rely on their ears, but you rely on their eyes coz they're going to be focused on what's like what you're talking about if they uh, miss something you're talking about. They're gonna see it on the screen."***(Participant 4)***

These three statements all revolve around adding highlights either to the system's output word inside the text, or to the part of the text where the user's speech is up to, or both of them. As the label widget has been selected for storing the user's notes in this project, the current Tkinter version disallows attributes of some specific characters to be modified for this widget, and other widgets such as the text widget unfortunately only possess the manual highlighting mechanism instead of the automatic highlight[71]. An alternative GUI platform may be adopted in future development in order to avoid that specific limation Tkinter gives.

Additionally, one of the participants has suggested making the button more obvious to the user for the sake of not being confused with bolded text *("Maybe I will make a button blue or different colour because this way it will be a bit more clear that it's button and not just text.")*. This is because not all participants during the usability testing were able to successfully spot that button, as one of the other participants has mentioned *("I think it's just like a text in the middle that is bold.")*.

8.3 System's Sensitivity

*"I think it can help probably, I guess it would have to be a little more polished in a way that maybe it could be a little quicker. So if it's quick, I think it could definitely help in real-world situations. In this state. I think maybe it would need a little more polished."***(Participant 2)**

This quote is selected as a reiteration of one of the points mentioned when verifying the fourth functional requirement in Chapter Six. A complete drop in the network speed constraint should be considered in future project iterations, in which the users are still able to use the system for situations when the network is unavailable, such as during a flight.

8.4 New Scenarios Application

After having an exploration of the areas in which the current state of the system can be improved, it is time to broaden out the project scope as stated in Chapter Three by considering two new scenarios:

- **People singing or holding the concert:** This requires the system to recognise the human singing voice as well as generate output as closer to the user's sound as possible, inferring that various tones must be added to both the speech detection and text-to-speech algorithm. Another challenge will be not to mis-detect the background music and the audience's noise as the actual user's input.

- **People with aphasia:** It is still worth taking into account this particular user group as they are a subset of people who have difficulty speaking in public, as outlined within the first chapter. However, besides communication impairment, some of them also struggle with reading, as one of the SLTs has mentioned during the interview *("Apart from speaking and understanding, it's also reading and writing but this is not the case for all of them. Some people will have problems with reading after the stroke. Some people will not. And also some people with writing. But this is not that much with learning, because before the stroke they knew the language. But now after the stroke, they have difficulties finding their proper writing or reading, but they can still learn probably in a slower way. But it's mostly about them losing the abilities they had before.").* Therefore, the user's note is supposed to be modified in a way that is more readable to people specifically with aphasia, such as increasing the font size of the main text shown inside the screenshot below plus the auto-scroll mechanism being introduced as the user's speech goes along.

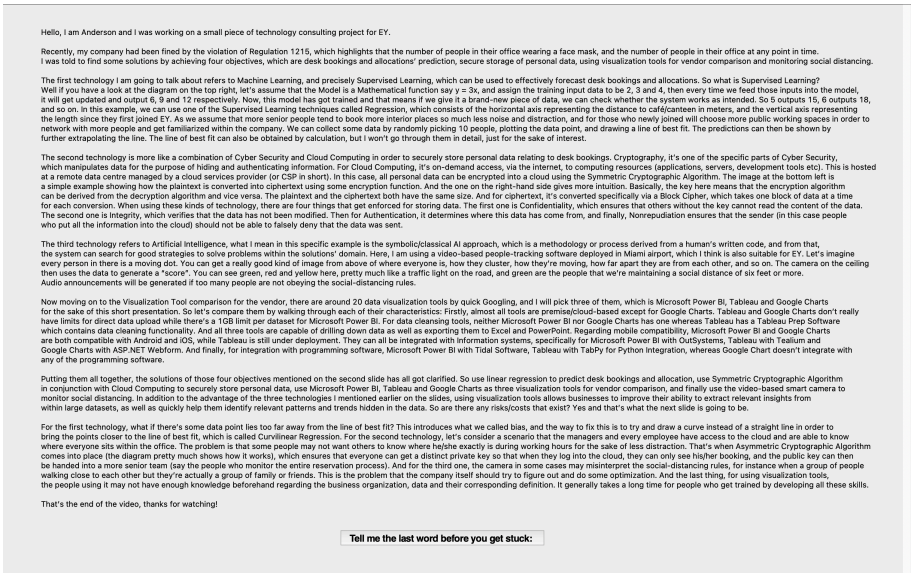

Figure 8.2: Main window of the final prototype.

