DUBLIN INSTITUTE of TECHNOLOGY
Institiúid Teicneolaíochta Bhaile Átha Cliath

AAC Using Natural Language Processing
Final Year Project Report

DT228
BSc in Computer Science

Mark Naylor
John Gilligan

School of Computing
Dublin Institute of Technology

06/04/2016

Abstract

AAC includes all forms of communication, other than oral speech. One such method is using pictorial symbols that represent words and phrases. AAC users can have issues with the variety of vocabulary and speed of communication allowed by existing solutions. The project aims to take natural language processing techniques and apply them to aided augmentative and alternative communications (AAC) systems to increase the speed of user's communication and the amount of vocabulary that they can access.

The objective of this project is to provide a mobile application that uses n-gram modelling to allow AAC users to communicate faster and access more language, while also achieving a high level of usability for the target end user. The application will present boards of these symbols to the user and speak the word or phrase the symbol represents when the user clicks it. It will also present a prediction of the user's next word in the form of a symbol, using by n-gram modelling.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

_____

Mark Naylor

03/04/2016

Acknowledgements

I would like to thank my supervisor John Gilligan for his advice and support with this project. I like to thank my second reader Damian Bourke his encouragement and support, as well as Orlagh Gregory for her feedback and input across the development of the project. I would also like to thank my family and friends for all their support throughout the year.

Table of Contents

Table of Figures

1 Introduction

1.1 Project Overview

The application is aimed at users of aided augmentative and alternative communications methods (AAC). AAC provides people with little or no speech a means to communicate and function in their daily lives. It will present various boards of symbols to the user that can be tapped to form sentences and it will speak the words and phrases that the symbols represent on clicking them, using text-to-speech. When one or more symbols are selected, the application will present a suggestion for the next symbols the user may wish to select. The application will be developed for iOS devices using Swift and Xcode, as research has shown iOS is the platform of choice for accessibility-focused users.

The suggestions will be based off the user's usage history, so it is learning from their use, as well as statistical analysis of a text corpus. The prediction from the text corpus will be configured so that the language is filtered and only words that are compatible with the users reading comprehension level are predicted. N-gram modelling will be used to produce the predictions and analyse past usage history. The project will be designed and developed with a user-centric design approach and also conform to standards set by existing AAC mobile applications in terms of the graphical user interface (GUI).

1.2 Project Objectives

To produce a design that meet users' needs and is accessible to users. Produce a system with a high level of usability for the target end user. Enable users to communicate faster, and with access to more vocabulary than they would have with existing solutions. To show that prediction can help AAC users communicate faster. To make an application that is easy to use and configure so that new boards can be easily added and customized to introduce new language to users.

1.3 Methodology and Rationale

A community project option was made available for final year students. A parent, Orlagh, whose daughter was diagnosed with Dyspraxia, Dyslexia and Autism, gave a talk. She detailed some of her daughter's struggles in everyday life, and in particular with communication. She demonstrated AAC software that she was using that was not well designed, was difficult to customize, and did not perform well for their needs. Her main concerns were around usability and configuration. The idea of using prediction to help her daughter access more symbols without the visual clutter of them always being on screen was one suggestion discussed during the meeting. This meeting, as well as many follow up meetings, have shaped the idea behind the objectives of and the design of the project.

A user-centric design approach was adopted for developing the project so that the development of the project was focused on meeting Orlagh's needs and solving her existing problems. Usability was identified as an important factor. User-centric design

is a process where the end users needs, wants and limitations are a focus during all stages of the design and development lifecycle. Emphasis is placed on how the end user wants to use a product as opposed to forcing the end user to change their behaviour to use the product. This methodology was chosen because the end user of the system has special requirements and this approach enabled Orlagh's active involvement and input to iteratively refine the system requirements and design. This was carried out so that the system would achieve a high level of usability for its target end user and all design decisions were taken with the user in mind and input from the user.

1.4 Document Structure

Chapter 2 Background Research: This chapter includes all the background research performed in order to gain more knowledge in the area.

Chapter 3 Design: This chapter goes into more detail on the methodology chosen and also details the design produced as a result of the work, and what contributed to this design.

Chapter 4 Architecture & Development: This chapter describes the development phase of the project, including the development of the server, the client-side iOS application and all software involved in the project.

Chapter 5 Implementation & System Delivered: This chapter provides a description of the final system delivered and an evaluation of how this system meets the requirements.

Chapter 6 Project Plan: This chapter describes how the project changed from the initial proposal and why.

Chapter 7 Conclusion & Future Work: This chapter discusses what future work could be carried out on the project and what ideas were discussed but not implemented.

2 Research

2.1 Overview of AAC

AAC are strategies for people who are non-speaking or have speech disabilities. AAC provides people with little or no speech a means to communicate and function in their daily lives. AAC can be aided or unaided. Unaided includes sign language and body language. Aided AAC varies greatly from paper communications boards with symbols to complex voice generating software (American Speech-Language-Hearing Association, 2015). Augmentative communication is a method to supplement natural speech when it is not sufficient. Alternative communication is an alternative for those who are non-speaking (Grandbois, 2012).

Some terms that are important in relation to AAC are speech, language and communication. Speech is the verbal means of communicating. It is motor movement where the communicator coordinates muscle movements in specific sequences to produce sounds. Language is a cognitive skill. It is a shared rule or symbol set. It includes vocabulary, grammar and many other rules. People communicate using language without speech daily, e.g. sign language, email and text messaging. Communication is the exchange of information using some medium such as speech, using a mutually agreed rule set, a language (Grandbois, 2012).

Users of AAC include people with intellectual disabilities, cerebral palsy, Parkinson's and motor neuron disease, among others (American Speech-Language-Hearing Association, 2016). In Orlagh's daughter's case, it is Dyspraxia, Dyslexia and Autism. Dyspraxia is a difficulty with thinking, planning and carrying out sensory or motor tasks. It has different effects on different people, as does Autism. This is why Autism is referred to as a spectrum disorder. In this case, her daughter can understand language that is spoken to her but cannot express her thoughts using the motor movement of speech. She also has reduced ability with the cognitive skill of language compared to other children of the same age, due to Autism and Dyslexia. Overall, this has a big impact on her ability to communicate.

2.2 Types of AAC

There are different ways that people access AAC: by direct selection using a finger, for example, or by scanning where different objects on an interface will light up and the user will make a selection with a switch or button. There are also eye gaze products that allow a user to make decisions with eye movements (RockyBay.org, 2016). Health professionals carry out an assessment and recommend a type of AAC based on a person's capabilities and needs.

Orlagh's daughter has the motor skills for direct selection and uses a tablet device already so the project is focused on that method of operation. She uses single-meaning picture based systems, as opposed to alphabet systems so the project is focused on creating a single-meaning picture system. This means that each symbol on a board will represent one word or short phrase as opposed to the user being presented with a keyboard in alphabet systems or semantic compaction systems where the user has to

use more than one pictorial symbol to form each word they wish to communicate (RockyBay.org, 2016). Semantic compaction systems are faster, but require more training and cognitive skill to use, whereas single-meaning picture systems offer access to more vocabulary (AAC Language Lab, 2008).

2.3 Goals of AAC

AAC users in general use AAC to compensate for a lack of functional speech. However, this should not limit their access to language, which is a cognitive skill as opposed to a motor skill. The main goal for AAC users is Spontaneous Novel Utterance Generation (SNUG). This means the user should be able to initiate the communication and the communication should not be scripted or predetermined by the medium. The user should be able to say exactly what they want. This requires access to core vocabulary; the few hundred very commonly used words, as well as fringe vocabulary; the many other seldom used words. It has been shown that the majority of our language is made up of a small number of core words, for oral communication two hundred words can make up as much as eighty percent of language (Cannon & Edmond, 2009). Along with SNUG, the goal of AAC, as with any communication, is to allow a user to communicate as quickly and effectively as possible (Grandbois, 2012).

The single-meaning picture approach revolves around having the user learn the core vocabulary symbols to express whatever they want to say. Some fringe vocabulary that is relevant to certain contexts might be also added to some of their communications boards by a parent or language therapist. They should then be able to put together a large number of expressions from a small number of symbols. Issues arise when they cannot find a suitable symbol or cannot navigate through the interface, or when no symbols exists to express what they are trying to communicate. This will be explained in more detail later when reviewing existing solutions.

2.4 Importance of AAC

Communication is very important for social interaction and forming relationships. It has been shown that social closeness is the purpose of communication (Light, 1988). AAC allows user to greet others, ask questions when they would like to know something, tell stories and much more (Grandbois, 2012). A big issue for Orlagh, that she expressed frustration with many times, is that her daughter can often not tell her about her day at school or any other time she is not with her, due to the lack of options she has to express her thoughts and the frustration she has with using the poorly designed tools she currently has.

2.5 Symbols

The application will require a library of the single-meaning picture symbols as mentioned earlier. In meetings with Orlagh, these were discussed and she stated that the SymbolStix library from n2y was her preferred choice. The developer contacted SymbolStix and requested use of their library for the project. Ms. Anne Johnson-Oliss, Director of Symbols at n2y, responded and a Skype meeting was held to discuss the use of the symbols.

Ms. Johnson-Oliss explained that SymbolStix is the largest library available with over thirty thousand symbols and that they employ eight graphic designers who are constantly adding new symbols. She stated that their symbols are the most child-friendly, and research has shown that many non-profit organisations agree and recommend them, such as the American Speech and Language Association (McClure and Rush, 2007). She granted free access to SymbolStix Prime, a search engine that allows a word or phrase to be typed in and gives the option to download all related symbols for that word, for the length of the project. She explained that if the application were to launch on an App Store that a commercial agreement would have to be signed and then the project would have full access to all their symbols and associated metadata and allow them to be preloaded into the application.

Since SymbolStix are Orlagh's preferred choice, and are more suitable for more children, they were selected as the number one choice of symbol library. A full fringe vocabulary does not need to be built in. The project will not require a library the size of the SymbolStix, so it can proceed even though access was not secured to the SymbolStix developer metadata, or the full symbols library. A big and time-consuming disadvantage in this is that symbols have to be manually downloaded from the SymbolStix web site, instead of taken from an API or preloaded, as they would be for commercial projects with full access.

2.6 Existing Solutions

2.6.1 Overview

There are many existing AAC applications available for tablet devices for single-meaning picture based AAC systems. Given that this is the type of AAC application that this project is focused on, due to Orlagh's requirements, these reviews focus on the application's features for single-meaning picture AAC users. Many of them target other users like eye gaze users with different features as well.

Traditionally, this software ran on dedicated purpose-built devices. With advances in technology, cheaper and more readily available devices like Apple's iPad can now run the software. As shown by McNaughton (2013), the iPad has brought AAC to mainstream technology and increased its acceptance. Tablet devices are required for their screen size. The applications are available on Apple's App Store for their devices and the Google Play Store for Android devices. Some free AAC applications have very small and limited features sets. The paid applications have more features but are expensive. Two solutions include predictive features based off a user's usage history, Proloquo2Go and AACorn.

2.6.2 AACorn Application

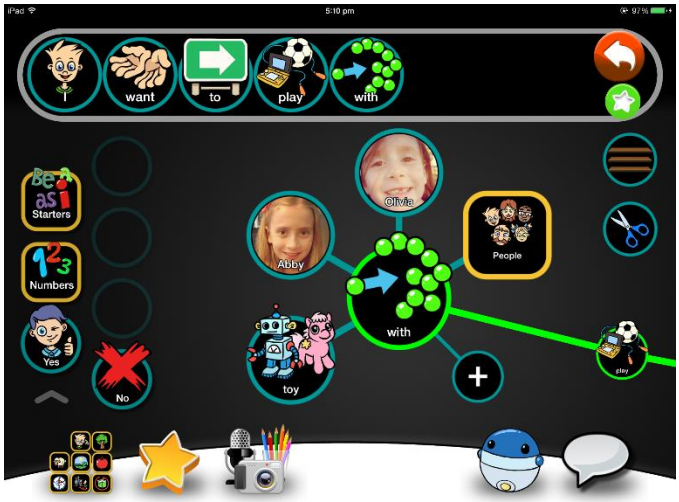

Figure 1 Screenshot from the AACorn app

AACorn, pictured above running on an iPad, is an application with a similar goal to this project. It learns from usage history to anticipate the next word a user may wish to select and presents predictions to the user in a web shape as seen above. The sentence will accumulate in the same place and the clear button of this application will be placed to the right of the sentence. It uses a dark background with bright coloured interface elements for a high contrast user interface (Smart Apps For Kids, 2013).

The application was developed by a parent whose daughter had a brain condition. The motivation for the application was to supplement various AAC tools that already exist but to offer something different by using prediction. This quote from their website summarizes the founders sentiment; “Our suspicion was that for many children - especially those who are pre-literate or have more severe cognitive issues - using systems that require hunting and pecking through grids and folders while certainly very helpful for lots of adults is far from ideal for children - not just because the words are hard to find, but because quite often many children don’t even know what they are looking for!” (AACorn AAC, 2016). This is a sentiment that was also echoed by Orlagh throughout several meetings and describes the goal of this project very well.

A big difference with how AACorn is laid out in the above screen to how this project will work is that the symbols change position with each click. This is in conflict with other solutions that keep symbols in the same place all the time so that users can remember their positioning. On installing the application, it does not have any prediction. It records users use and predicts the next word of a sentence, so that when a user selects a word it predicts words that previously followed the selected word. When a symbol is not featured in the predictions, the user has to add it to the predictions that should follow that word. This is a downside to using the application as it conflicts with the goal of SNUG. It requires learning and configuration. It costs two hundred dollars to purchase one license of AACorn.

2.2.3 Grid 3

Figure 2 Screenshot of the main screen

Grid 3 runs on the Windows operating system. It is one of the most advanced AAC applications available in terms of its feature set. It provides prediction based on a user's usage history. Control buttons are placed on the right hand side of the screen as seen above. It uses this home screen of categories that can be clicked into to enter into a board containing symbols. This method of navigation is similar to Orlagh's existing software and is something she expressed frustration with, as did the developer of AACorn, as mentioned earlier. There are different boards of symbols, and the user or a person helping them has to enter and exit boards to find the words and phrases they wish to select.

It has a mode where words change tense and from singular to plural based on the previous words selected, which is a unique feature. The quick talk button is always available by being placed on the right-hand side among other control buttons like delete and clear. It offers a library of different voices in different accents. An excellent feature that it performs is highlighting each word and symbol as it is being spoken by the application. It also offers a plethora of customization options, allowing the user to create new boards and add symbols to them. It allows users to share their Grids with other users of the same application. Grid 3 also allows vocabulary to be customized to varied levels of literacy and language and access (Think Smart Box, 2016).

2.6.3 Proloquo2go

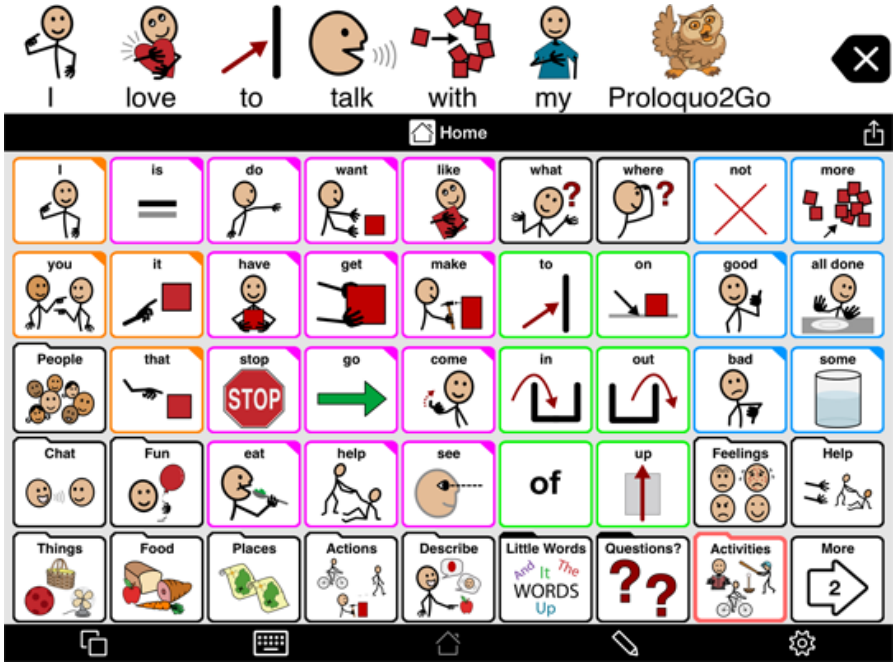

Figure 3 Sample board of Proloquo2Go

Proloquo2Go is an award winning AAC application. It costs two hundred and fifty euro to purchase on Apple's App Store. It includes a very large feature set and a lot of customization options. It allows a user to customize the grid size of the application. It allows over forty voices to be downloaded for a more natural voice tone for the user, for example boy or girl voice tones. It allows users to customize their vocabulary to their reading age and their board sizes to match their ability to process information. It features the SymbolStix symbols library built in, which contains over twenty thousand symbols in total.

Proloquo2Go, like other AAC applications, colour code different types of language, for example verbs or nouns have a coloured border that is consistent for all words of that type. It allows the creation of new symbols and boards from within the application. Proloquo2Go supports sharing sentences to messaging applications or emails. It has support for multiple users and multiple languages. Proloquo2Go also uses a folder structure for organising symbols and different parts of language (AssistiveWare, 2015).

2.6.4 Avaz

Figure 4 Screenshot of the Avaz App

Avaz, pictured above, is another AAC application for mobile devices.
It has a slightly different interface to Grid 3 and Proloquo2go, and places the colour coding of language behind the symbols instead of bordering them. A gap is also left between each symbol on the board, as opposed to each border being directly next to the following symbols border. It also uses a dark coloured background which, combined with the coloured background of each symbol, provides much greater contrast and clarity (Avaz, 2015). Outside of the user interface it offers the same feature set as the previous apps, such as customization and multiple text-to-speech options. It cost one hundred and sixty seven euro for Apple Devices (Avaz – AAC App for Autism, 2015).

2.6.5 TalkTablet

The final popular existing solution reviewed is called TalkTablet. It is the cheapest of the four at sixty euro (TalkTablet - Speech/AAC app, 2015). TalkTablet uses a similar interface to Proloquo2go. It also uses the same symbols library. Symbols have colour coding around their borders as well. One distinction that TalkTablet has is that it runs on multiple platforms including iOS, Android, Windows and Amazon Kindle. It offers many of the same features as the other apps as well, like customization, allowing the creation and resizing of boards. It offers an extra feature, allowing users to share custom symbols they create with other users of the same application (TalkTablet, 2015).

