Submitted for the Degree of M.Eng. in Computer Science 2016

Circular Watch Text

Author:
Cameron Drennan

Registration Number:
201126086

Supervisor:
Dr. Mark Dunlop

Second Marker:
Dr. Marilyn Lennon

Except where explicitly stated all the work in this report, including appendices, is my own and was carried out during my final year. It has not been submitted for assessment in any other context.

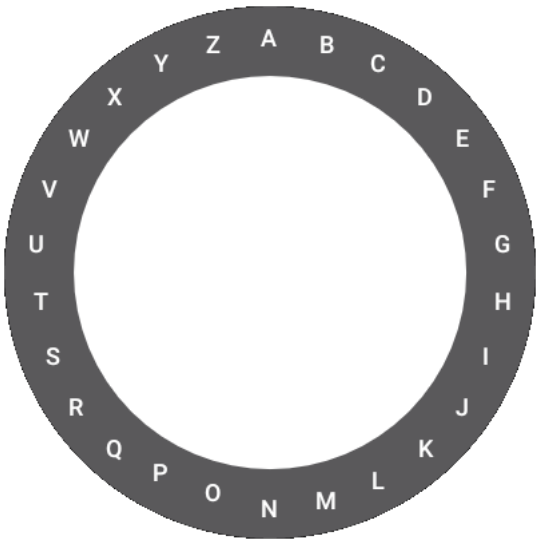
I agree to this material being made available in whole or in part to benefit the education of future students.

Signature: _____ Date: _____

Abstract

The size of smartwatch devices makes it difficult to enter text with them and the existing text entry methods are not often suited for devices with circular screens. This affects the usability and user experience of text entry on circular smartwatch devices. This work introduces a new circular keyboard design for circular smartwatches named Hoop Keyboard.

The Hoop Keyboard was evaluated using data gathered from a formal comparative user study involving a QWERTY keyboard. The data was created by recording the performance of 16 users as they completed a set of phrases for each keyboard. Each set of phrases contained 15 randomly selected phrases. This project will display and evaluate the results from the user studies, provide insight for future development as well as detailing the process of design and implementation of the keyboard.



Acknowledgments

I would like to thank my supervisor, Dr. Mark Dunlop, for all of his guidance & support throughout the entire project.

I want to also thank all of the staff and lecturers at The University of Strathclyde. Thank you for introducing me to the art of computer programming by allowing me to change my degree discipline from "Sport and Physical Activity" to "Computer Science" with no prior programming experience.

I would like to thank Lindsey for all her help & support throughout the duration of the project and beyond.

Furthermore, I want to thank all of the participants of the user studies for sacrificing their own precious time to assist in the results of this project.

Lastly, I would also like to thank my parents for their overwhelming support & patience throughout my entire university career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wearable electronic devices are becoming common gadgets worn by many people today with sales of smartwatch devices alone surpassing 30 million units worldwide for 2015 with a forecast of over 50 million units in 2016[1]. Text entry in the form of a keyboard on smartwatch devices is available through third-party applications, most of which, adopt a rectangular QWERTY key layout reduced in size to fit to the small screen of a smartwatch. However, when applied to a circular screen it is often found that the edges of the rectangular keyboard are cut off, therefore the keyboard requires to be moved from the bottom of the screen, thus using more of the limited screen space available.

The aim of this project is to develop a circular keyboard designed for circular wearable devices and to provide an analysis of the performance of said keyboard. The circular keyboard is named "Hoop Keyboard".

Users cannot be expected to type accurately on a small interface such as a smartwatch therefore, the Hoop Keyboard uses an advanced language model which predicts the key that was most likely intended to be pressed by the user based on the key pressed and its immediate neighbours. An auto-correction system was also integrated into the Hoop Keyboard to increase the accuracy of users typing. In order to measure the performance of the Hoop Keyboard; a comparative study against a rectangular QWERTY keyboard was undertaken.

This report will discuss, in detail, each stage of the project as well as demonstrating how this project is unique in relation to work that has already been done in the field.

[1]Gartner Wearable Device Sales Forecast 2016 - http://www.gartner.com/newsroom/id/3198018

Chapter 2 details the background research that was conducted before implementation and highlights the related work that has already been done.

Chapter 3 specifies the project itself and provides a clear description of the problems being addressed, as well as the approach taken during the development of the system.

Chapter 4 details the design approach and the system architecture along with the decisions that were made in order to achieve the ultimate goal of the project.

Chapter 5 provides a detailed insight into the software design as well as the specific details of design decisions that were taken. This section also highlights specific challenges that arose and the process of deriving a solution to solve them.

Chapter 6 details the verification, validation and testing methodology used throughout the project to ensure that it was successful.

Chapter 7 outlines the results of the user studies undertaken and provides an in depth evaluation of the of the Hoop Keyboard.

Chapter 8 will provide a summary of the performance and success of the project as a whole, highlight the potential of future development, and lastly form a conclusion from the main points in the report.

Chapter 2

Related Work

The purpose of this chapter is to identify the work that has already been conducted in the field as well as the research performed before and during the undertaking of this project. This chapter also illustrates how the project links with the previous work and the underlying technology used to create the practical components of the project.

2.1 Transparent User guided Prediction (TUP)

Morten Proschowsky's thesis on "Adaptive Text Entry for Mobile Devices" introduces transparent user guided prediction (TUP). TUP is a novel text entry method for devices without keyboards and was developed to allow easier text entry on small devices with touch sensitive wheels (Proschowsky, 2008). Proschowsky provides a description of the language modelling techniques that he used to predict which letter was most likely from the tapped letter and it's neighbours. The implementation of TUP was carried out on the iPod Classic device by Apple Computers, Inc using the touch wheel as a circular keyboard for text entry as shown in Figure 2.1 below.

TUP was identified as very closely related work from the outset and formed a large portion of the background study for this project. The design of the Hoop keyboard was largely taken from TUP due to success of the TUP alphabetical circular layout on the iPod device. Proschowsky included a space key, keys for special characters and a mode selection in the circular keyboard as shown in Figure 2.1.

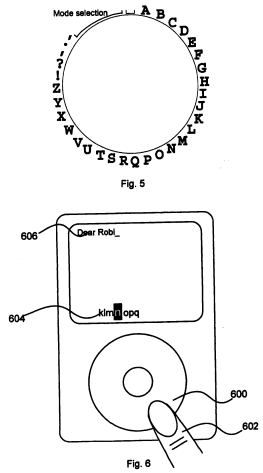


Figure 2.1: Image from the TUP United States of America (USA) patent application (Cui & Proschowsky, 2006).

2.2 YourText

Morten Proschowsky developed a new language model framework for TUP called "YourText". It is a context aware language model based on a set of prediction by partial match (PPM) models and it is a powerful language model created by combining different language models (Proschowsky, 2008). Proschowsky's implementation uses a novel adaptive language model called PPM*D where:

> "The PPM* refers to the compression method by Cleary, Teahan, and Witten [1995], where variable length n-graphs are used. The D refers to a decay function that is built into the language model. The decay function will reduce the counts of the characters, so that characters that have not been used recently will be less likely." (Proschowsky, 2008)

Adaptive language models tend to only be able to remember words previously mentioned but YourText is able to also forget with the addition of the decay function (Proschowsky, 2008). The ability for a language model to forget is just as important as the ability to remember since letter sequences that were previously very common may no longer be relevant. TUP and YourText were combined by Proschowsky to form his final keyboard implementation with a final estimated text entry speed of 10.5 words per minute (WPM) and an error rate of 5% (Proschowsky, 2008).

2.3 VelociTap

VelociTap is a state-of-the-art touchscreen keyboard sentence-level decoder developed by Keith Vertanen. The VelociTap keyboard uses a combination of a probabilistic keyboard model, a character model, and a word language model to produce a sentence-level decoder (Vertanen *et al.* , 2015). The decoder takes a sequence of taps from a user and searches for the most likely sentence given the sequence of touch events. VelociTap allows users to seamlessly choose from three word-delimiter actions: tapping the space key, swiping to the right or omitting spaces and leaving the decoder to insert spaces (Vertanen *et al.* , 2015). When compared to Google's Android keyboard VelociTap has a significantly lower error rate while retaining the same entry rate (Vertanen *et al.* , 2015). Vertanen shows that novice users can reach an entry rate of 41 WPM on a smartwatch-sized keyboard (see Figure 2.2) using VelociTap with a character error rate of 3% (Vertanen *et al.* , 2015). The text entry speed on VelociTap is significantly faster that the mere 10.5 WPM found on Proschowsky's TUP keyboard and the error rate is also lower with just 3% compared to 5% on TUP.

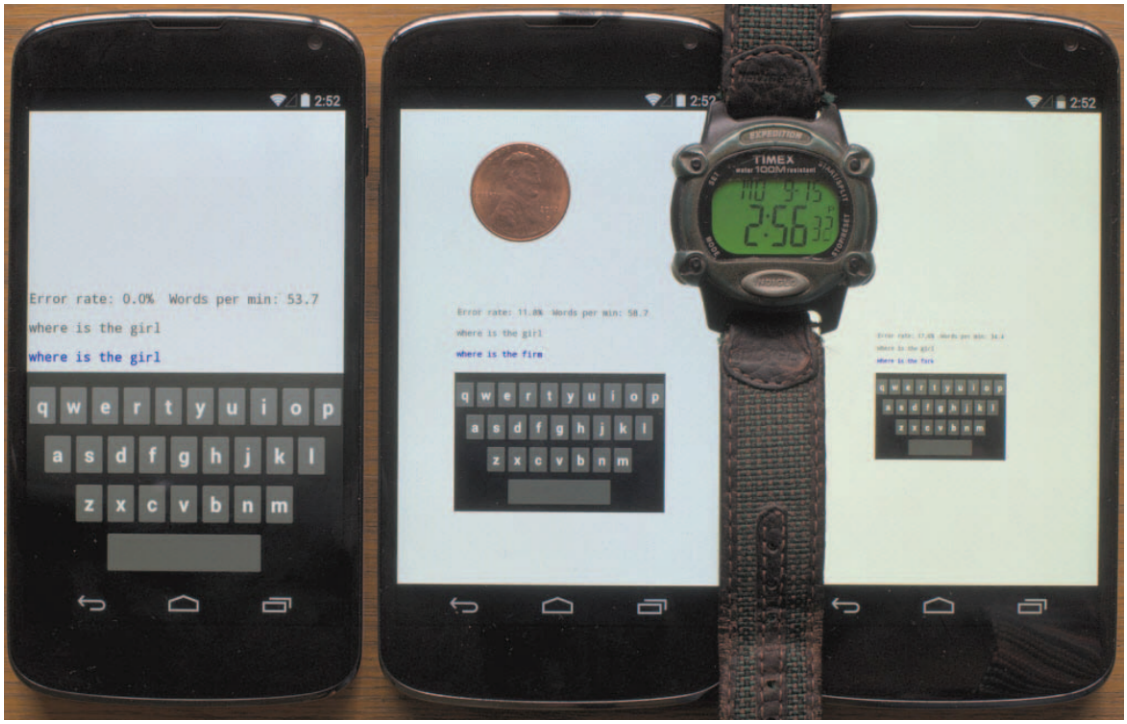


Figure 2.2: Three variations of the VelociTap keyboard layout on Nexus 4 devices. A penny and a Timex Expedition watch have been added for scale (Vertanen *et al.* , 2015).

2.4 Minuum Keyboard

The Minuum Keyboard[1] by Whirlscape Inc is a commercial QWERTY soft keyboard for Android and IOS devices. Whirlscape have also built a pre-release Android Wear version for Android smartwatches. Early access to the pre-release version was gained through the Minuum smart watch mailing list. The pre-release version of the Minuum keyboard input method editor (IME) is a simple QWERTY keyboard layout similar to the layout found on standard QWERTY keyboards on smartphones (see Figure 2.3). Minuum claim that after an hour of use, new users are able to type at speeds of 25 WPM on a standard smartwatch[2]. Text entry speed on the Minuum is significantly slower than the similar VelociTap keyboard which has proven to achieve 41 WPM as mentioned previously.

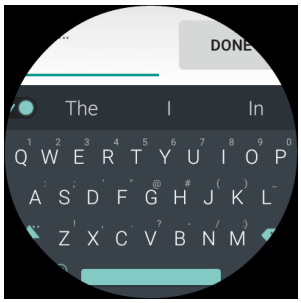


Figure 2.3: Screenshot of the Android Wear Minuum Keyboard with default settings.

2.5 English Letter Frequency Counts

In the field of computational linguistics, letter frequency counts is of utmost importance in the processing of a natural language. The frequency counts can be utilised to calculate the probability of a character occurring in a given sequence, therefore a prediction of the most likely character intended can be formed.

Peter Norvig the Directory of Research at Google (at time of writing) continued the research started by Mark Mayzner by using the Google Books corpus in order to gather more accurate data for the English letter frequency counts (Mayzner & Tresselt, 1965). Norvig consulted the 2012 English Version of the n-gram raw data set for the Google Books data[3] which provides word counts of the number of times each word appears in the books scanned by

[1]Minuum Website - http://minuum.com/
[2]Minuum Website - http://minuum.com/category/wearable/
[3]Google Books N-gram Data - http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

Google (Norvig, 2013). Norvig condensed the entries by combining the counts for all years, ignoring entries that used characters other than the 26 letters A-Z, treating words with different capitalisation as a single word and discarding any word with fewer than 100,000 mentions (Norvig, 2013).

The distillation of the Google Books n-gram data generated by Norvig resulted in 97,565 distinct words mentioned a total of 743,842,922,321 times (Norvig, 2013). This was 37 million more mentions than the 20,000-mention collection used by Mayzner (see Appendix A.1.2) (Norvig, 2013). The open source research conducted by Norvig also includes letter counts with a total of 3,563,505,777,820 letters mentioned (see Appendix A.1.1) and n-gram data from unigram to 9-gram as shown in Figure 2.4 (Norvig, 2013).

The n-gram data[4] sourced from Norvig's research form the basis of the underlying language model design for the Hoop keyboard. The advanced language model analyses the n-gram probabilities of the key pressed and its neighbours to determine the most likely character intended by the user. A complete description of the advanced language model applied to the Hoop keyboard can be found in chapter 5.2.

1	2grams	3grams	4-grams	5-grams	6-grams	7-grams	8-grams	9-grams
e	th	the	tion	ation	ations	present	differen	different
t	he	and	atio	tions	ration	ational	national	governmen
a	in	ing	that	which	tional	through	consider	overnment
o	er	ion	ther	ction	nation	between	position	formation
i	an	tio	with	other	ection	ication	ifferent	character
n	re	ent	ment	their	cation	differe	governme	velopment
s	on	ati	ions	there	lation	ifferen	vernment	developme
r	at	for	this	ition	though	general	overnmen	evelopmen
h	en	her	here	ement	presen	because	interest	condition
l	nd	ter	from	inter	tation	develop	importan	important

Figure 2.4: The 10 most common n-grams from unigram to 9-gram (Norvig, 2013).

2.6 Other Keyboards

2.6.1 ZoomBoard

ZoomBoard[5] is keyboard that allows text entry on ultra-small devices such as smartwatches. The layout of the ZoomBoard is based on the popular QWERTY layout and uses an iterative zooming technique to enlarge otherwise impossibly tiny keys to a comfortable size as shown by Figure 2.5 (Oney *et al.* , 2013).

[4]Norvig's N-gram Data - http://norvig.com/tsv/ngrams-all.tsv.zip
[5]ZoomBoard - http://www.chrisharrison.net/index.php/Research/Zoomboard

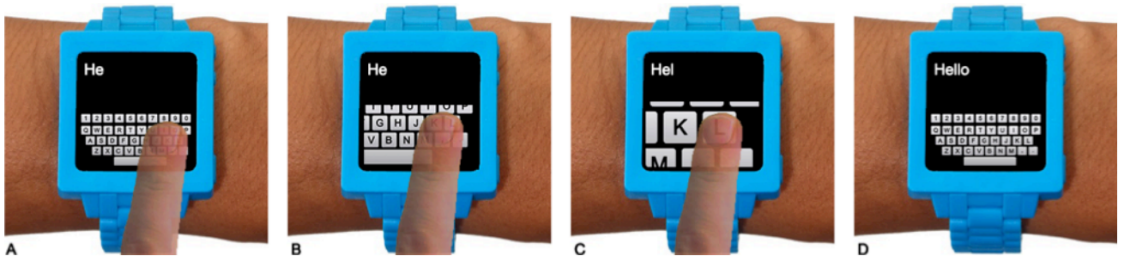


Figure 2.5: The ZoomBoard on a watch-sized device. The keyboard starts fully zoomed out (A) and when a key is press by the user, the keyboard zooms in iteratively (B & C), until the key is of a large enough size to be pressed accurately. Once the selected character has been entered the keyboard is reset to its default state (D) (Oney *et al.* , 2013).

ZoomBoard makes use of swiping motions by allowing users to swipe to the left to delete, to the right for a space and swipe up to change the keyboard to symbols. The results from the ZoomBoard user study showed that an average of 9.3 WPM was achieved with an error rate of 7% (Oney *et al.* , 2013). Proschowsky's TUP & YourText had better overall performance than the ZoomBoard with 10.5 WPM and an error rate of 5% showing that a circular layout can produce better performance than some implementations of the QWERTY layout.

2.6.2 Wear Messages

Wear messages by Appfour, also known as "Messages for Android Wear"[6] is a commercial text messaging application for Android Wear devices using a QWERTY keyboard as shown in Figure 2.6. Appfour have created a range of similar applications that use the swiping QWERTY keyboard such as email and web browser apps. The keyboard shown in Figure 2.6 illustrates an alternative design of the QWERTY keyboard to the design found on the ZoomBoard keyboard.

Figure 2.6: The QWERTY keyboard found on appfour applications. The keyboards default state is with the left third of the keyboard in view (left). The rest of the keys are available by swiping to the left (middle & right) and can be selected by tapping on a letter.

[6]Messages for Android Wear - https://play.google.com/store/apps/details?id=com.appfour.wearmessages

2.6.3 TouchOne

The TouchOne[7] smartwatch keyboard by Infiniti Technology features an award winning patented design for circular smartwatches. The TouchOne keyboard is currently in development and is at the beta phase of construction. The design is inspired by the T9 predictive keyboard (see Figure 2.7) and the company claims that text entry speeds of 32 WPM can be expected [8]. The keyboard features 8 keys around the edge of the screen and uses the swipe to the left action to delete and a tap on the central area to insert a space. With comparison to the similar TUP keyboard, the TouchOne keyboard is expected to outperform TUP by a considerable amount since text entry speeds on TUP managed 10.5 WPM as mentioned previously. The TouchOne keyboard is still not as fast as VelociTap as Keith Vertanen's keyboard outperforms the TouchOne keyboard by 9 WPM.

(a) Picture of Nokia 3210 mobile phone featuring the original T9 keyboard layout.

(b) The TouchOne smartwatch keyboard.

Figure 2.7: Illustration of TouchOne's adaptation of the original T9 predictive keyboard.

&hl=en_GB

[7] TouchOne Website - http://www.touchone.net/

[8] Tech in Asia article on the TouchOne keyboard - https://www.techinasia.com/touchone-smartwatch-keyboard

Chapter 3

Problem Description and Specification

In this chapter a detailed description of the problem will be provided along with a specification for the problem. An insight into the development approach taken to complete the project will also be discussed.

3.1 Problem Description

The default text input for Android Wear devices is a hands free voice input method provided by Google as standard. In Android Wear messaging applications such as Google's Hangouts application[1] users can only assemble custom messages using voice commands or they have to use pre-set messages in the form of emojis and fixed text such as "OK" (see Appendix L.1). Users of the Hangouts Wear application will not be able to send custom messages without allowing the intended text to be heard by members of the public that are within earshot. This results in a privacy problem therefore users tend to use their smartphones to send sensitive messages instead. Another issue with voice commands is the users accent. From personal experience, it has been found that the Scottish accent does not work well with voice input since resulting messages tend to get lost in translation. Background noise is also an issue with voice commands as the voice recognition software often gets confused when used in loud environments.

The solution to the default Android Wear OS input methods are keyboards. There are a range of keyboards available for Wear devices in the form of third party applications, some

[1]Google's Hangouts Android Application - https://play.google.com/store/apps/details?id=com.google.android.talk

of which have been previously introduced in Chapter 2. Most of the keyboards available for Wear devices are simply mobile QWERTY keyboards that have been reduced in size to fit on a smartwatch screen. The mobile QWERTY keyboard is designed for large rectangular devices which causes a problem when used on a small circular smartwatch screen.

The Hoop keyboard has been designed to remedy the issue of text entry on circular smartwatches such as the Huawei watch[2] (see Appendix G). A circular keyboard is the ideal input method for circular smartwatches since it allows users to send private messages as well as making maximum use of the small usable screen space available.

The most critical requirement for keyboards is performance. It is crucial that keyboards are responsive to users' taps and the overall textual input experience is seamless. In order to create a good user experience, computation needs to be optimal especially on devices with limited processing power & memory like smartwatches.

3.2 Problem Specification

A number of tasks were initially set for the completion of the project. A custom circular keyboard design was required to layout the keys in a circle around the edge of the screen. Due to the increasing number of circular Android Wear devices it was important to design an adaptive layout that responds to any given device. The keys should be as large as possible and by using the outer edge of the watch screen it will enable the keys to be bigger which will increase typing accuracy. The text that is being typed will appear in the inner circle (i.e. inside the circular keyboard). Allowing the user to use swiping gestures to insert spaces and to remove characters from the inner input box was also identified as a functional requirement.

The design of an advanced language model was necessary in order to help users type accurately on the keyboard. The keyboard buttons are very small, therefore users cannot be expected to type accurately meaning a good language model design was needed in order to pick the most likely character intended by the user based on their tap position.

A spell checker framework was also required in order to help correct users' typing errors. There are many third-party spell checkers available such as Aspell[3] but there is very little support for Android Wear devices, therefore a custom spell checker was required for the project.

[2]The Huawei Watch - http://consumer.huawei.com/minisite/worldwide/huawei-watch/
[3]Aspell Website - http://aspell.net/

Performance optimisation and data management was another major requirement for the project to be a success. The amount of dictionary data & letter sequence data required for a keyboard is massive (over half a million entries) and the amount of data increases the more complex the language models & spell checkers become. Android Wear devices such as the Huawei Watch only have 512 megabytes (mb) of random access memory (RAM) (see Appendix G.2) which means storing the data in RAM will result in a sluggish and unresponsive application. The system requires a large memory reserve with the addition of super-fast data retrieval speeds.

The addition of a NoSQL next generation database was an ideal solution to data storage and retrieval problems within the system. The System Architecture & Design and Detailed Design & Implementation chapters will provide an in-depth explanation of the usage of NoSQL within the project.

3.3 Development Approach

The development of this project required a great deal of research, experimentation and analysis. The plan from the outset was to develop 3 prototypes, starting with a keyboard with no language model and building up to a keyboard with an advanced language model (see Appendix C.1).

Having no prior experience with Android development the first stages of the project were used to create a better understanding of the concepts of Android application development and the Android Studio integrated development environment (IDE)[4]. By examining a range of sample Android applications provided by Google[5] and by exploring the Android Application Programming Interface (API)[6], an enhanced understanding of the technology was quickly formed.

In terms of experimentation the initial plan of building 3 separate prototypes was adhered to as shown by the project plans in Appendix C. Each prototype was used as a platform for the subsequent prototype eventually leading to a combination of the prototypes for the final implementation. The Hoop Prototype 2 & the Hoop Keyboard were evaluated in separate user studies which allowed analysis to be conducted to identify possible improvements for the later prototypes. More on evaluation and analysis can be found in Chapter 7.

[4]Android Studio Website - http://developer.android.com/sdk/index.html
[5]Android Samples - http://developer.android.com/samples/index.html
[6]Android API - http://developer.android.com/reference/classes.html

Chapter 4

System Architecture & Design

This chapter details the overall design approach for the system, as well as a high level overview of the purpose of each component used in each of the prototypes developed.

