University of Plymouth

School of Engineering, Computing, and Mathematics

COMP3000

Final Stage Computing Project

2020/2021

Hiccup Virtual Assistant

David Keay

10582332

Bsc (Hons) Computer Science

Acknowledgements

I would like to thank Thomas Wennekers for his guidance and advice throughout this project.

# Abstract

This report will outline a software development project for a virtual assistant Windows desktop application. The report begins with exploring the background information on virtual assistants, and the alternative technologies that currently exist in the market. It then investigate the technologies used within the application, such as Mozilla’s DeepSpeech speech-to-text engine. Next, the report explores the aims and objectives of the project, as well as a description of the legal, ethical, and professional issues surrounding it.

The development process of the project is explored, and describes the project management techniques used, such as the agile development environment, which allowed for incremental gain in functionality while also remaining flexible when it came to changes in development ideas or the schedule. Each sprint is described in the project management section, explaining the functionality added and the refining of the development style. The functional requirements for the project are then explored, and the user testing is described and reviewed.

Closing off the report is a post-mortem of the project, highlighting which requirements were included, which were removed due to constraints, what could have gone better within the project, and a critical evaluation of the entire project.

Contents

[Abstract 1](#_Toc79074753)

[Statement of Word Count 4](#_Toc79074754)

[Code Repository 4](#_Toc79074755)

[1. Introduction 5](#_Toc79074756)

[1.1. Background 5](#_Toc79074757)

[1.2. Existing Solutions 5](#_Toc79074758)

[1.3. Aims, Objectives, and Deliverables 6](#_Toc79074759)

[2. Tools and Technologies 7](#_Toc79074760)

[2.1. Languages Used 7](#_Toc79074761)

[2.2 Technological Justification 7](#_Toc79074762)

[3. Legal, Social, Ethical, and Professional Issues 9](#_Toc79074763)

[4. Method of Approach 10](#_Toc79074764)

[4.1. Agile 10](#_Toc79074765)

[5. Project Management 11](#_Toc79074766)

[5.1 Supervisor Meetings 11](#_Toc79074767)

[6. Functional Requirements 11](#_Toc79074768)

[6.1 Functional Requirements 11](#_Toc79074769)

[Function pseudocode 13](#_Toc79074770)

[7. Architecture and Design 13](#_Toc79074771)

[7.1 Interpreting Commands 13](#_Toc79074772)

[7.2 Speech Recognition 14](#_Toc79074773)

[7.3 Executing Commands 14](#_Toc79074774)

[7.4 Possible Extension of Architecture 15](#_Toc79074775)

[8. Project Development Sprints 16](#_Toc79074776)

[8.1 Sprint 1 16](#_Toc79074777)

[8.2 Sprint 2 16](#_Toc79074778)

[8.3 Sprint 3 16](#_Toc79074779)

[8.4 Sprint 4 17](#_Toc79074780)

[8.5 Sprint 5 17](#_Toc79074781)

[8.6 Sprint 6 17](#_Toc79074782)

[8.7 Sprint 7 17](#_Toc79074783)

[9. Sprint Retrospectives 18](#_Toc79074784)

[9.1 Sprint 1 18](#_Toc79074785)

[10. Testing 18](#_Toc79074786)

[11. End-Project Report 18](#_Toc79074787)

[12. Project Post-Mortem 18](#_Toc79074788)

[12.1 What did go well? 18](#_Toc79074789)

[12.2 What did not go well? 18](#_Toc79074790)

[12.3 What would be done differently? 19](#_Toc79074791)

[12.4 Potential Extensions of the Project 19](#_Toc79074792)

[13. Conclusions 20](#_Toc79074793)

[14. References 21](#_Toc79074794)

[15. Appendices 22](#_Toc79074795)

# Statement of Word Count

Word count:

# Code Repository

The code repository can be found at:

https://github.com/keayd/COMP3000-Hiccup-VA

# 1. Introduction

Virtual assistants are a fast-growing market in the modern day. In the United Kingdom alone, “22 per cent of UK households now own a voice-controlled digital home assistant device, such as an Amazon Echo or Google Home, doubling the 11 per cent figure recorded in 2017” (Stewart, 2019). Furthermore, it is predicted that “ by 2024, the number of digital voice assistants will reach 8.4 billion units” (Vailshery, 2021). With predictions of having more digital voice assistants than the world’s population, it is clear how essential it is that not only are these virtual assistants easy to use, but that they are able to have a wide range of utility and functionality if they are to simplify day-to-day life.

Being able to operate a computer is becoming an essential skill for the modern worker, so for the visually impaired members of our society, it becomes more difficult for them to operate computers when sight is such an important factor. However, with few tools available to assist them, visually impaired people require a tool that can make operating computers more simple.