2.6.6 Other Software Designed for Disabled Users

In order to get a better understanding of designing software for intellectually disabled users, some exisiting software was reviewed in addition to the AAC software mentioned earlier, from a design perspective. These include Waybuddy and Look at

Me. Each have different features that simplify the interface with the aim of making the app more accessible for intellectually disabled users.

Figure 5 Prototypes of Way Buddy

Way Buddy, pictured above, is an app that is being produced to help intellectually disabled users to travel on foot. Way Buddy is being produced by an Irish developer with input from intellectually disabled users, meaning it is also a user-centric design project. It has a very simplistic interface. Each screen has a maximum of three or four buttons. Font sizes are very large (Boland, 2014). Each button is depicted by an icon that mimics the action it performs. The colour scheme is consistent throughout and buttons are large and have a high contrast against background colours (Way Buddy, 2015). This project will look to emulate the large buttons and high contrast that Way Buddy achieved with its design.

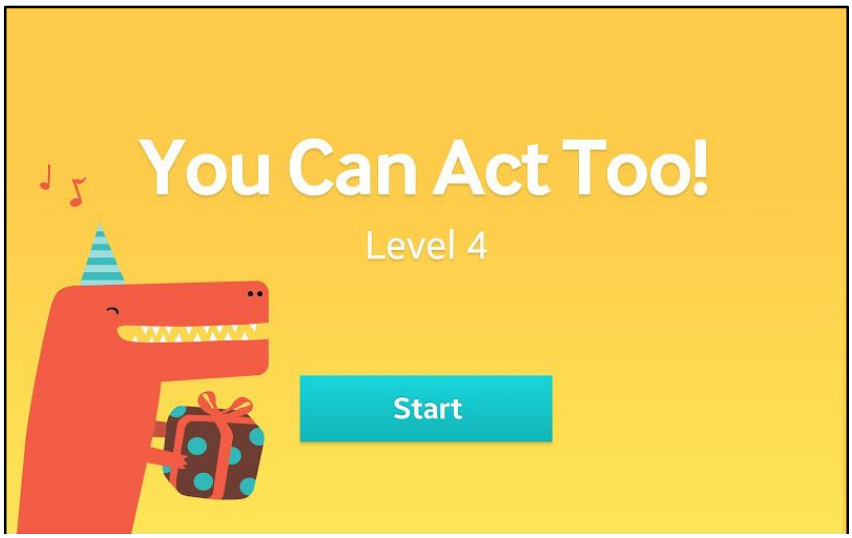

Figure 6 Screen from the Look at Me app

The Look at Me app, screenshot above, is another app designed to help people with intellectual disabilities. It comes from the Samsung backed Look at Me project and was developed with the help of many experts in the field and health professionals (The Look at Me Project, 2014). The app aims to help children with autism to improve their ability to make eye contact, which can be challenging for them. The app presents challenges to the user to line up their eyes with the targets on screen using the devices webcam to track the user's eyes. Again, this app also uses very large font sizes and large buttons. It uses a simple colour scheme with buttons that contrast

against the background colours (Samsung Look at Me, 2015). There are a very small number of elements in the user interface on each screen.

2.6.7 Summary

All of these apps have feature parity, as described throughout, and similarities in design. These features were all considered as features for this project. They do have some design choices that distinguish them from others, however. A problem with these solutions that this project aims to achieve is that they do not allow users to access new vocabulary. The applications that offer prediction only offer prediction from past usage. The symbols on the communications board are fixed, unless a parent or therapist alters it, and in many cases this is not easy. This limits or does not allow SNUG. Another issue, as described by the ACCorn founder and Orlagh, is that navigation is difficult for young or disabled users, who find a folder structure confusing and complicated to use.

2.7 N-Gram Modelling

2.7.1 Overview

The prediction feature of the application will be based off n-gram modelling. The aim of the application is to accurately predict the next word in a sentence for a previous word or words, except the word will be displayed as a symbol. With this in mind, the examples explained here refer to grams as words, even though they are often letters for other uses of n-gram modelling. A statistical approach was researched to do this. An n-gram model is a type of language model that will be used for predicting the next item in a sequence of text. The goal of language modelling is to assign a probability to a sentence, sequence of words, letter or an upcoming word. This has many uses including spell correction, speech recognition, natural language generation, machine translation and, in this case, text prediction. An n-gram model uses the previous n-1 words in a sentence to predict the next word (Jurafsky and Martin, 2014).

2.7.2 Background

N-gram models are based around mathematics probability principals. An IBM employee, Trim (2012), who are a pioneering company in this space with their Watson Artificial Intelligence engine, explained in a blog post the Markov assumption, an example of such a principle. It is the presumption that the future behaviour of a dynamical system only depends on its recent history. In particular, in a Markov model, the next state only depends on the most recent states. This principal will be applied to generate prediction. The application will use a user's recent use to generate the predictions.

Conditional probability is also used in n-gram modelling as explained by Jurafsky and Martin (2014). This is calculating the probability of event, like a word occurring, given another event, like the last word entered. The chain rule is used where the probability has to be calculated over a number of words. This means that if the probability of the third word to be added to a sentence is dependent on the probability of the second word to be added, and the probability of the second is dependent on the

probability of the first word, then the probability of the third word is dependent on the first word in that sentence.

2.7.3 Corpora

Construction of a language model is based on counting words in a corpus or corpora, a collection of text. The probabilities of an n-gram model come from the corpus that it was trained on. The model is therefore dependent on the training corpus. The corpus used to train the model should be representative of the application that the model will be used for. The corpus needs to be large enough to produce enough n-grams. It also needs to be general or specific relative to the needs of the application. For my application, it will need to be appropriate for the reading level of the user and general enough so that it can be used in any setting.

When n is one, this is called a unigram. This model assigns probability to the occurrence of single words. When n is two, it is called a bigram. The probability of a sentence when n is two is the conditional probability of its bigrams. This means the probability assigned to the next word will rely solely on the previous word. In the case of a trigram, when n is three, it will rely on the previous two words, and so on. With a larger value of n, a larger text corpus is needed to train the model.

2.7.4 Issues

There are issues with using an n-gram approach for prediction. One such issue is assigning a probability to a sentence that hasn't occurred in the training set. An n-gram model can be smoothed or unsmoothed. A unsmoothed or simple model would count the occurrence of words and assign a probability based on a words share of the total count of all words, for example, if the word 'the' occurred five times in a sample of one hundred, the probability a word would be 'the' is 0.05. This simple approach can be sufficient when unigrams are being examined, but when n is a larger number, the number of unseen events will be much larger than the number of observed events (Hockenmaier, 2012).

To combat this, the probability of previously seen events is decreased slightly, so that a probability can be assigned to unseen events. This is called smoothing. There are many methods of doing this. As explained by Jurafsky and Martin (2014), one such method involves assuming every event occurred once more than it did in the training data, called add-1 or Laplace smoothing. This model can be problematic in that it can assign too much probability mass from seen events to unseen events. There are other methods that try to solve this problem as well. There are also other smoothing algorithms such as Good-Turing and Kneser-Ney. These more advanced algorithms look at the count of events that happened, once to use this probability for events that never occurred.

2.7.5 Conclusion

Since the training material is important and influences the predictions produced, this project requires training sets that align with the reading age of the end user. The reading comprehension level is provided as a configurable option in the applications settings. The corpus used for n-gram generation is based on children's conversations

and writings, so that the predictions are more relevant for the target user. The n-grams are filtered against reading lists of words for each comprehension level. This will help the prediction be more accurate and appropriate to the language skills of the user. The user's usage history is also stored, and added to the applications training material so that the application's prediction is more tailored with time.

As Hockenmaier (2012) explained, the best language model is one that performs best on an unseen test set. A good language model is one that assigns a high probability to the next occurring word in the test corpus a high percentage of the time. The application taking both the users use as well as the corpus analysis should ensure it performs better over time and also gives some predictions that could introduce new symbols and language to the user.

2.8 Technologies Researched

2.8.1 Mobile Application

The first step in the research of technologies was to choose a platfrom to develop the client appplicaiton for. Both native and hybrid mobile applications were examined. The first option examined was whether to develop the app native for one platform or using a tool to publish it for multiple platforms. The first step was to review some of the multi-platform app development tools to see what they offer. There are multiple hybrid app development environments to choose from. There are both free and paid options. Two of the most popular are Phonegap and Xamarin.

Xamarin is the first hybrid option examined. Xamarin provide a free starter edition that allows development in C# for iOS, Android and Windows Mobile. C# is a language that is already familiar to the author so this was a big advantage. A downside to their starter edition is that it does not allow calls to third party libraries. It also limits the developer to small apps. To use Xamarin without restriction, a fee of twenty-five dollars a month would have to be paid (Xamarin, 2015).

Phonegap is the second hybrid option reviewed. Phonegap supports building apps with HTML5, CSS and Javascript for all platforms (Kohan, 2015). While these are powerful technologies, the developer has not used Javascript much, so this would have been a downside to choosing it. One big advantage that Phonegap has over Xamarin is that it is open source and free, as it is based on the Apache Cordova framework. They have other strengths and weaknesses as well, for instance Xamarin is better for native-looking UI performance for each platform, but Phonegap is better for non-native shared UI across platforms (Nel, 2014).

The next step was comparing developing using these solutions to developing natively for a platform. When using hybrid tools, the developer is dependent on the tools and framework to be up to date with each platform. Native platforms offer better tools for testing and debugging, which will be useful given the developers lack of experience in the area. Native apps provide better performance and a better experience for the user interacting with the interface (Nel, 2014). The user interface is a very important area of the project, the application needs to provide the best user experience possible.

Hybrid solutions are cheaper and faster when supporting multiple operating systems, but this is not an important requirement for the project as there will not have time to test and debug for more than one platform. The speed advantage is also negated by the fact that the developer will be learning these platforms, whether native or hybrid, for the first time. Due to the importance the project will have in a user's daily life should it achieve its goal, I think it is very important that the application performs well and is responsive and fluid, so for these reasons native development was selected over hybrid deveopment. This, along with the need for a constant internet connection, is also a reason why a web application is not suitable for the project.

Next, mobile operating systems were examined to decide what will be best to develop, for. iOS has a larger market share for tablet devices. According to StatCounter (2015), a global analytics firm, iOS commands sixty five percent of traffic on the internet from tablets, while Android accounts for just over thirty percent, for the period from January to August 2015. The other important factor is which of the two operating systems is better for accessibility. Apple's iOS is the leading OS in terms of accessibility. Google has added many features to Android to catch up, like colour inversion in Android 5.0, but it was long after Apple implemented them (Google, 2015). Apple pioneered many accessibility features for mainstream mobile operating systems, like zoom, which allows a user to enlarge elements on the devices screen. As can be seen on Assist Ireland's website, iOS has a larger feature set for disabled users (Assist Ireland, 2015). iOS is more established in the classroom and there have been studies in using iOS in a special needs setting (Campigotto, 2013). There have been no such studies of using Android in a classroom environment. iOS is the more popular platform where accessibility is concerned (Flewitt, 2014).

Native iOS development has to be done in the Objective-C or Swift programming languages, which are proprietary to Apple. The developer has no experience with these languages. Native Android development, on the other hand, is done in Java. The developer does have experience working with both Java as a language and Android as a platform. Weighing up all the options, the drawback of having to learn new languages for developing for iOS does not outweigh the benefits of it being the platform of choice for accessibility. The developer owns an Apple laptop and has access to an iPad tablet for testing, so all the tools required are at hand. Orlagh also mentioned in her presentation that she owns and her daughter uses an iPad on a regular basis, so it is also a good fit from that point of view, since she is the target end user.

Another in relation to the client application was whether to use objective-c or Swift to code the application. Swift is a relatively new language introduced by Apple in 2014. Objective-C is Apple's older language, which they developed thirty three years ago. Swift is easier to read, faster, requires less code, easier to maintain and more approachable (Solt, 2016). The only advantage Objective-C has over Swift is that there are more existing API's and code available due to the length of time it was the default language for development for Apple's ecosystem. However, for this project, there are no Objective-C API's or libraries needed, so Swift is clearly the better choice of language for the project.

The last decision regarding the client application was choosing a data persistence option or options for iOS. iOS offers standalone SQLite, Core Data (an Apple object

relational mapping solution using SQLite), property lists, user defaults and archived objects. Each has its own advantages and disadvantages. SQLite and Core Data are the fastest and most efficient, with the only difference between them being that SQLite scales slightly better, and has better performance when the number of records gets very large. Property lists, user defaults and archived objects all require all data to be read and rewrote on each change. They are not suitable for the applications n-gram data, as the memory usage would be very high. Core data was selected for its ease of use instead of standalone SQLite because of the developer's lack of past experience with iOS technologies (Jacobs, 2012).

Boards, symbols and configuration do not need to be modified as often as the n-gram data and are a much smaller collection of data, so Core Data or SQLite do not need to be used to store those. Property lists cannot store images, so they are not suitable. User defaults require converting image data to binary data and back to the correct image format again on retreival, which introduces complexity if data in mixed formats is retrieved from the database and needs to be converted back to an image object. Archived objects allow all types of data to be easily stored so they were chosen for storing the symbols, boards and settings (Jacobs, 2012).

2.8.2 Client Requirements

The decision to develop for iOS, as explained earlier, was made because it is the lead platform for accessibility and also Orlagh's preference. An Apple Macbook Air will be used to run the Xcode 7 IDE using the iOS 9 SDK to develop the mobile application. An iPad Mini will be used to test the application as well as several emulators available in Xcode for testing different resolutions. Apple devices only run programs that are signed with a developer's certificate issued by Apple (developer.apple.com, 2016). . Apple provides these certificates as part of provisioning profiles for developers who register with them, which cost ninety-nine dollars per annum. For this, Apple will sign a developers code, review their code and then make their application available for download on the store should it pass the review. This would not be an option, however, without signing an agreement with SymbolStix beforehand.

Thankfully, they recently added an exception to this rule, allowing developers to generate a certificate in Xcode, self-sign their code, and manually trust that certificate on testing devices in the devices settings so developers do not have to incur this expense any more (Mayo, 2015). It is something that would have to be considered going forward if the decision was made to distribute the application, however.

2.8.3 N-Gram Software

Following this, the other major decision around technologies to use for the project was an n-gram library. The time constraints of the project mean that it would be very difficult to write all the n-gram software, along with the client and server, and implement all the features required, so therefore an existing library had to be used. With iOS in mind as the client application, the first step taken was to search for n-gram libraries in objective-c or Swift, Apple's programming languages.

The supervisor on the project discovered a very simple n-gram library written in objective-c, made available on Github. Unfortunately, there was very little documentation accompanying it and it did not feature corpus readers or any of the more advanced functionality that comes with well-developed NLP libraries (Doubrovkine, 2014). With no other options of taking n-gram analysis on the device, the next step was to analysize options that could be ran remotely and relayed to the device via a server.

The first library to appear in this search was the Natural Language Toolkit (NLTK), written in Python. NLTK has an accompanying book called Natural Language Processing with Python (Bird, Klein, and Loper, 2009). The book gives a thorough introduction to NLTK and its different use cases, and also a guide to using the toolkit. This libarary had better documentation that all other options discovered, and more online resources. The developer does not have much experience with Python, so other libraries were also considered. The next most well established option examined was NlpTools, which is a PHP library. The developer has experience with PHP, but this did not have much documentation, and is still under development, so the decision was made to use NLTK (Php-nlp-tools.com, 2016).

2.8.4 Server Technologies

With the decision made to use the NLTK library, a server architecture had to be chosen to transport the n-gram analysis from NLTK to the iPad app. NLTK Server was the first option discussed. It is a small and open source project that wraps some of NLTK's functionality by allowing commands that would normally be typed into the Python interperter to be sent to the application using HTTP requests, and sends the data back in the JSON format. This program has limited documentation, however, and does not encapsulate all the functionality of NLTK (GitHub, 2015). It crucially does not support using the Collocations module of NLTK, which is needed for n-gram analysis. Another option examined was having a JSP servlet accept a request from the client application and then connect to the library using Jython, a tool for using Python in Java, but this was a complicated and time consuming process.

Next, using a PHP script with the "exec" call was examined. The reasoning for this was to try to use technology that the devloper has experience with for the server so that the developer did not have to learn new technologies for it as well, and consume a lot of time doing so. This runs any shell script on the server from the PHP program as if it were typed into the commad line by a user, and captures the output. This could be used to call a Python script to generate the n-grams and capture the output. While this is a viable option, the author found that the output was unformatted and it would have required a lot of work to parse the output before transporting it to the mobile application. To get around this problem, the final solution chosen was to generate the n-grams using a Python script and then upload the data to a database on the server. The mobile application can then send a request to the server, which will serve the data in response.

This approach avoids the problems previously mentioned of parsing text or working with time consuming libraries to bridge different languages such as Jython. It also means that requests will be served quickly as the n-grams will already be taken by the server, so it will not have to compile this data each time a request comes into it, it will

already be computed and inserted into a database. Both LAMP and Django were options for the server-side of the project. Django, which is based on Python, would allow the use of NLTK straight from the server-side scripts. It would require the developer to learn to use a fully new server stack, however, where LAMP would not.

The LAMP stack was chosen as the server achitecture for the project. This means that MySQL was chosen to store the n-grams on the server, and PHP was chosen as the server-side scripting language for accepting requests and responding to them. It runs on an Apache server. This is the world's most popular server architecture, and it is free, open source and secure. It is very easy to use and the developer of the project has a lot of experience with PHP and MySQL. It runs on all major platforms, so the application can be tested locally on Apple's OS X and deployed to a Linux server seemlessly. MySQL can handle much more than the required amount of data or number of rows required for n-gram analysis (Steidler-Dennison, 2016).

With the client-side of the application, the decision was made with the end user in mind. In order to focus on the user and dedicate as much time to design and user interface as possible, the back end technology was selected based on ease of use for the developer and the time taken to implement a solution. The client-side technology is all new for the developer so it would have been a large risk to choose to learn a new back end technology as well.

2.8.5 Server Requirements

Initially, for development, a temporary local server is required. The server has to run XAMPP, so it supports web site and database hosting using PHP and MySQL. It also has to run the NLTK library and Python. PHP will be used to develop the web scripts. The PHP scripts will read from the database and output JSON. XAMPP, NLTK and Python are all cross platform, but since the application will be developed for iOS the aforementioned software will all be ran on an Apple Macbook Air on OS X 10.11.