4.1 Design Methodology

4.1.1 Software Development Methodology

Agile Scrum Methodology

The software development approach taken to develop the Hoop Keyboard was the "Agile Scrum Methodology". The concept of Scrum was first introduced to me when attending the 2015 Spring week event at J.P. Morgan in Glasgow and it was also enforced by the university when representatives of Baillie Gifford conducted a lecture on Agile project management. The theory of Scrum was founded on empirical process control theory asserting that knowledge comes from experience and decision making is based on what is known (Schwaber & Sutherland, 2013). In order to optimise predictability and control risk, Scrum uses an iterative, incremental approach (Schwaber & Sutherland, 2013). Ken Schwaber and Jeff Sutherland give a simple definition of Scrum as:

> "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value." (Schwaber & Sutherland, 2013)

Agile methodologies, as opposed to more traditional software development approaches such as the Waterfall methodology, focus on producing the required functionality within a given time frame rather than attempting to develop a model in preparation for potential changes in the future.

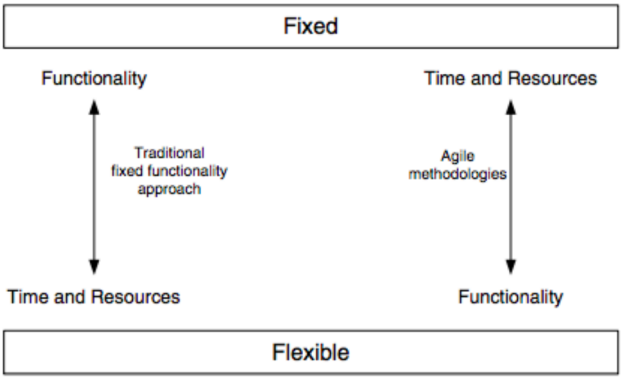


Figure 4.1: The comparison of Agile to Traditional Methodologies (Hunt, 2006).

Figure 4.1 illustrates the clear differences between Agile and Traditional approaches. The reversed model of Agile is well-suited to the requirements of the complete project as time is fixed in the form of a deadline and the resources available are also fixed as the task is an individual project. The approach to functionality also took the Agile method since the flexible implementation of the Hoop Keyboard facilitates expansion and effortless changes in the functional requirements. Therefore, new ideas, generated from background research or supervisor discussions (see Appendix D), were able to be applied to the final product without affecting the underlying architecture of the application.

Trello

Trello[1] was utilised in order to manage my Scrum boards. Trello is a popular tool used to manage tasks for projects of any size and is particularly useful for Scrum boards. By delivering fortnightly short iterations known as "Sprints" it was easy to demonstrate the progress of each planned task (see Appendix P). Generally sprints are of short lengths ranging from weekly to monthly and it is rare to use sprints with a length longer than a month.

By delivering fortnightly iterations of sprints I was able to clearly illustrate my progress to my supervisor and easily identify lessons & improvements for following sprints.

[1]Trello Website - https://trello.com/

By delivering fortnightly iterations of sprints, progress was able to be clearly illustrated to my supervisor. Lessons and areas for improvements were also able to be identified easily for following sprints.

4.1.2 Alternative Designs Considered

An important decision that was required to be made at the outset of the project was whether to build the Hoop Keyboard as an application specific keyboard or as a device global input method editor.

Input Method Editor (IME)

An example of an Android Wear IME is the Minuum keyboard[2] previously introduced in Chapter 2.4. One of the advantages of an IME is that the keyboard is able to be used throughout the device in all applications that have an input box. However, this becomes redundant for Android Wear devices due to the fact that standard Wear applications do not utilise the EditText[3] widget which allows users to insert text into a text area. There are third-party applications available on the Google Play Store[4] that incorporate an EditText widget such as the previously mentioned "Wear Message" application by Appfour, but these applications are few in number.

Installation of IMEs on Android Wear devices is awkward to say the least. Android Mobile devices are able to easily change the IME keyboard in the settings applications of the device, however the same support for Wear devices does not exist in the Wear settings application. Installation of an IME on a Wear device currently requires significant technical knowledge of how to operate the Android Debug Bridge (adb) and only a single IME can be installed at any given time. However, even though IMEs on Android Wear are possible, there is little support and little use for IMEs due to the design of the current underlying operating system.

Conclusion

The disadvantages of an Android Wear IME supersede the advantages due to the installation complexity of the technology. An application specific keyboard on the other hand can be

[2]Minuum Keyboard Website - http://minuum.com/
[3]Android EditText Documentation - http://developer.android.com/reference/android/widget/EditText.html
[4]Google Play Store - https://play.google.com/store?hl=en_GB

easily installed on the device through the support of an accompanying mobile application. The process of the user study also had to be taken into account at this stage since the process of comparing two IMEs on a single Wear device would be significantly more time consuming than comparing an application keyboard against an IME. This due to the fact that two IMEs cannot be enabled at the same time and would require manual changes within the adb shell. The initial plan was to compare my implementation of a circular keyboard against the QWERTY implementation of the Minuum IME. Therefore, the obvious choice was to build the Hoop Keyboard as an application specific keyboard.

4.1.3 Log Book

A WordPress[5] blog was used to keep a record of my personal progress and to document all the activity during the course of the project (see Appendix Q). The online blog allowed effortless logging & editing of the tasks and achievements with the advantage of being available for interested parties to view.

4.1.4 Version Control

To maintain version control over the course of the project, a Git[6] repository was used. GitHub[7] was used to store the remote repository since previous knowledge of the technology was gained from undertaking projects in past years such as the "GizmoBall" project in my 3rd year (see Appendix R).

4.2 User Interface

The basic overall design of the user interface for the Hoop Keyboard was defined by problem specification detailed in Chapter 3.2. The aim for the general look and feel of the keyboard was to keep the interface as simplistic as possible with the keys spread out in a circle around the edge of the screen (see Appendix L.2). The background colour of the keyboard was chosen to mimic the default colour of buttons on Android Wear devices so that the text on the keyboard buttons were clear and legible for the average user. It became apparent that

[5]WordPress Website - https://wordpress.com/
[6]Git Website - https://git-scm.com/
[7]GitHub Website - https://github.com/

it was not necessary to provide buttons for spaces & backspaces and that swiping motions could be utilised instead like the aforementioned VelociTap keyboard (i.e. swipe right for space and swipe left for backspace). The same could be said for special characters since there was no real reason to include them on the main keyboard. Therefore a menu was implemented that contained a range of special characters from "." to "!". The menu is accessible by swiping down from the top of the screen (note: This feature was disabled in test mode). The letter case can also be changed by swiping up from the bottom of the screen. The state transition network (STN) for the final Hoop Keyboard application can be found in Appendix N.

4.3 Data Management

One of the most challenging tasks of the project was to incorporate a technology capable of storing masses amounts of data. The advanced language model utilises a vast amount of n-gram probability data (243666 n-grams) along with a spell checker requiring 314358 unique words. With a combined total of 558024 possible objects it was clear that the data required to be stored within some sort of key to value data structure. A HashMap in Java is the go-to data structure for storing unordered data in a key to value representation. Unfortunately due to the huge dataset required, the amount of RAM available on Wear devices (512mb) is not capable of storing the amount of memory required. After extensive research, the solution to the data management problem was using a database and in particular a NoSQL (shorthand for "Not Only SQL") database called "SnappyDB".

4.3.1 SnappyDB

SnappyDB[8] is a key to value non-relational database for Android devices used in order to implement a NoSQL approach.

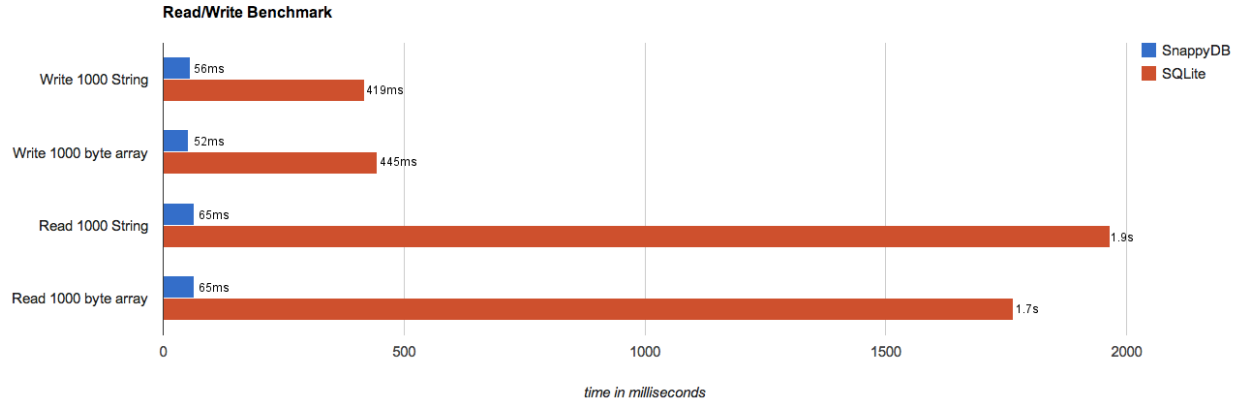


Figure 4.2: Illustration of the read/write benchmark comparison between SnappyDB and SQLite[9].

Figure 4.2 clearly demonstrates the speed performance benefit of the SnappyDB over the default SQLite[10] database that comes as standard with Android. The advanced language model requires super-fast data retrieval speeds in order to make a real-time decision thus enhancing the argument that SnappyDB is an ideal choice of data management. One of the main attractions of SnappyDB is that it is based on Google's LevelDB[11] database which in turn is used by the Google Chrome web browser[12]. LevelDB makes use of Google's very fast compression/decompression algorithm named Snappy[13] which is also used in SnappyDB.

The down side to SnappyDB and NoSQL databases in general is the fact that the data is unstructured and can only be used to stored key to value data. However, within this project, structured data is not necessary so the limitations of NoSQL does not affect the implementation of the Hoop Keyboard. Another disadvantage of SnappyDB in particular is the limited option of a local database. This means that the database can only be updated through the release of a new version of the whole application.

[8]SnappyDB GitHub Repository - https://github.com/nhachicha/SnappyDB
[9]SnappyDB vs SQLite Benchmarks - http://snappydb.com/img/benchmark_sqlite_with_transaction.png
[10]Android SQLite - http://developer.android.com/reference/android/database/sqlite/package-summary.html
[11]Google's LevelDB Website - http://leveldb.org/
[12]Google Chrome - https://www.google.com/chrome/
[13]Google Snappy Library - http://google.github.io/snappy/

4.3.2 Alternatives

SQLite

SQLite is a great option for storing relational data in a database management system (DBMS). Structuring of the data can be presented in ordered tables which is required by many applications but unfortunately does not apply to the circular watch text project. Due to the overwhelming performance benefit of SnappyDB in comparison to SQLite shown in Figure 4.2 then there is a clear indication that the overall performance of the keyboard would have decreased dramatically if the SQLite DBMS was used.

Couchbase-Lite-Android

Couchbase[14] is a NoSQL alternative to SnappyDB which offers the same performance benefits as SnappyDB but with added implementation complexity. Couchbase offers a fantastic product in their Android package and it was a difficult decision to choose between Couchbase and SnappyDB for the project. The main benefit of Couchbase is the option to use the Couchbase Sync Gateway & Server to keep the local database up to date with a remote database without the need to update the version of the application. However, the requirement of the NoSQL database for the Hoop Keyboard was simply a no-frills local database containing the required data for the underlying language model and spell checker framework. This reliability is exactly what SnappyDB offers, however the potential to use the Couchbase database in the future remains a serious possibility especially if the Hoop Keyboard was commercialised.

4.4 Prototypes

In total, three prototypes were constructed. Prototype 1 was constructed to provide a basic shell of the Hoop Keyboard for future prototypes. Prototype 2 applied a simple language model to Prototype 1. Prototype 3 was a development of the language model found in Prototype 2 to apply more advanced techniques. All of the prototypes were built with accompanying Mobile applications as well as a Wear application for simple installation.

[14]Couchbase Mobile - http://developer.couchbase.com/mobile/

4.4.1 Overall Structure

The Mobile and Wear applications for each prototype are linked together by using the same package name e.g. "com.camerondrennan.hoopkeyboard". This way, data & messages can be transferred between the devices easily through Bluetooth using the Android Wearable Data API[15]. Figure 4.3 demonstrates the process of sending data from a Android Mobile device to a paired Android Wear device. The process is simply reversed in order to send data from the smartwatch to the handheld device.

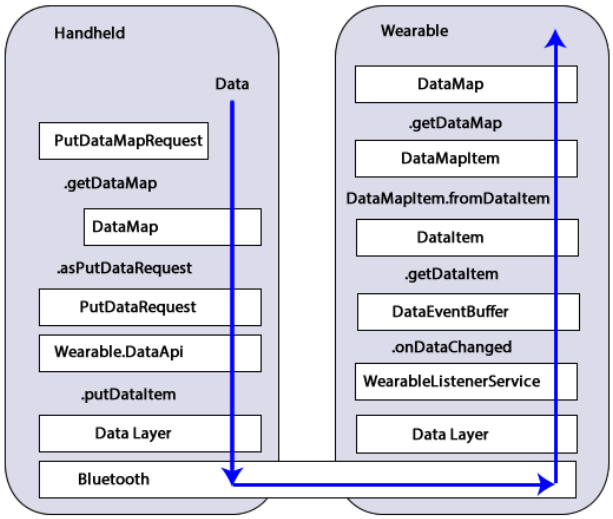


Figure 4.3: Illustration of the data flow process of sending data from a handheld device to a wearable device[16].

[15]WearableListenerService API - https://developers.google.com/android/reference/com/google/android/gms/wearable/package-summary

[16]Data Layer DataMap Objects - http://android-wear-docs.readthedocs.org/en/latest/data.html

4.4.2 Hoop Prototype 1: No Language Model

The Hoop Prototype 1 was used as learning process as a basis for more complicated prototypes. The prototype is simple a implementation of a circular keyboard that provides no predictive language model. The basic concepts of text entry via the keyboard were addressed and the design of the prototype can be seen in Figure 4.4.

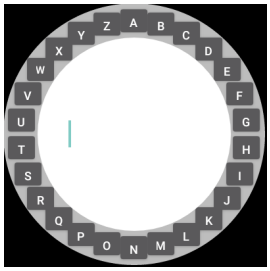


Figure 4.4: Screen shot of the Hoop Prototype 1 keyboard.

4.4.3 Hoop Prototype 2: Simple Language Model

The simple language model used for the Hoop Prototype 2 Keyboard was a bigram character language model. The bigram model was a suggestion brought up during a supervisor meeting and was presented as a good starting point by my project supervisor.

A bigram character language model is made up of unigrams & bigrams and the probability of their occurrence given a corpus of data. The term "unigram" refers to a n-gram of size 1 and "bigram" is a n-gram of size 2. Given that a character based language model was used then, the unigrams were simply made up from the 26 letter alphabet. Character bigrams are 2 consecutive letters that appear next to each anywhere in a word. For example the word "letter" has 5 bigrams: "le", "et", "tt", "te", "er" each bigram occurs on therefore, they each have an equal probability of 20% ($\frac{1}{5}$). The word "letter" also contains 4 unique unigrams: "l", "e", "t", "r" where probabilities of each letter are not equal:

- $p(\text{“l”}) = \frac{1}{6}$
- $p(\text{“e”}) = \frac{2}{6}$
- $p(\text{“t”}) = \frac{2}{6}$
- $p(\text{“r”}) = \frac{1}{6}$

The same process can be done n number of times to create an n-gram character model (Manning & Schütze, 1999).

The unigram & bigram data used for the simple language model of the Hoop Prototype 2 keyboard was the data created by the aforementioned Peter Norvig. The total number of

unigrams & bigrams used was relatively low with 702 n-grams in total. After retrieving the data from Norvig's fusion tables[17] the task of calculating the probabilities for each n-gram was relatively straight forward. The data was then exported to an XML document for use within the Android application as String values for fast retrieval speeds.

The language model was then applied to each button press by the user by comparing the probability of the character pressed to its immediate neighbours. The most likely n-gram was calculated based on their probabilities and the winning character was then inserted into the text.

The design of the Hoop Prototype 2 keyboard was similar to the design found in the first prototype and can be found via Appendix L.4.

4.4.4 Hoop Keyboard: Advanced Language Model

The advanced language model of the Hoop Keyboard expands on the bigram model developed in Hoop Prototype 2. The Hoop Keyboard adopts an n-gram model that is detailed in Chapter 5 Section 5.2. Figure 4.5 illustrates the design of the final implementation of the Hoop Keyboard.

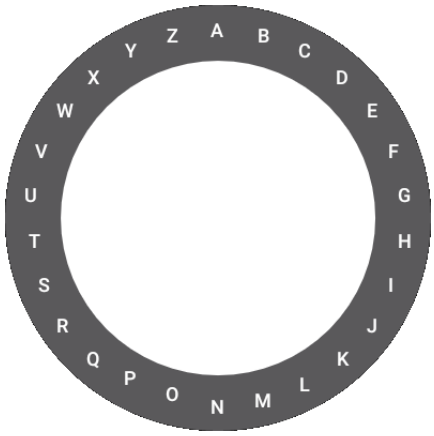


Figure 4.5: Screen shot of the Hoop Keyboard.

[17]Peter Norvig's N-gram Fusion Tables - https://docs.google.com/folder/d/0BxQoRSCXdeTTdEROWXVuaTQ5QkE/edit

Chapter 5

Detailed Design & Implementation

This chapter will illustrate the specific process of the design and implementation of project. The most demanding aspects of the project will also be examined. This chapter will also detail the justifications for the design choices and will describe the implementation process of the intricate components within the project.

5.1 Software Architecture

A software architectural pattern which is heavily integrated within Android is Model View Controller (MVC) (Gamma *et al.* , 1995). MVC is the process of splitting the software into three independent components: "Model", "View", & "Controller" in order to increase flexibility, adaptability and provide greater encapsulation of the software. The natural choice of architectural pattern for the Hoop Keyboard was MVC due to the underlying structure of the Android framework. The class diagrams in Appendix B illustrates the usage of MVC within the Mobile and Wear modules of the Hoop Keyboard application. In the context of the Hoop Keyboard (i.e. the Wear module), the Model component consists of the LanguageModel, SpellChecker, and the WearDatabase classes along with their respective interfaces. These classes provide the underlying logic of the application. The Controller component of the Hoop Keyboard is represented by the WearActivity class and the IWearKeyboard interface. The Controller handles the interactions between the View and the Model along with the handling of the user input. The View component in Android is represented by XML files that determine the layout of the user interface. The Hoop Keyboard does utilise a custom circular layout that is implemented in the CircleLayout class which represents the View.

5.2 Advanced Language Model

The advanced language model used for the final implementation of the Hoop Keyboard was a n-gram character language model with a spell checker (Manning & Schütze, 1999). The n-gram model was chosen due to the ease of building on top of the simple language model already developed.

5.2.1 N-gram Data

The n-gram data used for the language model of the Hoop Keyboard was the n-gram data created by aforementioned Google's Head of Research "Peter Norvig". As mentioned, Norvig used the Google Books n-gram data to generate the n-gram frequencies and provided them open source on his personal website[1]. The British National Corpus[2] (BNC) was an alternative source of data that could have been used for the language model. I decided to choose the corpus from Google Books over the BNC because the Google Books corpus was greater in volume and Norvig had already generated the n-gram data.

The initial task was to collate all of the 243,666 unique n-grams and their probabilities in to a single XML file for use in the Android Wear application. I initially made the XML file using String resources for the n-gram data and placed the file in the value resource folder within the Wear module. This resulted in a compilation error due to the application having too many constants. Further investigation in Oracles Java documentation[3] found that the "constant_pool_count" has a size limit of "u2" where u2 = 2 unsigned bytes, therefore the size limit of the number of constants in Java is 65,536. The n-gram data far exceeded the limit so it was necessary to find another location to store the large data set.

Android applications have an XML folder within the resource directory which allows fast and easy data retrieval from an XML file. After experimenting with the XML folder it became apparent that data in this directory gets compiled when the project is built. The one million lines of XML n-gram data took around three hours to compile which was not feasible. Further research into the matter found that the assets folder does not get pre-compiled by Android applications and works like a file system making it the ideal location to store the n-gram data. The compilation completed within 15 seconds after the data was simply moved into the assets directory.

[1]Peter Norvig's Website - http://norvig.com/mayzner.html
[2]British National Corpus Website - http://www.natcorp.ox.ac.uk/
[3]Oracle Documentation - https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html

5.2.2 Spell Checker

It was made aware to me by my supervisor that there are a number of 3rd-party open source spell checkers available. Having said this, I did however come across an essay on "How to Write a spelling Corrector" by Peter Norvig[4] when searching for his n-gram data. It was there that the SymSpell[5] spelling corrector was found as well as Rael Cunha's Java implementation[6] of Norvig's tutorial.

The SymSpell spelling corrector was applied to my Android Wear application but immediate problems were found due to the limited memory available on the Wear device. SymSpell stored the generated dictionary in a HashMap and the watch was unable to cope with the memory allocation required. It was realised that the SnappyDB database that was already implemented could be used to store the dictionary data. The SymSpell code was very complex and required a significant amount of time and effort to convert the framework to use the SnappyDB as storage. Therefore the spell checker was changed from SymSpell to Rael Cunha's spelling corrector since Rael's implementation was far easier to adapt and integrate into the current Wear application.

Like SymSpell, Rael also used a HashMap to store the dictionary so the same challenge was apparent but on a simpler scale. An email was sent to the developer Rael outlining my intention of applying an adapted version of his spell checker within my Wear application. Rael then replied with a faster version of his spelling corrector as an attachment and provided consent to use his code.

The adaptation of Rael's spell checker was relatively straightforward and since the database was already filled with n-gram data, with all the keys in uppercase, it made sense to insert the dictionary entries as lowercase. This allowed all the data to be stored in a single NoSQL database and removed the possibility of the dictionary data overwriting the n-gram data due to symmetrical keys.

The adapted spelling corrector was implemented using the insertion of a space as a word delimiter. When users swipe to the right, the immediate sequence of characters preceding the cursor will be attempted to be corrected. If the word is spelled correctly then no corrections will take place. However, if the word is not spelled correctly and the spelling corrector has a good suggestion, then the word will be corrected.

[4]Peter Norvig "How to Write a Spelling Corrector" - http://norvig.com/spell-correct.html
[5]SymSpell GitHub Repository - https://github.com/wolfgarbe/symspell
[6]Rael Cunha's Spell Correct - http://raelcunha.com/spell-correct/

5.2.3 Data Transfer & Storage

As mentioned, the Hoop Keyboard uses the SnappyDB NoSQL database in order to store the n-grams and their probabilities alongside the words in the dictionary and their number of occurrences. The number of key to value entries in the database totalled 558,024. The recommended storage location of the database was in the internal storage[7] of the Wear device since the data in this location is private and is only accessible from the application. The initial approach taken was to fill the database on the first launch of the app after installation by reading the data in from the assets folder, meaning that any subsequent launches did not have to fill the database. However, it took a considerable amount of time (about 10 minutes) to build the database on the first launch of the application.

In order to reduce the amount of processing required from the Wear device on the first launch of the application, the top priority was to remedy the problem with filling the database. An alternative solution was devised of pre-creating and pre-filling the database on the mobile device and then manually copy the database files into the assets folder of the Wear application. Then on the first launch of the app the pre-created database files were simply copied from the assets folder to the internal storage of the Wear device.

A side project called "Pre-Create-NoSQL-Database" was built in order to generate a SnappyDB NoSQL database that was pre-filled with all the n-gram & dictionary entries required. The database was saved to the external storage of an Android mobile device. Mobile devices generally have more processing power than Wear devices hence the reason the database was created with a mobile application then copied over to the Wear application. Also there is no simple way to access the internal storage of a Wear device from a Windows machine, whereas it is a painless exercise on an Android phone. In the end, the process of transferring the database files now only takes around 10 seconds which is a massive improvement when compared to the 10 minutes it was taking to build the database on the Wear device itself.

In order to maintain the database version an entry was added into the database called "database_version" which has an integer value of the database version number. The database version is checked against the String constant contained in the Wear application and if they are different then the database is destroyed and recreated using the files located in the database folder within the assets directory.

[7]Internal Storage - http://developer.android.com/guide/topics/data/data-storage.html#filesInternal

5.2.4 First Character Key Press Weighting

It was found during the development of the final prototype that the language model often corrected the first character of a word typed. Figure 5.1 shows the effect of a weighting being added to the first character using the derived algorithm:

$$WP(c) = p(c) + ((p(c) + p(lc) + p(rc)) \cdot \frac{x}{100}) \tag{5.1}$$

Where $WP(c)$ is the weighted probability of a given character c, p is the probability of a character, lc is the character to the left of c and rc is the character to the right of c, with x being the percentage of the weighting to be added.