Considering the project's maintainability being defined and verified in the earlier chapters, more experienced developers are on the way to bringing the current project to the next level...

References

[1] Petr Slovak, 2021, Lecture 1: Introduction to interaction design, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered September 2021.

[2] Petr Slovak, 2021, Lecture 2: Usability and cognition, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered October 2021.

[3] Petr Slovak, 2021, Lecture 4: Define: Identifying the right problem, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered October 2021.

[4] Petr Slovak, 2021, Lecture 3: Empathise: Understanding user needs, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered October 2021.

[5] Petr Slovak, 2021, Lecture 7: Heuristic evaluation, usability user test, field studies and appropriation, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered November 2021.

[6] Petr Slovak, 2021, Lecture 6: Prototyping week: Doing is the best thinking, lecture slides, Human-Computer Interaction 6CCS3HCI, King's College London, delivered November 2021.

[7] Sahar Al-Sudani and Leonardo Magela Cunha, 2019, Introductory Lecture: Module Overview, lecture slides, Introduction to Professional Practice 4CCS1IPP, King's College London, delivered October 2019.

[8] Sahar Al-Sudani, 2019, Lecture 2: Cyber Security, lecture slides, Introduction to Professional Practice 4CCS1IPP, King's College London, delivered October 2019.

[9] Leonardo Magela Cunha, 2019, Lecture 3: Project Management, lecture slides, Introduction to Professional Practice 4CCS1IPP, King's College London, delivered October 2019.

[10] Leonardo Magela Cunha, 2019, Lecture 4: Professional skills for a globalised world, GDPR and intellectual property rights, lecture slides, Introduction to Professional Practice 4CCS1IPP, King's College London, delivered November 2019.

[11] Helen Yannakoudakis, 2022, Lecture 1: Introduction to Machine Learning, lecture slides, Machine Learning 6CCS3ML1, King's College London, delivered January 2022.

[12] Speechrecognition 3.8.1, 2017. [Online]. Available at: `https://pypi.org/project/SpeechRecognition/`.

[13] Abhishek. "variable" not accessed, 2021. [Online]. Available at: `https://stackoverflow.com/questions/66074642/variable-not-accessed`.

[14] Meenakshi Agarwal. Multiline string in python with examples. [Online]. Available at: `https://www.techbeamers.com/python-multiline-string/`.

[15] Nikhil Aggarwal. Defaultdict in python, 2022. [Online]. Available at: `https://www.geeksforgeeks.org/defaultdict-in-python/`.

[16] Ghazi Alkhatib. The maintenance problem of application software: An empirical analysis. *Journal of Software Maintenance: Research and Practice*, 4(2):83–104, 1992.

[17] Lowell Jay Arthur. *Software evolution: the software maintenance challenge*. Wiley-Interscience, 1988.

[18] David J. Barnes and Michael Kolling. *Objects First with Java: a Practical Introduction Using BlueJ, Global Edition*. ProQuest Ebook Central, 2017.

[19] Keith H Bennett and Václav T Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, page 75, 2000.

[20] David R Beukelman, Pat Mirenda, et al. *Augmentative and alternative communication*. Paul H. Brookes Baltimore, 1998.

[21] Nigel Bevana, Jurek Kirakowskib, and Jonathan Maissela. What is usability. In *Proceedings of the 4th International Conference on HCI*, page 1. Citeseer, 1991.

[22] BIONITY. Aphasia. [Online]. Available at: `https://www.bionity.com/en/encyclopedia/Aphasia.html`.

[23] BIONITY. Glossophobia. [Online]. Available at: `https://www.bionity.com/en/encyclopedia/Glossophobia.html`.

[24] Trustee Board. Bcs, the chartered institute for it code of conduct for bcs members, 2021. [Online]. Available at: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf.

[25] Thomas Both. Design project guide, 2016. [Online]. Available at: https://static1.squarespace.com/static/57c6b79629687fde090a0fdd/t/6130fdb46ff0113eb66ba091/1630600647282/DESIGNPROJECTGUIDE-SEPT-2016-V3.pdf.

[26] Frank Bott. *Professional Issues in Information Technology*. ProQuest Ebook Central, 2014.

[27] Virginia Braun and Victoria Clarke. *Successful qualitative research: A practical guide for beginners*. sage, 2013.