To have the service available over the Internet, a live deployed server is required. The author was provided with six months of free usage of Microsoft's Azure infrastructure as a service platform through the college. Azure allows the creation of virtual machines using several different operating systems, including Ubuntu Linux, which supports XAMPP, Python and NLTK. For this reason, Azure was selected to host the deployed server.

2.8.6 Other Technology Requirements

Git is an open source version control system used to manage changes and allow multiple people to work on one project at the same time. It allows a developer to make changes and then revert back to older versions if the changes cause crashes or break functionality. It works by pushing changes to Github's servers when a developer commits code, and then should they wish to revert they can pull down an older version. It stores every version and its changes instead of overwriting all data on its server every time a developer commits.

2.9 Text-to-Speech

2.9.1 Overview

Text-to-speech (TTS) is software used to convert words from computer document or program into audible speech spoken from the computer's speaker. TTS is an important feature of ACC products, and the emphasis the reviewed existing solutions place on it demonstrates this. It is used as an assistive aid for people with difficulty reading, for example in screen readers for navigating through an interface, or reading a book or web page to the user. It is used for users with people with sight issues, cognitive issues that affect reading and for people with motor issues, operating controls or handling a book, for example (Morin, 2014).

Text-to-speech has evolved a great amount since it was first introduced to computer systems in the late 1950's in Japan. It has transitioned from being robotic sounding to being much more natural sounding. The last ten years in particular have produced some great improvements in TTS technology. The goal of a speech synthesizer is to be as similar as possible to a human voice, naturalness, and be understood by humans as clearly as possible, intelligibility (Gilligan, 2016).

There are two parts to a speech synthesis system. The first part takes text and transforms it into the fully written equivalent. This process is called tokenization. It involves converting abbreviations and numbers to their full form, for example. It then assigns phonetic transcriptions to each word, dividing it into units like phrases, clauses and sentences. The second part then each of these transcriptions into sound. This is known as the synthesizer.

The process of tokenizing text is complicated and difficult. Numbers and abbreviations can have different meanings based on their context, for example "St." could mean Saint or street in different scenarios. There are two approaches to this problem, a dictionary-based approach and a ruled-based approach. The dictionary approach involves querying a large dictionary containing all the words in a language for the pronunciation of each word. The rule-based approach involves applying the rules for the pronunciations of words to each word to pronounce them, just as children are thought to learn speech by sounding out syllables (Gilligan, 2016).

There are two main technologies for generating synthetic waveforms, concatenation synthesis and format synthesis. Format or rule based synthesis does not use human speech samples at runtime. Frequency, voicing and noise levels are varied using this technique to create a waveform of artificial speech. For this reason the speech generated is robotic sounding and not natural. It does produce reliably intelligible speech even when reading at high speeds, however. It is also a smaller system then a concatenation system because it does not have a stored database of speech. Concatenation synthesis involves stringing together parts of recorded speech. These parts can be different sizes and there are different types of concatenation synthesis based on the size of each part.

For concatenation synthesis, voices are recorded, split into variable sized units, and stored in a database. The smallest size unit splits each recording into phonemes, which are the smallest structural units that distinguish meaning in a language. They are the sounds of a language that can be individually produced. A phone is the actual pronunciation of a phoneme. Systems differ in the size of these units with some

systems storing phones or diphones while others are very advanced and store words or even sentences. The systems that store phones can provide a very large output range but may not sound natural, whereas the systems that store sentences have a narrow output range and must be used for a specific domain.

A diphone is a pair of adjacent phones. Diphone synthesis, one type of concatenation synthesis, uses a database of all diphones in a given language to produce all combinations of speech in that language. In diphone synthesis, only one sample of each diphone is stored in the speech database. A technique called unit selection allowed for natural sounding voices. This is concatenation synthesis, but the recording is split into many types of different unit. For example, a recording may be stored by both diphone and word using unit selection. To achieve a high level of naturalness, however, the speech unit databases have to be very large. The synthesizer has access to both long and short samples of speech and the best one is chosen based on the context (Gilligan, 2016).

A new technology is speech synthesis, based off hidden Markov models. Similar to the n-gram modelling being used for this project, it is a statistical method where the TTS is based on a hidden model that it is trained with, and it is refined by further usage. Probabilities are used to calculate the most likely pronunciations, given the context. A set of observations is taken and the next most probable path is calculated based off these observations. This approach is CPU intensive, due to the amount of calculations required by each step.

TTS is very important for accessible applications, and the transition to more advanced mathematical methods to improve TTS software is an important development. Good TTS improves word recognition, helps children recognize errors in their communication and increases their ability to pay attention and remember information (Morin, 2014). Other areas that are important for AAC users in TTS are the tone and accent of the voice produced. This is an issue that was raised by Orlagh during meetings. She mentioned that she and her daughter would prefer a female voice and a child's voice as opposed to an adult male voice, for example. The accent is another important feature.

Apple has a built in TTS API for iOS called AVSpeechSynthesizer. It allows various rates of speaking with different pitch and with various voices. It offers English in American, Irish and British voices (My iOS development monologue, 2014). They do not offer a choice between adult and child's voices. They also do not offer any details about how their TTS works or what method it uses. The Centre for Speech Technology in the University of Edinburgh works on creating TTS technology using different methods, like hidden Markov Models, unit selection and diphone synthesis, for different accents (The Festival Speech Synthesis System, 2016). They offer free software, the Festival Speech Synthesis System, which allows anybody to use their methods with a voice of their own. This could be used to create a synthesizer that is based on a young girl with an Irish accent, for example.

2.10 Functional Requirements

Based on the research conducted on existing solutions, as well as the extra functionality this project will add to them, the users will hopefully have all of the

following, ranked in terms of importance, available to them on completion of the project:

- Customisable communications board with symbols
- Customisable grid size for each board
- Text-to-speech reading of each word or phrase on clicking a symbol or clicking read of a sentence accumulated
- Prediction that is based on a user's comprehension level and use history
- Colour coding of different parts of language, e.g. verbs coloured red
- Customization of user interface colours, such as the background colour
- Allowing the creation of new symbols by the user
- Backup of application configuration to a server
- Allow recording of audio to play on click of symbol, instead of synthesized voice
- Highlight each word as it is being spoken on click of speak sentence button

The objective of the application is to present multiple communications boards of symbols, as seen earlier in the existing solutions. The application will present a prediction for the next word or symbol the user may wish to select after they add one or more to a sentence.

On launch, the application will ask the person configuring the application for the user's comprehension level. This comprehension level will be used to filter the list of predictions offered to the user so that the complexity of the words provided will be suitable for their language capabilities. It will then invite the person configuring to customize the communications boards to suit the user's needs. After initial configuration is complete, the application will be ready for the user to start communicating with.

There will be an area on screen where clicked symbols will accumulate until cleared, so that sentences can be formed, like a text box in a messaging application. The symbols will be image files inside of frames that can be enlarged or made smaller to change the size of the grid to suit the user and their capabilities. This will be available in the application's settings. When a symbol is clicked, the app will read out the word it is associated with using text-to-speech. Also in the settings area of the application, the communications board will be able to be customised so that symbols can be rearranged, new symbols can be added, or existing ones can be deleted. These image files will be saved in the app so they can be added at any time. The person configuring the application will also have the option of adding a symbol from the devices camera roll or photo library.

Boards will allow symbols to be customized, using a drag and drop gesture that is popular on mobile devices. A user will be able to drag a symbol from one location on the board to another and drop it to move it there. Each symbol will have a coloured background that will denote what type of word it is, as seen in the Avaz application earlier. The person configuring the application will be able to add symbols to boards or remove them from them as well.

A section of the screen will present the predicted next word and symbol. Using n-gram analysis gathered on the server side, using the NLTK library, the application will present multiple suggestions for the next symbol based on the input entered so far. The number of suggestions presented will depend on the size of the grid selected. To gather the n-gram analysis, the app will send a HTTP POST request to the server. The server will generate n-grams on a corpus that is suited towards the reading age. The server will respond with the n-grams encapsulated in JSON. The n-grams collected will be filtered against a list of words that are suitable for the comprehension level of the user. It will then store the filtered n-grams in a SQLite database on the iPad using Core Data. When a word is entered, the app will query the database for n-grams containing that word and then calculate the most likely words to follow it from the database. It will present the most likely words to follow the word entered as suggestions to the user. User input will be added to the database as n-grams so that the suggestions will improve with time.

Text-to-speech will be implemented, so that each word or phrase selected is spoken on click, and a sentence is spoken on click of a read button on screen. A feature other AAC applications have, that could be implemented as well, depending on time, is allowing audio to be recorded and played in place of the system's text-to-speech. An example of where this could be used is allowing a parent to replace the text-to-speech reading of a word with his or her own reading of it. This will allow the app to be more personal and the voice to be more recognisable for the end user.

Another feature that other applications offer that will be implemented is the backup of the application's data to the server. The MySQL database could be extended and user data could be stored in it as well. On a new or second device they would have the ability to download a backed-up configuration from the server.

3 Design

3.1 Overview

The user interface and its usability are very important aspects to the project, and are a large part of the reason that the project was taken up in the first place. For this reason user-centric design (UCD) was chosen as the user interface design process. UCD is a process that put specific emphasis on making systems usable as opposed to other design processes that focus on the activity or the business process being performed. All development focuses on the user. It is an iterative process where evaluation and feedback is incorporated throughout the development of the project, and changes are made based off it.

User-centric design allows a developer to achieve a high degree of usability by getting a better understanding of the end user's needs, wants and capabilities. In this case feedback sessions with Orlagh were used for getting an idea of her daughter's capabilities, which is particularly important given the fact that she has cognitive disabilities, as well as what she wants from the software and what problems she currently has, that can be solved.

Usability is defined as "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." The objectives of usability can be described as usefulness, effectiveness, learnability and attitude. Usefulness is the extent to which the product enables the user to achieve their goals. In this case, it is how well the user can communicate using the application. Effectiveness is the ease of use of the product and the speed of its performance or how many errors it produces. Learnability is the user's ability to operate the system with training and how much training is required. It also refers to the ability, or lack of, a user has to use the system without training. Attitude is the user's perception, feelings and opinion of the product and can be described as how much the user likes the product.

Prior to the feedback sessions, background research was carried out to come up with an initial design that would be presented to Orlagh for feedback. This included research on software that is designed for users with disabilities and research on Apple's human interface design specifications and guidelines. The findings from this research, the feedback, and finally from standards set by existing AAC software were combined for the final design.

3.2 UML

This section provides modelling of the system and its functionality. There are two types of user of the system so there is a UML diagram modelling each type's capability.

3.2.1 User

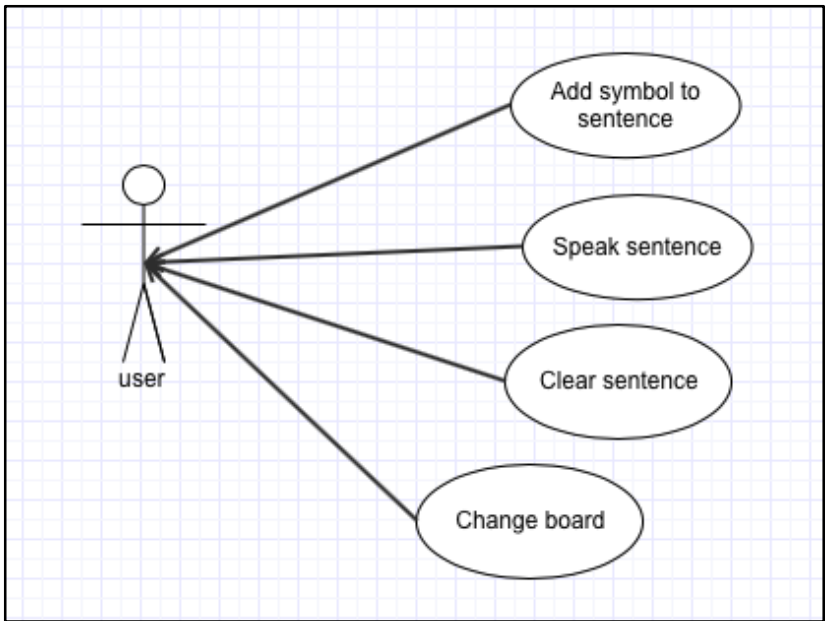

Figure 7 End user use case diagram

3.2.2 Configurer

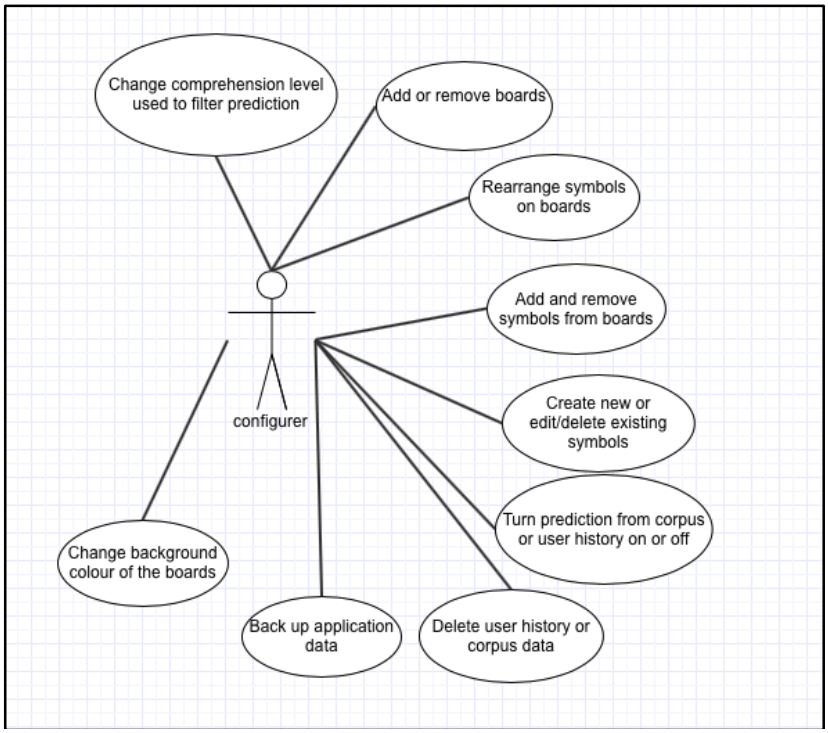

Figure 8 Configurer use case diagram

The UML diagrams illistrates the two types of user that the application has and the different capabilities that each has.

3.3 Apple Human Interface Guidelines

3.3.1 Overview

The Apple Human Interface Developer Guidelines were also reviewed. The guidelines have many human interface recommendations that apply to the development of this application. The first category the guidelines address is screen layout. They recommend that UI elements be placed on screen according to their importance, from the top left to the bottom right. This is consistent with research from the Nielson Norman Group that discovered, using eye tracking, that users pay more attention to content along the left and at the top of a web page (Neilson, 2006). They also recommend that elements be sized according to their importance. Important items should be larger than less important and items that will be used less frequently. They also suggest that elements with similar functionality are aligned together and look similar. Visual weighting is recommended for suggesting the importance of UI elements as well. Red might be used to suggest a button is important, for example.

Apple recommend not having any start up screens and limiting post-install input to a minimum. They recommend not asking for input, like login, for as long as possible. The guidelines also recommend getting as much information from other sources as possible and storing user input in the apps settings, so that it can be changed later on. They also recommend only providing an on boarding tutorial if absolutely necessary and limiting text in this tutorial as much as possible, and instead use animations to teach users how to use the app. Apple don't allow apps to have a stop or exit option. They recommend that apps can resume from their previous state to the lowest level of detail possible when revisited (Developer.apple.com, 2016).

The guidelines suggest that navigation should support the apps function without calling any attention to it. There are three main types of navigational structure: flat, hierarchical and content-driven. A hierarchical app is one where users navigate by making one choice per screen until they reach their destination. To navigate to another screen, users must go back or restart. In an app with a flat information structure, all categories are made available on the home screen, and users can navigate between them. Navigation in an app that uses content is typically driven by the content. This application will use the content driven structure. The guidelines state that users should always know where they are in a navigational structure and how to get to their next destination or return to a previous one (Developer.apple.com, 2016).

Apple recommends a variety of cues to signal that elements are interactive. Examples of this include setting a key colour, location and meaningful icons. Across the OS the icons represent their functionality. Built in apps each have a key colour to show what elements of the screen are buttons. It recommends that standard gestures, like swipe, tap and drag are only implemented consistent with their use in the OS and it's built in apps.

The guidelines recommend the careful use of animation to communicate status, enhancing direction manipulation, and to help users visualize their actions. They caution that animation should be consistent with built in animation, and not be used excessively to the point where it impacts performance or distracts users. They should also be consistent throughout the full application. They say any custom animations should be credible and realistic (Developer.apple.com, 2016).

The use of colour plays a big role in interface design for iOS. Apple recommends using the same font throughout an application. They also caution that colours used should work well together and that contrast should be considered. Apple recommends using one colour to define interactive elements, for example making all buttons blue, and avoid using this colour for non-interactive elements. The guidelines recommend prioritizing content by font size and never using fonts smaller than 11pt. The guidelines recommend that buttons are at least 44pt by 44pt and that there is 11pt between targets (Developer.apple.com, 2016).

3.3.2 Conlusion

Some elements were added to design upon completion of reviewing Apples guidelines. There include:

- Keeping the background of all board icons white, to make it clear to the user that they have a separate purpose to the symbols on the boards themselves, the role of navigation
- The addition of a blue border around the icon of the currently selected board to communicate status
- A gap of size 12pt is used between symbols on each board
- The speak and delete buttons are the same colour and same style to indicate interactivity. They are also coloured black for string contrast against the background so that their visual weighting makes them stand out

3.4 Existing Software

Existing solutions were reviewed from a design perspective in addition to the functionality, as was mentioned earlier. Avaz, pictured in figure 4, influenced the design in particular, as the research conducted on user interface design points to its design, as being the best among the software reviewed. It adheres to more of Apple's guidelines than the other products and it has much better use of contrast than the other products. Some examples of design decisions made based on existing solutions:

- To put the sentence at the top of the screen, as this is consistent with other AAC applications
- To place the delete and speak buttons were placed on the right hand side of the current sentence

3.5 Prototyping & Feedback Sessions

3.5.1 Overview

Following the user-centric design process, many meetings were held to explain the applications design and make changes based on user feedback. Many different prototypes were constructed and demonstrated at various meetings. This started out with a wireframe and a Powerpoint presentation and was developed until later on actual iOS application prototypes were demonstrated with the target end user. At each stage, details were discussed and changes made until the final design was arrived at. The focus of most of the meetings was on the main screen of the application, as this is

the only screen that the end user uses; a parent or another person configuring the application will only access the other screens.