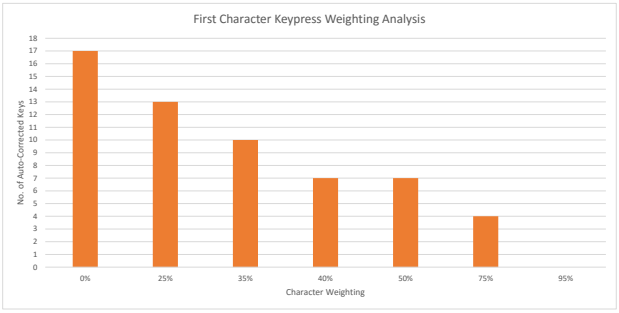


Figure 5.1: The results from the analysis of the effect of a weighting being added to the first character of a word.

As shown by Figure 5.1 there are 17 characters that are corrected to their neighbour when there is no weighting added (0%). The number of corrected characters reduces as the percentage of weighting applied increases.

The compromise between being able to select all keys and to still apply the language model to the first key press was apparent. A weighting of 75% was applied to the final implementation of the Hoop Keyboard. The weighting of 75% reduced the amount of characters corrected to just 4 which allowed 13 characters to be inserted that were previously not able to be selected as the first character of the word.

To counteract the negative effect the weighting algorithm has on the most common characters, the buttons for the common characters were enlarged. The most common keys were “A”, “I”, “O”, “P”, “S”, “T”, & “W”. As a result the language model now applies a weighting of 75% to characters that are the first character of a word and the common keys are around 10% wider or 10% taller (depending on where they lay on the keyboard) than the standard key size.

Chapter 6

Verification and Validation

This chapter will detail the testing and validation methodology for ensuring the outcome of the project satisfies the specification. The majority of this chapter will provide an overview of the testing concepts used to test the code of the application itself as well as a detailed insight into the user studies conducted as part of the verification & validation process.

6.1 Code Testing

A range of testing methods were used over the course of the project, and a brief overview of the testing process and technologies used is provided in this section. See Appendix F for the detailed test cases.

6.1.1 Android Instrumentation Testing - Espresso

Google's Espresso[1] is an Android UI testing tool allowing automated testing of UI components within an Android application. Espresso was used to create automated tests of the UI components of the Hoop Keyboard. Espresso was particularly helpful in identifying when the font size was too small and it picked up on the fact that the 'W' key was smaller than the recommended minimum font size, which is 12 Scale-independent pixels (sp). It also became apparent when testing some buttons (i.e. the "A" key) that Espresso tests would fail if the component that was requested to be clicked was not mostly visible to the user. It was found

[1]Espresso Support Library - https://google.github.io/android-testing-support-library/docs/espresso/index.html

that there is a constraint of 90% visibility of a view component[2] before Espresso will successfully click a component. This highlighted a scaling bug when using the Hoop Keyboard on devices/emulators other than the Huawei Watch as the keyboard buttons would overlap each other. Once the issue with the resizing of view components was resolved, all other tests passed.

6.1.2 Unit Testing

JUnit[3] is the most popular and widely-used unit testing framework for Java and it is integrated into the Android Studio IDE. Due to its popularity, ease of use and the fact I have developed a good understanding of JUnit throughout my university degree, JUnit was an obvious choice for unit testing. Version 4 of JUnit was used to test the functionality applications throughout the project including the Android applications, the Analysis Software and the database generator application. To test the Android applications, the JUnit tests and Espresso tests were combined in order to fully test the components of the applications. The Java applications were tested using a combination of JUnit & Eclemma[4] to provide an indication of code coverage tested. Further detail on all the tests performed can be found in Appendix F.

6.1.3 Android Lint

Android Lint[5] is a static code analysis integrated into the Android Studio IDE. Android Lint checks the files of an Android project for potential bugs along with optimisation improvements for: correctness, security, performance, usability, accessibility and internationalization. Android Lint was able to identify potential code optimisations with in the Hoop Keyboard project as well as removing unused imports and highlighted incorrect Javadoc documentation within the code.

[2]Espresso GeneralClickAction - https://android.googlesource.com/platform/frameworks/testing/+/android-support-test/espresso/core/src/main/java/android/support/test/espresso/action/GeneralClickAction.java

[3]JUnit Website - http://junit.org/

[4]Eclemma Website - http://eclemma.org/

[5]Android Lint - http://developer.android.com/tools/help/lint.html

6.2 User Studies

User studies were completed to analyse the performance of the Hoop Keyboard in comparison to related keyboards. Ethical approval was required from the university before any participants could take part in a user study. An initial user study was conducted to gather insight into the required improvements for the final implementation of the Hoop Keyboard. A final user study was carried out to analyse the performance of the Hoop Keyboard using a comparison against the Minuum Keyboard.

The participants recruited for the user studies were aged between 20 and 25 years old where 2 were female and the rest were male. The main type of mobile text entry for all of the users was an on-screen QWERTY keyboard. All bar one of the participants type several messages/emails/social media posts on their phones a day. Finally, 3 of the users regularly use a smartwatch whereas the rest of the participants had no prior experience using a smartwatch before the user study. All of the background information for each participant involved in the studies can be found via Appendix J.2.

The user studies were conducted using the Huawei Watch and Google's Nexus 5[6] (built by LG) mobile phone and the hardware specifications for both devices can be found in Appendices G.2 & H.1 respectfully. In order to preserve the data logged and reduce the possibility of fatal problems when conducting the user study, all internet connectivity on the Nexus 5 device was turned off and it was put into "Do not disturb" mode. This blocked all possible notifications on the phone during testing while also allowing the phone and the watch to communicate with each other through Bluetooth[7] technology.

The phrases were displayed on the Nexus 5 device and the user was instructed to type the displayed phrase on the Huawei Watch as accurately and as quickly as possible for all keyboards used. The exact instructions given prior to each user study can be found in the "User Guide" section of the pre-participation form in Appendix J.1. A storyboard portraying a typical task on the Hoop Keyboard in the final user study can be found in Appendix M.2. The state transition network demonstrating a typical task on the Hoop Keyboard can also be found in Appendix N.

[6]Google's Nexus 5 built by LG - http://www.lg.com/uk/mobile-phones/lg-D821
[7]Bluetooth Technology Website - https://www.bluetooth.com/

6.2.1 Data Recorded & Formulae

The following metrics were recorded for all participants during both of the user studies and were used to evaluate the performance of each keyboard tested:

- The speed of each user in raw words per minute (WPM) (see Equation 6.1). This was used to calculate the raw speed of the keyboard which in turn was used to evaluate the performance of the keyboard.
- The speed of each user in adjusted words per minute (WPM) (see Equation 6.2). This was calculated as well as the raw WPM to calculate the speed while taking into account the number of errors made by the user.
- The number of errors that were corrected by the user. This was used to calculate the corrected error rate.
- The number of errors that were not corrected by the user. This was used to calculate the uncorrected error rate.
- The average inter-keypress time for each user (NOTE: The average inter-keypress time was not used in the initial study involving the Hoop Prototype 2 keyboard).

Formulae

The key used for the following equations:
$|T|$ - The sum of characters typed
$|C|$ - The sum of correct characters
$|INF|$ - The sum of incorrect characters that are not fixed
S - The time elapsed in seconds calculated as the difference between the time stamps for the first and last character typed.

The words per minute measures the time taken to produce a number of words. The speed of each user in raw words per minute, was calculated as (Arif & Stuerzlinger, 2009):

$$rawWPM = \frac{|T|}{S} \cdot 60 \cdot \frac{1}{5} \tag{6.1}$$

Here, the constant 60 is the number of seconds per minute, and the factor of one fifth represents the average length of a word in characters.

The speed of each user in adjusted words per minute is the similar to the raw WPM but also taking into account the number of errors made by the user (Proschowsky, 2008):

$$adjustedWPM = \frac{|C| + |INF| - 1}{S} \cdot 60 \cdot \frac{1}{5} \tag{6.2}$$

Here, again, the constant 60 and the factor of one fifth represents the same as in the Equation 6.1.

6.2.2 Ethical Approval

The University of Strathclyde have a very strict approach to testing which involves participants. Ethical clearance from the Department of Computer and Information Science Ethics Committee is necessary before any tests that involve human subjects are undertaken. Ethics approval was acquired well in advance of the launch of the initial user study (see Appendix I).

6.2.3 Initial User Study

The initial user study involved 10 participants and was carried out immediately after the completion of the Hoop Prototype 2 keyboard. Each participant was required to complete and sign a consent form (see Appendix J.1) before they were allowed to participate in the user tests. Once the form was completed participants were then instructed to complete the 2 phrase sets illustrated in Appendix J.6.1.

Half of the participants were randomly chosen to start on phrase set 1 and the other half started on phrase set 2 but in the end, all participants completed the same phrases. The purpose of randomising the participants' starting phrase set was to produce unbiased results and to reduce the effect of "easier" phrases within the phrase sets.

6.2.4 Final User Study

The final user study involved 16 participants in total and was carried out immediately after the completion of the final Hoop Keyboard implementation. I was able to get 9 returning participants from the initial user study and 7 new participants. The returning participants did not require to sign another consent form whereas the new participants had to provide

consent before they were able to take part in the study. All the participants were then required to complete the 2 phrase sets shown in Appendix J.6.2.

The process of randomisation in order to produce unbiased results in the final user study was a more complex problem than the one found in the initial user study. Like the initial study, this study also used 2 phrase sets but also had the addition of another keyboard to test resulting in a total of 2 keyboards. Thus, the 4 possible user conditions were:

- "M1→H2" - Start on phrase set 1 with the Minuum Keyboard, then do phrase set 2 on the Hoop Keyboard.
- "H1→M2" - Start on phrase set 1 with the Hoop Keyboard, then do phrase set 2 on the Minuum Keyboard.
- "M2→H1" - Start on phrase set 2 with the Minuum Keyboard, then do phrase set 1 on the Hoop Keyboard.
- "H2→M1" - Start on phrase set 2 with the Hoop Keyboard, then do phrase set 1 on the Minuum Keyboard.

The random allocation of a user to their given user condition was done by getting an independent body to shuffle the forms, they were kept in this order over the remaining course of the study. To assure that users completed their given user condition accurately I personally set up each and every phrase set on the mobile application throughout the study.

Participants were also handed a NASA Task Load Index (NASA-TLX) form to fill out immediately after the completion of a phrase set (see Appendix J.3). That way, each user would complete a NASA-TLX for each keyboard so that the results could be used to compare the two keyboards. Once a participant completed their user condition they were then provided with an online link to the exit questionnaire illustrated in Appendix J.5. The questionnaires were recorded in order to analyse each users opinion of the Hoop Keyboard and to provide a comparison with the Minuum Keyboard.

6.2.5 Results Format

The data recorded during the course of the user studies was saved to simple text files on the mobile device. A new file was created for each phrase submitted by a user, resulting in 30 files per user in the final study and 40 files per user in the initial study. Research

and experimentation found that it was not possible to access the files in internal storage of the Wear device, however it was possible to access the files in the external storage of the mobile device through a Microsoft Windows[8] computer. An alternative solution was to upload the data to a remote database such as MySQL[9] but this would rely on a steady internet connection as well as requiring more time to construct.

The folder directory of the saved log files was generated based on the user number and user condition given to the user as shown by Table 6.1.

Log_Data	*user condition*	*user number*	*keyboard*	*phrase set & phrase number*	*filename*

Table 6.1: The format of the directory structure of the log data.

The filename for each text file was a unique timestamp so that there was no possibility of data getting overwritten or lost. Table 6.2 illustrates the format of the text file for the final user study.

Fixed Text	**Separator**	**Variable Values**
Final Text:	\t	*The phrase attempt submitted by the user*
Backspaces:	\t	*The number of times the user pressed/swiped for backspace*
Time Taken (milliseconds):	\t	*The time taken to complete the phrase*
Average Inter-Keypress Time (milliseconds):	\t	*The average time taken in between key presses*
Raw WPM:	\t	*A raw measure of the speed of the user*
Uncorrected errors:	\t \n	*A list of mistakes present in the phrase attempt. Separated by the "\n \t" characters*
No. of uncorrected errors:	\t	*The number of mistakes present in the phrase attempt*
Adjusted WPM:	\t	*An adjust measure of the speed of the user*
X Y Coordinates:	\t \n	*A list of X & Y coordinates of the users' tap. The X & Y values are separated by a space(" "). Coordinates are separated by the "\n \t" characters*

Table 6.2: The format of the text file containing the usage data gathered from the user studies.

[8]Microsoft Windows - www.microsoft.com/en-gb/windows
[9]MySQL Website - https://www.mysql.com/

There were some exceptions to the format as there were no X & Y coordinates recorded for the Minuum Keyboard and the initial study log data did not include the Average Inter-Keypress Time or the Adjusted WPM data. Another thing to note is that the initial study recorded the Time Taken in "(HH:MM:SS:sss)" rather than milliseconds and the wording of "Raw WPM:" was simply "WPM:". The WPM calculation made by the device turned out to be wrong therefore the final WPM for the Hoop Prototype 2 keyboard was calculated by the User Data Analyser software.

The data acquired from the keyboards was harnessed and converted using the User Data Analyser software (see Appendix O). The software written in Java was specifically functioned to retrieve the metrics detailed above. The application was able to parse the data logged for all of the participants and output comma-separated values (CSV) files for all metrics for each keyboard. The CSV format was chosen for ease of portability for Microsoft Excel[10] so that graph representations of the data can be easily generated.

The X & Y coordinate data was used to produce an R[11] program that generates a heatmap of all the coordinates recorded. RStudio[12] is a powerful IDE for R and it was able to handle the vast amount of coordinates logged from the user studies. R is a great programming language for small projects such as heatmaps and RStudio is also great for plotting heatmaps and graphs alike.

MatLab[13] was an alternative utility considered when investigating for the best tool to generate heatmaps from X & Y coordinates. In contrast to the open source availability of R & RStudio, MatLab is not open source and only trail software is available for free. R and MatLab are both very powerful tools to solve complex problems but I found the syntax of R easier to understand than that of MatLab and the fact it was open source persuaded me to choose R over MatLab.

[10]Microsoft Excel - https://products.office.com/en-gb/excel
[11]R - https://www.r-project.org/
[12]RStudio - https://www.rstudio.com/
[13]http://uk.mathworks.com/products/matlab/

Chapter 7

Results and Evaluation

This chapter presents and reviews the results gathered from the user studies. An overall evaluation of the project is also provided as well as a comparison against other systems.

7.1 User Study Results

7.1.1 Raw Words Per Minute

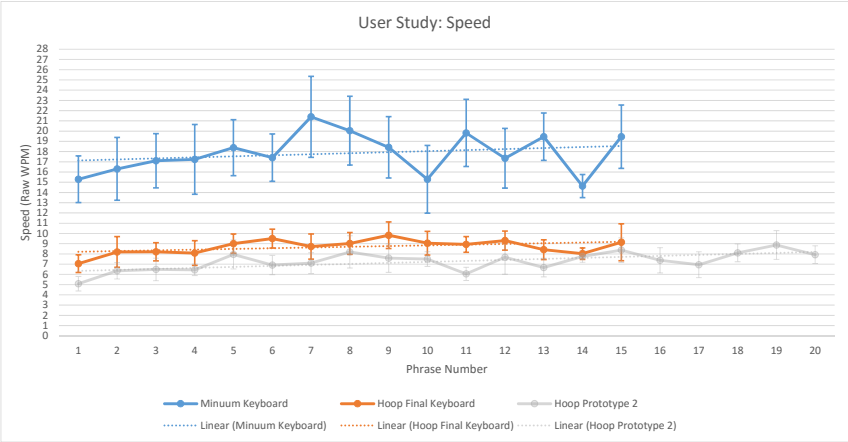


Figure 7.1: The average speed recorded for each phrase using the Raw WPM calculation. Higher is better. Error bars show 95% confidence intervals.

Figure 7.1 illustrates that the overall mean speed of the Hoop Final Keyboard is slightly faster than the Hoop Prototype 2. The fastest average speed recorded for a phrase on the

Hoop Final Keyboard was 9.83 RawWPM whereas the fastest on the Hoop Prototype 2 keyboard was slower by ∼1 word with only 8.87 RawWPM. The fastest user on the Hoop Final Keyboard was also faster than the fastest on the Hoop Prototype 2 keyboard with 11.03 RawWPM and 8.71 RawWPM respectfully.

It can also be concluded that the mean speed of the Hoop Final Keyboard was significantly slower than the Minuum Keyboard by ∼50%. The fastest average speed recorded for a phrase on the Minuum Keyboard was twice as fast as the Hoop Final Keyboard with 21.4 RawWPM. Similarly, the fastest user on the Minuum Keyboard was significantly faster than the Hoop Keyboard's fastest user with 23.98 RawWPM. Unsurprisingly, the difference between the Hoop Final Keyboard & the Minuum Keyboard for this series was significant (paired t-test, n=15, $p<0.001$).

The linear trend-lines for all keyboards show an increasing trend till around Phrase 9 where the mean speed of each user starts to tail off.

7.1.2 Adjusted Words Per Minute

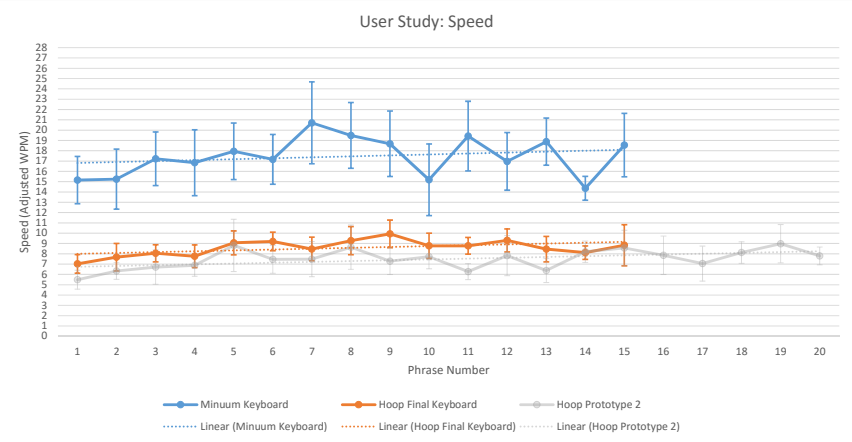


Figure 7.2: The average speed recorded for each phrase using the Adjusted WPM calculation. Higher is better. Error bars show 95% confidence intervals.

Figure 7.2 illustrates that the overall mean speed of the Hoop Final Keyboard is slightly faster than the Hoop Prototype 2. The fastest average speed recorded for a phrase on the Hoop Final Keyboard was 9.94 AdjustedWPM whereas the fastest on the Hoop Prototype 2 keyboard was slower by ∼1 word with only 8.98 AdjustedWPM. The fastest user on the

Hoop Final Keyboard was also faster than the fastest on the Hoop Prototype 2 keyboard with 11.3 AdjustedWPM and 10.22 AdjustedWPM respectfully.

It can also be concluded that the mean speed of the Hoop Final Keyboard was significantly slower than the Minuum Keyboard by ∼50%. The fastest average speed recorded for a phrase on the Minuum Keyboard was twice as fast as the Hoop Final Keyboard with 20.7 AdjustedWPM. Similarly, the fastest user on the Minuum Keyboard was significantly faster than the Hoop Keyboard's fastest user with 24.83 AdjustedWPM. Unsurprisingly, the difference between the Hoop Final Keyboard & the Minuum Keyboard for this series was significant (paired t-test, n=15, $p<0.001$).

The linear trend-lines for all keyboards show an increasing trend till around Phrase 9 where the mean speed of each user starts to tail off.

7.1.3 Corrected Error Rate

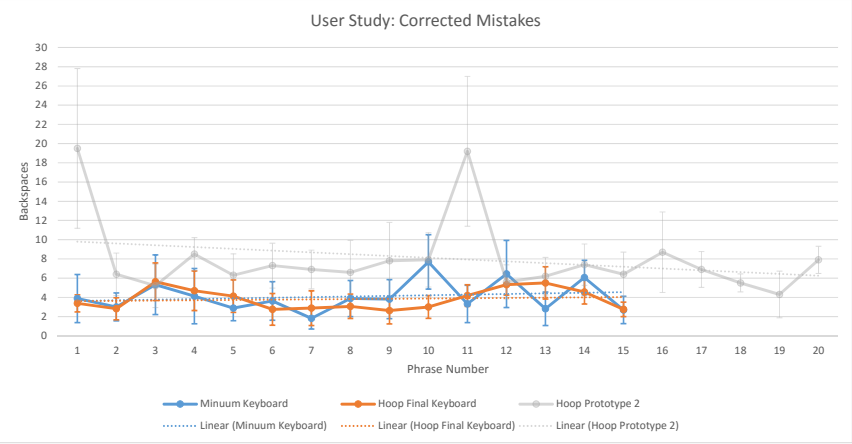


Figure 7.3: The average corrected error rate recorded for each phrase. Lower is better. Error bars show 95% confidence intervals.

As expected, Figure 7.3 shows a dramatic reduction in corrected mistakes when the Hoop Final Keyboard is compared to the Hoop Prototype 2 keyboard. It would appear that there was a particularly bad phrase in the study involving the Hoop Prototype 2 keyboard that users either faced in Phrase 1 or Phrase 11. Regardless, the best average corrected error rate for a phrase recorded on the Hoop Final Keyboard is still significantly less than the Hoop Prototype 2 keyboard with only 2.63 backspaces compared to 4.3 backspaces on the Hoop Prototype 2 keyboard. The user with the best average for corrected error rate on the

Hoop Final Keyboard again, outperforms the best user on the Hoop Prototype 2 keyboard with average of just 2.6 backspaces per phrase compared to 4.25 backspaces on the Hoop Prototype 2 keyboard.

Interestingly, as shown by Figure 7.3 the overall difference between the corrected error rate on the Hoop Final Keyboard and the Minuum Keyboard is very small. The best average corrected error rate recorded for a phrase on the Minuum Keyboard was better than the Hoop Final Keyboard with an average of 1.81 backspaces. The worst phrase average on the Hoop Final Keyboard demonstrates a less bad average backspace count than the Minuum Keyboard with 5.63 backspaces compared to 7.69 backspaces. Similarly, the same trend is apparent when comparing the best and worst user averages on the keyboards. The Minuum Keyboard again, has the better "best" user average with 1.27 backspaces when compared to 2.6 backspaces found on the Hoop Final Keyboard. However, the "worst" user average on the Hoop Final Keyboard is significantly lower than the Minuum Keyboard's with just 5.4 backspaces compared to 10.07 backspaces. There was no significant difference between the Hoop Final Keyboard & the Minuum Keyboard for this series (paired t-test, n=15, p=0.64).

The linear trend-lines for the Hoop Final & Minuum Keyboards show a decreasing trend till around Phrase 9 where the mean corrected error rate of each user starts to increase. The trend-line for the Hoop Prototype 2 keyboard demonstrates a general decrease in average corrected error rate over the course of the study.

7.1.4 Uncorrected Error Rate

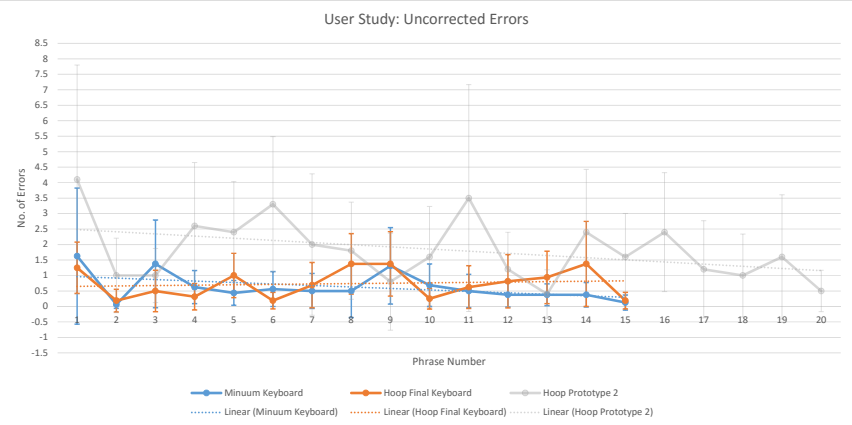


Figure 7.4: The average uncorrected error rate recorded for each phrase. Lower is better. Error bars show 95% confidence intervals.