[28] John Brooke, Nigel Bevan, Fred Brigham, Susan Harker, and David Youmans. Usability statements and standardisation: Work in progress in iso. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 357–361, 1990.

[29] Kyla Brown, Linda Worrall, Bronwyn Davidson, and Tami Howe. Living successfully with aphasia: Family members share their views. *Topics in stroke rehabilitation*, 18(5):542, 2011.

[30] Claire D. Costa. Top 10 python gui frameworks for developers, 2020. [Online]. Available at: https://towardsdatascience.com/top-10-python-gui-frameworks-for-developers-adca32fbe6fc.

[31] crasic. How to find word next to a word in python, 2013. [Online]. Available at: https://stackoverflow.com/questions/17686809/how-to-find-word-next-to-a-word-in-python.

[32] daouzli. What are the most common python docstring formats? [closed], 2014. [Online]. Available at: https://stackoverflow.com/questions/3898572/what-are-the-most-common-python-docstring-formats.

[33] Murial Daran and Pascale Thévenod-Fosse. Software error analysis: A real case study involving real faults and mutations. *ACM SIGSOFT Software Engineering Notes*, 21(3):158, 1996.

[34] John Dattilo, Gus Estrella, Laura J Estrella, Janice Light, David McNaughton, and Meagan Seabury. “i have chosen to live life abundantly”: Perceptions of leisure by adults

who use augmentative and alternative communication. *Augmentative and alternative communication*, 24(1):21, 2008.

[35] Sayed Mehdi Hejazi Dehaghani and Nafiseh Hajrahimi. Which factors affect software projects maintenance cost more? *Acta Informatica Medica*, 21(1):63, 2013.

[36] DelftStack. Set font of tkinter text widget, 2020. [Online]. Available at: https://www.delftstack.com/howto/python-tkinter/how-to-set-font-of-tkinter-text-widget/#:~:text=Set%20Font%20for%20Tkinter%20Text%20Widget%20With%20tkFont,-We%20could%20also&text=family%20%2D%20font%20family%2C%20like%20Arial,font%20slant%3A%20roman%20or%20italic.

[37] DelftStack. Remove punctuation from python list, 2021. [Online]. Available at: https://www.delftstack.com/howto/python/python-remove-punctuation-from-list/.

[38] Joseph S Dumas, Joseph S Dumas, and Janice Redish. *A practical guide to usability testing*. Intellect books, 1999.

[39] Pierre Nicolas Durette. Module (gtts), 2022. [Online]. Available at: https://gtts.readthedocs.io/en/latest/module.html#playing-sound-directly.

[40] D Vera Edelstein. Report on the ieee std 1219–1993—standard for software maintenance. *ACM SIGSOFT Software Engineering Notes*, 18(4):94, 1993.

[41] John Elder. Adding a full screen scrollbar - python tkinter gui tutorial #96, 2020. [Online]. Available at: https://www.youtube.com/watch?v=0WafQCaok6g.

[42] John Elder. Overwrite grid labels - python tkinter gui tutorial #42, 2020. [Online]. Available at: https://www.youtube.com/watch?v=Q-rRF6c8kJM&t=36s.

[43] Ethan Field. Python : How to center label in tkinter window, 2017. [Online]. Available at: https://stackoverflow.com/questions/46069531/python-how-to-center-label-in-tkinter-window.

[44] P Floris and H Vogt Harald. How to save on software maintenance costs. *Omnext white paper, SOURCE*, 2, 2010.

[45] Association for Project Management. What is project management? [Online]. Available at: https://www.apm.org.uk/resources/what-is-project-management/#:

~:text=Project%20management%20is%20the%20application,a%20finite%20timescale%20and%20budget.

[46] Lisa Fritscher. Glossophobia or the fear of public speaking? symptoms, complications, and treatments, 2021. [Online]. Available at: https://www.verywellmind.com/glossophobia-2671860#:~:text=Glossophobia%2C%20or%20the%20fear%20of,manage%20and%20control%20the%20fear.

[47] Bhavya Ghai and Klaus Mueller. Fluent: An ai augmented writing tool for people who stutter. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, pages 2,3, 2021.

[48] Sharon Glennen and Denise C DeCoste. *The handbook of augmentative and alternative communication*. Cengage Learning, 1997.

[49] Adam Gordon, René Rohrbeck, and Jan Oliver Schwarz. Escaping the" faster horses" trap: bridging strategic foresight and design-based innovation. page 32, 2019.