## 1.1. Background

For the visually impaired and general consumers who need assistance in day-to-day operation of their PC, the ‘Hiccup’ Virtual Assistant is a general use virtual assistant that can open programs, search the web, dictate, and read to them, all controlled by the user’s voice.

Hiccup is a windows desktop application that uses a microphone for the user to communicate with the virtual assistant, the user is also able to communicate using the keyboard in case they do not have a microphone available. Hiccup can produce multiple responses to small talk lines such as ‘hello’ and ‘goodbye’, and can answer questions the user has such as “what time is it?” and “what is the weather like?”

## 1.2. Existing Solutions

Virtual assistants are a very popular market in the modern day. There are many different solutions currently available from large tech companies such as Google’s Google Assistant, which is used in more than 90 countries by over 500 million people each month (Bronstein, 2020). Another well-known solution is the Amazon Alexa virtual assistant, which was initially available in the Amazon Echo, Echo Dot, Echo Studio, and Amazon Tap speakers developed by the company, but has since expanded to be supported on over 20,000 devices (Heater, 2018). Both of these products are able to connect to and control smart devices within a user’s home, using themselves as a home automation system. Google Assistant uses natural language processing, designed to process the spoken instruction from the user in a literal sense; whereas the Amazon Alexa uses natural language understanding, which is built to extract the context and intent of the instruction rather than the literal meaning.

One advantage that the Hiccup Virtual Assistant has over the other existing products currently on the market is that Hiccup does not collect personal data from its users. In 2019, Fowler wrote that “Amazon keeps a copy of everything Alexa records after it hears its name… Google’s Assistant, by default also keep recordings to help train their artificial intelligences.” This is an issue with privacy, as Amazon employees were given access to these recordings for the purpose of improving the artificial intelligence, but these recordings could have confidential information in them, which is a major issue. Hiccup, however, does not collect data, after the spoken command is processed by the pattern matching functions, it is not stored, ensuring that the users personal data remains safe.

## 1.3. Aims, Objectives, and Deliverables

Three development goals: Core, Desired Functionality, Extended Functionality

Core

The minimum viable product for the project. Contains speech recognition; GUI; speech to text writing, basic commands such as current time and hello world. Commands are hard coded and do not allow for interpretation.

Desired functionality

Application opener; web search functionality; text to speech responses; Some commands are hard coded, others allow for limited interpretation. Small talk functionality is included.

Extended functionality

Natural Language understanding system to ‘read into’ command spoken by user; learning from answers; recognizing voice of main user only using machine learning; and an ‘Eliza’ type system is implemented.

The final deliverable of this project will consist of a windows desktop application which should have successfully fulfilled the aims and objectives of the project.

# 2. Tools and Technologies

## 2.1. Languages Used

This project uses C# for the backend functionality. C# is used here as it has native garbage collection, allowing for more time and effort to be focused on building functionality. C# is also deeply integrated with windows, which is extremely beneficial for this project as it is exclusively a windows desktop application.

C# is an object-oriented programming language, which has four basic principles: Abstraction, which is creating classes using relevant attributes to define an abstract representation of a system; Encapsulation, which is about hiding the internal functionality of an object and only allowing access through public functions; Utilizing inheritance allows child objects to ‘inherit’ attributes and functions of their parent objects, while also being able to possess new attributes and functions not shared by the parent; and Polymorphism, which allows for the implementation of properties or methods in different ways across multiple abstractions

For command interpretation, we use F# for it’s powerful pattern matching functionality to accurately match what kind of functionality the user wishes to use. F# is used here rather than C#, as it is capable of creating partial active pattern recognizers in a lightweight, easy, and flexible manner.

F# is a functional programming language, meaning that, unlike object-oriented programming languages, it does not manipulate the state of an object and it’s functions do not have any side effects, instead it uses pure functions, which is one whose results are only dependent upon the input parameters, and make no external impact other than the result.

## 2.2 Technological Justification

Early on in development, it was decided that using an external library for the speech recognition aspect of the project would be of best interest. This is because speech recognition software requires a significant amount of time and resources to develop to a high-quality standard wherein it has a low word-error rate, which would not be permissible given the amount of time allocated towards the project. It was decided that using an external library would allow for more time and focus to be put towards developing the different levels of functionality of the application, as having a broader scope of functionality allows for the application to be more flexible and have a wider reach.

Initially, the Microsoft Azure Speech libraries were used, as they are extremely powerful libraries that have a low word-error rate for their speech to text library, as well as high quality ‘neural’ text to speech voices which are capable of “highly expressive and human-like voices.” (Microsoft, 2021). This was valuable as it allows for easier interaction between the user and the application, and allowed for the application to be more flexible with the way in which it interpreted commands.