Similarly to the corrected error rate results, Figure 7.4 shows a dramatic reduction in uncorrected mistakes when the Hoop Final Keyboard is compared to the Hoop Prototype 2 keyboard. The best average uncorrected error rate for a phrase recorded on the Hoop Final Keyboard is significantly less than the Hoop Prototype 2 keyboard with an average of only 0.19 errors compared to 0.4 error rate on the Hoop Prototype 2 keyboard.

The overall difference between the uncorrected error rate on the Hoop Final Keyboard and the Minuum Keyboard is very small. The best average uncorrected error rate recorded for a phrase on the Minuum Keyboard was better than the Hoop Final Keyboard with an average of 0.06 errors. Interestingly, the worst phrase average on the Hoop Final Keyboard demonstrates a less bad average error count than the Minuum Keyboard with an average of just 1.4 errors compared to 1.6 errors. Similarly, the same trend is apparent when comparing the worst user averages on the keyboards. The "worst" user average on the Hoop Final Keyboard is significantly lower than the Minuum Keyboard's with just 2.87 errors compared to 3.07 errors. There was no significant difference between the Hoop Final Keyboard & the Minuum Keyboard for this series (paired t-test, n=15, p=0.44).

The linear trend-line for the Hoop Final Keyboard shows a generally increasing trend for the uncorrected error rate. The trend-lines for the Minuum Keyboard and the Hoop Prototype 2 keyboard demonstrate a general decrease in the average uncorrected error rate over the course of the study.

7.1.5 Average Inter-keypress Time

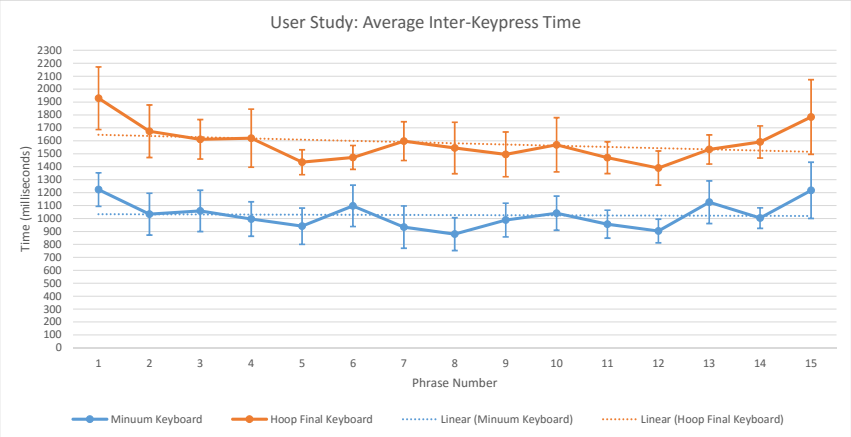


Figure 7.5: The average inter-keypress time recorded for each phrase. Lower is better. Error bars show 95% confidence intervals.

The average inter-keypress time (IKT) was only recorded during the final study hence the reason why there is no line for the Hoop Prototype 2 keyboard in Figure 7.5. The average IKT results are shown in Figure 7.5 for the Hoop Final Keyboard and the Minuum Keyboard. It is clear from the graph above that the Minuum Keyboard had a significantly lower average IKT than the Hoop Final Keyboard. The best average IKT for a phrase recorded on the Hoop Final Keyboard is significantly more than the Minuum Keyboard with an average time of 1389.98milliseconds (ms) compared to 879.31ms on the Minuum Keyboard. Similarly, the best average IKT for a user recorded on the Hoop Final Keyboard is more than the Minuum Keyboard with an average time of 1280.53ms compare to 782.63ms on the Minuum Keyboard. Unsurprisingly, the difference between the Hoop Final Keyboard & the Minuum Keyboard for this series was significant (paired t-test, $n=15$, $p<0.001$).

The linear trend-lines for both keyboards show a decrease in average IKT over the phrases. It would appear that the average IKT starts to increase after the 12th phrase for both keyboards till the end of the phrase set.

7.1.6 Heat map - XY Coordinates

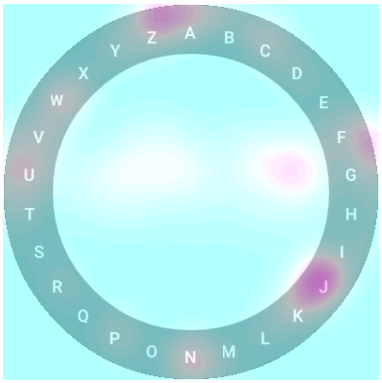


Figure 7.6: The heatmap of taps recorded during the user studies for the Hoop Keyboard. Blue is little to no taps, white is some taps, and pink is many taps.

Figure 7.6 illustrates a heatmap of the participants' taps on the Hoop Keyboard over the course of the final study. The blue areas are the "cold" sections where there are little to no taps in this region. The while areas are the "warm" sections where there are some taps in this region. The pink areas are the "hot" sections where there are many taps in this region. The figure above shows that the most common areas for taps are around the "I", "J" and "K" letters along the the area near the "Z" and "A" letters. Sections around "E", "F", "G", "N", "O", "U" and "T" also received a significant amount of taps as well as a large area to the right of centre of the inner circle.

7.2 Questionnaire Results

7.2.1 NASA Task Load Index (NASA-TLX)

As mentioned, each user was instructed to complete a NASA-TLX form immediately after the conclusion of a phrase set. The format of the NASA-TLX form can be found in Appendix J.3 and the data recorded from these questionnaires is available via Appendix J.4. The results for both, the Hoop and the Minuum, keyboards have been included on the same graph to facilitate easy comparison (see Appendix J.4). An cumulative total has also been added for both extremes, starting at the middle point for both directions. The addition of a polynomial trend-line for the accumulative total data further enhances the visualisation of the data and clearly illustrates the results of the NASA-TLX in a comparative sense.

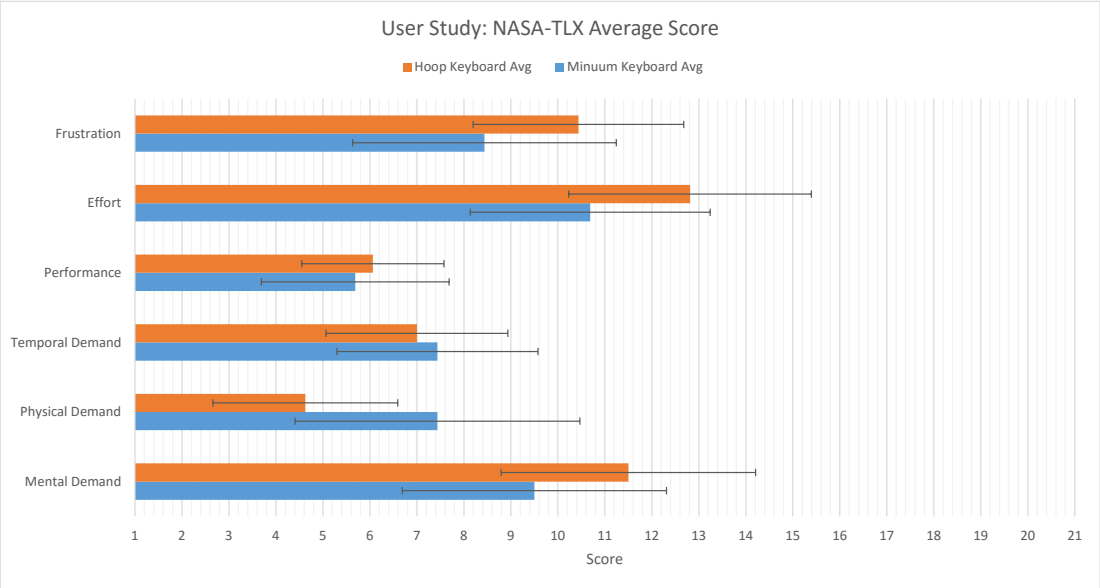


Figure 7.7: NASA-TLX Results - The users average score for each metric measured. Lower is better. Error bars show 95% confidence intervals.

Figure 7.7 presents an overall average score for all the metrics measured in the NASA-TLX forms for both of the keyboards involved in the final user study. It is clear from figure 7.7 that the participants found the Hoop Keyboard significantly more mentally demanding than that of the Minuum Keyboard. Surprisingly, the results of the physical demand scale in Figure 7.7 demonstrate that the Hoop Keyboard took a considerable lower physical demand than the Minuum Keyboard but both were still showing a low physical demand on average.

There was nothing to separate the keyboards in terms of the temporal demand results gathered as Figure 7.7 clearly shows that the tasks on both keyboards were not hurried or rushed in the slightest. Again there was not much to separate the keyboards on the NASA-TLX results on performance. Figure 7.7 illustrates that the participants strongly felt they excelled on the given tasks, and it appears that the overall performance average indicates that the users felt they had done better on the Minuum Keyboard. However, from the detailed results in Figure J.6 from Appendix J.4 the trend-lines show there is a consensus that the participants felt they performed slightly better on the Hoop Keyboard in comparison to the Minuum Keyboard. The overall level of effort required for the study was generally high as seen in Figure 7.7. Lastly, as shown in Figure 7.7 the overall average frustration levels reported by the participants was on the lower end of the spectrum however, the users did feel they were more frustrated with the Hoop Keyboard than the Minuum Keyboard.

7.2.2 Exit Questionnaire

The aim of the exit questionnaire was to ascertain a clear indication of the participant's preference to the definitive aspects of the Hoop Keyboard in a compared to the Minuum Keyboard. The questions took the form of a 7-point likert scale, as studies indicate that 7-point scale is the optimal (Jacoby & Matell, 1971). For consistency, all of the answer scales were displayed in the same format with "1" being the features of the Minuum Keyboard, "7" being the features of the Hoop Keyboard and "4" being neutral as shown by Appendix J.5. This way users were able to easily fill in their opinion accurately without the real need to pay too much attention to the change in the scales. All of the figures displayed in this section below illustrate the total number of responses for each point in the scale for each question. Each figure also includes a cumulative total for each extreme on the likert scale starting from the neutral value. A polynomial trend-line was also added in order to easily present a clear comparison of the overall participants' preference for the surveyed aspects of the keyboards.

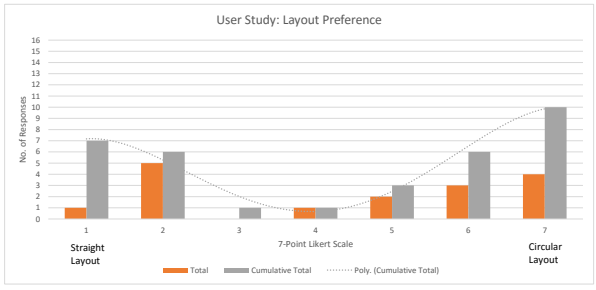


Figure 7.8: Exit Questionnaire Results - Layout Preference.

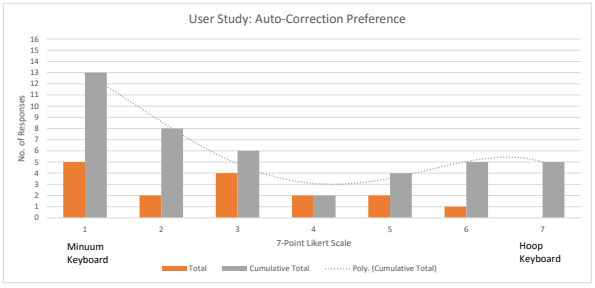


Figure 7.9: Exit Questionnaire Results - Auto-Correction Preference.

Figure 7.8 shows that the participants displayed a a mixed preference to the keyboard layout when asked if they preferred a "Straight" layout or a "Circular" layout. There was slightly more people in favour of the circular layout which demonstrates that there was a real divide in the users, where some much preferred the Hoop Keyboard but others favoured the traditional QWERTY layout of the Minuum Keyboard.

The results displayed in Figure 7.9 demonstrate the clear preference for the Minuum Keyboard auto-correction over the Hoop Keyboard. There is no doubt that the option to select a word from a suggested list is the favoured option when it comes to word auto-correction.

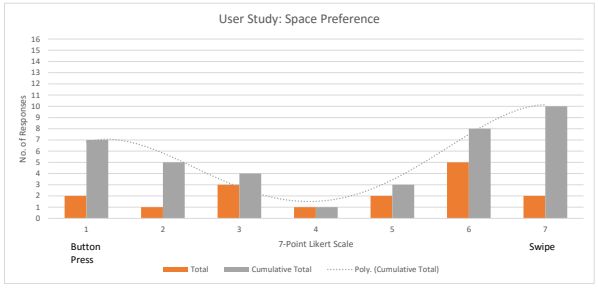


Figure 7.10: Exit Questionnaire Results - Space Preference.

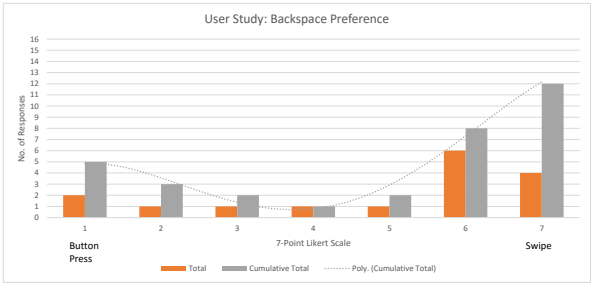


Figure 7.11: Exit Questionnaire Results - Backspace Preference.

As mentioned previously the Hoop Keyboard takes advantage of swipe gestures to insert spaces and to backspace thus, removing the need for virtual keys to be displayed on the precious screen real estate (see Appendix L.5). However, the Minuum Keyboard uses buttons to detect when the user wishes to insert a space or to backspace (see Appendix L.6).

The users were asked for their favourite method of inserting a space character (“ ”) and the result in Figure 7.10 displays a similar trend to the layout preference results in Figure 7.8. There is a slight preference towards the swipe method on the Hoop Keyboard over the space bar button on the Minuum Keyboard. Again, the results are inconclusive to prove that there is a clear preference for the swipe method.

The results of users' overall backspace method preference surprisingly contrasts the results of the space method survey. It is obvious from Figure 7.11 that the favourable backspace method is the swipe.

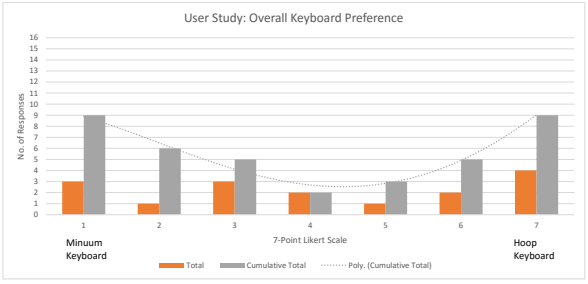


Figure 7.12: Exit Questionnaire Results - Overall Keyboard Preference.

Figure 7.12 illustrates the overall keyboard preference of the participants. The polynomial trend-line demonstrates that there is a split decision for the overall keyboard preference with half of the users preferring the Hoop Keyboard and the other half displaying a preference to the Minuum Keyboard.

7.3 System Evaluation

7.3.1 Speed

The results from the user studies for the RawWPM, AdjustedWPM and the average IKT can be used to evaluate the text entry speed performance of each keyboard. The overall raw speed of Hoop Prototype 2 was 7.3 WPM, the Hoop Keyboard was 8.7 WPM and the Minuum Keyboard was 17.8 WPM.

Over the course of development of the Hoop Keyboard the speed of text entry on the keyboard has steadily increased. However, the better performance of text entry on the Minuum Keyboard suggests that the Hoop Keyboard still requires improvements. The word suggestions on the Minuum Keyboard allowed users to quickly select a word from a list. This meant the user did not have to manually type every letter of each word leading to faster text entry on the Minuum Keyboard. The Hoop Keyboard on the other hand only provided a simple auto-correction after the word was finished which had no effect on increasing the text entry speed. With the addition of a suggestive auto-correction to the Hoop Keyboard and further enhancements to the language model there remains the possibility of the Hoop Keyboard getting faster text entry results.

Proschowsky's TUP keyboard outperforms the Hoop Keyboard on text entry speed. As mentioned, the TUP keyboard had an overall speed of 10.5 WPM which is 1.8 WPM better than the Hoop Keyboard. The performance difference between the two keyboards could be put down to the differing user study formats. Proschowsky conducted a user study over the course of 4 weeks with over 25,000 messages (Proschowsky, 2008). The Hoop Keyboard was evaluated using 240 phrases over the course of 10 minutes per participants. The sharp contrast in keyboard exposure alone suggests that given a longer user study, the text entry speed of the Hoop Keyboard may produce better results than the TUP keyboard.

Optimisation of the key layout has potential for increasing the text entry speed as Figure 7.6 shows that the most active areas are almost at opposite sides of the keyboard. By reducing the amount of distance the finger has to travel by using an enhanced layout may also have a favourable effect on text entry speed. Further research and development will be required in order to evaluate the effect of a suggestive auto-correction and/or an alternative key layout.

The main type of text entry for all of the users was an "On screen QWERTY keyboard" suggesting that the most decisive factor of the success of the Minuum Keyboard was familiarity. With more time and practice on the Hoop Keyboard along with the previously mentioned

enhancements, it is likely that the text entry speed of the users will improve.

7.3.2 Error Rate

The results from the corrected error rate and the uncorrected error rate for the user study can be used to evaluate the overall error rate of the keyboards. As expected, the Hoop Keyboard demonstrates a significant improvement during its development, as this was a issue identified after the initial high error rates found during the initial user study. The Hoop Keyboard and the Minuum Keyboard both have low error rates and the t-tests suggest that there is very little difference between the error rates for both of the keyboards. It would appear that the suggestive auto-correction again, had a positive effect on the error rate for the Minuum Keyboard. The auto-correction suggests correctly spelled words from a dictionary which allows users to be more accurate with spelling and reduce the backspace usage. The n-gram character language model on the Hoop Keyboard has dramatically reduced the backspace usage and decreased the number of errors typed by the user to the same level as the Minuum Keyboard. This suggests that the language model is effective at reducing the error rates but there is reason to believe that a suggestive auto-correction system would further improve the results.

7.3.3 User Feedback

NASA-TLX

Generally, for both the Hoop Keyboard and Minuum Keyboard, the NASA-TLX scores were close to the lower end of the spectrum with some differences between the keyboards.

The difference between the mental demand required for Hoop Keyboard and the Minuum Keyboard was apparent with the results clearly in favour of the Minuum Keyboard. Further speculation suggests that users may have found it more difficult to find keys in the implemented alphabetical layout in comparison to the well known QWERTY layout. This resulted in the Hoop Keyboard requiring a higher degree of concentration. Strangely the physical demand of the Hoop Keyboard was significantly less than that of the Minuum Keyboard. The difference between the keyboards here is odd since the same simple tasks were carried out on both keyboards but overall both average scores were on the lower end of the spectrum.

The temporal demand on both keyboards were relatively low due to the non-rushed nature of

the tasks since the users were able to take breaks in between phrases. The users also felt they performed very well on the tasks given on both keyboards as there was not much difference between the estimated performance. The average estimated performance on the Minuum Keyboard was only slightly better than the Hoop Keyboard. This suggests that the users thought they performed better on the Hoop Keyboard than they actually did. In saying that, the accuracy of the typing on the Hoop Keyboard was very good, however the speed of the typing was rather slow when compared to the Minuum Keyboard. Therefore, the estimated performance by the users suggests that they measured their performance primarily through error rate which correlates with the average corrected & uncorrected error rates discussed earlier.

The level of effort required on both keyboards was relatively high when compared to the other measurements measured by the NASA-TLX. The average scores for effort were around the neutral scale. The effort required by the users on the Hoop Keyboard was again greater than that of the Minuum Keyboard. Again, the difference is most likely due to the familiarity of the QWERTY key layout as the users found that it took more effort to find keys on the Hoop Keyboard. Further speculation suggests that the Minuum Keyboard required less effort because the user was able to select a word from a list when a word was still being typed, whereas on the Hoop Keyboard every letter was required to be typed in order to complete each word. Overall the users did find it relatively difficult to type on the Android Wear device. This, and the large amount of phrases used during the final user study (30 in total), may have contributed to the high level of effort required by the users. The same trend was found when evaluating the frustration levels of the users on both keyboards with the key layout being the most likely differing factor.

The frustration levels reported by the participants was mixed with an equal amount of users on both sides of the spectrum. There is however a trend displaying a greater amount of participants being frustrated with the Hoop Keyboard when compared with the Minuum Keyboard. This again could be due to the simple fact that the users were unfamiliar with the circular & alphabetical layout of the Hoop Keyboard.

Exit Questionnaire

The results from the exit questionnaire show that the users generally favoured the layout of the Hoop Keyboard. This suggests that the circular layout of the keyboard is appropriate for circular smartwatches. It is a new keyboard layout and it was not expected that everyone in the demographic would like it. As previously mentioned the demographic of 20-25 year

olds tend to be reluctant to change, therefore the layout result can be considered a success for the Hoop Keyboard.

The clear preference towards a suggestive auto-correction framework was an expected result, as plans to implement a spell checker framework similar to the method used in the MaxieKeyboard (see Appendix A.2) had already been discussed during supervisory meetings. It is apparent that the suggestive auto-correction increases the overall performance of a keyboard, as it not only reduces the error rate by suggesting words from a dictionary, but it also improves the text entry speed allowing users to auto-complete words without having to type all the letters.

It is interesting that the results from the surveys on space and backspace differ by so much. An explanation maybe due to the space button being very large whereas the backspace button was very small in comparison on the Minuum Keyboard, (see Appendix L.6) therefore users were experiencing more problems with the backspace button. Another reasoning may be down to the fact that the space button is part of natural typing whereas the backspace is not, meaning that some users may have found it faster to type with an available space key and easier to delete with a swiping motion. However, in general the swipe method was preferred but further investigation may find that the text entry speed will improve if the Hoop Keyboard adopted a space bar.

The layout of the Hoop keyboard leaves 31.7% of usable screen space available to display the typed text. In comparison, the Minuum Keyboard only leaves 11% of usable screen space to display the typed text. Therefore the Hoop Keyboard is able to display three times as much text on the device. A comment from User 7 from the study said that "the qwerty keyboard was pretty small and hard to press the buttons, plus it obfuscated the text on the screen so it was hard to see what you were typing". This illustrates the main issue that the users found when typing on the Minuum Keyboard was that the keyboard took up too much of the usable screen space.

It can be concluded that 50% of the participants preferred the Hoop Keyboard and 50% preferred the Minuum Keyboard. The equal divide in preference suggests that the Hoop Keyboard has a real potential to be a success with further research and development.

Chapter 8

Summary and Conclusions

This chapter summaries the performance and success of the project in relation to the original objectives. Potential improvements in the form of future development are also discussed in this chapter.

8.1 Summary

Overall the results attained from the user studies proved that text entry on Hoop Keyboard is significantly slower than a QWERTY keyboard and that it requires further development in order to be considered a competitor keyboard for Android Wear devices. There is a possibility that the differences in speed were mostly down to the users being already very familiar with the QWERTY layout. However, the low error rates on the Hoop Keyboard were competitive with the QWERTY layout which suggests that the language model was allowing users to type accurately but not quickly.

From analysis of the all the results gathered from the exit questionnaire of the final user study it can be concluded that there was a mixed response as to whether the users preferred the Hoop Keyboard over the Minuum Keyboard. It can be drawn that the swipe features for space and backspace were more popular as a whole than the buttons. However, the auto-correction on the Hoop keyboard will need to be addressed as the results suggest that the current implementation is very poor when compared with a auto-correction system that offers the user suggested words.

8.2 Future Work

8.2.1 Input Method Editor (IME)

As the Android Wear operating system becomes more advanced and more features are available, it is likely that support for input method editors on smartwatches are more prominent. With this in mind, creating an IME for the Hoop Keyboard could potentially reap the benefits of being able to use the keyboard throughout all application within a Wear device.

8.2.2 Sentence Level Decoder

Background research has found that the VelociTap keyboard by Keith Vertanen is by far the best keyboard available for use on smartwatch sized devices. The sentence level decoder is the backbone of VelociTap's success and implementing a similar framework for the Hoop Keyboard has to be taken into consideration. With text entry speeds of over 40 WPM, VelociTap has set the new standard of performance to be expected from smartwatch devices.