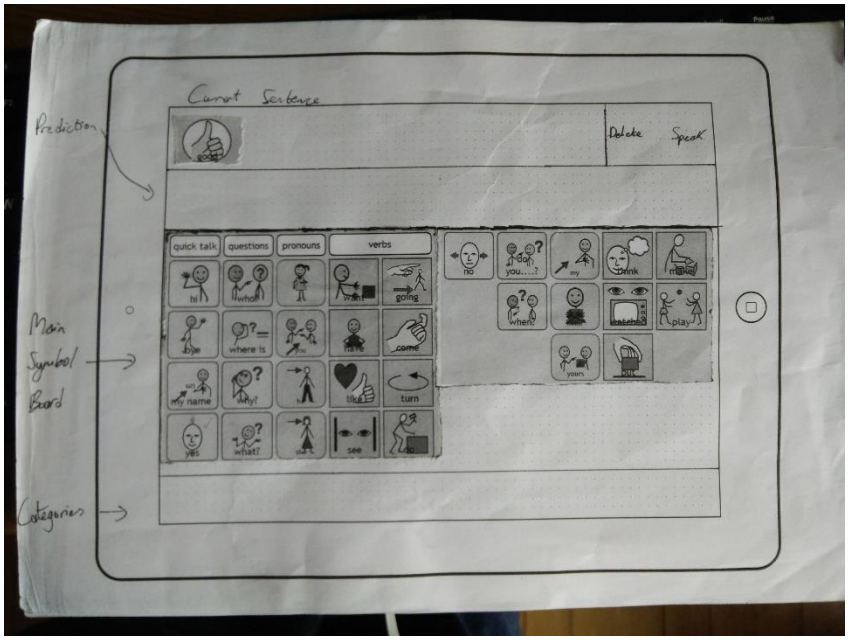

Figure 9 Initial wireframe of the main scene

An example of a paper wireframe put together during the first stage of discussion about user interface design. The sentence is at the top, with the control buttons placed to the right of it. Under that, there is a panel for predictions. The current board takes up the middle of the screen, with a selection of symbols added in the picture for illustration. Navigation from board to board was in a panel at the bottom of the screen labelled categories in the picture.

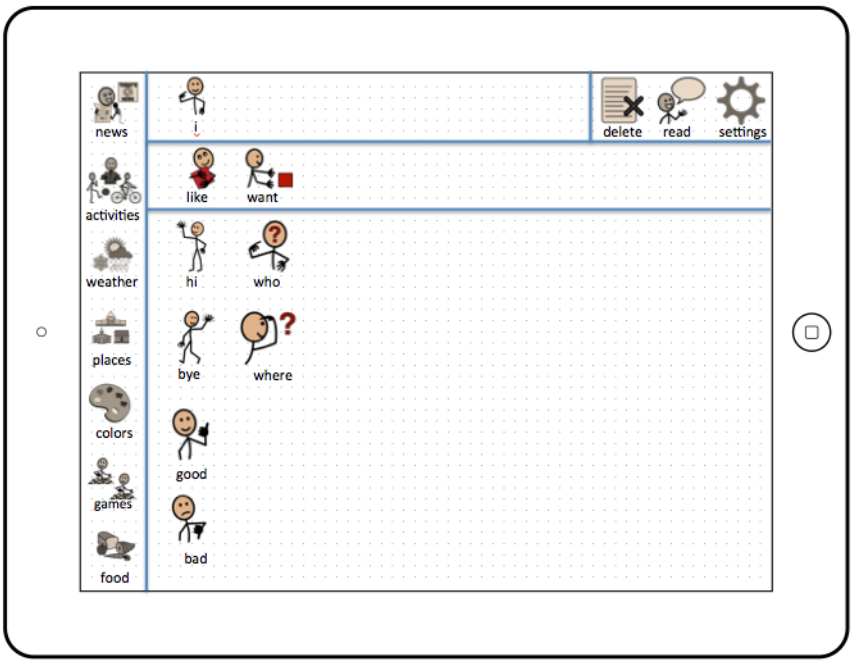

Figure 10 First prototype constructed with Powerpoint

This is a more detailed prototype used in a later meeting. The decision was made to move the navigation between boards from the bottom of the screen to the left hand side, on discovery of the research that concluded that the most important interface elements should be placed across the top and along the left hand side of an interface, and after reviewing Apple's developer guidelines.

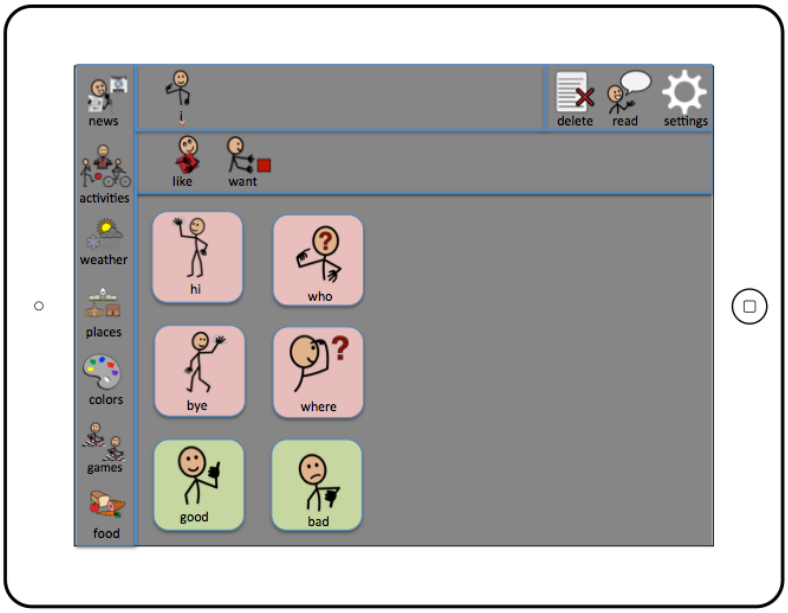

Figure 11 First of two final prototypes constructed before development commenced

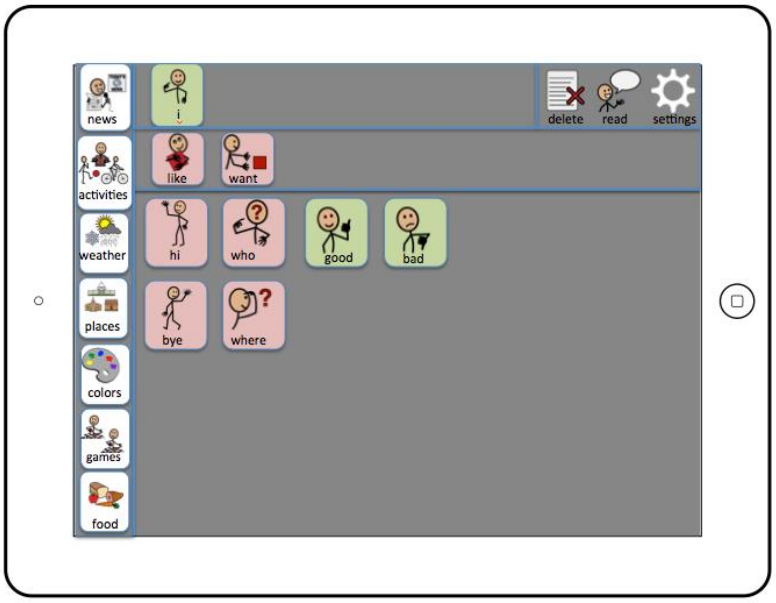

Figure 12 Second of two final prototypes constructed before development begun

These were the last two prototypes discussed before programming began. The symbols on each board in the images were made different sizes to demonstrate that they could be configured in the settings of the application.

3.5.2 Conclusion

While discussing these prototypes, and over the course of all meetings held, Orlagh gave some directions on how the interface should behave and directions that influenced the design. These are all included as part of the design:

- That the current sentence be spoken on click of anywhere in the panel where it is displayed in the user interface, as this is a feature her current software has, and is useful
- That the settings in the application are not accessible to her daughter, as she will change them and may delete data. She suggested that they're locked with a passcode
- That her daughter is able to process thirty-two cells on screen at once, so this should be the approximate default value
- That words or symbols related to people are coloured yellow, verbs are coloured green, descriptions blue, nouns orange, social words pink and miscellaneous white
- That she can easily customize the boards and their content
- That the scroll gesture was appropriate should the space required for the icons of boards, or symbols on them, exceed the available space on screen. A scroll bar will appear in this case. Orlagh confirmed that her daughter would be comfortable with scrolling

3.6 User Interface Colour

Another element that was important in the design was the choice of colours for the user interface. Contrast is very important in particular for disabled users. The software Orlagh demoed had very poor contrast and this was initially targeted as an area for improvement. In order to achieve a good level of contrast and produce a usable design colour-wise, a colour contrast checker was used to test combinations of user interface colours. The tool, provided by WebAIM, a non-profit based in Utah State University that has worked on web accessibility for over a decade, allows the input of a background and a foreground colour, and then gives a contrast ratio of the two colours (Webaim.org, 2016). The background colour is configurable in the applications settings, but the default colour is selected because it has a high contrast ratio against all the other colours that feature in the interface.

4 Architecture

4.1 Overview

This section will discuss the overall layout of the architecture of the project and its components. It will describe the tiers to the system and how each tier interacts with the others.

4.2 Architecture

4.2.1 Full System Architecture

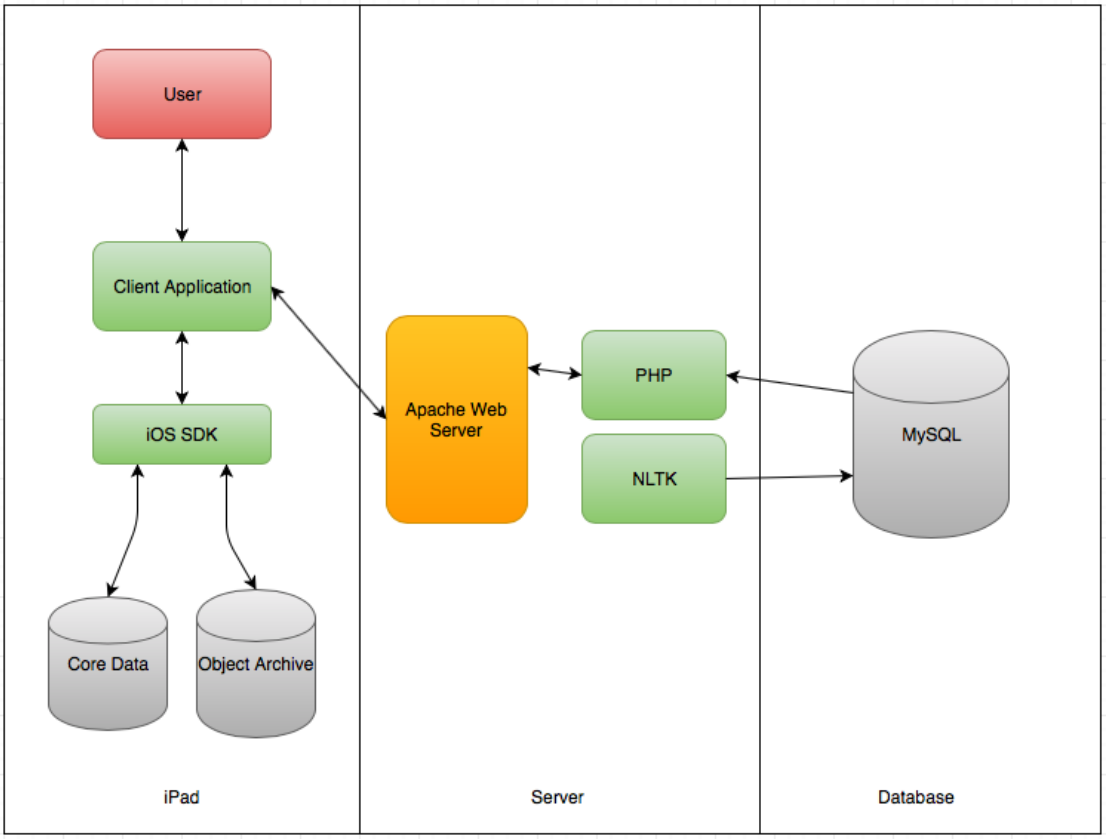

Figure 13 Full system architecture diagram

4.2.2 Client Layer

The client layer is the only level of the system that the user interacts with. It consists of:

- **iOS SDK:** The software development kit provided by Apple that contains all classes, libraries and frameworks for iOS development. Code will be implemented in Xcode using the SDK
- **Core Data:** An object relational mapping framework provided by Apple that uses a local SQLite database for persistence

- **Object Archives:** Technique provided by Apple for the archiving of objects to files. Objects are converted to a stream of bytes and then written to a file

4.2.3 Web Server

The web server layer provides the client application with a means of connecting to the database layer. It consists of:

- **PHP:** PHP scripts will accept requests from the client and send the n-gram data in response using JSON
- **NLTK/Python:** A Python script will be used to call NLTK library calls to generate the n-grams, and then insert the data into the database

4.2.4 Database

The database, MySQL, will store n-grams uploaded from Python and make them available to the PHP scripts. Security measures, like encryption and hashing will need to be deployed, if user data is backed up to the database. On the device Core Data will store the n-gram data.

4.3 Hardware

Hardware used for the development of the project:

- **Apple MacBook Air running OS X 10.11 El Capitan:** used to run Xcode with the iOS SDK, NLTK and Python, as well as a local server using Apache, PHP and MySQL (XAMPP)
- **Apple iPad Mini running iOS 9.3:** Used to develop and test the iOS application

4.4 Web Hosting

Microsoft's Azure infrastructure as a service cloud platform was used for the deployed server. Microsoft gives each virtual machine created, using the service a free domain name. When setting up the virtual machine, both port 80 and port 22 were opened for SSH and HTTP requests. SSH, or secure shell, was used to connect to the virtual machine using its domain name. A virtual machine was set up because all the software used for local development also runs on Ubuntu. The domain used for the project was **prepict.cloudapp.net**.

5 Development and Implementation

5.1 Overview

This section of the report will describe in detail the development and implementation of each section of the project and its functionality. The chapter is divided into two sections, the two hardware tiers of the project: client and server.

5.2 Server-side Development

5.2.1 PHP

```php
<?php
    $connection = mysqli_connect ( "localhost", "fyp", "fypuser", "DIT" ) or die ( "Cannot establish connection" );
    $readingAge = "5";
    if (isset( $_POST['readingAge'] ) ) {
        $readingAge = $_POST['readingAge'];
    }
    $words = array();
    $file = $readingAge .= ".txt";
    $handle = fopen($file, 'r') or die('Cannot open file:  '.$my_file);
    while (!feof($handle)) {
        $words[] = trim(fgets($handle));
    }
    $words = array_map('strtolower', $words);
    $sql = "SELECT * FROM Bigram";
    $rows = array();
    $sth = mysqli_query ( $connection, $sql ) or die ( "Syntax error in SQL statement" );
    $rows = array();
    while($r = mysqli_fetch_assoc($sth)) {
        $r = array_map('strtolower', $r);
        if (in_array(trim($r["word1"]), $words) && in_array(trim($r["word2"]), $words)) {
            $rows[] = $r;
        }
    }
    echo json_encode($rows);
?>
```

Figure 14 PHP script that queries database and filters bigrams for reading age

A script was used to accept and respond to requests on the server. It accepts the comprehension level as a HTTP POST value from an incoming request and reads a text file containing a list of suitable words for that comprehension level into an array. It then connects to the database and selects all rows from the bigrams table. It then filters the database results, only using rows where all the bigram words are in the comprehension list. It encodes the bigram in JSON and responds with the JSON results.

The built in PHP method in_array was used to check if each word in a bigram is in the array that has the reading list, which is read in from a file. The database results are captured as an associative array, so that the field name is also part of each row of the results. This means that the data is in the correct format automatically for converting to JSON and this process just takes a call to PHP's json_encode method.

5.2.2 Python

```
#!/usr/bin/python
import MySQLdb
import nltk

db = MySQLdb.connect("localhost","python","password","DIT")

cursor = db.cursor()

words = nltk.corpus.nps_chat.words('11-08-teens_706posts.xml')
bgs = nltk.bigrams(words)
for n in bgs:
    word1 = str(n[0])
    word2 = str(n[1])
    cursor.execute('''INSERT INTO Bigram (word1, word2) VALUES (%s, %s)''',  (word1, word2))

# Make sure data is committed to the database
db.commit()

db.close()
```

Figure 15 Example of a Python script that generates bigrams and inserted them into the database

The script opens up a connection to the database used, named DIT. It then uses a corpus reader built into NLTK by referencing nltk.corpus. The reader it uses in this case is for the NPS Chat corpus, a corpus of text conversations grouped by the ages of the users who generated them. It uses the 'words' method to return all the individual words in that corpus into the variable named words. A method built in as part of NLTK's Collocations module called bigrams, called here with 'nltk.bigrams', is then used to generate bigrams from the list of words in the 'words' variable. The 'bgs' collection is then looped through, and a row is inserted into the database for each bigram.

The changes are committed outside the loop, as opposed to inside for performance reasons, and the connection is closed. These scripts were run for multiple corpuses to build up the necessary number of records in the bigram table to produce prediction for a large number of words in the client. This was tested locally and then deployed using the Unix SCP command to copy the files from the computer to the server.

5.2.3 MySQL

For bigrams to be stored in the MySQL database, a single table with two columns was required. The PHPMyAdmin graphical user interface (GUI) was used to create and test the database functionality locally. This GUI is turned off on the Azure server for security reasons. For deployment, all commands were typed into the MySQL command prompt, after using SSH to connect to the server from the MacBook. To protect against SQL inject, the PHP script is run from a user who has read only access to the database.