Ultimately, it was decided that the Microsoft Azure Speech libraries would not be used in the application, as it is not a free library, and the ‘free’ version of the plan is extremely limited in how much it is able to be used. Instead, the Mozilla DeepSpeech library was chosen as the best alternative for the speech recognition library. DeepSpeech is an open-source library, allowing unlimited use at no cost. The price-performance ratio of DeepSpeech is significantly better for a project such as this as, despite it not containing the extra features that the Microsoft Azure Speech library contains, it is not restricted in the amount of time it is able to be used for, meaning that more testing will be able to be performed to improve the overall project.

# 3. Legal, Social, Ethical, and Professional Issues

When developing a virtual assistant, there are many legal, social, ethical, and professional issues to address.

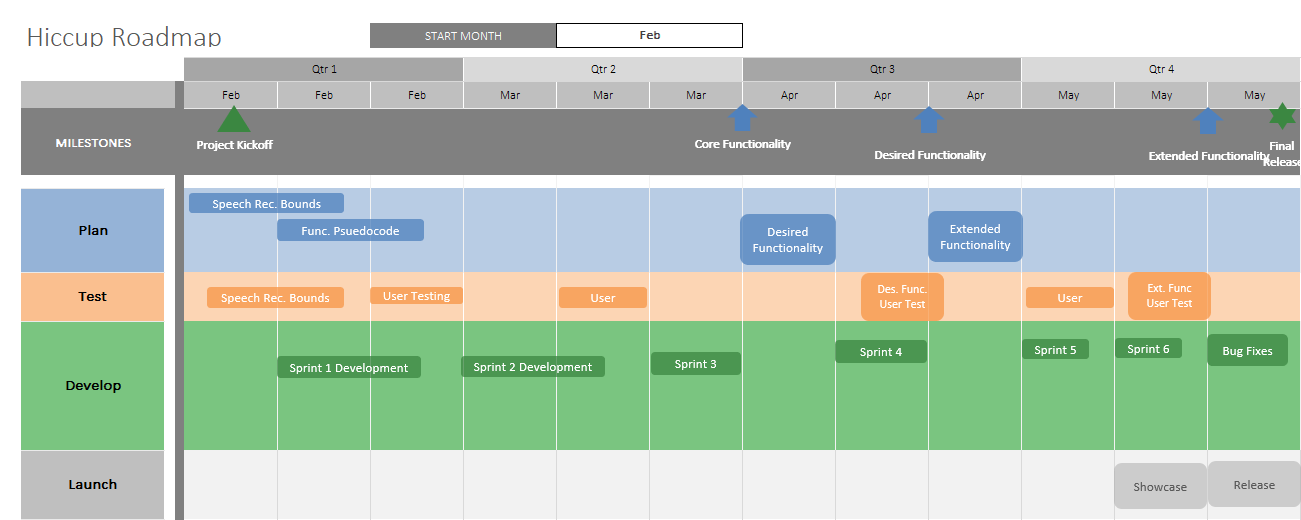
* Reading out the words on the page when the page in question is their online banking may be a breach of the data protection act.

# 4. Method of Approach

## 4.1. Agile

The Hiccup Virtual assistant was developed in an agile environment. The Agile manifesto was created in February of 2001, and laid out a set of core values behind agile development. As described by Atlassian, “Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.” (Atlassian, n.d.)

This project was developed within a set of two-week long sprints, with each sprint adding functionality to the software and progressively building it up with back-end improvements, front-end functionality, and multi-language integration. Two weeks was decided as the optimal length for the sprints as it allowed for significant development within that time frame, as well as accommodating for potential unexpected issues that may arise.

The initial sprint of the project was dedicated to planning the project. An Office365 planner page was created which details which tasks needed to be finished within each sprint, as well as planning the ‘Research Phase’ of the project, and the project backlog for tasks that were not completed during individual sprints. A project roadmap was also created, for the purpose of visualizing what is detailed in the planner, as well as being a visual aid for the testing side of the application. This is shown by figure 1.

*Figure 1: The project roadmap created during the initial sprint.*

# 5. Project Management

## 5.1 Supervisor Meetings

At the start of development, there were supervisor meetings held every two weeks, in order to keep track of the development of the application, as well as to receive advice on fully outlining the plan of the application, as well as gaining clarity on best practices for development. Within these meetings, there were a small group developing different applications, and each member would detail which stage they are currently at in development, as well as what will be done during the next sprints. These meetings were extremely valuable, as the solution to a problem that one of the other members was having could spark inspiration for different methods of approaching problems with this application. The supervisor meetings were also useful to gain a broader knowledge of the different technologies that may be used within this application, such as the implementation of F# for its powerful pattern matching capabilities.

# 6. Functional Requirements

The functional requirements for the application are listed here, there are four columns, one being the label of the task, the user story for the requirement, the functional requirement, and the priority level, which is between 1 and 5, 5 is the highest priority level, and 1 is the lowest.