8.2.3 Optimal Keyboard Layout

Mark Dunlop and John Levine have created a concept using multidimensional pareto optimisation to design an optimal keyboard layout for speed, familiarity and improved spell checking (Dunlop & Levine, 2012). Work has already been undertaken to produce an optimal keyboard layout in the Keyboard Layout Optimiser project found in Appendix O. The optimiser uses badgram tables to try and separate the common characters so that the accuracy of the language model can be enhanced. The lower the badgram score the better the layout. The program generated over 2 billion random keyboard layouts for the Hoop Keyboard and the best layout generated had a badgram score that was half of the score found on the current alphabetical layout:

Current Keyboard

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
SCORE - 11941744

Best Keyboard

X H Y A G V J S U T Q W I D O L P B E F C R K M Z N
SCORE - 6146690

With further development on the Keyboard Layout Optimiser using the concept detailed by Dunlop and Levine then an optimised alphabetical keyboard can be introduced so that the users are still familiar with the key layout as well as increasing the performance of the keyboard.

8.2.4 MaxiKeyboard

The MaxieKeyboard[1] is a Android IME with an enhanced spell-checking framework and predictive text system to offer context-sensitive suggestions along with a highlighting scheme to help support error correction (Komninos *et al.* , 2015). The keyboard was designed to improve error correction on touch-screen smartphone keyboards for older adults (Komninos *et al.* , 2015). The MaxieKeyboard highlights words that have: "serious" uncorrected errors in red & displays a red bar, "slight" auto-corrected errors in orange & displays an orange bar, been picked from the word suggestions in blue (see Appendix A.2) (Komninos *et al.* , 2015). The highlighting feedback model of the MaxieKeyboard provides excellent user feedback on errors & auto-corrections making mistakes obvious to users.

An auto-correction system based on the MaxiKeyboard was a planned discussion held during a supervisor meeting. The MaxiKeyboard had great success in helping the typing accuracy of users and could easily be applied to the Hoop Keyboard.

8.2.5 Circular Gesture to Exit

With the auto-correction system of the MaxiKeyboad in mind, the long press to exit feature of the app would be overridden by the long press to correct feature of the auto-correction. Therefore, a circular gesture to exit was devised during a supervisor meeting. The concept is simple and easy to adopt, the user would simply run their finger around the circular keyboard in order to exit the application, allowing the long press gesture to be available for a different use.

[1]MaxieKeyboard GitHub - https://github.com/komis1/MaxieKeyboard

8.3 Conclusions

A circular keyboard for circular Android Wear devices still has a potential application even in the presence of mixed results. The results suggest that the users liked the keyboard layout and with further research & development there is the possibility of the keyboard being a success. This project presented prototypes of the Hoop Keyboard to users in two formal studies, both of which were performed to a high standard. Overall the user studies were successful in collecting a vast amount of data on the performance of the users. The design of the Hoop Keyboard proved to be robust & resilient during the course of the user studies and throughout development through its use of software architectural patterns. The application showed no issues during the user studies and successfully saved all of the usage data recorded. The absence of bugs and the reliability of the design demonstrates the rigorous process followed during development.

In conclusion, the overall project and implementation of the circular keyboard can be considered successful. A number of challenges have been addressed and the ultimate goal of producing complete functional circular keyboard was achieved.

References

Arif, Ahmed Sabbir, & Stuerzlinger, Wolfgang. 2009. Analysis of text entry performance metrics. *Pages 100–105 of: 2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)*. IEEE.

Cui, Y., & Proschowsky, M. 2006. *Mobile communication terminal, system and method*.

Dunlop, Mark, & Levine, John. 2012. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, 2669.

Gamma, Erich, Helm, Richard, Johnson, Ralph, & Vlissides, John. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc.

Hunt, John. 2006. *Agile software construction*. Springer London.

Jacoby, Jacob, & Matell, Michael S. 1971. Three-Point Likert Scales Are Good Enough. *Journal of Marketing Research*, **8**(4), 495.

Komninos, Andreas, Nicol, Emma, & Dunlop, Mark D. 2015. Designed with Older Adults to SupportBetter Error Correction in SmartPhone Text Entry. *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct - MobileHCI '15*, 797–802.

Manning, CD, & Schütze, H. 1999. *Foundations of statistical natural language processing*.

Mayzner, M S, & Tresselt, M E. 1965. *Tables of Single-letter and Digram Frequency Counts for Various Word-length and Letter-position Combinations*. Psychonomic monograph supplements. Psychonomic Press.

Norvig, Peter. 2013. *English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU.*

Oney, Stephen, Harrison, Chris, Ogan, Amy, & Wiese, Jason. 2013. ZoomBoard. *Page 2799 of: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13.* ACM Press.

Proschowsky, Morten. 2008. Adaptive Text Entry for Mobile Devices.

Schwaber, Ken, & Sutherland, Jeff. 2013. The Scrum Guide. *Scrum.Org and ScrumInc*, 17.

Vertanen, Keith, Memmi, Haythem, Emge, Justin, Reyal, Shyam, & Kristensson, Per Ola. 2015. VelociTap : Investigating Fast Mobile Text Entry using Sentence-Based Decoding of Touchscreen Keyboard Input. *Chi 2015*, 659–668.

Appendices

Appendix A

Related Work

A.1 English Letter Frequency Counts

A.1.1 Letter Counts

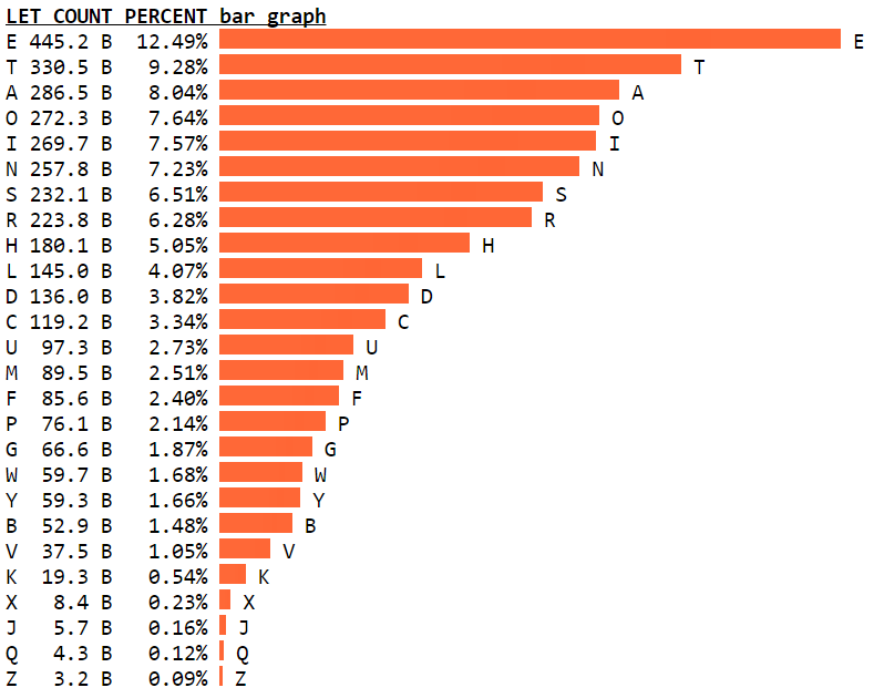


Figure A.1: The letter counts for each letter in billions (Norvig, 2013).

A.1.2 Top 50 Word Counts

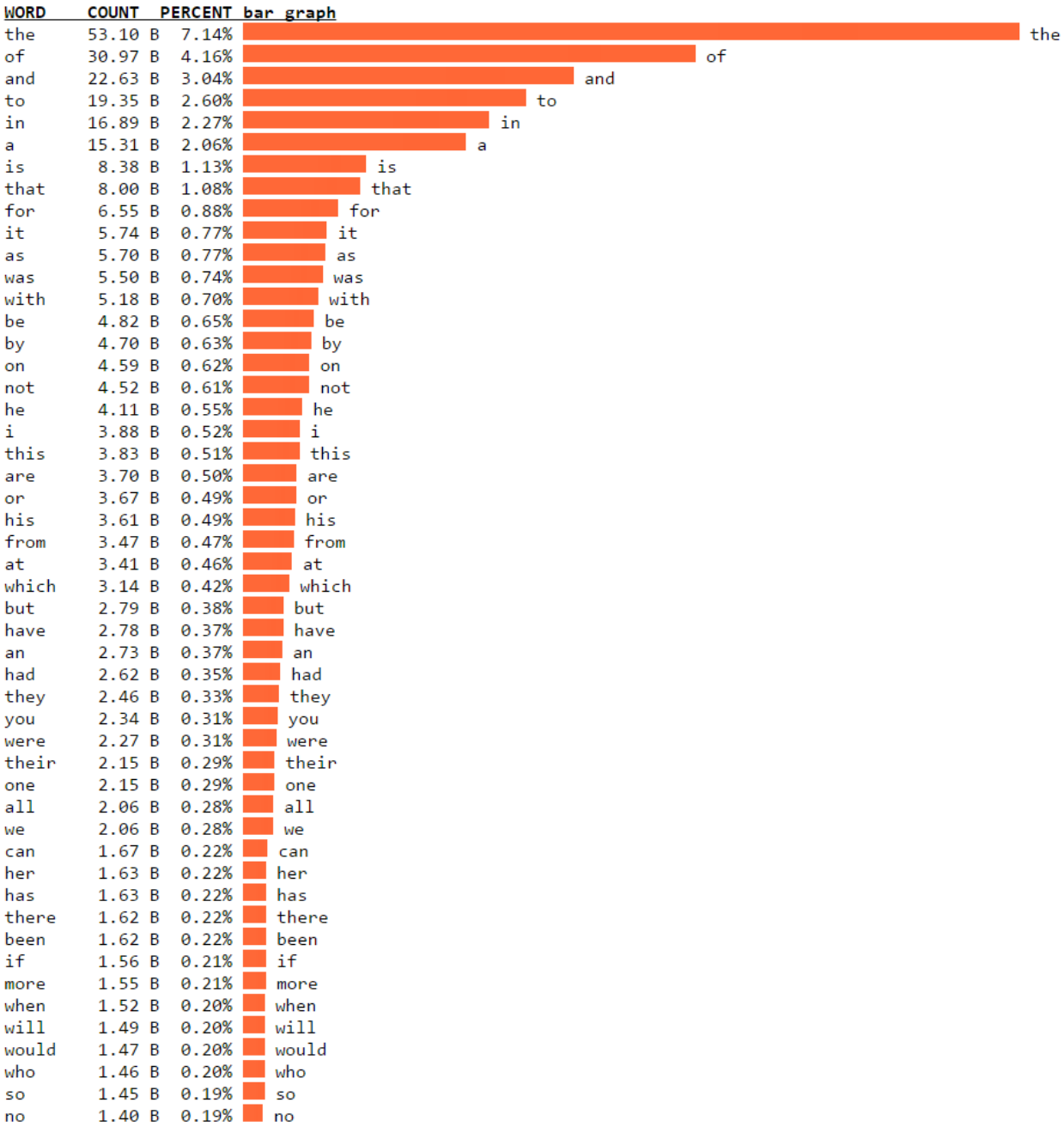


Figure A.2: The top 50 word counts in billions (Norvig, 2013).

A.1.3 Top 50 N-grams

1	2grams	3grams	4-grams	5-grams	6-grams	7-grams	8-grams	9-grams
e	th	the	tion	ation	ations	present	differen	different
t	he	and	atio	tions	ration	ational	national	governmen
a	in	ing	that	which	tional	through	consider	overnment
o	er	ion	ther	ction	nation	between	position	formation
i	an	tio	with	other	ection	ication	ifferent	character
n	re	ent	ment	their	cation	differe	governme	velopment
s	on	ati	ions	there	lation	ifferen	vernment	developme
r	at	for	this	ition	though	general	overnmen	evelopmen
h	en	her	here	ement	presen	because	interest	condition
l	nd	ter	from	inter	tation	develop	importan	important
d	ti	hat	ould	ional	should	america	ormation	articular
c	es	tha	ting	ratio	resent	however	formatio	particula
u	or	ere	hich	would	genera	eration	relation	represent
m	te	ate	whic	tiona	dition	nationa	question	individua
f	of	his	ctio	these	ationa	conside	american	ndividual
p	ed	con	ence	state	produc	onsider	characte	relations
g	is	res	have	natio	throug	ference	haracter	political
w	it	ver	othe	thing	hrough	positio	articula	informati
y	al	all	ight	under	etween	osition	possible	nformatio
b	ar	ons	sion	ssion	betwee	ization	children	universit
v	st	nce	ever	ectio	differ	fferent	elopment	following
k	to	men	ical	catio	icatio	without	velopmen	experienc
x	nt	ith	they	latio	people	ernment	developm	stitution
j	ng	ted	inte	about	iffere	vernmen	evelopme	xperience
q	se	ers	ough	count	fferen	overnme	conditio	education
z	ha	pro	ance	ments	struct	governm	ondition	roduction
	as	thi	were	rough	action	ulation	mportant	niversity
	ou	wit	tive	ative	person	another	rticular	therefore
	io	are	over	prese	eneral	importa	particul	nstitutio
	le	ess	ding	feren	system	interes	epresent	ification
	ve	not	pres	hough	relati	nterest	represen	establish
	co	ive	nter	ution	ctions	elation	increase	understan
	me	was	comp	roduc	ecause	rmation	individu	nderstand
	de	ect	able	resen	becaus	mportan	ndividua	difficult
	hi	rea	heir	thoug	before	product	dividual	structure
	ri	com	thei	press	ession	formati	elations	knowledge
	ro	eve	ally	first	develo	communi	nformati	struction
	ic	per	ated	after	evelop	lations	politica	something
	ne	int	ring	cause	uction	ormatio	olitical	necessary
	ea	est	ture	where	change	certain	universi	hemselves
	ra	sta	cont	tatio	follow	increas	function	themselve
	ce	cti	ents	could	positi	relatio	informat	plication
	li	ica	cons	efore	govern	special	niversit	anization
	ch	ist	rati	contr	sition	process	iversity	according
	ll	ear	thin	hould	merica	against	lication	differenc
	be	ain	part	shoul	direct	problem	experien	operation
	ma	one	form	tical	bility	nstitut	structur	ifference
	si	our	ning	gener	effect	politic	determin	rganizati
	om	iti	ecti	esent	americ	ination	ollowing	organizat
	ur	rat	some	great	public	univers	followin	ganizatio

Figure A.3: The 50 most common n-grams from unigram to 9-gram (Norvig, 2013).

A.2 MaxieKeyboard

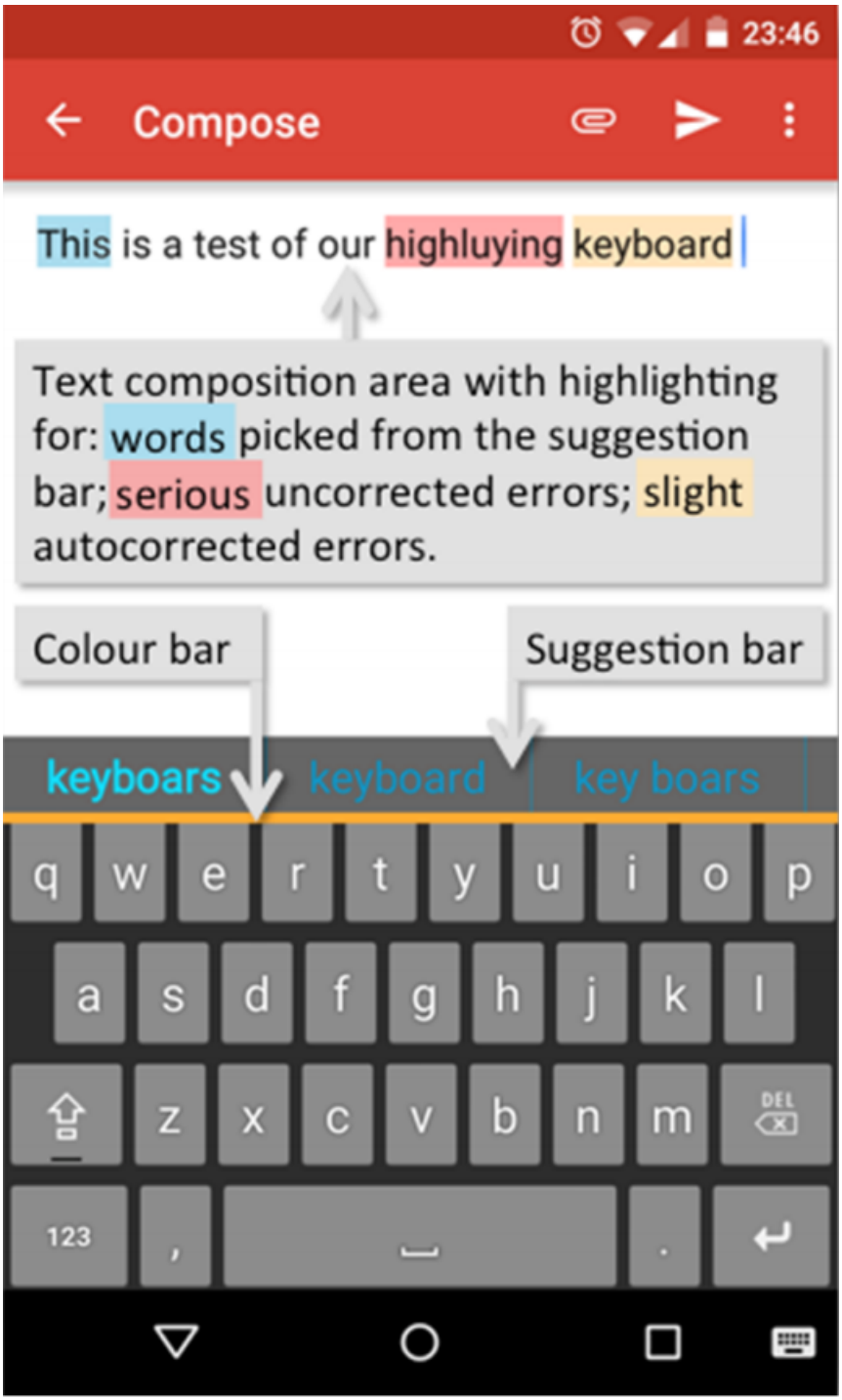
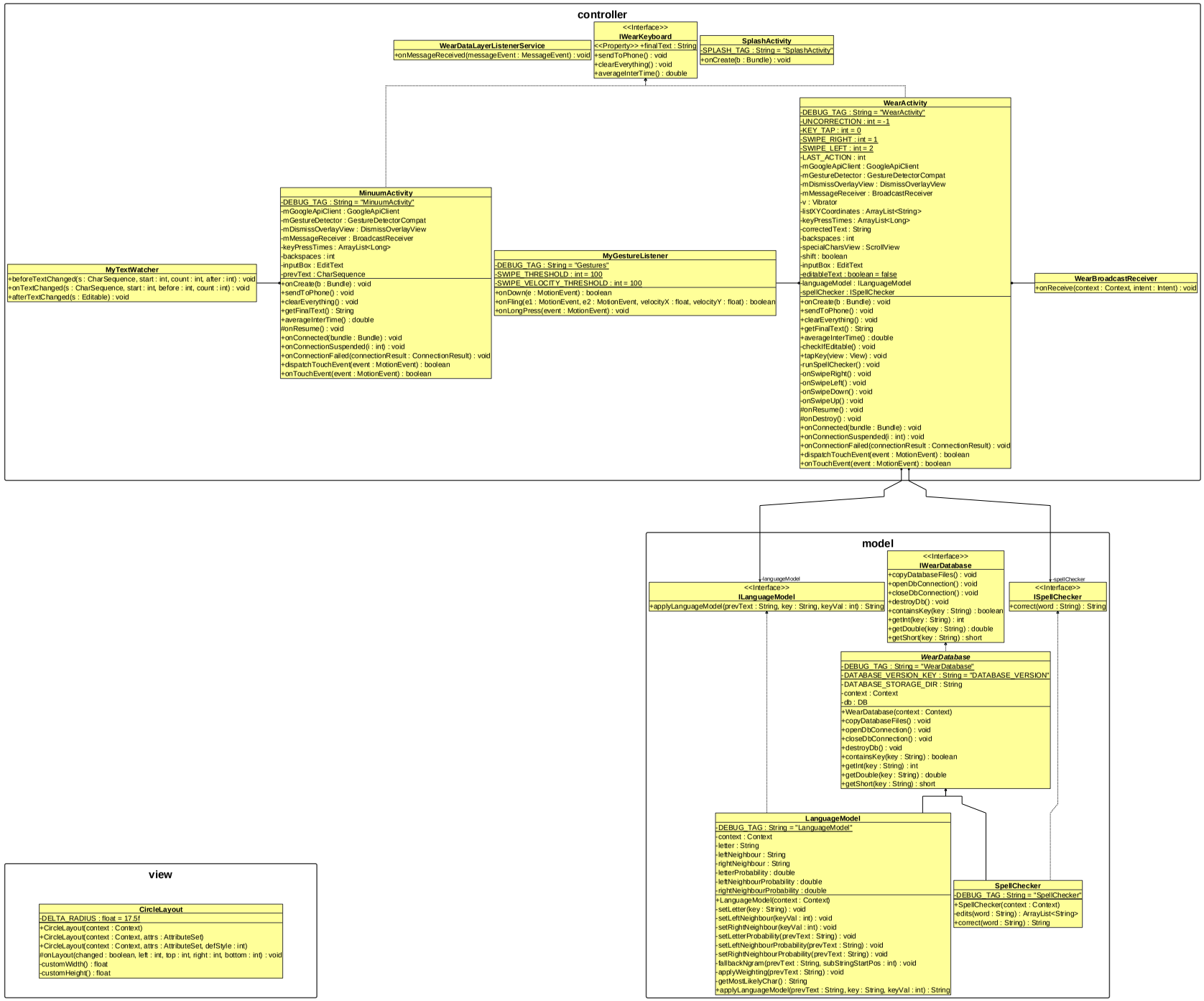


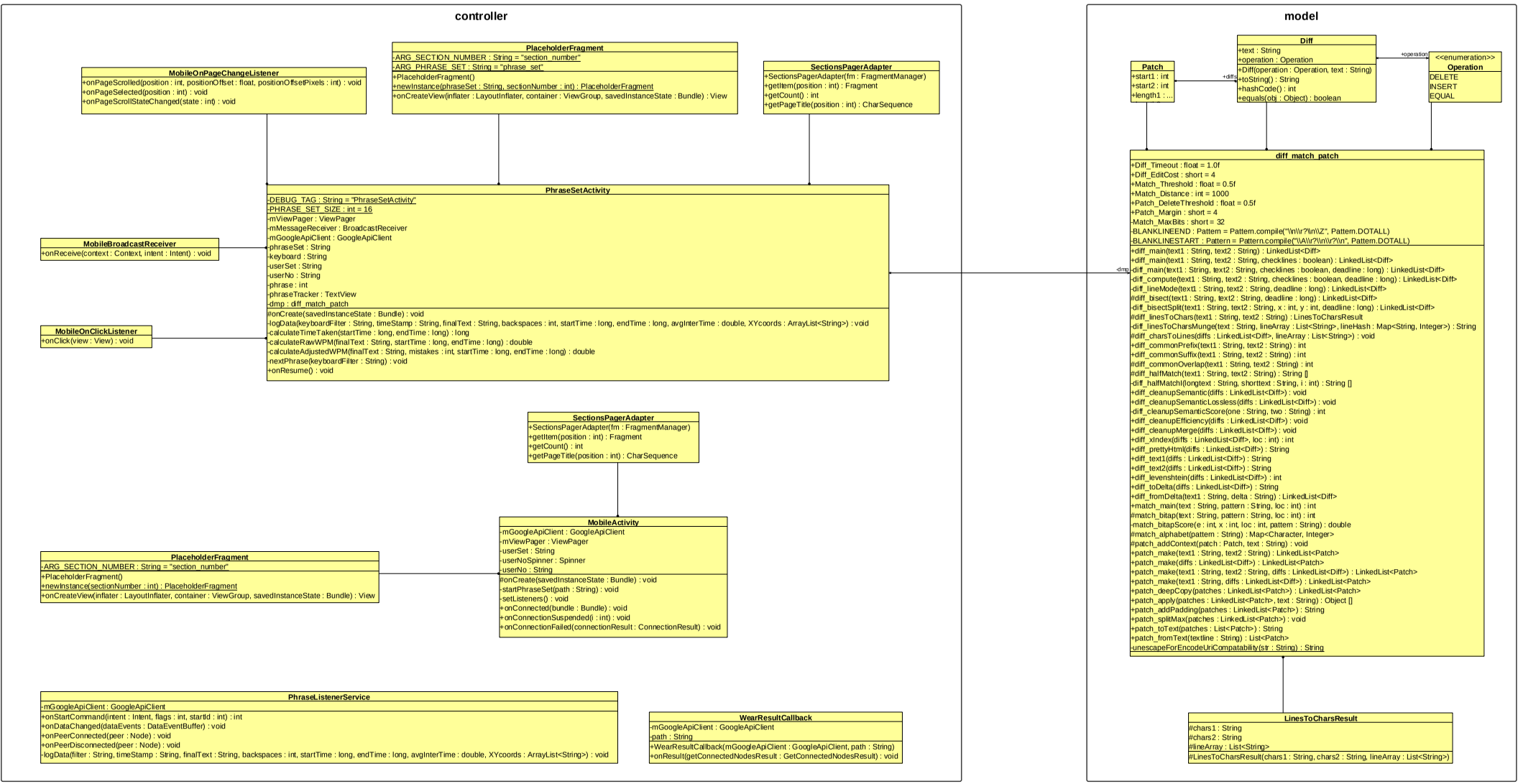
Figure A.4: Overview of the MaxieKeyboard features (Komninos *et al.* , 2015).