5.3 Client-side Development

Swift was the language selected to code the client application in the Xcode IDE. This is a relatively new language provided by Apple that allows access to the full iOS SDK. iOS development, similar to Android and other mobile SDK's, uses XML storyboards that provide containers for the view, which are then referenced from the Swift code and populated by the view controllers. iOS development using Swift and

storyboards follows the model-view-controller architectural pattern (Developer.apple.com, 2016). Swift was used for the following:

- Provide the graphical user interface
- Read and write to the object archive files
- Fetch, insert and delete from Core Data
- Allow the drag and drop moving of symbols on each board
- Create new symbols from the device camera or gallery
- Create new boards from a user provided icon and name
- Request, receive and parse JSON bigrams from the server
- Calculate the predicted next word from n-gram data
- Generate bigrams from a user's sentences and insert into Core Data
- Speak each word or phrase selected or a full sentence when the user taps the speak button

All requirements were implemented with the exception of backing a user's data up to the server.

5.3.1 Views: Overview

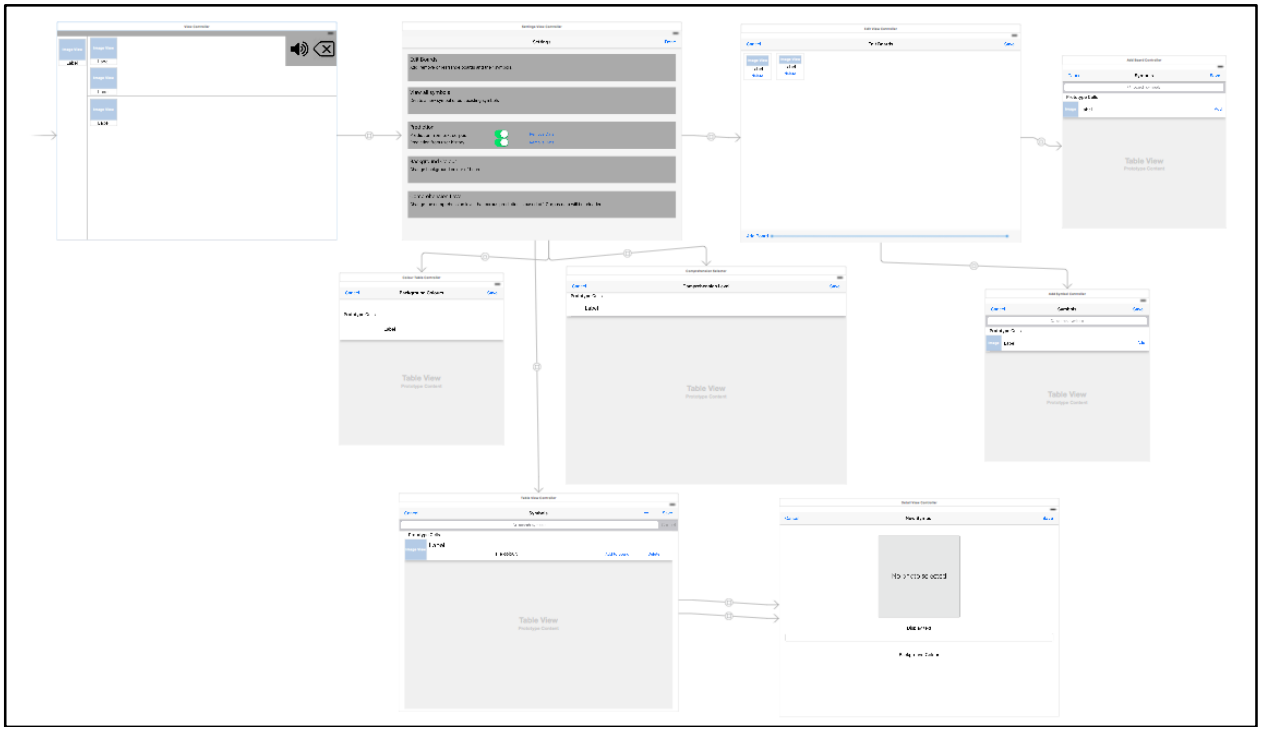

Figure 16 Overview of the client application's full storyboard

The view was constructed using storyboarding in Xcode. Storyboards contain multiple scenes. A scene represents an on-screen area. Each screen can contain one or more scenes. Storyboarding allows for scenes to be constructed by dragging and dropping interfaces objects on to views in the interface builder, shown in figure 16. For each scene, view containers were added to the storyboard and constraints were added to them to give them specific sizing. Scenes are connected by segues. Segues are transitions between scenes, which can be animated, and are represented by the lines shown.

A container allows the dynamic creation of objects in the view at runtime. An example of a container is a table view, where table rows are constructed from a view controller class that holds a reference to the table view at run time. Templates are constructed for each type of object that will be dynamically created, both in the view and in the controller. For example, in each table in the storyboard, a prototype table row is added, which acts as a template for each row created from the Swift code. A corresponding controller class is added for the prototype row that holds references to any elements in the view, like text labels or image views. For each row an object is created from controller template and the row is populated on screen from the associated table view methods using the reference to the table view.

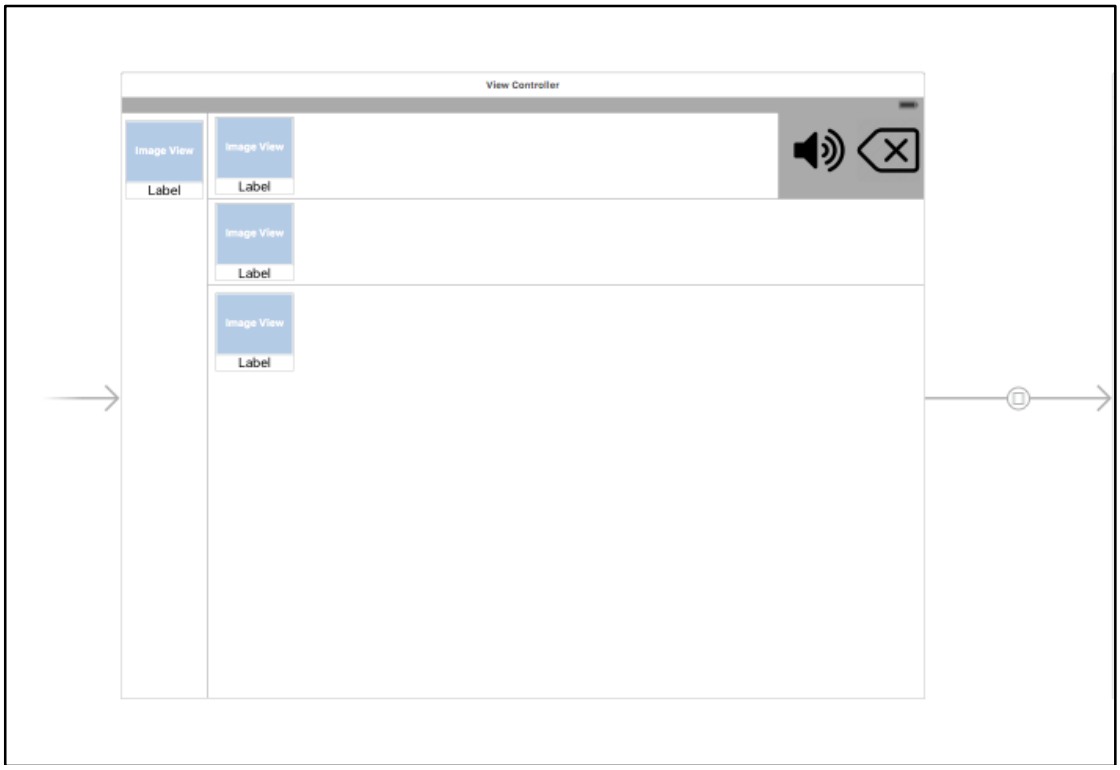

Figure 17 Storyboard of the applications main scene with the various collection views

Figure 17 shows the applications home screen, the screen that the end user uses to select symbols, navigate between different boards and create sentences. The arrow to the left indicates it is the root view controller for the application. That means this scene is displayed when the application launches. This view contains four UICollectionViews, which were chosen to display the board icons, symbols on each board, the current sentence and also the prediction next word. They are containers that arrange objects in a grid pattern, allow for easy reordering and automatically allow scrolling when the number of objects created does not fit inside the space the grid is allocated in the scene. It also contains the speak and delete buttons to the top right-hand side, as was designed earlier in the prototypes. A segue, illustrated by the arrow to the right of the view controller, connects this scene to the settings of the application.

5.3.2 Views: Issues Encountered

Each view of the application is constructed in the same way as above. Each scene has an associated view controller that populates the templates contained on the storyboard and handles the actions that take place when a button is pressed or an item in the grid is selected. Creating the views using storyboarding was initially a frustrating process and considerably slowed development progress at the start of the project. Many problems were encountered connecting the view layer to various controllers as there are many connections from the view containers that have to be maintained and any inconsistency will cause the application to crash with very little indication of the cause.

An example of an issue encountered was connecting a table view to a table view controller class. To connect a table view, it must have an outlet to the viewing controlling it, which is a reference to the table view held in a variable. The controller must also implement the table view and table view data source delegates, the Swift term for what other object oriented languages call interfaces, which means that specific methods have to be implemented. The name of the class must be entered as the controlling class in a field in the storyboard. The table view cells have to have outlets for each item contained in them. The table view itself has to be connected to the class as delegate and data source in the storyboard. All of these have to be maintained and change the name of a variable will cause the application to crash or an empty table to appear.

Every container, like the table view, must have all these connections maintained in the same way. This illustrates the amount of maintenance required between the view and the controllers when developing for iOS, and why it has a steep learning curve and can be difficult to debug.

Another area of difficulty with developing the view was constraints. Constraints are rules applied to the layout of objects in the view. An example would be setting a rule on a table to specify that it must be aligned along the left hand margin. Constraints on iOS are complicated when trying to support many screen sizes and resolutions. When developing for one screen size or resolution, a fixed height and width can be provided for each interface object. This is not a viable option any more with the variation of devices that Apple currently sells. In order to support various different resolutions, objects have to be organised relative to each other. These constraints can easily contradict each other and this will then cause the view to appear empty on screen or objects to overlap. This required a lot of testing both on the device and in various emulators, in different orientations. This was both difficult and time consuming.

5.3.3 Controllers: Overview

The controllers for the project were programmed entirely in Swift. Each scene in the storyboard has an associated view controller class written in Swift. Each object in the view, like a button, has an outlet or reference in the view controller that handles all the actions that must be performed when the user makes a selection, gesture or any other control used in the interface. View controllers are independent of each other, except in the case where segues connect view controllers and objects are passed to the new scene. In this section any important functionality performed by view controllers will be explained. Particular focus is placed on the initial view controller since it is the only scene the end user will interact with; only the configurer will use the others.

5.3.4 Initial View Controller: Main Scene

The first scene launch when the application launches, the only screen the user interacts with, is the initial scene launched on opening the application. For this reason this class in particular is described thoroughly. Shown in figure 18, it contains four UICollectionViews and two buttons.

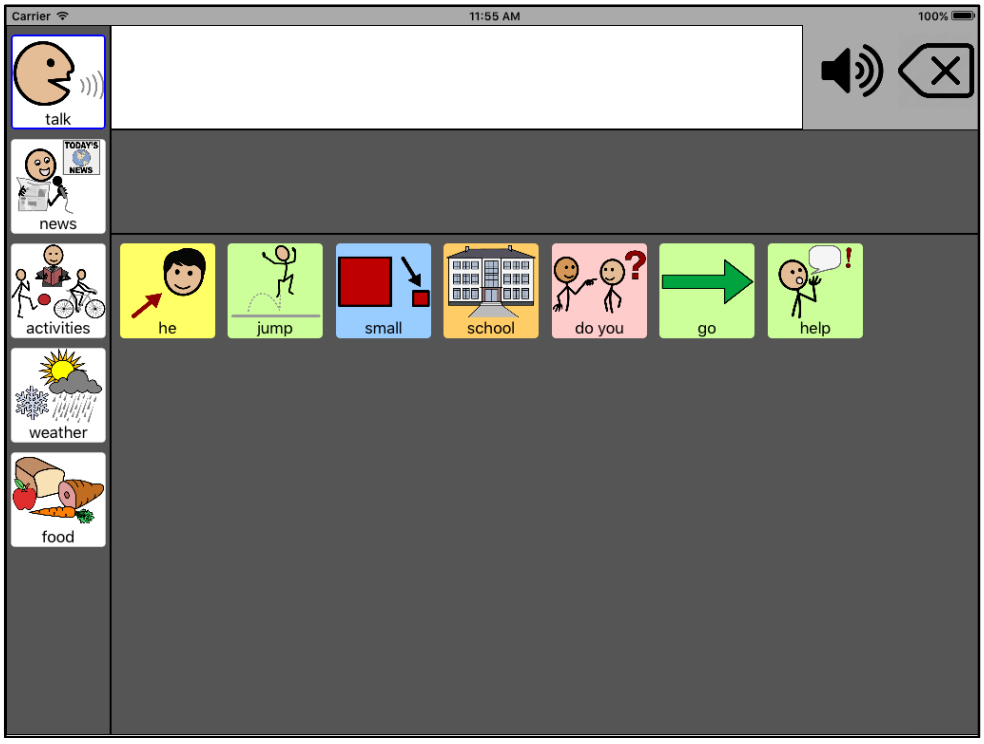

Figure 18 View of initial scene, the applications main screen

```
class ViewController: UIViewController, UICollectionViewDelegate, UICollectionViewDataSource, UIGestureRecognizerDelegate {

    var categories = [Board]()
    var sentence = [Symbol]()
    var suggestions = [Symbol]()
    var currentBoard: Board?
    var allSymbols = [Symbol]()
    var settings: Settings?

    // Outlets for collection views used for boards, their symbols and the current sentence
    @IBOutlet weak var suggestionCollection: UICollectionView!
    @IBOutlet weak var boardCollection: UICollectionView!
    @IBOutlet weak var sentenceCollection: UICollectionView!
    @IBOutlet weak var categoryCollection: UICollectionView!
```

Figure 19 Definition of main view controller class

The main view controller class, shown in figure 19, is a collection view delegate, data source and a gesture recognizer delegate. These are the delegates that have to be implemented to control a collection view, as well as attaching a gesture recognizer to part of a view, which was required to speak the sentence on clicking anywhere in the view, as requested by Orlagh during meetings. Also shown are the outlets to each of the collection views, which link to the corresponding element in the view described earlier.

```
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    var noCells = 0
    if collectionView == self.boardCollection {
        noCells = self.currentBoard!.symbols.count
    } else if collectionView == self.categoryCollection {
        noCells = self.categories.count
    } else if collectionView == self.sentenceCollection {
        noCells = self.sentence.count
    } else if collectionView == self.suggestionCollection {
        noCells = self.suggestions.count
    }
    return noCells
}
```

Figure 20 numberOfItemInSection defines the number of items in each collection view

The first method of this view controller, shown in figure 20, tells the application how many cells it needs to construct for each collection view. The data source for each collection view is an array, categories for the boards, sentence for the accumulated sentence, suggestions for the prediction and symbols for the symbols on the currently displayed board. In each case the count of the number of elements in the array is returned as the number of cells.

```
func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {

    var cell = CollectionViewCell()

    if collectionView == self.boardCollection {
        cell = collectionView.dequeueReusableCellWithReuseIdentifier("cell", forIndexPath: indexPath) as! CollectionViewCell
        cell.image?.image = self.currentBoard!.symbols[indexPath.row].photo
        cell.word?.text = self.currentBoard!.symbols[indexPath.row].word
        cell.backgroundColor = self.currentBoard!.symbols[indexPath.row].bgColor
        cell.layer.masksToBounds = true;
        cell.layer.cornerRadius = 4
    } else if collectionView == self.categoryCollection {
        cell = collectionView.dequeueReusableCellWithReuseIdentifier("cat", forIndexPath: indexPath) as! CollectionViewCell
        cell.image?.image = self.categories[indexPath.row].icon.photo
        cell.word?.text = self.categories[indexPath.row].name
        if currentBoard == self.categories[indexPath.row] {
            cell.layer.borderColor = UIColor.blueColor().CGColor
            cell.layer.borderWidth = 2.0
        } else {
            cell.layer.borderWidth = 0.2
            cell.layer.borderColor = UIColor.blackColor().CGColor
        }
        cell.layer.masksToBounds = true;
        cell.layer.cornerRadius = 4
    } else if collectionView == self.sentenceCollection {
        cell = collectionView.dequeueReusableCellWithReuseIdentifier("sen", forIndexPath: indexPath) as! CollectionViewCell
        cell.image?.image = self.sentence[indexPath.row].photo
        cell.word?.text = self.sentence[indexPath.row].word
        cell.backgroundColor = self.sentence[indexPath.row].bgColor
        cell.layer.masksToBounds = true;
        cell.layer.cornerRadius = 4
    } else if collectionView == self.suggestionCollection {
        cell = collectionView.dequeueReusableCellWithReuseIdentifier("sug", forIndexPath: indexPath) as! CollectionViewCell
        cell.image?.image = self.suggestions[indexPath.row].photo
        cell.word?.text = self.suggestions[indexPath.row].word
        cell.backgroundColor = self.suggestions[indexPath.row].bgColor
        cell.layer.masksToBounds = true;
        cell.layer.cornerRadius = 4
    }
    return cell
}
```

Figure 21 cellForItemAtIndexPath method builds collection view cells

The method shown in figure 21 constructs each cell in each of the collection views. The dequeueReusableCellWithReuseIdentifier method is a reference to the cell in the storyboard. The attributed are added to each cell from the corresponding array and the cell is then returned. Each of the four collection views uses the same CollectionViewCell Swift template, as the icon for a board is stored as a symbol in the program. The current board selected has a thicker blue border drawn around its icon by checking if the array number currently being constructed is the board currently being displayed. This was implemented to give the user a clear understanding of their current position in the navigational structure, something recommended in Apple's design guidelines, and also not following in many existing solutions.

```
func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath: NSIndexPath) {
    if collectionView == self.boardCollection {
        sentence.append(self.currentBoard!.symbols[indexPath.row])
        generateSuggestion((self.sentence.last?.word)!)
        self.speech = AVSpeechUtterance(string: self.currentBoard!.symbols[indexPath.row].word)
        self.speech.rate = 0.4
        self.synth.speakUtterance(speech)
        self.sentenceCollection.reloadData()
        self.sentenceCollection.setNeedsDisplay()
        self.suggestionCollection.reloadData()
        self.suggestionCollection.setNeedsDisplay()
    } else if collectionView == self.categoryCollection {
        currentBoard = self.categories[indexPath.row]
        self.speech = AVSpeechUtterance(string: self.categories[indexPath.row].name)
        self.speech.rate = 0.5
        self.synth.speakUtterance(speech)
        self.categoryCollection.reloadData()
        self.boardCollection.reloadData()
        self.boardCollection.setNeedsDisplay()
    } else if collectionView == self.suggestionCollection {
        sentence.append(self.suggestions[indexPath.row])
        self.speech = AVSpeechUtterance(string: self.suggestions[indexPath.row].word)
        self.speech.rate = 0.4
        self.synth.speakUtterance(speech)
        generateSuggestion((self.sentence.last?.word)!)
        self.sentenceCollection.reloadData()
        self.sentenceCollection.setNeedsDisplay()
        self.suggestionCollection.reloadData()
        self.suggestionCollection.setNeedsDisplay()
    }
}
```

Figure 22 didSelectItemAtIndexPath method handles selection of collection view cells

The method shown above, in figure 22, is automatically called when a cell in any collection view in the scene is selected. When a symbol or prediction is selected, that symbol is added to the current sentence and a new prediction is generated. The word or phrase is spoken by the application using an AVSpeechSynthesis object called speech, instantiated earlier in the class definition. The current sentence and prediction collection views are reloaded and displayed again. When another board is selected, the board name is spoken and the symbols on screen are changed to that boards symbols.

```
func generateBigrams(sentence: Array<Symbol>) {
    var insert = [BigramModel]()
    if sentence.count > 1 {
        for (i, item) in sentence.enumerate() {
            if i != sentence.endIndex - 1 {
                insert.append(BigramModel(word1:item.word, word2: sentence[i + 1].word))
            }
        }
    }
    DataAccess.insertToCoreData(insert, table: "History")
} // End generate bigrams method
```

Figure 23 method used to capture and store user history as bigrams

The method shown in figure 23 is called when a user presses the delete button. On pressing the button, bigrams are generated from the current sentence and inserted into the Core Data ‘History’ entity. The insert array of type BigramModel, which has two attributes ‘word1’ and ‘word2’, both strings, is an array of bigrams that is generated from the users sentence. This is the same process that occurs on the server, except that it is done on a corpus using NLTK instead of the user’s history.