## 6.1 Functional Requirements

*Table 1 – Functional Requirements*

|  |  |  |  |
| --- | --- | --- | --- |
| Task | User Story | Requirement | Priority |
| Speech Recognition | As a user, I want to be able to speak to the application, and have it understand me. | The application must have speech recognition functionality by pressing ‘Listen’. | 5 |
| Speech Synthesis | As a user, I want to hear a spoken response to my commands. | The system must have speech synthesis to do text to speech responses. | 5 |
| Small Talk - Basic | As a user, I want to talk to the system casually. | The system must have responses to basic phrases such as ‘Hello’ and ‘Goodbye’. | 2 |
| Small Talk - Advanced | As a user, I want to talk to the system and hear different responses so it feels more human. | The system must have multiple responses to the same phrases such as ‘Hello’ and ‘Goodbye’. | 2 |
| Text Recognition | As a user without a microphone, I want to be able to type my commands rather than speak them. | The system must allow for typed commands through a text box in the GUI. | 4 |
| UI | As a user, I wish to interact with the system using a mouse and keyboard | The system must have an intuitive UI which allows for manipulation of the system. | 3 |
| Time | As a user, I wish to know what time it is. | The user should be able to ask the current time, and have the system respond. | 2 |
| Date | As a user, I wish to know what the date is. | The system must be able to respond with the current date. | 2 |
| Day | As a user, I wish to know what the current day is. | The system must be able to return the current day of the week | 1 |
| Volume | As a user, I want to know the current volume level. | The system must be able to respond with the current volume level of the device. | 2 |
| Open Application | As a user, I want to be able to ask for an application to be opened and have it open. | The application must be able to open external applications. | 2 |
| Close Application | As a user, I want to be able to ask for an application to be closed and have it close. | The application must be able to close external applications | 2 |
| Internet Search | As a user, I want to be able to search the internet using my voice. | The application must be able to search the internet with the user’s stated query. | 2 |
| Dictation | As a user, I want to dictate what I want written down and have it be written. | The system must be able to convert the user’s spoken sentence to text and write it down. | 2 |
| Reading | As a visually impaired user, I want for the words on the screen to be read out to me. | The system must be able to synthesize the text on the screen. | 2 |
| Timer | As a user, I want to be able to set a timer for a specific time | The system must be able to set timers for a specific time, and make a sound once that time is reached. | 2 |
| Reminder | As a user, I want to be able to set a reminder for a specific time. | The system must be able to speak a sentence given by the user reminder at a specific time. | 2 |
| Name Recognition | As a user, I want for the system to remember what my name is. | The system must be able to ask for, and save the name of, the user. | 1 |

# 7. Architecture and Design

## 7.1 Interpreting Commands

The interpretation section of this project was handled using F# code and partial active listeners for key words in the sentence. Once the interpretation function is finished, it returns a string and a token back to the C# code which would then handle the implementation of what the string does. The string would be used to determine which ‘theme’ of response would be used. For example, token group ‘1’ would be some form of date or time-based response based on what the user said, while ‘2’ would be to do something with an external application such as open it, scroll down on a web page, or search using google. This allows for the code to have stronger structure and would operate faster as grouping functions together means there would be less if-statements to check against compared to having it check an if-statement for every single bit of functionality it can perform.

Using F#, we can use its powerful pattern matching features to set pattern identifiers, for example the ‘Greeting’ pattern identifier would match the input with words such as ‘Hello’, ‘Good Morning’, or ‘Hi’. This is called an active recognizer. Active recognizers act as discriminators that divides the input into different categories. We use this to our advantage here to generate multiple different responses based on these key subjects. Partial active patterns are a special class of single-case active patterns, which can return either ‘Some’ or ‘None’. We use partial active patterns here to verify whether the query passed to the F# code contains specific key words which define specific functions. The main partial active pattern used is RegexContains, which uses Regex’s ‘matches’ function to see if the input query contains a key word. This is used for when the user says or types a full sentence, as opposed to using single words. This is necessary as an average user would rarely say the single word ‘Time’ when asking for what the current time is.

This interpretation method can also be used for an ‘Eliza’-type system. Eliza was a chatbot therapist developed in 1966 at the MIT Artificial Intelligence Laboratory by Joseph Weizenbaum which used pattern matching to simulate a conversation between the user and a therapist. “Eliza was programmed to spot key phrases in its interlocutor’s sentences and plug them in to preformed sentences of its own” (New Scientist, 2017, p. 13) Having Eliza-like functionality in this system would be an effective tool to have the user feel more of a connection with the system, rendering them more likely to continue to use it.

## 7.2 Speech Recognition

The built-in Windows System.Speech library is decidedly outdated, and has significant issues with dictation and has a high word-error rate. For a virtual assistant, having a low word-error rate is extremely important to avoid confusion from the user, and ensuring that the assistant operates at an optimal level. This lack of recognition ability necessitates using an alternative. It was decided that utilizing a pre-made library would be best for the time constrictions associated with the project, so multiple technologies were investigated.