Appendix B

Hoop Keyboard Class Diagram

Android Wear applications require a linked mobile module in order to install the Android package file on the Wear device. I have included the final class diagrams for both the Wear module and the Mobile module below respectfully.


controller
WearDataLayerListenerService
+onMessageReceived(messageEvent : MessageEvent) : void
<<Interface>>
IWearKeyboard
<<Property>> +finalText : String
+sendToPhone() : void
+clearEverything() : void
+averageInterTime() : double
SplashActivity
-SPLASH_TAG : String = "SplashActivity"
+onCreate(b : Bundle) : void
WearActivity
-DEBUG_TAG : String = "WearActivity"
-UNCORRECTION : int = -1
-KEY_TAP : int = 0
-SWIPE_RIGHT : int = 1
-SWIPE_LEFT : int = 2
-LAST_ACTION : int
-mGoogleApiClient : GoogleApiClient
-mGestureDetector : GestureDetectorCompat
-mDismissOverlayView : DismissOverlayView
-mMessageReceiver : BroadcastReceiver
-v : Vibrator
-listXYCoordinates : ArrayList<String>
-keyPressTimes : ArrayList<Long>
-correctedText : String
-backspaces : int
-specialCharsView : ScrollView
-shift : boolean
-inputBox : EditText
-editableText : boolean = false
-languageModel : ILanguageModel
-spellChecker : ISpellChecker
+onCreate(b : Bundle) : void
+sendToPhone() : void
+clearEverything() : void
+getFinalText() : String
+averageInterTime() : double
-checkIfEditable() : void
+tapKey(view : View) : void
-runSpellChecker() : void
-onSwipeRight() : void
-onSwipeLeft() : void
-onSwipeDown() : void
-onSwipeUp() : void
#onResume() : void
#onDestroy() : void
+onConnected(bundle : Bundle) : void
+onConnectionSuspended(i : int) : void
+onConnectionFailed(connectionResult : ConnectionResult) : void
+dispatchTouchEvent(event : MotionEvent) : boolean
+onTouchEvent(event : MotionEvent) : boolean
MinuumActivity
-DEBUG_TAG : String = "MinuumActivity"
-mGoogleApiClient : GoogleApiClient
-mGestureDetector : GestureDetectorCompat
-mDismissOverlayView : DismissOverlayView
-mMessageReceiver : BroadcastReceiver
-keyPressTimes : ArrayList<Long>
-backspaces : int
-inputBox : EditText
-prevText : CharSequence
+onCreate(b : Bundle) : void
+sendToPhone() : void
+clearEverything() : void
+getFinalText() : String
+averageInterTime() : double
#onResume() : void
+onConnected(bundle : Bundle) : void
+onConnectionSuspended(i : int) : void
+onConnectionFailed(connectionResult : ConnectionResult) : void
+dispatchTouchEvent(event : MotionEvent) : boolean
+onTouchEvent(event : MotionEvent) : boolean
MyGestureListener
-DEBUG_TAG : String = "Gestures"
-SWIPE_THRESHOLD : int = 100
-SWIPE_VELOCITY_THRESHOLD : int = 100
+onDown(e : MotionEvent) : boolean
+onFling(e1 : MotionEvent, e2 : MotionEvent, velocityX : float, velocityY : float) : boolean
+onLongPress(event : MotionEvent) : void
MyTextWatcher
+beforeTextChanged(s : CharSequence, start : int, count : int, after : int) : void
+onTextChanged(s : CharSequence, start : int, before : int, count : int) : void
+afterTextChanged(s : Editable) : void
WearBroadcastReceiver
+onReceive(context : Context, intent : Intent) : void
model
<<Interface>>
ILanguageModel
+applyLanguageModel(prevText : String, key : String, keyVal : int) : String
-languageModel
<<Interface>>
IWearDatabase
+copyDatabaseFiles() : void
+openDbConnection() : void
+closeDbConnection() : void
+destroyDb() : void
+containsKey(key : String) : boolean
+getInt(key : String) : int
+getDouble(key : String) : double
+getShort(key : String) : short
-spellChecker
<<Interface>>
ISpellChecker
+correct(word : String) : String
WearDatabase
-DEBUG_TAG : String = "WearDatabase"
-DATABASE_VERSION_KEY : String = "DATABASE_VERSION"
-DATABASE_STORAGE_DIR : String
-context : Context
-db : DB
+WearDatabase(context : Context)
+copyDatabaseFiles() : void
+openDbConnection() : void
+closeDbConnection() : void
+destroyDb() : void
+containsKey(key : String) : boolean
+getInt(key : String) : int
+getDouble(key : String) : double
+getShort(key : String) : short
LanguageModel
-DEBUG_TAG : String = "LanguageModel"
-context : Context
-letter : String
-leftNeighbour : String
-rightNeighbour : String
-letterProbability : double
-leftNeighbourProbability : double
-rightNeighbourProbability : double
+LanguageModel(context : Context)
-setLetter(key : String) : void
-setLeftNeighbour(keyVal : int) : void
-setRightNeighbour(keyVal : int) : void
-setLetterProbability(prevText : String) : void
-setLeftNeighbourProbability(prevText : String) : void
-setRightNeighbourProbability(prevText : String) : void
-fallbackNgram(prevText : String, subStringStartPos : int) : void
-applyWeighting(prevText : String) : void
-getMostLikelyChar() : String
+applyLanguageModel(prevText : String, key : String, keyVal : int) : String
SpellChecker
-DEBUG_TAG : String = "SpellChecker"
+SpellChecker(context : Context)
-edits(word : String) : ArrayList<String>
+correct(word : String) : String
view
CircleLayout
-DELTA_RADIUS : float = 17.5f
+CircleLayout(context : Context)
+CircleLayout(context : Context, attrs : AttributeSet)
+CircleLayout(context : Context, attrs : AttributeSet, defStyle : int)
#onLayout(changed : boolean, left : int, top : int, right : int, bottom : int) : void
-customWidth() : float
-customHeight() : float


controller
model
MobileOnPageChangeListener
+onPageScrolled(position : int, positionOffset : float, positionOffsetPixels : int) : void
+onPageSelected(position : int) : void
+onPageScrollStateChanged(state : int) : void
PlaceholderFragment
-ARG_SECTION_NUMBER : String = "section_number"
-ARG_PHRASE_SET : String = "phrase_set"
+PlaceholderFragment()
+newInstance(phraseSet : String, sectionNumber : int) : PlaceholderFragment
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View
SectionsPagerAdapter
+SectionsPagerAdapter(fm : FragmentManager)
+getItem(position : int) : Fragment
+getCount() : int
+getPageTitle(position : int) : CharSequence
PhraseSetActivity
-DEBUG_TAG : String = "PhraseSetActivity"
-PHRASE_SET_SIZE : int = 16
-mViewPager : ViewPager
-mMessageReceiver : BroadcastReceiver
-mGoogleApiClient : GoogleApiClient
-phraseSet : String
-keyboard : String
-userSet : String
-userNo : String
-phrase : int
-phraseTracker : TextView
-dmp : diff_match_patch
#onCreate(savedInstanceState : Bundle) : void
-logData(keyboardFilter : String, timeStamp : String, finalText : String, backspaces : int, startTime : long, endTime : long, avgInterTime : double, XYcoords : ArrayList<String>) : void
-calculateTimeTaken(startTime : long, endTime : long) : long
-calculateRawWPM(finalText : String, startTime : long, endTime : long) : double
-calculateAdjustedWPM(finalText : String, mistakes : int, startTime : long, endTime : long) : double
-nextPhrase(keyboardFiller : String) : void
+onResume() : void
MobileBroadcastReceiver
+onReceive(context : Context, intent : Intent) : void
MobileOnClickListener
+onClick(view : View) : void
SectionsPagerAdapter
+SectionsPagerAdapter(fm : FragmentManager)
+getItem(position : int) : Fragment
+getCount() : int
+getPageTitle(position : int) : CharSequence
MobileActivity
-mGoogleApiClient : GoogleApiClient
-mViewPager : ViewPager
-userSet : String
-userNoSpinner : Spinner
-userNo : String
#onCreate(savedInstanceState : Bundle) : void
-startPhraseSet(path : String) : void
-setListeners() : void
+onConnected(bundle : Bundle) : void
+onConnectionSuspended(i : int) : void
+onConnectionFailed(connectionResult : ConnectionResult) : void
PlaceholderFragment
-ARG_SECTION_NUMBER : String = "section_number"
+PlaceholderFragment()
+newInstance(sectionNumber : int) : PlaceholderFragment
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View
PhraseListenerService
-mGoogleApiClient : GoogleApiClient
+onStartCommand(intent : Intent, flags : int, startId : int) : int
+onDataChanged(dataEvents : DataEventBuffer) : void
+onPeerConnected(peer : Node) : void
+onPeerDisconnected(peer : Node) : void
-logData(filter : String, timeStamp : String, finalText : String, backspaces : int, startTime : long, endTime : long, avgInterTime : double, XYcoords : ArrayList<String>) : void
WearResultCallback
-mGoogleApiClient : GoogleApiClient
-path : String
+WearResultCallback(mGoogleApiClient : GoogleApiClient, path : String)
+onResult(getConnectedNodesResult : GetConnectedNodesResult) : void
Diff
+text : String
+operation : Operation
+Diff(operation : Operation, text : String)
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
+operation
<<enumeration>>
Operation
DELETE
INSERT
EQUAL
Patch
+start1 : int
+start2 : int
+length1 : ...
+diffs
diff_match_patch
+Diff_Timeout : float = 1.0f
+Diff_EditCost : short = 4
+Match_Threshold : float = 0.5f
+Match_Distance : int = 1000
+Patch_DeleteThreshold : float = 0.5f
+Patch_Margin : short = 4
-Match_MaxBits : short = 32
-BLANKLINEEND : Pattern = Pattern.compile("\\n\\r?\\n\\Z", Pattern.DOTALL)
-BLANKLINESTART : Pattern = Pattern.compile("\\A\\r?\\n\\r?\\n", Pattern.DOTALL)
+diff_main(text1 : String, text2 : String) : LinkedList<Diff>
+diff_main(text1 : String, text2 : String, checklines : boolean) : LinkedList<Diff>
-diff_main(text1 : String, text2 : String, checklines : boolean, deadline : long) : LinkedList<Diff>
-diff_compute(text1 : String, text2 : String, checklines : boolean, deadline : long) : LinkedList<Diff>
-diff_lineMode(text1 : String, text2 : String, deadline : long) : LinkedList<Diff>
#diff_bisect(text1 : String, text2 : String, deadline : long) : LinkedList<Diff>
-diff_bisectSplit(text1 : String, text2 : String, x : int, y : int, deadline : long) : LinkedList<Diff>
#diff_linesToChars(text1 : String, text2 : String) : LinesToCharsResult
-diff_linesToCharsMunge(text : String, lineArray : List<String>, lineHash : Map<String, Integer>) : String
#diff_charsToLines(diffs : LinkedList<Diff>, lineArray : List<String>) : void
+diff_commonPrefix(text1 : String, text2 : String) : int
+diff_commonSuffix(text1 : String, text2 : String) : int
#diff_commonOverlap(text1 : String, text2 : String) : int
#diff_halfMatch(text1 : String, text2 : String) : String []
-diff_halfMatchI(longtext : String, shorttext : String, i : int) : String []
+diff_cleanupSemantic(diffs : LinkedList<Diff>) : void
+diff_cleanupSemanticLossless(diffs : LinkedList<Diff>) : void
-diff_cleanupSemanticScore(one : String, two : String) : int
+diff_cleanupEfficiency(diffs : LinkedList<Diff>) : void
+diff_cleanupMerge(diffs : LinkedList<Diff>) : void
+diff_xIndex(diffs : LinkedList<Diff>, loc : int) : int
+diff_prettyHtml(diffs : LinkedList<Diff>) : String
+diff_text1(diffs : LinkedList<Diff>) : String
+diff_text2(diffs : LinkedList<Diff>) : String
+diff_levenshtein(diffs : LinkedList<Diff>) : int
+diff_toDelta(diffs : LinkedList<Diff>) : String
+diff_fromDelta(text1 : String, delta : String) : LinkedList<Diff>
+match_main(text : String, pattern : String, loc : int) : int
#match_bitap(text : String, pattern : String, loc : int) : int
-match_bitapScore(e : int, x : int, loc : int, pattern : String) : double
#match_alphabet(pattern : String) : Map<Character, Integer>
#patch_addContext(patch : Patch, text : String) : void
+patch_make(text1 : String, text2 : String) : LinkedList<Patch>
+patch_make(diffs : LinkedList<Diff>) : LinkedList<Patch>
+patch_make(text1 : String, text2 : String, diffs : LinkedList<Diff>) : LinkedList<Patch>
+patch_make(text1 : String, diffs : LinkedList<Diff>) : LinkedList<Patch>
+patch_deepCopy(patches : LinkedList<Patch>) : LinkedList<Patch>
+patch_apply(patches : LinkedList<Patch>, text : String) : Object []
+patch_addPadding(patches : LinkedList<Patch>) : String
+patch_splitMax(patches : LinkedList<Patch>) : void
+patch_toText(patches : List<Patch>) : String
+patch_fromText(textline : String) : List<Patch>
-unescapeForEncodeUriCompatability(str : String) : String
-dmp
LinesToCharsResult
#chars1 : String
#chars2 : String
#lineArray : List<String>
#LinesToCharsResult(chars1 : String, chars2 : String, lineArray : List<String>)

Appendix C

Project Plan

C.1 Original Project Plan

Circular Watch Text		2015																						
Deliverables	**W**	Nov	46	47	48	Dec	50	51	52	53	Jan	2	3	4	Feb	6	7	8	Mar	10	11	12	13	
Planning phase	**6**	**1**	**2**	**3**	**4**	**5**	**6**																	
Identifying The Unknown Unknowns	2	n	n																					
Background Research of Language Models	6	n	n	n	n	n	n																	
Investigation of IME on Android Wear	2	n	n																					
Android Studio IDE Setup	1	n																						
Creating My First Android Wear App	1	n																						
Exploration Of Android APIs	2	n	n																					
Requirement Specification	2	n	n																					
Design Specification	3	n	n	n																				
Test Specification	2		n	n																				
Project Poster (DUE 09/12/2015)	3				n	n	n																	
Prototype 1	**3**							**1**	**2**	**3**														
Create Empty Android Wear App/IME	1							n																
Develop Circular Keyboard In App/IME	2							n	n															
Display Text Inputted	2								n	n														
Prototype 2	**3**										**1**	**2**	**3**											
Implement Simple Language Model	3										n	n	n											
Prototype 3	**3**													**1**	**2**	**3**								
Implement Complex Language Model	3													n	n	n								
Evaluation phase	**3**																**1**	**2**	**3**					
User Testing	1																n							
Performance Analysis	3																n	n	n					
Project Report	**10**													**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Project Progress Report 1 (05/02/16)	2													n	n									
Project Progress Report 2 (04/03/16)	2																	n	n					
Project Report (DUE 30/03/2016)	4																			n	n	n	n	

Figure C.1: The original project plan designed for the project

C.2 Final Project Plan

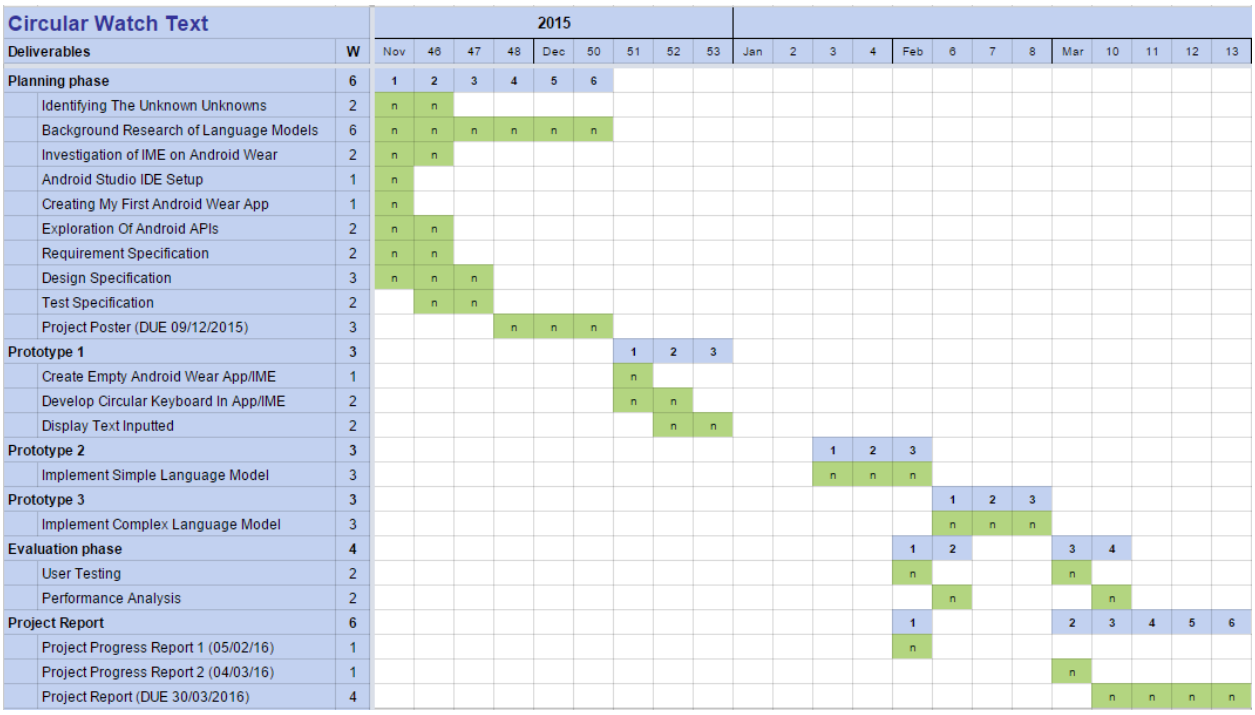


Figure C.2: The final project plan undertaken during the project

Appendix D

Summary of Meetings

02/11/15

This was an initial project meeting with all of the students undertaking one of Mark's projects. This was an introductory meeting and an opportunity for Mark to hand out the hardware for development purposes.

09/11/15

I showed Mark my Project scope and outline plan which he was generally pleased with. Mark suggested a few alterations to the project and tasked me with choosing either developing an input method editor (IME) or an app from the outset. I decided to go with the app for now. I also had to change my Gantt chart because I put in the wrong starting date.

23/11/15

I demonstrated my first Android Wear Skeleton app and the showed the Minuum keyboard installed on my watch (which Mark believes would be a great comparison for my evaluation).

08/12/15

I showed Mark my progress on the poster which was to be demonstrated to Marilyn on 09/12/15. Mark felt there was too much text on the slides and to change them to small bullet points and talk about them instead. Mark also suggested including a storyboard into the poster to demonstrate a simple task carried out by the user. Mark also suggested to

only have buttons for the letters in the alphabet in the keyboard and to have functionality to swipe down to access a menu of other characters.

20/01/16

I demonstrated the progress I made over the holidays for prototype 1 and he was happy enough to move onto prototype 2. We decided the best "Simple" language model would be to go for an English bigram language model. Mark provided me with a link for the bigram data. I also tried to send the app package .apk file for Mark to install on his own devices but I realised I had to build a mobile module for this to work.

27/01/16

We decided what we would do for prototype 3, I will implement an advanced language model using trigram data with smoothing with the addition of a spell checker. Mark also suggested to add a flag in to disable the cursor handler of the edittext box for user testing. We also decided to do user testing on prototype 2 over the next week, which meant I will have to apply for ethics approval and also extend the current mobile app to display phrases for the user to type, and log usage data. This meant I had to download and randomise phrases taken from the Vertanen and Kristensson standard Enron phrase set that didn't have any special characters. Mark also suggested to add more bigram data which would take into account the likelihood of letters appearing after a space (" "). This would mean I have to parse the British National Corpus (BNC) data in order to get the frequency of words beginning with each letter of the alphabet. We also decided to integrate Levenshtein's distance into prototype 3 which will detect how far away the user's tap was from the intended target and use this to make a more accurate prediction.

03/02/16

During today's meeting I gave Mark the first opportunity to test out my new application for logging user's data by giving them random phrases to complete. Mark gave me guidance for what to include in my user participation consent form which I was tasked with making this week. Mark also suggested adding the "—" as a caret so the user has a better idea of where their typed text will be inserted. Also for the next prototype Mark suggested to put a small red dot for the last position of user's tap on keyboard (display for first 4 phrases).

17/02/16

The meeting with Mark today was really good, and he was pleased with the progress I had made on Prototype 3. The language model is much better and we discussed the next stage of user testing. I will be carrying out another study using the same participants and I will be doing the study on Prototype 3 (with spell checker) vs The Minuum Keyboard. We also discussed what the Spell checker should have the following features:

- If the typed word is not a valid word, and the spell checker has a really good suggestion, then autocorrect word (and vibrate).
- A Really good suggestion is:
  - Word length ¡= 5 and levenshtein dist = 1
  - Word length ¿ 5 && ¡= 10 and levenshtein dist = 2
- Run the spell checker on swipe right (Insert Space)
- Unchange correction on swipe left (Backspace)
- Use Keith Vertanens Big English Word Lists for the dictionary. Need to watch out for american spellings (manually add in British spellings when needed).

Time to get started with the user study.

22/02/16

I had a skype/phone call with Mark today just to get some advice/guidance on the next steps. After the phone call it became apparent that there was a few changes that could be made to the application:

- Reluctant to change keyboard layout (keep it as it is).
- Use a different weighting for first character pressed. Use 75% instead of 40% but make the common keys (“A”, “I”, “O”, “P”, “S”, “T”, “W”) buttons bigger (2dp top, 1dp bottom).
- Create an app for the Minuum keyboard as a comparison.
- Use the same phrases for all user studies.

I have changed the language model to give a weighting of 75% and I have made the common keys 10% wider and 10% taller than the standard key size.

25/02/16

Had my meeting with Mark today. We discussed making some alterations to the current app and how to proceed with the user testing. List of things discussed:

- The name of the keyboard: "The Hoop Keyboard"
- Screenshot long text on the Minuum to demonstrate hidden text and how much of the screen the qwerty keyboard takes up
- Calculate the inter-time (time between two characters) for the report
- Detect the X Y coordinates of the user's tap on the grey circle and select the nearest button
- Get 16 people for user testing, 1 phrase set (15 phrases) per keyboard
- Provide a practice phrase on both keyboards
- Don't worry about capitalization, punctuation, and only use backspace to delete
- Randomise half of the users start with Hoop, the other half on Minuum
  - One phrase set on each
  - M1→H2
  - H1→M2
  - M2→H1
  - H2→M1
- Allow user to use word suggestion on Minuum
- The User Testing Process
  - Do phrase set
  - Nasa tlx
  - Second phrase set

- another Nasa tlx
- Exit Questionnaire
  * "Which method of space did you prefer?" swipe — button
  * "Which method of backspace did you prefer?" swipe — button
  * "Which Layout did you prefer?" straight — circular
  * "Did you have any issues with certain letters or inputs?"
  * "The hoop keyboard does auto correct, the minuum keyboard get you to pick words, which do you prefer"
  * "Overall what keyboard do you prefer?" Hoop — Minuum
  * "Comments. . . ."