A bigram, as stated earlier in the research, is an n-gram with the element size of two. For the purpose of this project each gram is a word because the application needs to predict the next word. This means that each one is a combination of two words. The sentence ‘I am the author’ contains bigrams: ‘I am’, ‘am the’, ‘the author’, for example. When a word is entered, the next most probable word can be calculated by getting all bigrams where ‘word1’ is the entered word, and calculating the most

frequently occurring ‘word2’s in those bigrams. These are used in the next method to generate a prediction.

```
func generateSuggestion(word: String)
{
    suggestions.removeAll()
    let results1 = DataAccess.selectCoreDataCorpus(word)
    let results2 = DataAccess.selectCoreDataHistory(word)
    var resultsSet = [BigramModel]()
    if results1.count > 0 || results2.count > 0 {
        for item in results1 {
            resultsSet.append(BigramModel(word1: item.word1!, word2: item.word2!))
        }
        for item in results2 {
            resultsSet.append(BigramModel(word1: item.word1!, word2: item.word2!))
        }
        for object in resultsSet {
            for j in resultsSet {
                if object.word2.lowercaseString == j.word2.lowercaseString && object != j {
                    object.count = object.count! + 1
                }
            }
        }
        resultsSet = resultsSet.sort { $0.count > $1.count }

        for item in resultsSet {
            let i = getSymbol(item.word2)
            if i > -1 {
                if !suggestions.contains(allSymbols[i!]) {
                    suggestions.append(allSymbols[i!])
                }
            }
        }
        suggestionCollection.reloadData()
        suggestionCollection.setNeedsDisplay()

    }
}
```

Figure 24 method that predicts the next word from bigram data

The method shown in figure 24 generates a prediction for the next word a user may wish to select by using their last entered word. It is called when a symbol or suggestion is selected in the earlier methods. The first line of the method removes current predictions. It then calls the select from Core Data method of the model class responsible for interacting with Core Data, passing in the previous word or phrase entered, which returns an array of bigrams where that word is the first of the two in the bigram. It does this for both the user history and corpus entities of Core Data and loops through each of the results arrays and appends every element to the ‘resultsSet’ as a ‘BigramModel’ object array so that it can be used to calculate the next most likely word. Each ‘BigramModel’ has two strings ‘word1’ and ‘word2’ and an integer ‘count’.

Next, the results array is looped through twice, and the count of how many times each ‘word2’ element occurs is calculated. The ‘count’ property of each object therefore represents the number of times each ‘word2’ occurs in the array of results. The array is then sorted using the built in Swift array sort method, with the object with the largest count value being sorted to the front of the array.

The results set is then looped through and any word or phrase that is part of the applications symbols library is added to the prediction array to be displayed to the user, with the object with the highest count being added first, so that the most probable word is presented first. This is done by passing the word predicted into the method, which then returns the position at which that symbol is available in the list of all symbols available. The list is an attribute of the class. The suggestions collection view is then updated on screen so that the values are presented to the user. Lastly, the method to check if the application has the symbol for the word predicted is explained.

```
func getSymbol(word: String) -> Int? {
    var result: Int = -1
    for (sym, i) in allSymbols.enumerate() {
        if i.word == word{
            result = sym
        }
    }
    return result
}
```

Figure 25 Method that checks if application has the symbol for a word predicted

This method carries out the aforementioned work of checking if a symbol is part of the applications symbols set, and if it does exist it returns the index in the array where it exists, otherwise it returns minus one so the previous method knows it does not exist. It is necessary for two reasons. First, it checks if the symbol exists in the application for the word predicted. This is necessary because words predicted from the text corpuses may not have symbols available for them. Secondly, it allows the prediction method to get a symbol object for a given word. It does this by looping through all symbols in the application, returning the index of the symbol whose word is the same as the word passed in.

5.3.5 Symbols Table View Controller

The symbols table view controller, which can be launched from the applications settings menu, lists all the symbols that the application contains and allows editing, deletion and adding symbols to boards. The table allows for searching, to make locating a particular word or phrase faster. It also allows a new symbol to be added by clicking the plus button in the top right corner as shown in figure 26.

Figure 26 Symbols table view controller

The table is populated in the same way that the collection views were for the previous view controller. One difference is the use of search on the table. To perform search, a separate empty array of symbol objects is instantiated when the view controller is created called ‘filteredResults’. When the text changes in the search bar, the text is matched against all the word attributes of the symbols array and any matching symbols are copied into the other array. The table view source is changed from the

array of all symbols to the ‘filtered results’ array. All other methods such as delete or adding to a board also reference the results array instead of the full array. When the search bar is empty again, the source is switched back.

The save button on the top right saves all changes made since the view launched. The cancel button discards all changes made since the view launched. This is consistent with Apple’s behaviour across the operating system, as well as their recommendation that users always have an option to abort making changes to an application’s data. This is implemented in all views in the application.

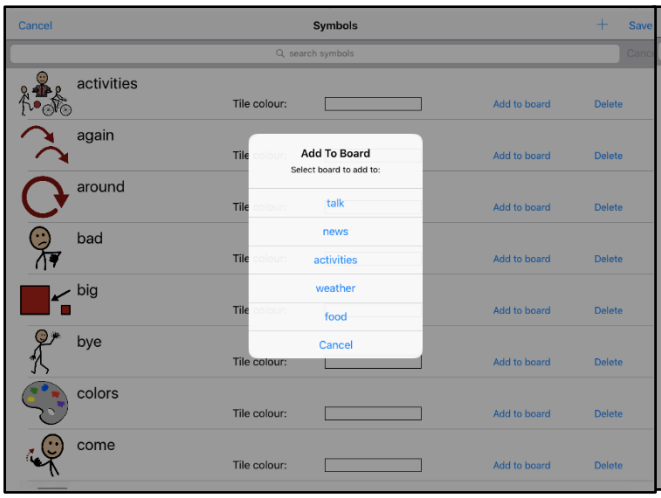

Figure 27 Add to board button

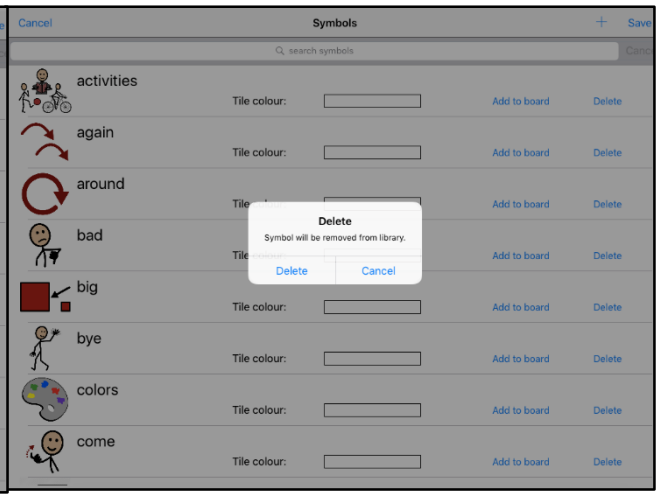

Figure 28 Delete button

On click of the ‘add to board’ button, a pop up dialogue appears that allows a user to select a board to add the symbol to. A dialogue also launches when a user clicks the delete symbol button. This is another Apple recommendation so that the user is aware they are removing data from the application. It allows a user the chance to abort as well.

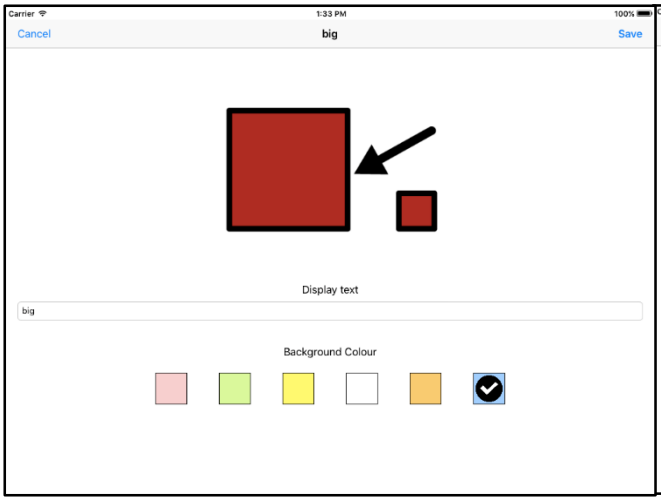

Figure 30 Edit Symbol

Figure 29 Add Symbol

When an existing symbol is selected from the table of symbols in the application, the view shown in figure 30 is displayed. The image view is populated with the image for the symbol, the text box contains its current text, and the custom colour picker control shows its current background colour as selected. All these can be changed, and again when changes are made, the save button becomes active. That is all the functionality available from the symbols table view.

When the plus button is clicked in the symbols table view, the screen shown in figure 29 launches. It features an image view, which can be tapped to select a photo, a text box for entering the word, and a custom control that allows the user to select a background colour for the symbol. The code for this control is in the 'ColourControll' file. This approach allows all the colour buttons to be place as one object on screen, removing the difficulty of adding constraints to size buttons and spacing them equally. The save button is inactive as pictured above, but becomes active when the user enters data. This same view is used when the user taps on a row in the table to edit an existing symbol.

5.3.6 Edit Boards View Controller

The edit boards view controller, which can be launched from the applications settings menu, displays all the applications boards and their symbols and allows reordering, editing and deletion of both the boards and the symbols they contain. This is constructed from a similar view to the one used for the home screen, with different cells as shown below in figure 31. Another difference is the cells can be reordered on this page but not on the home screen. They are also shown in the same form as they are on the home screen, using collection views, with the exception of a delete button being shown on each. This is done so that the person editing does not have to exit the view to visualize how the changes will appear.

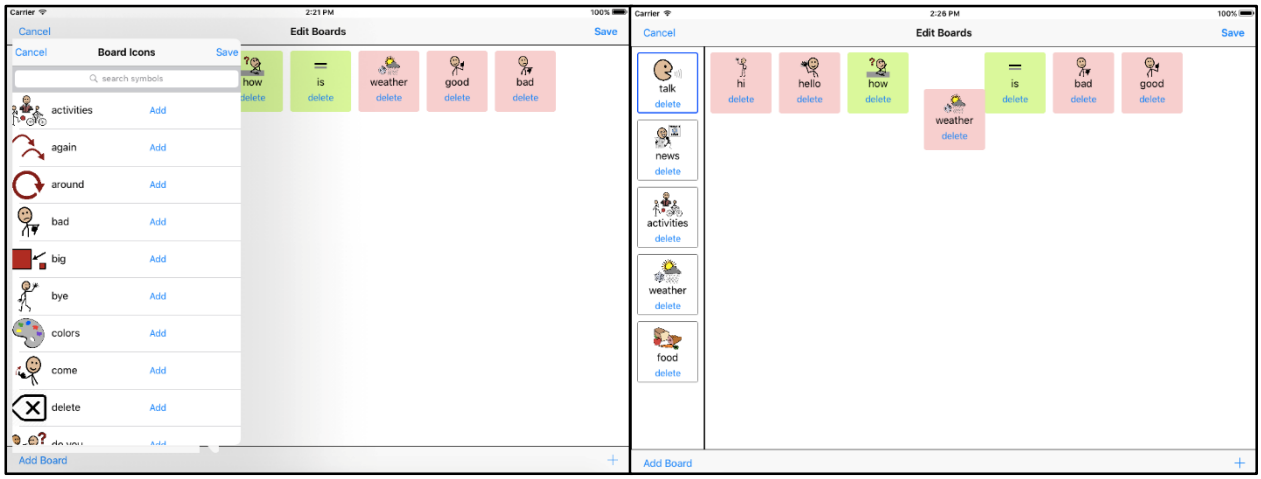

Figure 32 Add board *Figure 31 Moving cells*

The edit board's view is shown in figure 32. A cell is dragged out of place as a demonstration of how cells are reordered. The user can long press a cell, then move

their finger to another position on screen, to drop it there. The same can be done for boards. This was chosen as the method to reorder cells because every iOS device uses this gesture on its home screen so users are familiar with it and it is natural to them.

Also shown is a pop up view that allows the addition of a new board by selecting an icon to use from the list of symbols. The plus button on the right hand side also behaves the same way except it allows symbols to be added. The add button in the list becomes inactive after it is clicked, so a user knows that they have already added the symbol or board.

```
func handleLongGesture1(gesture: UILongPressGestureRecognizer) {

    switch(gesture.state) {

    case UIGestureRecognizerState.Began:
        guard let selectedIndexPath = self.boardCollection.indexPathForItemAtPoint(gesture.locationInView(self.boardCollection)) else {
            break
        }
        boardCollection.beginInteractiveMovementForItemAtIndexPath(selectedIndexPath)
    case UIGestureRecognizerState.Changed:
        boardCollection.updateInteractiveMovementTargetPosition(gesture.locationInView(gesture.view!))
    case UIGestureRecognizerState.Ended:
        boardCollection.endInteractiveMovement()
    default:
        boardCollection.cancelInteractiveMovement()
    }
}
```

Figure 33 Long press gesture added to collection view

The code pictured above in figure 33 is the definition of a method used to add a gesture recognizer to the symbols in the edit view controller, it was added to the collection view earlier in the class. It is executed on long press of a cell in the collection view it was added to, which are the symbols on the board currently open, pictured in figure 31. On execution, if the position of the selected cell changes, its position in the array used as a data source is changed. This same code is used for the collection of boards at the left hand side of the screen to allow those to be reordered using drag and drop.

5.3.7 Other View Controllers: Additional Settings

The applications settings page contains many settings options and many of them have their own view controllers. The comprehension level can be changed, which will remove bigrams from the Core Data entity for corpus data, and reload Corpus data from the server for the new comprehension level. An option is provided for changing the background colour of the main screen as well as the cell size of the symbols on the home screen. The view controllers for the aforementioned features are simplistic but some of the more complex code is explained in the next section, as it is part of the model tier of the application.

Prediction can be turned on and offer for both user history and text corpus data. The data associated with each of these features can also be deleted. A confirmation box also appears to make sure that the user wants to remove the data. These options are part of the settings view controller itself.

5.3.8 View Controllers: Issued Encountered

There were countless issues to solve while programming the view controllers. Many of them overlap with the storyboarding issue mentioned earlier. One feature that took

many attempts to get working and was difficult to overcome problems with was the prediction. That algorithm took much iteration before it would work consistently and combine the data from both the corpuses and the user history. Trying to assign decimal probability values performed the first attempt at prediction but this proved error prone, and was abandoned in favour of the counting approach implemented.

Another controller that was very difficult was the symbols table view controller. Combining a search bar with a table that contains action buttons on each row, and segues that are carried out on selection of rows, was complicated. Two sources of table data were needed because of the search bar and this led to many crashes that took a lot of testing to fix. Every table view method had to be updated with logic to deal with search being used and it being inactive. The segue methods also had to handle both cases and work with both table data sources.

5.3.9 Model, Data Persistence and Server Connectivity

The last remaining tier of the client application is the model and its data classes. The model consists of four classes: Symbol, Board, Settings and BigramModel. There is also a class for accessing and storing archived objects and Core Data, and also for sending and receiving requests to and from the server.

5.3.10 Symbol

```
class Symbol: NSObject, NSCoding {

    // MARK: Properties

    var word: String
    var photo: UIImage?
    var bgColor: UIColor

    static let DocumentsDirectory = NSFileManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask).first!
    static let ArchiveURL = DocumentsDirectory.URLByAppendingPathComponent("symbols")

    struct PropertyKey {
        static let wordKey = "word"
        static let photoKey = "photo"
        static let bgColorKey = "color"
    }
    // MARK: Initialization
    init?(word: String, photo: UIImage?, bgColor: UIColor) {
        // Initialize stored properties.
        self.word = word
        self.photo = photo
        self.bgColor = bgColor
        super.init()
    }
    func encodeWithCoder(aCoder: NSCoder) {
        aCoder.encodeObject(word, forKey: PropertyKey.wordKey)
        aCoder.encodeObject(photo, forKey: PropertyKey.photoKey)
        aCoder.encodeObject(bgColor, forKey: PropertyKey.bgColorKey)
    }
    required convenience init?(coder aDecoder: NSCoder) {
        let word = aDecoder.decodeObjectForKey(PropertyKey.wordKey) as! String
        let photo = aDecoder.decodeObjectForKey(PropertyKey.photoKey) as! UIImage
        let bgColor = aDecoder.decodeObjectForKey(PropertyKey.bgColorKey) as! UIColor
        self.init(word: word, photo: photo, bgColor: bgColor)
    }

}
```

Figure 34 Symbol class

The symbol class, shown in figure 34, has three variables: word, photo and background colour. It also has a path to its object archive location and a URL to the archive file itself. The PropertyKey struct contains the keywords needed for keyed encoding and decoding of the objects to object archives. It also has a constructor and encoding and decoding methods that also use said keys. The keys are declared as a struct type to avoid simple errors like typographic errors between encoding and decoding causing the application to crash.