[50] Penny Grubb and Armstrong A Takang. *Software maintenance: concepts and practice*. World Scientific, 2003.

[51] Greg Hewgill. Understanding slice notation, 2009. [Online]. Available at: https://stackoverflow.com/questions/509211/understanding-slice-notation.

[52] Garry Hurst. set window icon tkinter macosx, 2018. [Online]. Available at: https://stackoverflow.com/questions/48981184/set-window-icon-tkinter-macosx.

[53] Bernard J Jansen. The graphical user interface. *ACM SIGCHI Bulletin*, 30(2):22, 1998.

[54] Jeanne M Johnson, Ella Inglebret, Carla Jones, and Jayanti Ray. Perspectives of speech language pathologists regarding success versus abandonment of aac. *Augmentative and Alternative Communication*, 22(2):86, 2006.

[55] Robert L Kahn and Charles F Cannell. The dynamics of interviewing; theory, technique, and cases. 1957.

[56] Brenden Kromhout. python function default parameter is evaluated only once? [duplicate], 2015. [Online]. Available at: https://stackoverflow.com/questions/13087344/python-function-default-parameter-is-evaluated-only-once.

[57] Data Science Learner. python defaultdict overview with example : Implementation. [Online]. Available at: https://www.datasciencelearner.com/python-defaultdict-example-implementation/.

[58] Manny M Lehman and Laszlo A Belady. *Program evolution: processes of software change*. Academic Press Professional, Inc., 1985.

[59] Bennett P Lientz and E Burton Swanson. *Software maintenance management*. Addison-Wesley Longman Publishing Co., Inc., 1980.

[60] Janice Light, Betty Stoltz, and David McNaughton. Community-based employment: Experiences of adults who use aac. *Augmentative and alternative communication*, 12(4):224, 1996.

[61] David H Longstreet. *Software maintenance and computers*. IEEE Computer Society Press, 1990.

[62] Hoa Loranger. Checklist for planning usability studies, 2016. [Online]. Available at: https://www.nngroup.com/articles/usability-test-checklist/.

[63] Fredrik Lundh. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, page 16, 1999.

[64] macOS User Guide. Open a mac app from an unidentified developer. [Online]. Available at: https://support.apple.com/en-gb/guide/mac-help/mh40616/mac.

[65] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[66] David McNaughton, Tracy Rackensperger, Elizabeth Benedek-Wood, Carole Krezman, Michael B Williams, and Janice Light. "a child needs to be given a chance to succeed": Parents of individuals who use aac describe the benefits and challenges of learning aac technologies. *Augmentative and alternative communication*, 24(1):50, 2008.

[67] A Moorcroft, N Scarinci, and C Meyer. A systematic review of the barriers and facilitators to the provision and use of low-tech and unaided aac systems for people with complex communication needs and their families. *Disability and Rehabilitation: Assistive Technology*, 14(7):711, 2019.

[68] Joan Murphy, Ivana Marková, Sarah Collins, and Eleanor Moodie. Aac systems*: obstacles to effective use. *International Journal of Language & Communication Disorders*, 31(1):31,39,43, 1996.

[69] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[70] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 255, 1990.

[71] Bryan Oakley. Python tkinter single label with bold and normal text, 2016. [Online]. Available at: `https://stackoverflow.com/questions/40237671/python-tkinter-single-label-with-bold-and-normal-text`.

[72] Bryan Oakley. What does 'weight' do in tkinter?, 2019. [Online]. Available at: `https://stackoverflow.com/questions/45847313/what-does-weight-do-in-tkinter`.

[73] Institute of Research, Action on Fraud, and Plagiarism in Academia. The ten consequences of plagiarism. [Online]. Available at: `https://irafpa.org/en/methods/investigating-cases-of-plagiarism/the-ten-consequences-of-plagiarism/`.

[74] Sidekick Therapy Partners. Faqs about your aac device, 2020. [Online]. Available at: `https://www.mysidekicktherapy.com/blog/articles/faqs-augmentative-alternative-communication-aac-device`.

[75] Betsy Phillips and Hongxin Zhao. Predictors of assistive technology abandonment. *Assistive technology*, 5(1):36–45, 1993.

[76] Programiz. Python exception handling using try, except and finally statement. [Online]. Available at: `https://www.programiz.com/python-programming/exception-handling`.