Instead of implementing the X Y coordinates I made the circle of buttons bigger, so they touched the outer edge of the screen of the device.

03/03/16

I had my meeting with Mark today, we discussed how the user study was going so far with 3 participants completed so far. Mark also suggested to calculate the screen coverage of each keyboard and include the result in the report. Future work was also a topic of conversation and Mark suggested citing his "Maxiekeyboard" paper for the future adaptation for the spell checker. The spell checker highlights words that are still mistakes and a long press is used to get suggestions. A circular gesture to Exit was also a possibility for future work as the current long press is used to exit the app and a suggestion to replace this with a circular gesture is a possibility. Mark also recommended including state transition networks (STN) for the current flow of the app.

09/03/16

Mark suggested to build additional graphs:

- The 8 fastest users on Hoop — The 8 slowest users on Hoop
- 8 fastest on Minuum — 8 slowest on Minuum (Might be a greater separation in this graph)

Corrected Mistakes will be an interesting comparison (i.e. for 8 slowest on Hoop vs 8 slowest on Minuum). Mark touched on the fact that 18-25 year olds is a hard demographic and work

has identified that this group is reluctant to change due to the upbringing on QWERTY keyboards. Mark suggested to use "I" in the report instead of "We" or to go neutral "This shows...". The names of the keyboards should be:

- "Hoop Prototype 1"
- "Hoop Prototype 2"
- "Hoop Keyboard"
- "Minuum Keyboard"

Mark also suggested to construct "Pair-Wise T.Tests" on participants means:

- One Test - 16 pairs, Minuum mean vs Hoop mean
- Excel command: T.TEST(array1, array2, tails, type)
- Expected results: ¡= 0.10 or ¡=0.05

Mark said it was a good idea to include the main results in the main report and to include minor results in the appendix. We also touched on some future work in the form of:

- Optimised Keyboard Layout
  - Badgrams (Available on Mark Dunlops page)
  - Badgram score : "A" = (AB + AZ)
  - Identify 6 worst keys and separate them
  - Steps:
    1. Sum badgram scores : "A" - "Z"
    2. Randomly swap two keys with their neighbour, repeat Step 1.
    3. If score is better then keep layout and repeat Step 2. Else revert change back and randomly swap another two keys.

16/03/16

I showed Mark the testing that I had conducted on the system. He suggested changing the package name to a unique domain such as "hoopkeyboard.camerondrennan.com". Mark also suggested looking at another couple of keyboards: "Tilt text" and "Zoomboard". The paper "Clio and the economics of QWERTY" was also recommended to read.

Appendix E

Code References

The following are the sources of various code that were used within the project, as well as any frameworks or 3rd party software that was used. The Javadoc documentation includes the exact location of where each code snippet was applied.

- Data Syncing tutorial by E John Feig:
  `https://www.youtube.com/watch?v=jbbKCfCIyFA`
- Android Processes and Threads Guide by Google:
  `http://developer.android.com/guide/components/processes-and-threads.html`
- Method onFling obtained from Mirek Rusin:
  `http://stackoverflow.com/questions/4139288/android-how-to-handle-right-to-left-swipe-gestures`
- Android Sending and Receiving Messages Tutorial by Google:
  `http://developer.android.com/training/wearables/data-layer/messages.html`
- Spell Checker by Rael Cunha:
  `http://raelcunha.com/spell-correct/`
- Method createFileFromInputStream obtained from yugidroid: `http://stackoverflow.com/questions/11820142/how-to-pass-a-file-path-which-is-in-assets-folder-to-filestring-path`
- SnappyDB by Nabil Hachicha:
  `https://github.com/nhachicha/SnappyDB`
- Class "CircleLayout" obtained from tutorial by Matt Stoker
  `https://youtu.be/S2YojyVhZcY?t=17m46s`
- Android Spinner Tutorial by Google:
  `http://developer.android.com/guide/topics/ui/controls/spinner.html`

- Handling Data Layer Events Tutorial by Google:
  http://developer.android.com/training/wearables/data-layer/events.html
- Diff-Match-Patch library by Google:
  https://code.google.com/archive/p/google-diff-match-patch/
- JUnit 4:
  http://junit.org/junit4/
- Mockito:
  https://code.google.com/archive/p/mockito/
- Espresso:
  https://google.github.io/android-testing-support-library/docs/espresso/
- Hamcrest library:
  http://hamcrest.org/JavaHamcrest/
- Heatmap from X Y values in R by Aniko:
  http://stackoverflow.com/questions/5004063/creating-a-heat-map-from-x-y-corrdinates-in-r
- Android Lint:
  http://developer.android.com/tools/help/lint.html
- EclEmma
  http://eclemma.org/installation.html

Appendix F

Detailed Test Cases

These tests were carried out to test components of the Hoop Keyboard project. The code for the following tests can be found in the "androidTest" folders within the "Prototype3" Android project. Tests were also conducted on the analysis software and they can be found within the code of their respective Java projects included in the zip file.

F.1 Hoop Keyboard

Test	Test all the keys of the Hoop Keyboard
Input	"A b c d e f g h i j k l m n o p q r s t u v w x y z"
Expected Result	The text field will contain "A b c d e f g h i i k l m n o p q r s t t v w w y a" because the language model will correct the 'j', 'u' 'x', 'z' characters to one of their neighbours.
Result	Pass

Test	Test the insertion of a space
Input	Swipe right
Expected Result	The text field will contain the space character ' '
Result	Pass

Test	Test the deletion of a character
Input	'B' + swipe left
Expected Result	The text field will contain the empty String ""
Result	Pass

Test	Test the application of the language model on a misspelled word
Input	"Worling"
Expected Result	"Working"
Result	Pass

Test	Test the application of the spelling corrector on a misspelled word
Input	"Worp" + swipe right
Expected Result	"Work"
Result	Pass

Test	Test the application of the spelling corrector on a completely unrecognisable input
Input	"Wwrpwwrpwwrp" + swipe right
Expected Result	"Wwrpwwrpwwrp"
Result	Pass

Test	The most probable character returned by the language model with previously typed text
Input	"O" with "hell" previously typed
Expected Result	"O"
Result	Pass

Test	The most probable character returned by the language model at the start of a word
Input	"I"
Expected Result	"I"
Result	Pass

Test	The most probable character returned by the language model at the start of a word
Input	“J”
Expected Result	“I” - The language model should correct ‘J’ to ‘I’ based on their probabilities
Result	Pass

Test	The most probable character returned by the language model when it doesn’t recognise the n-gram
Input	“E” with “icaptur” previously typed
Expected Result	“E”
Result	Pass

Test	The most probable word returned by the spelling corrector
Input	“helloo”
Expected Result	“hello”
Result	Pass

Test	An unknown input for the spelling corrector
Input	“ieblviabbaekcakckvjebview”
Expected Result	“ieblviabbaekcakckvjebview”
Result	Pass

F.2 Minuum Keyboard

Test	Test all the keys of the Minuum Keyboard
Input	“a b c d e f g h i j k l m n o p q r s t u v w x y z”
Expected Result	“a b c d e f g h i j k l m n o p q r s t u v w x y z”
Result	Pass

Test	Test all the keys of the Minuum Keyboard including the space bar
Input	“a b c d e f g h i j k l m n o p q r s t u v w x y z”
Expected Result	“a b c d e f g h i j k l m n o p q r s t u v w x y z”
Result	Pass

Test	Test the back button of the Minuum Keyboard
Input	“abc” + back button
Expected Result	“ab”
Result	Pass

F.3 Mobile Application

Test	Test the spinner to select a user number
Input	Select each of the user numbers 1 - 16
Expected Result	The text of the spinner matches the selected user number at each iteration
Result	Pass

Test	Test the swipe to change phrase set functionality
Input	Swipe left then swipe right
Expected Result	The text of the phrase set number changes to “Phrase Set: 2” on swipe left & changes back to “Phrase Set: 1” on swipe right
Result	Pass

Test	Test the user condition floating action button (fab)
Input	1. Select “User_1” 2. Click on the user condition fab 3. Select a user condition mini-fab
Expected Result	• The user condition fab becomes available after step 1. Otherwise its hidden • All of the user condition mini-fabs become available after step 2 • The keyboard fab is hidden until after step 3, when it becomes available
Result	Pass

Test	Test the keyboard floating action button. Then the phrase set activity.
Input	1. Select "User_1" 2. Click on the user condition fab 3. Select a user condition mini-fab 4. Click on the keyboard fab 5. Select a keyboard mini-fab 6. Click on the next button fab in the phrase set activity 7. Create mock phrase data 8. Send mock data to listener
Expected Result	• The user condition fab becomes available after step 1. Otherwise its hidden • All of the user condition mini-fabs become available after step 2 • The keyboard fab is hidden until after step 3, when it becomes available • The keyboard opens on the Wear device and the phrase set activity is started after step 5 • The phrase tracker on the phrase set activity is set to "0/15" • The phrase text matches "Neil has been asking around" • The phrase text changes to "What a jerk" after step 8
Result	Pass

Appendix G

The Huawei Watch

Figure G.1: Image of the Huawei Watch[1]

.

[1]Image of the Huawei Watch - http://new-techonline.com/wp-content/uploads/2015/10/huawei_watch_01.png

G.1 The Huawei Watch with a scale

Figure G.2: Image of the Huawei Watch with a a ten pence coin added for scale.

G.2 Hardware Specification

Size	42mm diameter, 11.3mm thick
Display	1.4-inch full circle AMOLED display, 400x400 screen resolution, 286 ppi 10,000:1 high contrast ratio
Material	Cold-Forged 316L Stainless Steel Sapphire crystal
Battery	300mAh
Connectivity	Bluetooth 4.1 BLE, WiFi
Sensors	6-Axis motion sensor (Gyroscope + Accelerometer), Heart Rate Sensor (PPG), Barometer, Vibration Motor
OS Compatibility Requirements	Android 4.3+ /iOS 8.2+
Memory	512MB RAM + 4GB ROM

Table G.1: The Huawei Watch hardware specification[2]

[2]The Huawei Watch Hardware Specification - http://consumer.huawei.com/en/wearables/tech-specs/huawei-watch.htm

Appendix H

Nexus 5

H.1 Hardware Specification

Screen	• 4.95" 1920 x 1080 display (445 ppi) • Full HD IPS • Corning® Gorilla® Glass 3
Size	69.17 x 137.84 x 8.59 mm
Weight	4.59 ounces (130 g)
Cameras	• 1.3 MP front facing • 8 MP rear facing with Optical Image Stabilisation

Memory	• 16 GB or 32 GB (actual formatted capacity will be less) • 2 GB RAM
Processing	• CPU: Qualcomm Snapdragon™ 800, 2.26 GHz • GPU: Adreno 330, 450 MHz
Sensors	• GPS • Gyroscope • Accelerometer • Compass • Proximity/Ambient light • Pressure • Hall effect
Connectivity	• 2G/3G/4G LTE • Dual-band Wi-Fi (2.4G/5G) 802.11 a/b/g/n/ac • NFC (Android Beam) • Bluetooth 4.0 LE

Battery	2,300 mAh non-removable battery
OS	Android 4.4 (KitKat)

Table H.1: The Nexus 5 hardware specification[1]

[1]The Nexus 5 Hardware Specification - https://support.google.com/nexus/answer/6102470?hl=en-GB

Appendix I

Ethical Approval

Below is a copy of the ethics approval form submitted to the University of Strathclyde's department of Computer and Information Science Ethics Committee in order to gain ethics approval to conduct a study involving multiple participants.

Circular Watch Text: Ethics Approval

Title of research:

Circular Watch Text

Summary of research:

My project is to develop an Android Wear application with a circular keyboard. The keyboard will spread in a circle around the edge of a circular watch. I plan to build an advanced language model which will predict the key that was most likely intended to be pressed by the user based on the key pressed and its neighbours. I will be developing the application on my own smartwatch “The Huawei Watch”. The project will involve extensive Android development in Java and considerable user testing towards the end of the project. I will then analyse the performance of my keyboard through comparison against text entry on smartphones and alternative text entry on smartwatches. User studies will initially be for feedback on the keyboard and text entry, later they will be comparative with a standard watch text entry method.

How will participants be recruited?

For initial studies, I will personally ask 10 friends, colleagues, and family if they wish to participate in my text entry study. For final studies, 20 participants will be recruited. I will only be recruiting people aged 18 and over. It will be made clear to everyone this is voluntary and they will be given the option not to take part.

How will consent be demonstrated?

Participants will be given an outline of the study on paper and be asked background questions on their text entry usage. I will get the participants to fill in and sign a form of detailing the consent required.

What will the participants be told about the conduct of the research?

The participants will be informed that the information collected during the study will be presented in my final report, potential research publications and presentations but that no personal data will be collected or used in the process. They will also be told they can stop at any point if they wish.

What will participants be expected to do?

Participants will be presented with common phrases (displayed on my smartphone) and they will be expected to type these phrases on my smartwatch using my circular keyboard in an initial study. In later study a different set will be used for standard and circular. Text entry tasks will take a around 20-30 minutes. Phrases will be taken from the Vertanen and Kristensson standard Enron phrase set.

How will data be stored?

Data will be stored on my smartphone's internal memory, this data includes:
- X and Y coordinates of each tap

- Time taken
- Number of backspaces
- Number of uncorrected mistakes/errors
- Number auto-corrected errors
- Incorrect key pressed & corrected key
- Time of first character press of each phrase
- Time of last character press of each phrase

No personal data is needed nor will it be recorded.

How will data be processed? (e.g. analysed, reported, visualised, integrated with other data, etc.)

Data will be analysed for improvements to existing and future prototypes. Data will be presented in my report in various formats (i.e. tables, heat-maps, graphs, charts) and used for performance analysis.

How and when will data be disposed of?

Data will be kept for analysis by myself and my project supervisor (Dr Dunlop). If any publication follows it will be made open access.
Signed consent forms will be kept secure and then handed over to Dr Dunlop at end of project. He will destroy them by April 2017 .

Appendix J

User Study

J.1 User Pre-Participation & Consent Form

Below is the format of the consent form handed to users before they participated in the study. Users that participated in the first study did not have to complete the form again but it was a requirement for new participants in the final study.

CS408 User Participation Form

Circular Watch Text

Class
CS408 Individual Project

Course
MEng Computer Science

Project Supervisor
Mark Dunlop

Student
Cameron Drennan

Reg No.
201126086

I confirm that I understand my participation in this study is voluntary and that I have the option to not take part and that I can withdraw my participation at any time during the study. Also I understand that the information collected during the study will be presented in the final report of the project, potential research publications and presentations but that no personal data will be collected or used in the process.

Signature: ..... Date:.....

Project

Circular Watch Text

Project Study Outline

The purpose of this study is to gather usage data on circular keyboard prototypes developed for circular Android Wear devices. You will be asked to complete this brief form then type some phrases on our smartwatch.

User Guide

- Type as accurately and as quickly as possible for each phrase then you can take a short break between phrases if you wish.
- If you spot a mistake just after making it correct it by backspacing, otherwise just leave it.
- Do not worry about capitalisation and punctuation.
- To backspace: Swipe from right -> left. A small vibration will occur when a backspace is successful and the caret will also move back once.
- To insert space: Swipe from left -> right. A small vibration will occur when a space is successful and the caret will also move forwards once.

Form

- Gender: .....
- Age: .....
- Main Type of Text entry on mobile phones (Select one):
  - ○ On screen QWERTY keyboard
  - ○ Small physical keyboard (Blackberry style)
  - ○ Other:.....
- How often do you type messages, emails or social posts on your phone (Select one):
  - ○ Several times a day
  - ○ Few times a week
  - ○ One a week or fewer
  - ○ Never
- What is your experience of using a Smart Watches (Select one):
  - ○ Regularly use one
  - ○ Some
  - ○ None

J.2 User Pre-Participation & Consent Form Data

J.2.1 Initial User Study

User	Gender	Age	Main Type of Text entry on mobile phones	How often do you type messages, emails or social posts on your phone	What is your experience of using Smart Watches	Phrase Set Sequence
1	Male	21	On screen QWERTY keyboard	Several times a day	Regularly use one	set1->set2
2	Male	21	On screen QWERTY keyboard	Several times a day	Regularly use one	set2->set1
3	Male	20	On screen QWERTY keyboard	Several times a day	None	set1->set2
4	Female	25	On screen QWERTY keyboard	Several times a day	None	set2->set1
5	Female	20	On screen QWERTY keyboard	Several times a day	None	set1->set2
6	Male	21	On screen QWERTY keyboard	Several times a day	None	set2->set1
7	Male	21	On screen QWERTY keyboard	Several times a day	None	set1->set2
8	Male	21	On screen QWERTY keyboard	Several times a day	None	set2->set1
9	Male	21	On screen QWERTY keyboard	Several times a day	None	set1->set2
10	Male	21	On screen QWERTY keyboard	Several times a day	None	set2->set1

Figure J.1: The background information of the participants involved in the initial user study.

J.2.2 Final User Study

User	Gender	Age	Main Type of Text entry on mobile phones	How often do you type messages, emails or social posts on your phone	What is your experience of using Smart Watches	Phrase Set Sequence
1	Male	21	On screen QWERTY keyboard	Several times a day	Regularly use one	M2->H1
2	Male	21	On screen QWERTY keyboard	Several times a day	Regularly use one	H2->M1
3	Male	20	On screen QWERTY keyboard	Several times a day	None	H1->M2
4	Female	25	On screen QWERTY keyboard	Several times a day	None	H1->M2
5	Female	20	On screen QWERTY keyboard	Several times a day	None	H2->M1
6	Male	21	On screen QWERTY keyboard	Several times a day	None	M1->H2
~~7~~	~~Male~~	~~21~~	~~On screen QWERTY keyboard~~	~~Several times a day~~	~~None~~	
7	Male	21	On screen QWERTY keyboard	Several times a day	Regularly use one	M2->H1
8	Male	21	On screen QWERTY keyboard	Several times a day	None	H1->M2
9	Male	21	On screen QWERTY keyboard	Several times a day	None	M2->H1
10	Male	21	On screen QWERTY keyboard	Several times a day	None	M1->H2
11	Male	21	On screen QWERTY keyboard	Several times a day	None	H1->M2
12	Male	21	On screen QWERTY keyboard	Several times a day	None	M2->H1
13	Male	22	On screen QWERTY keyboard	Several times a day	None	H2->M1
14	Male	24	On screen QWERTY keyboard	One a week or fewer	None	M1->H2
15	Male	21	On screen QWERTY keyboard	Several times a day	None	H2->M1
16	Male	21	On screen QWERTY keyboard	Several times a day	None	M1->H2

Figure J.2: The background information of the participants involved in the final user study. User 7 (highlighted in red) withdrew his participation from the final study so they were replaced by another participant (highlighted in green).

J.3 NASA Task Load Index (TLX)

Below is the NASA TLX[1] form that was handed to users to complete immediately after they completed each phrase set during the final study.

[1]NASA TLX - http://humansystems.arc.nasa.gov/groups/tlx/downloads/TLXScale.pdf

NASA Task Load Index

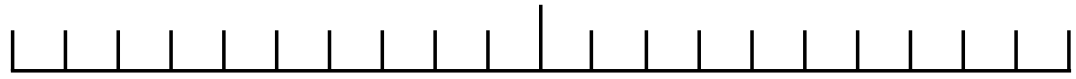
Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

Name	Task	Date

Mental Demand — How mentally demanding was the task?

Very Low — Very High

Physical Demand — How physically demanding was the task?



Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?

Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?

Very Low — Very High

J.4 NASA Task Load Index (TLX) Results

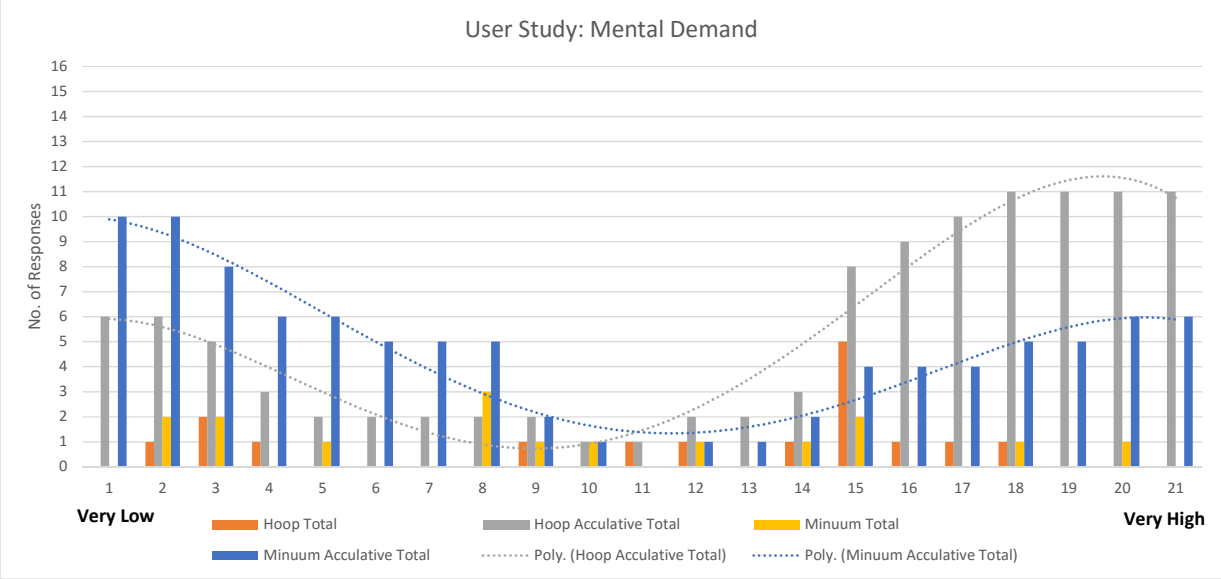


Figure J.3: The results for the users "Mental Demand" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

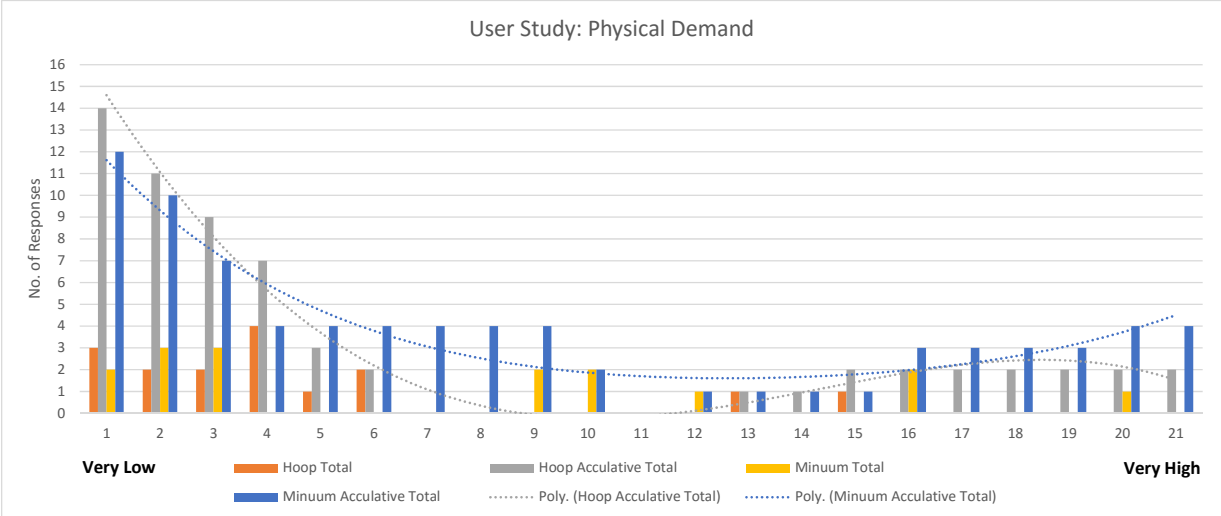


Figure J.4: The results for the users "Physical Demand" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