5.3.11 Board

```
class Board: NSObject, NSCoding {
    var symbols = [Symbol]()
    var icon: Symbol
    var name: String
    var cellSize: Int

    static let DocumentsDirectory = NSFileManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask).first!
    static let ArchiveURL = DocumentsDirectory.URLByAppendingPathComponent("boards")

    struct PropertyKey {
        static let symbolsKey = "symbols"
        static let iconKey = "icon"
        static let nameKey = "name"
        static let cellSizeKey = "cellSize"
    }

    init?(symbols: Array<Symbol>, icon: Symbol, name: String, cellSize: Int) {
        self.symbols = symbols
        self.icon = icon
        self.name = name
        self.cellSize = cellSize
        super.init()
    }

    required convenience init?(coder aDecoder: NSCoder) {
        let symbols = aDecoder.decodeObjectForKey(PropertyKey.symbolsKey) as! Array<Symbol>
        let icon = aDecoder.decodeObjectForKey(PropertyKey.iconKey) as! Symbol
        let name = aDecoder.decodeObjectForKey(PropertyKey.nameKey) as! String
        let cellSize = aDecoder.decodeObjectForKey(PropertyKey.cellSizeKey) as! Int
        self.init(symbols: symbols, icon: icon, name: name, cellSize: cellSize)
    }

    func encodeWithCoder(aCoder: NSCoder) {
        aCoder.encodeObject(symbols, forKey: PropertyKey.symbolsKey)
        aCoder.encodeObject(icon, forKey: PropertyKey.iconKey)
        aCoder.encodeObject(name, forKey: PropertyKey.nameKey)
        aCoder.encodeObject(cellSize, forKey: PropertyKey.cellSizeKey)
    }
}
```

Figure 35 Board class

The board class shares the same structure as the symbol class. Each board object has an array of symbols, a symbol object for its icon, a board name and a cell size property that is an integer. The board class has a constructor and all the methods and properties required for keyed encoding and decoding of objects to its object archive. The settings class follows the same structure and the various settings for the application are also stored in an object archive in the same manner.

5.3.12 BigramModel

```
class BigramModel: NSObject {

    //properties

    var word1: String
    var word2: String
    var count: Int?

    //constructor

    init(word1: String, word2: String) {

        self.word1 = word1
        self.word2 = word2
        self.count = 1
    }
```

Figure 36 BigramModel class

The BigramModel class is used for receiving data from the server, inserting into Core Data, fetching from Core Data and generating predictions. The count property, initially set to one in the constructor, is used for counting the number of occurrences of each word fetched from Core Data as described earlier and shown in figure 23.

5.3.13 Core Data

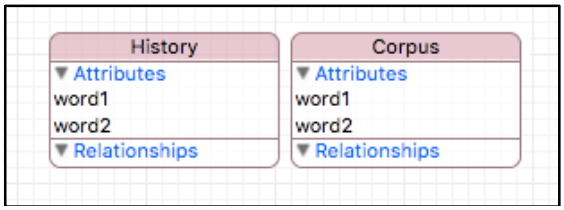

Figure 37 Core Data entity relationship diagram

To use Core Data, an entity relation diagram was first drawn in Xcode as shown in figure 37. There are two separate entities because the history and corpus data needs to be stored separately so that each can be switched on and off as needed, and so that data can be removed from each separately. Xcode then automatically generated NSManagedObject classes for these entities, called History.swift and Corpus.swift. When selecting from Core Data, the objects fetched have to be same type as the class specified in Core Data settings, as well as being NSManagedObjects, so that is why these two classes are used. The class used for accessing Core Data is called DataAccess. It has multiple public class methods, which means they can be accessed in any place in the application without a reference to a CoreDataAccess object, like using static in other object oriented languages.

```
class func insertToCoreData(items: Array<BigramModel>, table: String) {
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
    let managedContext = appDelegate.managedObjectContext
    let entity =  NSEntityDescription.entityForName(table, inManagedObjectContext:managedContext)
    for item in items {
        let newRow = NSManagedObject(entity: entity!, insertIntoManagedObjectContext: managedContext)
        newRow.setValue(item.word1.lowercaseString, forKey: "word1")
        newRow.setValue(item.word2.lowercaseString, forKey: "word2")
        do {
            try managedContext.save()
        } catch let error as NSError  {
            print("Could not save \(error), \(error.userInfo)")
        }
    }
}
```

Figure 38 First of three Core Data access methods

This is the method used to insert to Core Data, both using the user's history and the data downloaded from the server. An array of BigramModel objects and the entity name that needs to be inserted into is passed into the method. A managed object context is required for using Core Data, which is part of the applications AppDelegate class. Specifying the entity name, along with the managed object context, then gets a reference to the entity. Then each object in the array is looped through and the insert statement is executed for each using the save method as shown in figure 38.

```
class func clearCoreData(table: String) {
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
    let managedContext = appDelegate.managedObjectContext
    let fetchRequest = NSFetchRequest(entityName: table)
    fetchRequest.returnsObjectsAsFaults = false
    do
    {
        let results = try managedContext.executeFetchRequest(fetchRequest)
        for managedObject in results
        {
            let managedObjectData:NSManagedObject = managedObject as! NSManagedObject
            managedContext.deleteObject(managedObjectData)
        }
        print("Core data deleted")
    } catch let error as NSError {
        print("Detele all data in \(table) error : \(error) \(error.userInfo)")
    }
}
```

Figure 39 Remove all entity data from Core Data

There is no built in method for emptying all data from a Core Data entity. To do this, all rows were first fetched, and then each object fetched was deleted from the entity so the entity would be empty on completion.

```
class func selectCoreDataCorpus(word: String) -> Array<Corpus> {
    var results = [Corpus]()
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
    let managedContext = appDelegate.managedObjectContext
    let fetchRequest = NSFetchRequest(entityName: "Corpus")
    fetchRequest.predicate = NSPredicate(format: "word1 == %@", word.lowercaseString)
    do {
        results = try managedContext.executeFetchRequest(fetchRequest) as! [Corpus]
    } catch {
        fatalError("Failed to fetch corpus \(error)")
    }
    return results
}
```

Figure 40 Fetch bigrams from Core Data 1

There are two fetch methods for return bigrams from Core Data for the reason already mentioned: the class automatically generated for each entity must be used to capture the results from fetch requests on that entity. Both methods perform the same functionality; they take in a word and return all bigrams where that word is equal to the value of 'word1' in the bigram object. That is all the code wrote for using Core Data in the application.

5.3.14 Object Archives

In the same way the CoreDataAccess class works with Core Data, the ArchiveAccess class is used for accessing and storing objects in object archives. Each method is a class method, which means an ArchiveAccess object does not need to be instantiated to use the methods and they are available throughout all classes at any time.

```
class func checkForFile(fileName: String) -> Bool {
    let DocumentsDirectory = NSFileManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask).first!
    let myFilePath = DocumentsDirectory.URLByAppendingPathComponent(fileName).path
    let manager = NSFileManager.defaultManager()
    if (manager.fileExistsAtPath(myFilePath!)) {
        return true
    } else {
        return false
    }
}
```

Figure 41 checkForFile method that checks if an object archive exists

For each of the three types of objects that are archived in the application, the method shown in figure 41 is used to check if the archive file already exists for that type,

before trying to load the objects from it in each view controller, which would crash the application. It returns true if the archive file exists and false if it does not.

```
class func saveSettings(settings: Settings) {
    let isSuccessfulSave = NSKeyedArchiver.archiveRootObject(settings, toFile: Settings.ArchiveURL.path!)
    if !isSuccessfulSave {
        print("Failed to save settings...")
    }
    print("Saved settings")
}

class func loadSettings() -> Settings {
    return (NSKeyedUnarchiver.unarchiveObjectWithFile(Settings.ArchiveURL.path!) as? Settings)!
}
```

Figure 42 Load and save methods used for each archive

The two methods pictured in figure 42 are used for saving and loading each of the three types of archived objects to and from their archive file. The save method calls the encode method of each objects and the load method calls the decode method of each object, which were discussed earlier. They return all objects stored in the archive file for each.

5.3.15 Server Connectivity

The last class belonging to the model tier of the application is the server connectivity class, called DataModel. This class sends asynchronous requests to the server using the HTTP POST method with the comprehension level of the user included in the body of the request.

```
let urlPath: String = "http://prepict.cloudapp.net/bigram.php"
func downloadItems(readingAge: String) {

    let url: NSURL = NSURL(string: urlPath)!
    let request:NSMutableURLRequest = NSMutableURLRequest(URL:url)
    request.HTTPMethod = "POST"
    let bodyData = readingAge
    request.HTTPBody = bodyData.dataUsingEncoding(NSUTF8StringEncoding)
    var session: NSURLSession!
    let configuration = NSURLSessionConfiguration.defaultSessionConfiguration()


    session = NSURLSession(configuration: configuration, delegate: self, delegateQueue: nil)

    let task = session.dataTaskWithRequest(request)

    task.resume()

}
```

Figure 43 downloadItems method of DataModel

```
protocol DataModelProtocol: class {
    func itemsDownloaded(items: NSArray)
}
```

Figure 44 DataModelProtocol

The download items class is the method called by any objects using this class. This method created the request that is sent to the server. First, it specifies the URL, which is a property of the class defined above the method in figure 44. Any object using the DataModel class has to implement the DataModel protocol, shown in figure 43, which

is like an object-oriented interface, meaning it specifies methods the class must implement.

```
func URLSession(session: NSURLSession, dataTask: NSURLSessionDataTask, didReceiveData data: NSData) {
    self.data.appendData(data);

}
```

Figure 45 didReceiveData method

When new data is received the method shown in figure 45 is called, and it is added to data, a property of the class that stores all data received before it is parsed.

```
func URLSession(session: NSURLSession, task: NSURLSessionTask, didCompleteWithError error: NSError?) {
    if error != nil {
        print("Failed to download data")
    }else {
        print("Data downloaded")
        self.parseJSON()
    }

}
```

Figure 46 didComplete method

The above method runs on completion of the request. If there is an error it is printed to the console, otherwise the parseJSON method is called, which parses and stores the data and is described below.

```
func parseJSON() {

    var jsonResult: NSMutableArray = NSMutableArray()

    do {
        jsonResult = try NSJSONSerialization.JSONObjectWithData(self.data, options:NSJSONReadingOptions.AllowFragments) as! NSMutableArray

    } catch let error as NSError {
        print(error)

    }

    var jsonElement: NSDictionary = NSDictionary()
    let bigrams: NSMutableArray = NSMutableArray()

    for i in 0..<jsonResult.count {

        jsonElement = jsonResult[i] as! NSDictionary
        let bigram = BigramModel(word1: "", word2: "")

        //the following insures none of the JsonElement values are nil through optional binding
        if let word1 = jsonElement["word1"] as? String, let word2 = jsonElement["word2"] as? String {

            bigram.word1 = word1
            bigram.word2 = word2

        }

        bigrams.addObject(bigram)

    }

    dispatch_async(dispatch_get_main_queue(), { () -> Void in

        self.delegate.itemsDownloaded(bigrams)

    })
}
```

Figure 47 parseJson method called on successful receiving of data from server

First the result is parsed into an NSMutableArray. jsonElement is declared as a NSDictionary for parsing each item in that array as this allows referring to key value pairs, which is what the JSON from the server is. The array received is looped through and each element is added to a result array called bigrams. This is passed to the delegate method in the dispatch_async method at the bottom of figure 47, which will be implemented in the calling object, because it had to implement the protocol to execute the code in the first place.

5.3.16 Model Issues Overcame

The model, with the exception of Core Data, went much better than the development of the other two tiers. The objects archived were very easy to use and development of them went smoothly. The request to the server took a lot of time to get right but it is asynchronous so this is very good because it will not slow down the application even with very large amounts of data.

Core Data was originally used for storing the board and symbol data as well but this never worked correctly and was abandoned midway through development. A lot of time was wasted using the wrong classes on the fetch methods of Core Data for bigrams as well, which was causing crashes and empty results sets. It turned out to be a setting that was missed in the side bar of Xcode for the Core Data entity relationship diagram; each entity had to have a linked class used to instantiate objects from as results are fetched. Once these classes were generated for the Corpus and History entities, Core Data worked very well and the rest of the model development was fast.

5.4 Code Used from Outside Sources

5.4.1 Reachability Class

The reachability class simply has once class method that returns a Boolean value, true if the device has an internet connection, and false if it does not. It is used to check if the iPad has a network connection before contacting the server for the n-gram data. This code was taken from a StackOverFlow thread on the issue (Stackoverflow.com, 2016).

5.4.2 UIColor Extension

The UIColour extension at the bottom of ArchiveAccess file was also taken from StackOverFlow. This allowed values taken from the WebAIM accessibility checker, referenced in the design section, to be used for the background colours. It accepts a hexadecimal value as an input and returns a UIColor object of the colour that hexadecimal value represents. It also returns a hexadecimal value for a given UIColor, which is used in the colour control class for comparing colours, because UIColor objects cannot be compared directly in Swift (Stackoverflow.com, 2016).

6 System Validation

6.1 Overview

This chapter details all the testing and validation that was carried out to examine and verify whether the system meets the requirements and specifications and whether or not it achieved a high level of usability as it set out to do. The testing focused on the client application, as this is where most of the functionality is performed.

The testing on the client application includes graphical user interface (GUI), black box and usability testing. GUI testing tests whether the application meets the design requirements set by Orlagh, and set out after the design research. Black box testing tests the whether the application meets the functional requirements. Usability testing tests how user friendly the software is. Orlagh tested the GUI and performed usability testing before development was completed. With this completed, and post development, focus switched to testing the applications functionality.

6.2 Black Box Testing

Black box testing does not test the underlying code or application structure, but only the visible results of what the software did versus what it was supposed to do. For this reason black testing has limited ability to find bugs. The testing did ensure the software behaves how it was intended to do, however. Test cases were used to measure the applications performance against its functional requirements. The test cases and their results can be viewed in Appendix A. The following functionality was tested using test cases:

- Adding a symbol to a sentence
- Reordering a symbol on a board
- Deleting a symbol from a board
- Adding a symbol to a board
- Edit symbol details
- Delete symbol from library
- Add symbol to library
- Reordering boards
- Adding a new board
- Deleting a board
- Add a prediction to a sentence
- Deleting prediction data
- Change board background colour
- Change symbols cell sizes on boards

6.3 GUI and Usability Testing

Following the user-centric design process, Orlagh tested the application at each phase, and changes were made in iteratively based on her feedback. In addition to the design and prototyping described earlier in the design section, Orlagh also tested the GUI of the application to ensure it met requirements over two final meetings held shortly

before completion of the project. In the final phase, she approved the interface's colours, layouts, fonts, icons and buttons. iOS prototypes were used in the meetings.

Meetings were organised by the community's projects supervisors, and students presented their work and discussed their projects one by one. Orlagh, the students, supervisors and also Mr. Damon Berry, a DIT electrical engineering lecturer and Assistive Technologies researcher, attended. Much of the feedback Orlagh provided was included earlier in the design chapter in section 3.5. The usability of the application was the main focus of the meetings. UI behaviour was discussed in an effort to make the application as efficient as possible, for example. One drawback to the testing was that, because of a lack of features developed, much of the applications user interface for the configurer was not tested as part of the meeting. All the interface for the end user was tested, however.

6.4 Issues Discovered

Issue	View	Cause	Solution	Status
Application crashes when all boards are deleted and will not reopen	Main scene	There is no check to see if a board exists	Check size of boards array before loading	Fixed
Row appears blank after symbol is edited	Symbols table	Reload row data after editing not working correctly	Fix index of row reloaded	Unfixed
Add buttons disabled without being pressed	Edit boards pop up	Unknown	Unknown	Unfixed
Application crashes when search bar cancel button is selected	Symbols table	Search bar cancel button action method was missing	Implement cancel button method	Fixed
Application crashes when search bar cancel button is selected	Edit board	Search bar cancel button action method was missing	Implement cancel button method	Fixed
View goes off edge of screen in portrait	Symbols table	Wrong view constraints	Implement dynamic constraints	Fixed
View does not display correctly in portrait	Settings	Incorrect view constraints	Change view layout – layout is not suitable	Fixed

Both unfixed issues have workarounds and do not affect the user's ability to carry out the affected functionality.

7 Project Plan

7.1 Overview

This section will discuss and evaluate the project its differences between the end product and the initial project proposal. Any requirements not implemented are discussed, as well as features discussed that were not requirements. A review of the project and changes made are also discussed.

7.2 Project Plan

Most requirements were implemented but not all due to the time taken to become familiar with developing in Swift and for iOS. This delay in making progress on the client application had a knock-on effect and led to the lower priority requirements not being completed. The project did not follow the plan set out in the proposal for many reasons. Discussion between the author and supervisor on the project led to the focus of the project moving to usability and the change to the user-centric design approach. With this, there were many more iterations of design, prototyping and user feedback. This pushed the development of much of the projects functionality out much later than originally scheduled.

As part of the project research and planning many different people were consulted for various issues and guidance outside of the feedback from Orlagh. These include fellow community's projects students, Kieran Hogan and Laura McGovern, DIT lecturer Dr Robert Ross and n2y Director of Symbols, Ms. Anne Johnson-Oliss. The project supervisor set up an informal meeting with Mr. Ross. N-gram modelling was discussed with reference to this project in particular. Mr. Ross's insight helped guide the direction of the project and give the author a better understanding of natural language processing. Meetings between the author and supervisor were also held weekly, and biweekly, on occasion to discuss the design and development of the project. The majority of decisions regarding the project detailed throughout the report were discussed in these meetings.

The meeting between the students was held to discuss areas of each of their projects that overlapped, such as the use of symbols and colours used to colour code language, many resources were shared between students for both research and implementation. As mentioned in the research chapter earlier, a meeting was held with Ms. Johnson-Oliss from n2y. She expressed interest in the project and explained the process by which they design their symbols and sent the author some of the research behind their work.