[77] BCS Qualifications, Standards Board, and Trustee Board. The british computer society code of good practice, 2004. [Online]. Available at: `http://www.inf.ed.ac.uk/teaching/courses/pi/2013_2014/notes/1/cop.pdf`.

[78] Raja. Python list index(), 2021. [Online]. Available at: `https://www.geeksforgeeks.org/python-list-index/`.

[79] Abdou Rockikz. How to convert speech to text in python, 2021. [Online]. Available at: `https://www.thepythoncode.com/article/using-speech-recognition-to-convert-speech-to-text-python`.

[80] Abdou Rockikz. How to convert text to speech in python, 2022. [Online]. Available at: https://www.thepythoncode.com/article/convert-text-to-speech-in-python.

[81] Spencer Rugaber, Stephen B. Ornburn, and Richard J LeBlanc. Recognizing design decisions in programs. *IEEE Software*, 7(1):50, 1990.

[82] Marcia Joslyn Scherer. *Living in the state of stuck: How assistive technology impacts the lives of people with disabilities*. Brookline Books, 2005.

[83] Janet Scott. Low tech methods of augmentative communication. *Augmentative Communication in Practice 2*, pages 5,13, 1998.

[84] Akash Kumar Sen. How to set font for text in tkinter?, 2020. [Online]. Available at: https://www.geeksforgeeks.org/how-to-set-font-for-text-in-tkinter/.

[85] Dev Prakash Sharma. How to justify text in label in tkinter in python need justify in tkinter?, 2021. [Online]. Available at: https://www.tutorialspoint.com/how-to-justify-text-in-label-in-tkinter-in-python-need-justify-in-tkinter.

[86] Helen Sharp, Jennifer Preece, and Yvonne Rogers. *Interaction Design: beyond human-computer interaction*. ProQuest Ebook Central, 2019.

[87] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[88] Jeff Sigafoos and Erik Drasgow. Conditional use of aided and unaided aac: A review and clinical case demonstration. *Focus on Autism and Other Developmental Disabilities*, 16(3):152, 2001.

[89] Ian Sommerville. *Software Engineering*. ProQuest Ebook Central, 2016.

[90] Dean E Sutherland, Gail G Gillon, and David E Yoder. Aac use and service provision: A survey of new zealand speech-language therapists. *Augmentative and Alternative Communication*, 21(4):303, 2005.

[91] Wei-siong Tan, Dahai Liu, and Ram Bishu. Web evaluation: Heuristic evaluation vs. user testing. *International Journal of Industrial Ergonomics*, 39(4):626, 2009.

[92] Thales. The 4 types of software maintenance. [Online]. Available at: https://cpl.thalesgroup.com/software-monetization/

four-types-of-software-maintenance#:~:text=Software%20maintenance%20is%20the%20process,to%20boost%20performance%2C%20and%20more.

[93] Tactus Therapy. What is aphasia? [Online]. Available at: https://tactustherapy.com/what-is-aphasia/.

[94] Tactus Therapy. Speech flipbook, 2022. [Online]. Available at: https://tactustherapy.com/app/speech-flipbook/.

[95] Nadia Roumani Thomas Both. Ethnography fieldguide, 2019. [Online]. Available at: https://static1.squarespace.com/static/57c6b79629687fde090a0fdd/t/5d980bbeef5b5717b186539f/1570245598860/Ethnography+Fieldguide-DSS-Aug-2019-screen.pdf.

[96] Data to Fish. Create executable from python script using pyinstaller, 2021. [Online]. Available at: https://datatofish.com/executable-pyinstaller/.

[97] TobiiDynavox. Td snap, 2022. [Online]. Available at: https://www.mytobiidynavox.com/Store/Snap.

[98] TutorialKart. Tkinter button font. [Online]. Available at: https://www.tutorialkart.com/python/tkinter/button/font/.

[99] UKGov. Disability prevalence estimates 2011/12, 2012. [Online]. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/321594/disability-prevalence.pdf.

[100] user14203455. Pyinstaller exe not working on other computer(with other windows ver.), 2021. [Online]. Available at: https://stackoverflow.com/questions/65704901/pyinstaller-exe-not-working-on-other-computerwith-other-windows-ver.

[101] Stephanie Wilson, Abi Roper, Jane Marshall, Julia Galliers, Niamh Devane, Tracey Booth, and Celia Woolf. Codesign for people with aphasia through tangible design languages. *CoDesign*, 11(1):21, 2015.