Mozilla have developed an open-source speech recognition API called DeepSpeech, which can be used instead of the .NET speech recognition software. As it is open source it can be used freely and is a significantly stronger speech recognition candidate than the older .NET System.Speech libraries. Reuben Morais states that the “word error rate on LibriSpeech’s test-clean set is 6.5%” (Morais, 2017). Using a pre-made library such as DeepSpeech allows us to focus on building the functionality and not spend a significant amount of development time on trying to build a speech interpretation engine from scratch.

A better solution for speech recognition would be to use Microsoft Azure or Google cloud’s speech libraries for recognition and synthesis, but these libraries are not free. If this project were produced for an organization, these speech libraries would be a very strong choice for the recognition aspect especially, as they interpret what the user says with significant accuracy.

## 7.3 Executing Commands

After setting pattern identifiers for each of the key subjects, we can create multiple possible responses that the assistant can use. Using the ‘Greeting’ example again, responses can vary from ‘Hello’, to ‘Good morning/evening/night’ depending on what time of day it is. To choose these responses, we use a rand() function to select which response to use at random. Using a random number allows the system to have as many variants on the responses as it would realistically need, and having varied responses to the same input makes the assistant seem more human-like and welcoming for the user.

To execute a function from a spoken command from the user, once the F# code has matched the command to the function they want to use, it returns a string which contains the ‘theme’, or subject, of the command followed by a number, referred to as ‘theme’ and ‘number’ respectively. The string is then checked to see if it contains a digit, if it does, then the digit is copied to the number variable, and is removed from the original string. These two values are then passed to the InterpretQuestion function. The InterpretQuestion function finds the function to implement using two nested if-statements. The first if-statement checks for the broad ‘theme’ of the command, be it ‘datetime’, ‘application’, or ‘question’, dictated by the string ‘theme’. If the function is unable to match the spoken command to a function theme, the response will default to “I’m sorry, I don’t recognize that command, please try again.” Within these themed sections, there are other if-statements checking the value of ‘number’, which is an integer, and once the corresponding value matches, that specific function will be executed. An example of this process would be if the command was “What time is it?”, the F# function would match the sentence to the right command, and return ‘theme’ as ‘datetime’, and ‘number’ as ‘1’. The C# code would then find ‘datetime’ and ‘1’ and execute the function, then would return the response.

For opening external applications, the original query made by the user is set to a public string in the responses code called ‘originalQuery’. A substring is then made from this, which is set as everything after ‘open ‘followed by a space character. This substring is used for the fileName variable of the Process object. It allows for simple applications such as notepad, chrome, and the calculator to be opened using just their names. More specific applications and files require the file path to inputted.

For searching something on the internet, the user enters ‘search’ followed by what they want to search. A new process is then opened of their default browser with the Google search result of their query. The default browser is used over a specific application as user’s may have a specific preference, and this allows for their preferences to remain intact while using the virtual assistant. The search engine used is Google, as it is the most common and widely used search engine in the world, “maintaining a 92.47 percent market share as of June 2021” (Johnson, 2021), which allows for its primary use over alternative search engines justifiable, due to restrictions in time permitted for this project.

## 7.4 Possible Extension of Architecture

The architecture is currently designed so that the language assessment logic is modular, the current implementation uses a simple rule-based system to interpret commands. This could be extended to utilize other forms of artificial intelligence such as neural networks for more sophisticated forms of natural language processing or understanding. This would allow for more intuitive pattern recognition, allowing the system to interpret the core intent behind what the user is asking.

The way the architecture is designed also allows for it to be extended for localization for other languages. Each localized response could be placed in an array, with an index value being set based on the language selected in the settings menu. This index value – set as an integer such as 0, 1, or 2, would be used to select the appropriate language response from the array. This would also be the case for the matching process, as each language may have a different word for ‘time’, changing the language index value would also change the words required to match for the appropriate function.

# 8. Project Development Sprints

Retrospectives- What have I learned, what would I have done differently.

## 8.1 Sprint 1

The initial sprint was dedicated to detailing the outline of the application, and the plan for the overall project development stages. A Microsoft planner page was created, which was used to organize the different requirements into appropriate sprints, which will make the development of the project proceed much smoother. A project roadmap was also created, which helps to visualize the overall project stages and can help identify potential issues with the timeframes for development. Also in this sprint, a rough concept of the UI for the application was created, which will be expanded upon in a further sprint. Within this sprint, three development goals were decided upon: Core functionality, which is the minimum viable product for the application, which has basic commands and speech recognition and synthesis; Desired functionality, which has more advanced functionality; and Extended functionality, which contains very advanced functionality and would be able to be operated by a blind or visually impaired person without the need for assistance.

After the development of the plan for the application was finished, research began into important areas for the application such as alternative technologies and their strengths and weaknesses, as well as different external libraries for speech recognition software, as it was decided that developing a speech recognition engine from scratch would take up too much time and resources for the project, when that time could be focused upon developing the functionality instead.