7.3 Features Not Implemented

The backing of user data up to the server was not implemented. Although the server was in place and a layer was implemented for interacting with it, it would have taken extra time to implement this feature, as it would have required the serialization of images and other complex data types. It also requires careful consideration from a privacy point of view and would require a lot of extra security protection such as

encryption and hashing. This was not possible in the time frame available for development of the project. The database design is not very difficult, as it would mirror the classes defined in the application for the objects that need to be backed up. The data could be sent to the server using JSON, as is done from client to server already for the bigram data. The PHP files would also be relatively simple because of how easy it is to encode and parse data in PHP, as is evident from the length of the script shown earlier in the development chapter.

Another feature that was not implemented was allowing users to record audio to play instead of the synthesized system voice for each word. The development time was focused on fixing any issues with more important requirements and fixing bugs as opposing to implementing all the features, but leaving bugs in the final system. This approach was taken to ensure the system delivered was useful to Orlagh and her daughter. It is very important for a system that provides such a vital service to the user to perform its functionality error-free. Apple provides an API for audio recording called AVAudioRecorder sop this requirement would not take a lot of development time to add to the application.

The last feature that was not implemented was the selection of each word symbol or word as it is being spoken by the application. This was researched for development and the API provides a delegate that allows for fine-grained tracking of the progress of speech being read, but this feature was not implemented due to time constraints.

An existing feature that was not implemented as originally intended due to its difficulty was the method used to delete symbols or boards. There was supposed to be an area at the button of the screen, pictured in figure 31, which would act as a trash can as it does on a desktop operating system, so that symbols or boards could be dragged and dropped on to the area to be deleted. This was attempted but the code for dragging collection view cells outside of their collection view is complicated and the feature was scrapped in place of the less aesthetic but functional current method.

7.4 Future Work

Future work on the project includes requirements not implemented, expansion and further development of existing features, as well as many ideas that were discussed in various meetings but could not be developed due to the limited time available and other reasons explained below.

The implementation for features not developed includes all the functionalities mentioned in the last section. These features were not developed for a variety of reasons but the implementation of each of them was researched so the time required to implement them would be relatively short. The tracking of spoken text would improve the applications usability and also the applications ability to teach the user new language and vocabulary. The recording of audio would allow a parent or therapist to assist where the speech synthesis is not working very well. The backing up of user data would make the project more commercially viable as this is a feature that would be required for larger deployments of the application more so than one specific user. All these features would allow the application to achieve feature parity with even the most advanced current software, and also surpass the current solutions in many

respects. In addition to this a full built-in symbols set would be required for core and fringe vocabulary.

Existing functionality could also be improved and developed further. Remaining bugs explained earlier could be fixed. The n-gram modeling could be made much more advanced. There are many areas for exploration and development, many of which were researched, with natural language processing. Firstly, a different or multiple values of n can be used for n-gram modeling. Two may well work the best, especially in the case of a child, but three and four and other values could be implemented alongside or instead of two. The same code already wrote could easily be adopted for a larger value of n.

Another natural language processing option researched but not implemented was using stemming or lemmatization carried out on the text corpus by an algorithm like Porter's algorithm. This involves reducing down all combinations of a word to the stem or lemma, so for example, 'walks' would become 'walk' (Manning, Raghavan and Schütze, 2008). This could be useful for AAC because in many cases the user does not have the language skills to distinguish between the two forms, so it increase the number of predictions generated and the relevancy of each prediction for this use case by counting all occurrences of each words lemma in the corpus instead of each words full form, because symbols board generally contain a words lemma.

More options could be added for voices to be spoken by the application. The Festival speech system could be utilized to create a variety of different synthesizers for users of different genders and ages. This is something that was discussed at various stages and the Edinburgh University have open sourced the software so that their process can be used by another developer to create a synthesizer from recorded audio, using the unit selection method, for example (The Festival Speech Synthesis System, 2016).

Other issues discussed at various meetings, but not researched include implementing more context-aware prediction and using a different interface layout that places predictions in columns allowing a user to construct a grammatically correct sentence by selected symbols from left to right, choosing one from each column to construct a sentence. Context aware prediction could work by taking into account what board the user is currently using, which could indicate the top they wish to communicate about, and then changing the values predicted to be more specific to this topic. One way of implementing this would be, for example to separate user history by board that was active when the history was created, and then use the history specific to each board while it is the current one on screen. This approach was not adopted for the project as part of the idea behind the project was to remove the need for navigation, but this approach would require navigation by only predicting words in the current context.

The other idea discussed was making an application or a screen of the application that could help to teach the user sentence structure using the prediction data. The interface would use columns where the first column would be on the left hand side and feature the first predicted word, and the second would be on the right of it and its contents would change based on the first selection. This can be easily achieved using the n-gram modeling approach, but for this project the aim was to build an application for general purpose communication that allows SNUG uses the boards approach so that users can learn the position of each symbol on a board, while also having the

prediction panel to select words on different boards or even not on any board, that are probable to follow the previous word.

The columns that change based on a previous word might be useful for certain cases but there are many issues with the theory. An example of one is that the n-gram modeling only produces probable prediction, and this may not be grammatically correct. Another is that core vocabulary is not always accessible for the user, which is one of the foundations of single-meaning picture based AAC. If a user knows the symbol they want to select, but it is not displayed, they will have to use navigate to it using another method, which introduces complexity.

8 Conclusion

8.1 Reflection and Evaluation

Looking back over the last few months since the initial proposal, there are many decisions that would be taken differently. Swift and iOS development should have commenced far sooner so that all features and requirements were developed. All features would have been completed had the iOS development been tackled earlier. Major risks were well identified in the proposal, and design and development did not hit any major roadblocks, but the proposal risk assessment overlooked the number of new technologies and techniques being used and the time required and learning curve associated with each.

After I submitted my proposal I was skeptical I would be able to complete the project at all. I am happy with and proud of the project developed. I think it offers a user the opportunity to access more vocabulary and to communicate faster than existing systems. Even with development unfinished, it offers more features than many paid AAC systems that are costly. It was interesting and enjoyable to design and develop a project with a real end user in mind, and using the user-centric design process. I had a lot of help on the way and put a lot of time into the whole project process but I feel a sense of achievement from it and I am happy with what I have learned from it.

8.2 Conclusion

The project achieved what it set out to achieve; using n-gram modelling for predicting words for a single-meaning picture based AAC system. Time should have been managed better and the requirements would have then been fully implemented. Although there are small bugs remaining in the implementation, testing showed a system with a high degree of usability was delivered so the projects most important goals were reached. Further development and expansion of the project could yield an application that is worthy of release.

9 Bibliography

1. American Speech-Language-Hearing Association (2015). ASHA. Available at: http://www.asha.org/public/speech/disorders/AAC/. [Accessed 3 December 2015].
2. Grandbois, K. (2012) *Augmentative Alternative Communication*, (*Autism Speaks Technology and Autism Webinar Series*). Autism Speaks. Available at: https://www.autismspeaks.org/sites/default/files/augmentative_alternative_communication_webinar.pdf. [Accessed 29 March 2016].
3. Cannon, B., & Edmond, G. (2009). A few good words: Using core vocabulary to support nonverbal students. The ASHA Leader, 15 (5), 20-24. Available at: http://leader.pubs.asha.org/article.aspx?articleid=2289630. [Accessed 31 March 2016].
4. Light, J. (1988). Interaction involving individuals using augmentative and alternative communication systems: State of the art and future directions. Augmentative and Alternative Communication, 4, 66-82.
5. American Speech-Language-Hearing Association (2016). ASHA. Available at: http://www.asha.org/public/speech/disorders/InfoAACUsers.htm. [Accessed 30 March 2016].
6. RockyBay.org. (2016). *Selecting a Communication Device*. [online] Available at: http://www.rockybay.org.au/wp-content/uploads/2013/04/3.3-Words-and-Messages-on-a-Communication-Device.pdf [Accessed 31 March 2016].
7. AAC Language Lab. (2008). *What's Important in AAC*. [online] Available at: https://aaclanguagelab.com/files/whatsimportantinaac.pdf [Accessed 1 Apr. 2016].
8. SymbolStix (2015). N2y Software. Available at: https://www.n2y.com/products/symbolstix. [Accessed 7 December 2015].
9. McNaughton, D. and Light, J. (2013). '*The iPad and Mobile Technology Revolution: Benefits and Challenges for Individuals who require Augmentative and Alternative Communication*', Augmentative and Alternative Communication, 29:2, 107-116. Available at: http://www.tandfonline.com/doi/full/10.3109/07434618.2013.784930. [Accessed 5 December 2015].
10. Smart Apps For Kids. (2013). *Review: Assistive communication with Aacorn*. [online] Available at: http://www.smartappsforkids.com/2013/12/review-aacorn.html [Accessed 31 Mar. 2016].
11. AACorn AAC. (2016). *About*. [online] Available at: http://aacornapp.com/about/ [Accessed 31 Mar. 2016].
12. Think Smart Box. (2016). *Grid 3*. [online] Available at: https://thinksmartbox.com/product/grid-3/ [Accessed 31 Mar. 2016].
13. AssistiveWare (2015). Available at: http://www.assistiveware.com/product/proloquo2go. [Accessed 2 December 2015].
14. Avaz – AAC App for Autism (2015). Google Play. Available at: https://play.google.com/store/apps/details?id=com.avazapp.autism.en.avaz&hl=en. [Accessed 4 December 2015].
15. Avaz (2015). Avaz App. Available at: http://www.avazapp.com/features/. [Accessed 3 December 2015].

16. TalkTablet - Speech/AAC app (2015). Google Play. Available at: https://play.google.com/store/apps/details?id=com.gusinc.talktableta&hl=en. [Accessed 3 December 2015].
17. Jurafsky, D. and Martin, J. H. (2014) Speech and Language Processing. New Jersey: Prentice Hall.
18. Trim, C. (2012) 'The Chain Rule of Probability', *NLP Blog*, 4 November. Available at: https://www.ibm.com/developerworks/community/blogs/nlp/entry/the_chain_rule_of_probability?lang=en. [Accessed 17 November 2015].
19. Hockenmaier, J. (2012) *Lecture 3: Smoothing*, (*CS498JH: Introduction to NLP*). University of Illinois.
20. Xamarin (2015). Xamarin. Available at: https://store.xamarin.com/. [Accessed 4 December 2015].
21. Nel, H. (2014) 'Picking between Xamarin and Apache Cordova', *MSDN New Zealand Blog*. Available at: http://blogs.msdn.com/b/nzdev/archive/2014/12/22/picking-between-xamarin-and-apache-cordova.aspx. [Accessed 4 December 2015].
22. Kohan, B. (2015) Native vs Hybrid / PhoneGap App Development Comparison. Available at: http://www.comentum.com/phonegap-vs-native-app-development.html. [Accessed 4 December 2015].
23. StatCounter (2015). StatCounter. Available at: http://gs.statcounter.com/#tablet-os-ww-monthly-201501-201508. [Accessed 1 December 2015].
24. Assist Ireland (2015). *Apps for People with Disabilities and Older People - Assist Ireland*. [online] Assistireland.ie. Available at: http://www.assistireland.ie/eng/Information/Information_Sheets/Apps_for_People_with_Disabilities_and_Older_People.html [Accessed 5 Dec. 2015].
25. Flewitt, R., Kucirkova, N., & Messer, D. (2014). 'Touching the virtual, touching the real: iPads and enabling literacy for students experiencing disability',*Australian Journal Of Language & Literacy*, 37, 2, pp. 107-116. Available at: http://web.a.ebscohost.com.elib.tcd.ie/ehost/pdfviewer/pdfviewer?sid=eaabf37d-021e-4961-87f5-48f3e2137b84%40sessionmgr4005&vid=13&hid=4209. [Accessed 5 December 2015].
26. Solt, P. (2016). *Swift vs. Objective-C: 10 reasons the future favors Swift*. [online] InfoWorld. Available at: http://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html [Accessed 31 Mar. 2016].
27. Bird, S., Klein, E. and Loper, E. (2009) Natural Language Processing with Python, Sebastopol: O'Reilly Media, Inc.
28. Php-nlp-tools.com. (2016). *Natural language processing tools | NlpTools PHP*. [online] Available at: http://php-nlp-tools.com [Accessed 31 Mar. 2016].
29. Steidler-Dennison, T. (2016). *LAMP Web development | Red Hat*. [online] Redhat.com. Available at: https://www.redhat.com/magazine/003jan05/features/lamp/ [Accessed 31 Mar. 2016].
30. Morin, A. (2014). *Text-to-Speech Software: What It Is and How It Works*. [online] Understood.org. Available at: https://www.understood.org/en/school-learning/assistive-technology/assistive-technologies-basics/text-to-speech-software-what-it-is-and-how-it-works [Accessed 31 Mar. 2016].

31. Gilligan, J. (2016) *Overview of Text-to-Speech* [Lecture to BSc Computing], *DT211/3: Universal Design and Assistive ICT*. Dublin Institute of Technology.
32. The Festival Speech Synthesis System (2016). *Festival*. [online] Available at: http://www.cstr.ed.ac.uk/projects/festival/ [Accessed 31 Mar. 2016].
33. Jacobs, B. (2012). *Data Persistence and Sandboxing on iOS - Envato Tuts+ Code Tutorial*. [online] Code Envato Tuts+. Available at: http://code.tutsplus.com/tutorials/data-persistence-and-sandboxing-on-ios--mobile-14078 [Accessed 2 Apr. 2016].
34. Developer.apple.com. (2016). *Code Signing - Support - Apple Developer*. [online] Available at: https://developer.apple.com/support/code-signing/ [Accessed 1 Apr. 2016].
35. Mayo, B. (2015). *Xcode 7 allows anyone to download, build and 'sideload' iOS apps for free*. [online] 9to5Mac. Available at: http://9to5mac.com/2015/06/10/xcode-7-allows-anyone-to-download-build-and-sideload-ios-apps-for-free/ [Accessed 1 Apr. 2016].
36. McClure, M. J. and Rush E. (2007) *Selecting Symbol Sets: Implications for AAC Users, Clinicians, & Researchers* [online]. *2007 ASHA Convention*. Available at: http://www.asha.org/Events/convention/handouts/2007/0914_Rush_Elizabeth_2/ [Accessed 1 Apr. 2016].
37. My iOS development monologue. (2014). *Text to speech - AVSpeechSynthesizer - My iOS development monologue*. [online] Available at: http://devmonologue.com/ios/tutorials/text-speech-ios-avspeechsynthesizer-tutorial/ [Accessed 1 Apr. 2016].
38. Developer.apple.com. (2016). *iOS Human Interface Guidelines: Designing for iOS*. [online] Available at: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/ [Accessed 2 Apr. 2016].
39. Nielsen, J. (2006). *F-Shaped Pattern For Reading Web Content*. [online] Nngroup.com. Available at: https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/ [Accessed 2 Apr. 2016].
40. Webaim.org. (2016). *WebAIM: Color Contrast Checker*. [online] Available at: http://webaim.org/resources/contrastchecker/ [Accessed 2 Apr. 2016].
41. Manning, C., Raghavan, P. and Schütze, H. (2008). *Introduction to information retrieval*. New York: Cambridge University Press.
42. Stackoverflow.com. (2016). Check for internet connection in Swift 2 (iOS 9). [online] Available at: http://stackoverflow.com/questions/30743408/check-for-internet-connection-in-swift-2-ios-9 [Accessed 6 Apr. 2016].
43. Stackoverflow.com. (2016). How to use hex colour values in Swift, iOS. [online] Available at: http://stackoverflow.com/questions/24263007/how-to-use-hex-colour-values-in-swift-ios [Accessed 6 Apr. 2016].
44. Developer.apple.com. (2016). About the iOS Technologies. [online] Available at: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html [Accessed 6 Apr. 2016].

10 Appendices

10.1 Appendix A

Test Case	Steps to execute	Expected Result	Actual Result	Pass or fail
Add symbol to sentence	1. Click symbol on current board	Symbol is added to sentence and word is spoken	Symbol was added	**Pass**
Reorder symbols on board	1. Long press delete button 2. Click edit boards 3. Long press a symbol, move it and let go 4. Press save and then done in settings	Symbol is in new position on home screen	Symbol was moved	**Pass**
Delete symbol from board	1. Long press delete button 2. Click edit boards 3. Delete a symbol and save 4. Press done in settings	Symbol is deleted	Symbol was deleted	**Pass**
Add symbol to board	1. Long press delete button 2. Click edit boards 3. Tap plus in bottom right corner 4. Click add button on a row 5. Press save and then done in settings	Symbol is added to board	Symbol was added	**Pass**
Edit symbol details	1. Long press delete button 2. Click view all symbols 3. Tap a row 4. Edit the name, background colour and photo 5. Tap save	Symbol details are updated in table	Row is blank	**Fail**
Delete symbol from library	1. Long press delete button 2. Click view all symbols 3. Click delete 4. Tap save	Symbol is deleted	Symbol was removed	**Pass**
Add symbol to library	1. Long press delete button 2. Click view all symbols 3. Tap the plus button 4. Provide a name and image and select a background colour 5. Tap save	Symbol is added to end of table	Symbol was added	**Pass**
Reorder boards	1. Long press delete button 2. Click edit boards 3. Long press a board, move it and let go	Board is in new position in left panel	Board was moved	**Pass**

	4. Press save and then done in settings			
Add prediction to sentence	1. Click two symbols on current board 2. Click delete 3. Click first of the two symbols again 4. Click second in prediction panel	Symbol is added and word spoken	Symbol added and word spoken	**Pass**
Add a new board	1. Long press delete button 2. Click edit boards 3. Tap add board on botton left 4. Click add button on a row 5. Press save and then done in settings	Board is added to end of collection of boards	Board was added	**Pass**
Delete a board	1. Long press delete button 2. Click edit boards 3. Click delete on a board 4. Tap save	Board is removed from boards collection	Board was deleted	**Pass**
Delete prediction data	1. Click two symbols on current board 2. Click delete 3. Click first of the two symbols again 4. Click delete button 5. Long press delete 6. Click remove user history prediction data 7. Click ok in dialogue then done 8. Tap same first word again	Second word is not available as a predicted option	Word was not available	**Pass**
Change background colour	1. Long press delete button 2. Click change background colour 3. Choose a different colour 4. Click save and then done	Background colour is changed	Background changed	**Pass**
Change symbols cell sizes on boards	1. Long press delete button 2. Choose change symbol size 3. Select a different value	Cell size is changed	Cell size was changed	**Pass**