## 8.2 Sprint 2

The main goal of the second sprint was to work on the back-end foundations of the project. Within this sprint, a basic UI was created which contains buttons for speaking, listening, and changing the settings of the application, and the speech synthesis library was integrated, allowing for text to speech functionality from the application.

## 8.3 Sprint 3

Within this sprint, the basic Windows speech recognition library was integrated into the project for the purposes of user testing. The speech recognition did work to a degree, but many words were misunderstood by the system, leading to frustrations. Here it was decided that an alternative library would need to be used, and that the Microsoft Azure libraries would be a significantly better choice for the short-term purposes of testing the application’s functionality.

Fixed responses were also developed, for basic inputs such as single words and short sentences. Alongside these, the UI was improved, with two textboxes now present so the user is able to read what the system is interpreting their speech as, as well as read the response from the application in the second text box.

## 8.4 Sprint 4

The main goal of sprint 4 was to integrate Microsoft Azure’s Speech recognition libraries in to the application. These libraries are used as they are extremely powerful, and have a very low word-error rate while being easy to set up with different input devices.

## 8.5 Sprint 5

The focus of sprint 4 was to develop the code to match the spoken command from the user to the appropriate function within the system. This was done by developing F# code to utilize its pattern matching capabilities. The speech to text functionality would pass the string of the spoken command to the F# code, which would then match the string with the appropriate ‘theme’ for functionality. Within the different ‘small talk‘ themes, multiple responses were developed which would be selected based on a random number between 1 and 10, so that the application now has a variety of different responses it is able to return.

## 8.6 Sprint 6

Sprint 6 was dedicated to removing the Microsoft Azure Speech recognition library, and to look into alternatives to replace the library. The alternative decided upon was the Mozilla DeepSpeech library. Also during this time, the F# code was reworked to return the ‘theme’ of the function, as well as an index number so that the code for the appropriate function can be found in the Responses.cs file. This file contains an indexing system to identify the correct function that does not require iterating through a significant number of if-else statements. Instead, the code iterates through a few if-else statements which match the ‘theme’, and within those theme if-statements, there are more if statements to find the appropriate function.

## 8.7 Sprint 7

The focus of this sprint was to implement the remaining functionality. First, the opening and closing applications functionality was implemented. This involved using the System.Diagnostics library in order to start and stop the appropriate processes. Some processes, such as notepad, chrome, and calc (calculator) do not require a .exe extension in order to be opened, and can be opened simply with the command: “open \_\_\_”. Other files and applications can be opened using the file path where the file is located, which can be done by typing the path into the text box. Also implemented was the search functionality, which uses the default browser that the user uses, and creates a new window – or a new tab if the window is already open for certain browsers such as chrome – with the google result for whatever the user searches. For example, if a user were to write “search University of Plymouth”, a new tab or window would open with the google result for the University of Plymouth displayed.

## 8.8 Sprint 8

This sprint dealt primarily with various improvements from user testing. Different system voices were added, for which the source of the voices are the currently installed voices in windows, and the volume and rate of speaking can also be set. This is all set within the settings form, accessed by a button on the main form. The colour of the main form was also changed from a light grey to a light blue to look more ‘friendly’ for the user. It was decided to use the default System.Speech recognition for the speech-to-text aspect, because while it suffers from a high word-error rate so the accuracy is low, it is extremely lightweight, as it is integrated within the system, so the size of the application is drastically lowered over using a voice recognition system such as DeepSpeech, of which the pre-trained model alone has a very large file size.

Due to the low accuracy of the System.Speech recognition, an alternative method of passing commands to the interpreter was devised. The user can check a check box below the listen button, which changes the method of delivery from using voice recognition to get the command, to typing it within the listening text box, and having that used as the command. This will allow for testing the functionality of the system without having to deal with the frustrations encountered by the speech recognition. The speech recognition can be made more accurate, as it uses the built-in SpeechRecognitionEngine in Windows, so by training the engine within Windows it will adapt to the user’s voice, and become more accurate as a result.

# 9. Sprint Retrospectives

## 9.1 Sprint 1

# 10. Testing

The user testing for the project was made up of 3 different testers, who would be instructed to perform simple commands within the application. Afterwards, they would answer questions about what they thought of the application’s user interface, functionality, and what they believed could be improved to make their experience better.

One such response was, when asked about what the user disliked about the interface, “The user interface could be friendlier, there is too much grey and it’s too dull-looking.” This prompted a change to the background colour scheme of the UI, which was changed from a light grey to a light blue. When asked about the likelihood to use the search function of the virtual assistant, one user responded with “If it was a long question then 8/10 chance, but if it was short I would just open google and search it myself.” This demonstrates how the application would need to be streamlined further, to be seen as more convenient to search over typing the question in to a web browser. One such way to do this would be making the application completely hands-free by implementing a wake-up word, such as “Hey Hiccup”, to trigger the listening function rather than pressing the ‘Listen’ button. When asked if this would improve their experience and make them more likely to use the application over manually searching, all users responded positively.

# 11. End-Project Report

Overall, this project has been a success. There were three project goals set out early in the development of this project. The initial goal is ‘Core’, which is the minimum viable product for the project. It contains the basic features for the system, namely speech recognition; a functioning user interface; speech to text writing; and basic commands such as stating the current time or saying hello. The next goal is ‘Desired Functionality’, which contained all previously mentioned features, as well as the ability to open external processes; web search functionality; text to speech responses; and small talk functionality. The final goal is ‘Extended Functionality’, which uses a neural network to create a Natural Language Understanding system for interpreting commands; learning from the user’s answers; and recognizing the voice of the user so that others cannot control it at the same time. The final result fulfills the goal of ‘Desired functionality’, as it contains all of the features stated in both Core functionality, and Desired Functionality, but does not contain any from Extended Functionality. The reason for not fulfilling the goal of Extended Functionality is mainly due to time restraints. The time it would take for implementing an NLU system would significantly affect the amount of functionality that could be developed for the application, which would ultimately render it redundant to implement, as the purpose of an NLU system is interpreting commands for a wide range of features, so it is necessary to have the functionality to support the more advanced system. For the purpose of this project, the current system of using partial active pattern matching works for the functionality that is currently implemented. This reasoning also applies to the other features of Extended Functionality, as a more advanced system would require more time to implement to a high enough degree of quality.

The speech recognition aspect of the project was a significant problem in terms of time spent on it, as well as the end quality of the project. While the system does have speech recognition, the quality of recognition is extremely subpar. While there are many alternative technologies available for speech recognition, a large portion of them are paid products, and there is a direct correlation with price and performance. As a single-developer project, this is not viable, though as a product produced by a business, the paid technologies would be the best solution.

# 12. Project Post-Mortem

## 12.1 What did go well?

The command interpretation and execution aspect of the project went very well. At the start of the project, the unfamiliarity with using the F# language was worrying, but ultimately it resulted in a fast and powerful method of interpreting the spoken command from the user. The system for processing which functionality to use also went well. Using a theme with an index number allowed for less time to be used checking each if-statement, and by narrowing down the number of initial if-statements using the theme, it resulted in a faster processing time. It also allows for the system to be expanded upon in future development, as a new command can be implemented more simply as the structure of the interpretation and execution aspects is clear.

Utilizing the agile project management methodology within the project was also beneficial towards the development of the project. Using agile, any functionality left unimplemented during a sprint could be placed on the backlog and brought into a future sprint, allowing for a more flexible approach, which is essential for a single developer project.

## 12.2 What did not go well?

While the project largely went to plan, there were several issues which resulted in significant losses of time, resulting in the loss of several low-priority pieces of functionality. While these would result in a stronger project, luckily the development goals created at the start of development meant that these losses did not massively impact the overall plan.

Early on in the project it was recognized that the default Windows System.Speech libraries would not be viable due to their high word-error rate. An alternative was decided upon – the Microsoft Azure Cognitive Speech services, but it was not foreseen that the free plan that they provided would be too limited in terms of usable testing time, as well as only allowing the free plan to be used for one month. This was a major loss in time, as the speech recognition system had been focused around using these libraries, and the system had to be reworked from the ground up, and an alternative solution needed to be used. The issue with this alternative, however, was that it was significantly more complex to implement over the Microsoft Azure system, which also cost valuable time.

User testing was another area which did not go well. Due to the coronavirus pandemic, there was a very small user pool available to test the program. This, coupled with the issues with the speech recognition functionality costing a large portion of the time, led to a massive reduction in the amount of user testing that took place. The goal of having user testing take place after each sprint, as demonstrated in the project roadmap, was not achieved, instead having a small amount of user testing being performed in the last stages of the project.

## 12.3 What would be done differently?

The issues with the Azure speech recognition service would be recognized much earlier if this project were to be started again, and a new system would be implemented. This would result in a significant amount of development time being made more readily available, allowing for the development goal of ‘Extended Functionality’ to be achieved. The development of the project would also be started earlier, as while that would reduce the amount of time available during the research phase, it would allow for a wider range of functionality, resulting in a more flexible system which would be able to assist more in day to day use of the user’s computer.

With the lost time being recovered, more time would be spent on user testing. The tests would take place after each sprint with the opinions of the testers influencing each development sprint, allowing focus on what works well and what doesn’t. This would have lead to a stronger overall project, as these improvements would allow for a reduction in wasted efforts on ultimately redundant pieces of functionality or front end aspects.

## 12.4 Potential Extensions of the Project

One such piece of functionality would be a system resembling the Eliza chatbot, which would allow for the user to have a basic conversation with the virtual assistant. This would enhance the user experience, as being able to communicate casually with the system would make a user want to use it more often.

Another piece of functionality that would be implemented is using a neural network to interpret the spoken commands from the user using natural language understanding, as opposed to the current system which utilized hard coded key words recognized with active partial pattern matching. This would render the system able to interpret the intention behind the command, rather than just the exact wording given. This would make the system significantly better at understanding casual language, and would make using the system easier for the user, as they do not have to follow the rules of saying the exact command to utilize a specific piece of functionality.

Another extension of functionality for the project would be having the system constantly listen for a ‘wake up word’, much like many digital home assistants use today. This would make the system much more usable for visually impaired people, as they would not need to press the ‘Listen’ button in order to activate the speech recognition. This would come with it’s own potential issues, however, as if the system has a false positive for the wake up word it may misinterpret what the user is asking for. Ultimately, while this would require a large amount of development time, it would make the use of the system almost entirely usable with just the user’s voice, making it viable to be used by visually impaired people without the need for outside assistance.

These improvements, combined with a built-in screen reader for vocalizing the results of internet searches, opening of applications, and reading dictated sentences, would fully realize the goal of making this system a fully functional piece of software utilizable for the visually impaired.

# 13. Conclusions

This project was developed for general users, as well as the visually impaired.

# 14. References

Atlassian, n.d. *What is Agile?.* [Online]   
Available at: https://www.atlassian.com/agile  
[Accessed 2021].

Bronstein, M., 2020. *A more helpful Google Assistant for your every day.* [Online]   
Available at: https://blog.google/products/assistant/ces-2020-google-assistant/

Heater, B., 2018. *Alexa now works with 20,000 devices.* [Online]   
Available at: https://techcrunch.com/2018/09/01/alexa-is-now-available-on-20000-devices/  
[Accessed 2021].

Johnson, J., 2021. *Worldwide desktop market share of leading search engines from January 2010 to June 2021.* [Online]   
Available at: https://www.statista.com/statistics/216573/worldwide-market-share-of-search-engines/  
[Accessed 2021].

Microsoft, 2021. *Text to Speech.* [Online]   
Available at: https://azure.microsoft.com/en-gb/services/cognitive-services/text-to-speech/#overview  
[Accessed 2021].

Morais, R., 2017. *A Journey to <10% Word Error Rate.* [Online]   
Available at: https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/  
[Accessed April 2021].

New Scientist, 2017. *Machines That Think.* s.l.:New Scientist.

Stewart, T., 2019. *Almost a quarter of UK households are home to a smart assistant device.* [Online]   
Available at: https://mobilemarketingmagazine.com/uk-households-smart-assistant-ey-digital-home  
[Accessed July 2021].

Vailshery, L. S., 2021. *Number of digital voice assistants in use worldwide from 2019 to 2024 (in billions)\*.* [Online]   
Available at: https://www.statista.com/statistics/973815/worldwide-digital-voice-assistant-in-use/  
[Accessed July 2021].

# 15. Appendices

## Function pseudocode

**Function List**

Initialize

Using System.Speech.Synthesis;

SpeechSynthesizer synth = new SpeechSynthesizer();

synth.SetOutputToDefaultAudioDevice();

synth.Rate = -2; - This will need testing to find a good rate (-10 to 10)

synth.Speak(“Welcome”);

SpeechRecognizer speechRecognizer = new SpeechRecognizer();

https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/december/voice-recognition-speech-recognition-with-net-desktop-applications)

**Command List**

You just said “WORDS”, is that right

**Basic Questions**

What time is it?

Time = System.time();

synth.Speak(Time);

What is the date?

DateTime = ToString(System.DateTime());

synth.Speak(DateTime);

What is the current volume level?

AudioSwitcher.AudioAPI.CoreAudio

CoreAudioDevice defaultPlaybackDevice = new CoreAudioController().DefaultPlaybackDevice;

Volume = (“Current Volume:” + defaultPlaybackDevice.Volume);

synth.Speak(Volume);

Set volume to x%

SetVolume = ToInt(x)

defaultPlaybackDevice.Volume = Setvolume;

**Open and Search**

Open x application

Load x;

synth.Speak(“Application Opened”);

Close x application

Close x;

synth.Speak(“Application Closed”);

Search Google for x

Load browser;

Search x;

**Reading and Writing**

Read this to me

ToRead = (Highlighted Text);

synth.Speak(ToRead);

Dictate for me

While(CurrentWord != “Stop”){

CurrentWord = InterpretSpeech();

Write(CurrentWord);

}

**Small Talk**

What is my name?

If(UsersName = “”)

TextToSpeech(“I don’t have a name set up for you, would you like to do it now”)

Answer = InterpretSpeech();

If(Answer = “Yes”){

TextToSpeech(“Okay, what is your name?”)

UsersName = InterpretSpeech();

}

Else{

End;