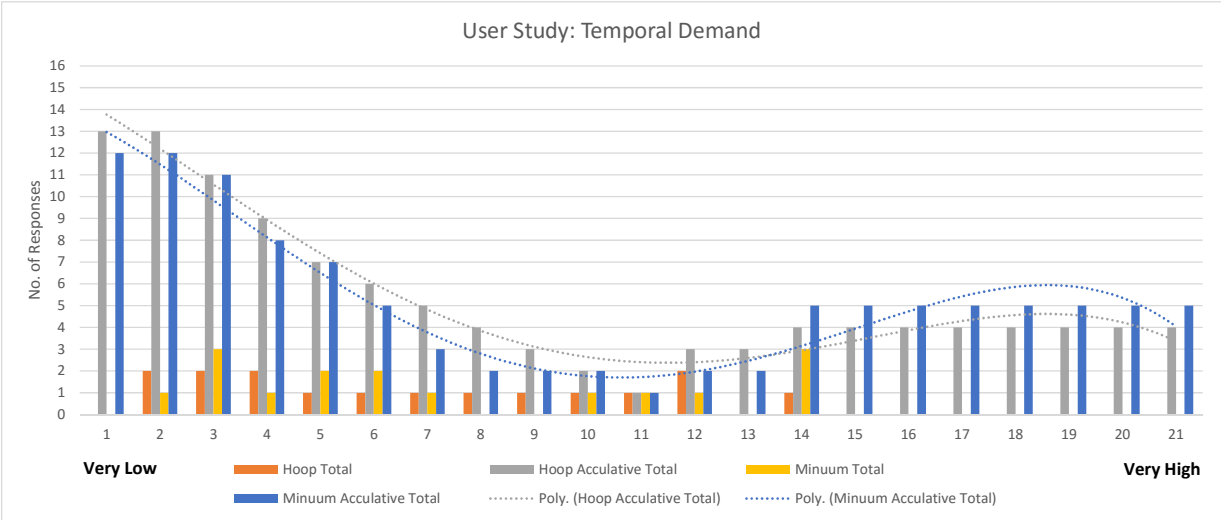


Figure J.5: The results for the users "Temporal Demand" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

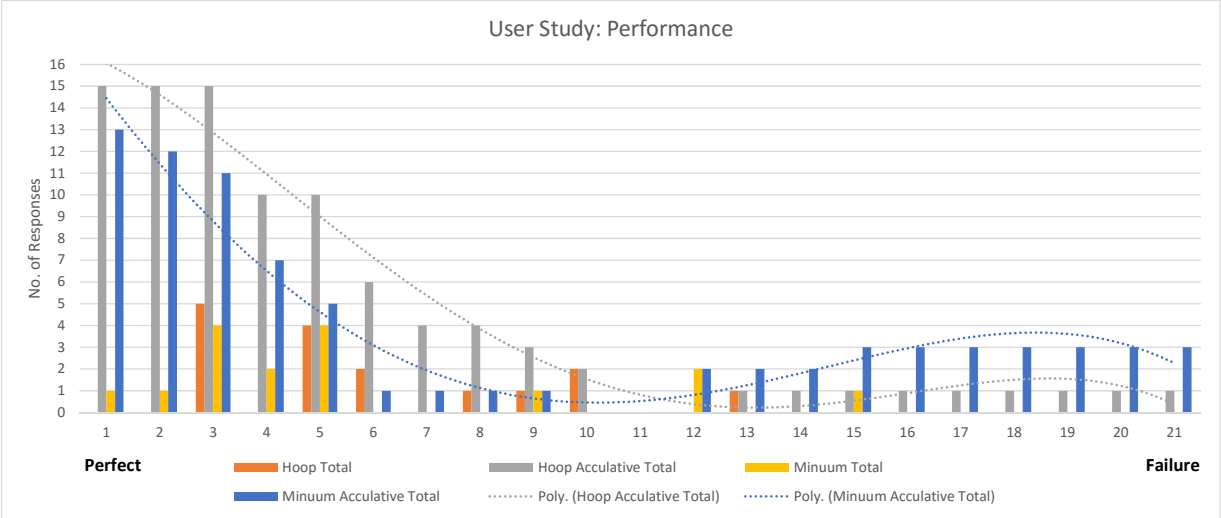


Figure J.6: The results for users "Performance" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

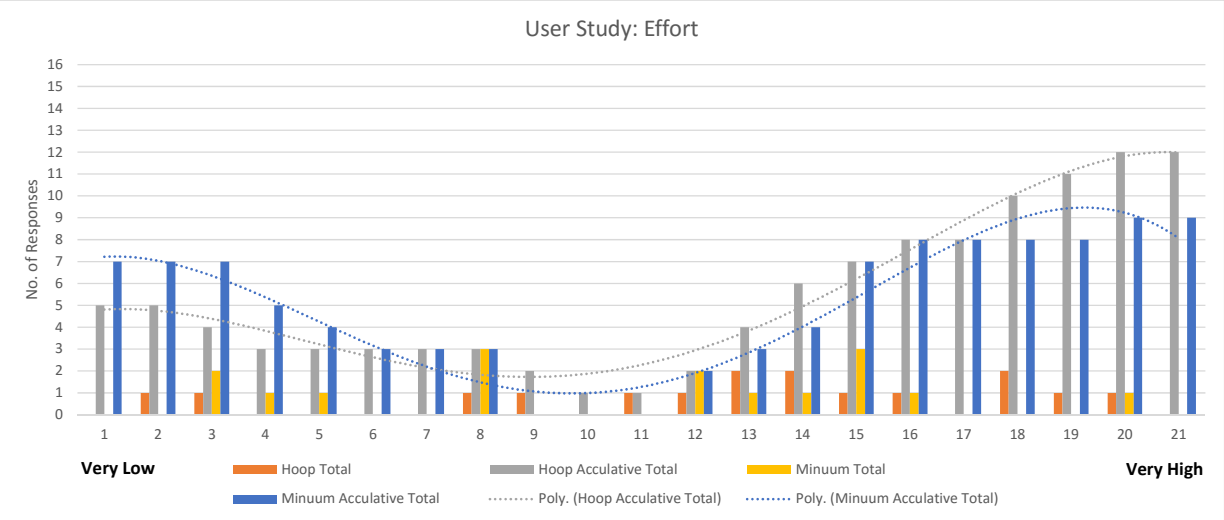


Figure J.7: The results for users "Effort" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

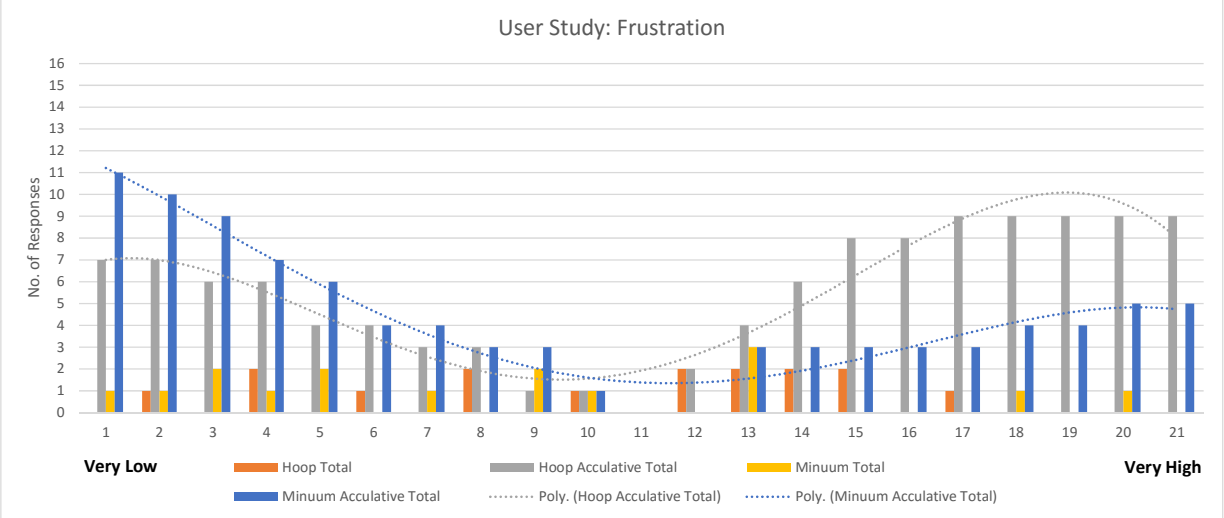


Figure J.8: The results for users "Frustration" shown as a comparison of the Hoop keyboard and the Minuum keyboard.

J.5 Exit Questionnaire

Below is the format of the exit questionnaire handed to participants at the end of the final study. The participants were sent a link to the online version of the form made using Google Forms[2].

[2]Google Forms - https://www.google.co.uk/forms/about/

Circular Watch Text: Questionnaire

This exit questionnaire is filled out by participants at the end of the user study.

* Required

1. **What was your given user number?** *
   Mark only one oval.

   - ◯ User_1
   - ◯ User_2
   - ◯ User_3
   - ◯ User_4
   - ◯ User_5
   - ◯ User_6
   - ◯ User_7
   - ◯ User_8
   - ◯ User_9
   - ◯ User_10
   - ◯ User_11
   - ◯ User_12
   - ◯ User_13
   - ◯ User_14
   - ◯ User_15
   - ◯ User_16

2. **Which type of layout did you prefer?** *
   Mark only one oval.

	1	2	3	4	5	6	7	
Straight layout	◯	◯	◯	◯	◯	◯	◯	Circular layout

3. **Which method of space did you prefer?** *
   Mark only one oval.

	1	2	3	4	5	6	7	
Button press	◯	◯	◯	◯	◯	◯	◯	Swipe

4. **Which method of backspace did you prefer?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Button press	◯	◯	◯	◯	◯	◯	◯	Swipe

5. **The Hoop keyboard does word auto-correction whereas the Minuum keyboard gets you to pick words. Which do you prefer?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Minuum keyboard (QWERTY)	◯	◯	◯	◯	◯	◯	◯	Hoop keyboard (Circular)

6. **Overall, which keyboard do you prefer?** *
Mark only one oval.

	1	2	3	4	5	6	7	
Minuum keyboard (QWERTY)	◯	◯	◯	◯	◯	◯	◯	Hoop keyboard (Circular)

7. **Additional Comments**

.....

.....

.....

.....

.....

J.6 Phrase Sets

The phrases used in both user studies are listed below. The phrases were randomly picked from Keith Vertanen's memorable test sets[3].

J.6.1 Initial User Study

Phrase Set 1

Phrase 1	Hope your trip to Florida was good
Phrase 2	No material impact
Phrase 3	OK with me
Phrase 4	I would like to attend if so
Phrase 5	I think those are the right dates
Phrase 6	Please revise accordingly
Phrase 7	No there will be plenty of others
Phrase 8	I agreed terms with Greg
Phrase 9	Neil has been asking around
Phrase 10	I hope you are feeling better

Table J.1: The 10 phrases included in Phrase Set 1 for the initial user study.

Phrase Set 2

Phrase 1	Their key decision maker did not show which is not a good sign
Phrase 2	We must be consistent
Phrase 3	And leave my school alone
Phrase 4	I would like to attend if so
Phrase 5	I will call
Phrase 6	I worked on the grade level promotion
Phrase 7	I have a request
Phrase 8	I was planning to attend
Phrase 9	What a pain
Phrase 10	I have a high level in my office

Table J.2: The 10 phrases included in Phrase Set 2 for the initial user study.

[3]Memorable test sets - https://keithv.com/software/enronmobile/mem.zip

J.6.2 Final User Study

Phrase Set 1

Practice Phrase	Neil has been asking around
Phrase 1	What a jerk
Phrase 2	No surprise there
Phrase 3	I changed that in one prior draft
Phrase 4	Hope your trip to Florida was good
Phrase 5	Please send me an email
Phrase 6	I was planning to attend
Phrase 7	Not even close
Phrase 8	Ken agreed yesterday
Phrase 9	I wanted to go drinking with you
Phrase 10	See you on the third
Phrase 11	Need to watch closely
Phrase 12	Please call tomorrow if possible
Phrase 13	I am walking in now
Phrase 14	That would likely be an expensive option
Phrase 15	You have a nice holiday too

Table J.3: The 15 phrases + practice phrase included in Phrase Set 1 for the final user study.

Phrase Set 2

Practice Phrase	Call me to give me a heads up
Phrase 1	Thank you for your prompt reply
Phrase 2	Good for you
Phrase 3	I hope you are feeling better
Phrase 4	This looks fine
Phrase 5	Thanks for your concern
Phrase 6	I would be glad to participate
Phrase 7	I have a request
Phrase 8	Both of us are still here
Phrase 9	I worked on the grade level promotion
Phrase 10	And leave my school alone
Phrase 11	It reads like she is in
Phrase 12	We need to talk about this month
Phrase 13	Jan has a lot of detail
Phrase 14	She has absolutely everything
Phrase 15	I am on my way

Table J.4: The 15 phrases + practice phrase included in Phrase Set 2 for the final user study.

Appendix K

First Character Key Press Weighting Analysis

For the first character key press weighting analysis data, please refer to the fourth_year_project.zip file under the directory "Other Analysis/First Character Weighting Analysis".

Appendix L

Screen Shots

L.1 Google Hangouts Application

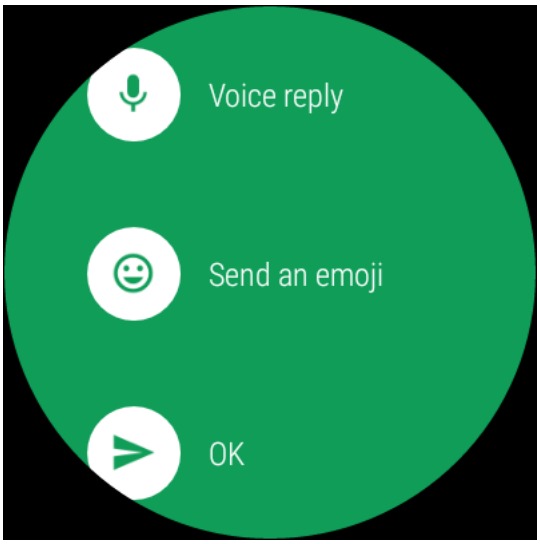


Figure L.1: Screen shot of the reply options in Google's Hangouts Android Wear application.

L.2 Initial Design Mock Up

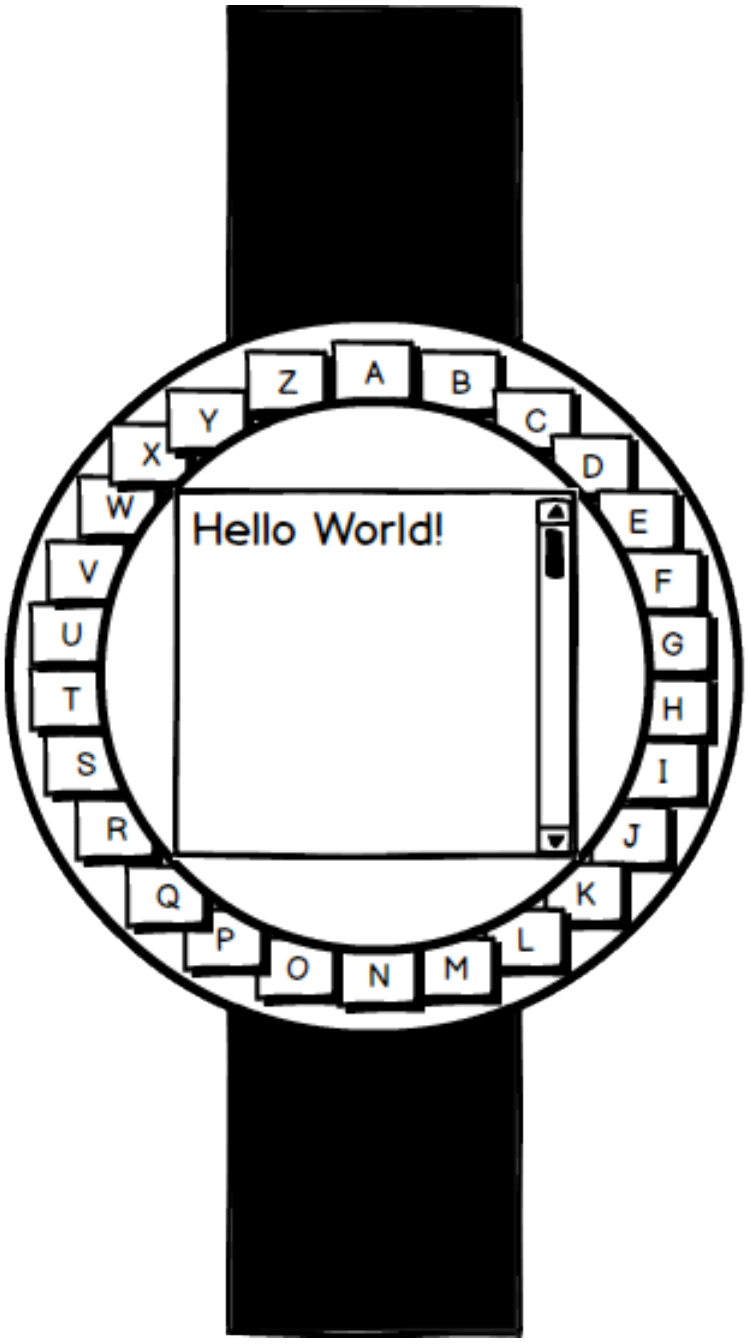


Figure L.2: The initial mock up design of the Hoop keyboard.

L.3 Prototype 1

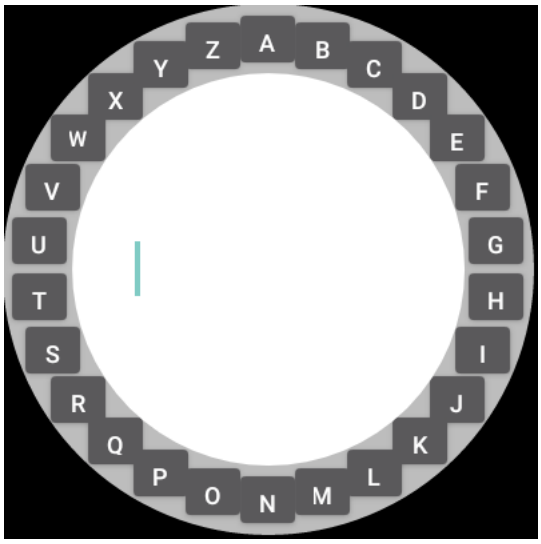


Figure L.3: Screen shot of the Hoop Prototype 1 keyboard.

L.4 Prototype 2

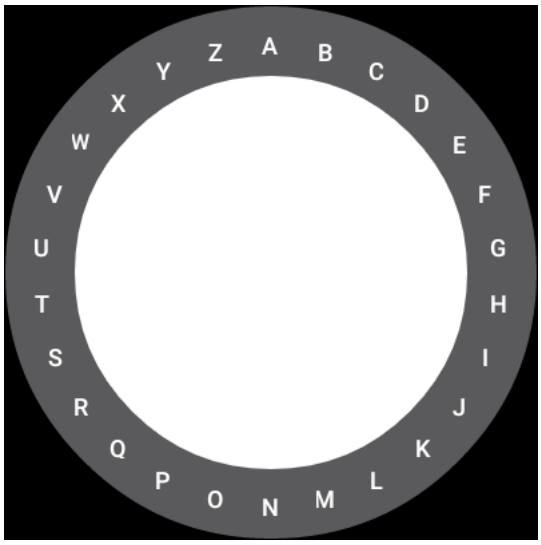


Figure L.4: Screen shot of the Hoop Prototype 2 keyboard.

L.5 Hoop Keyboard

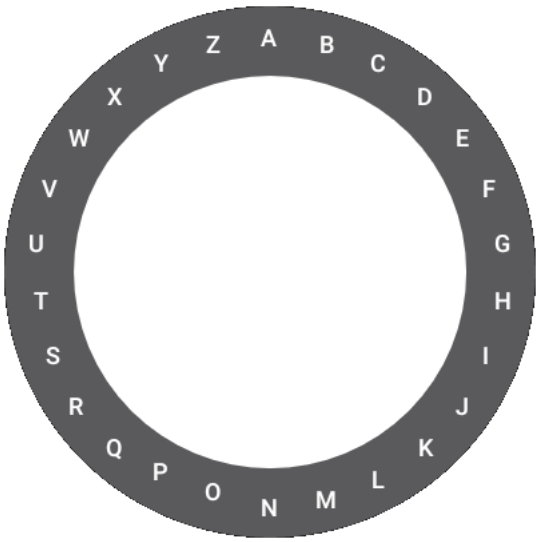


Figure L.5: Screen shot of the Hoop Keyboard.

L.5.1 Short Text

Figure L.6: Screen shot of the Hoop Keyboard with short text.

L.5.2 Long Text

Figure L.7: Screen shot of the Hoop Keyboard with long text.

L.6 Minuum Keyboard

Figure L.8: Screen shot of the Minuum Keyboard.

L.6.1 Short Text

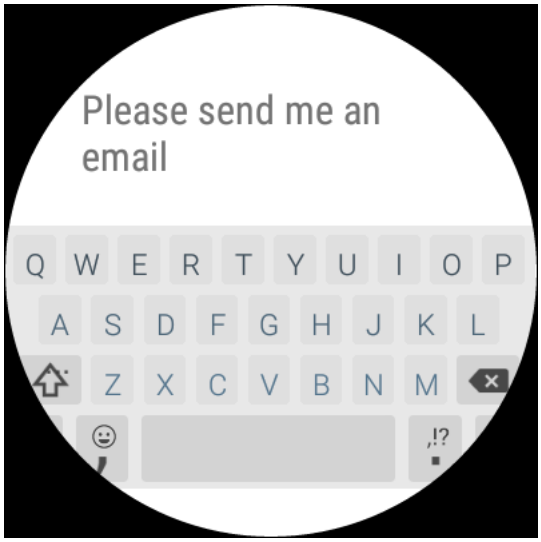


Figure L.9: Screen shot of the Minuum Keyboard with short text.

L.6.2 Long Text

Figure L.10: Screen shot of the Minuum Keyboard with long text.

L.7 Mobile Application

Figure L.11: Screen shot of the accompanying mobile application for the Hoop Keyboard.

Appendix M

Storyboards

M.1 Initial Mock Up Storyboard

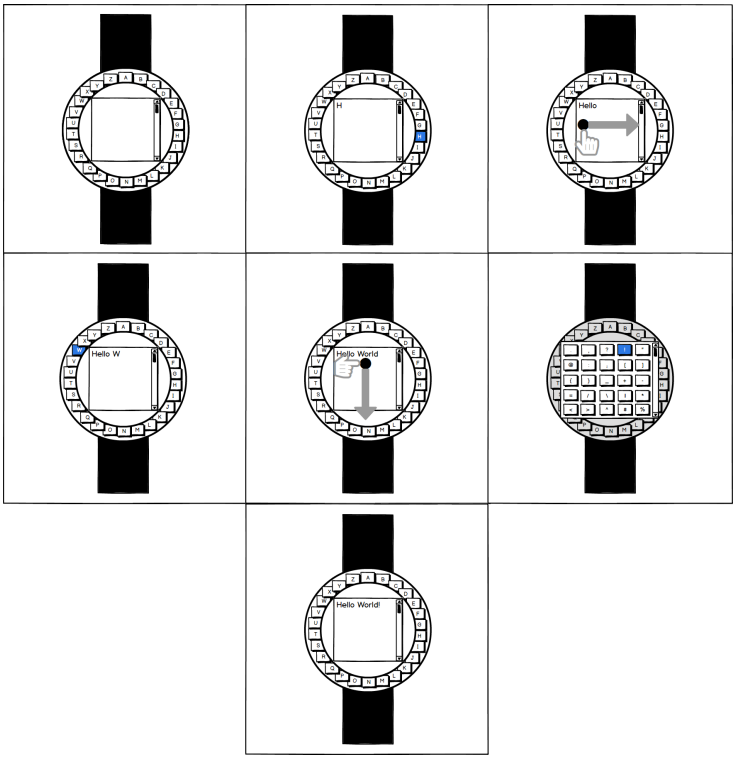
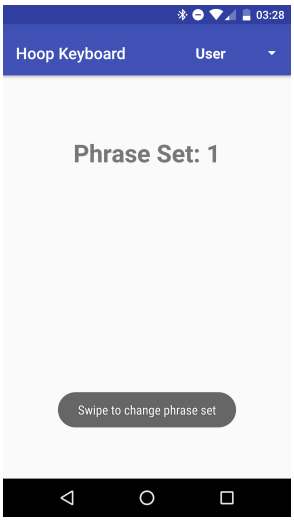
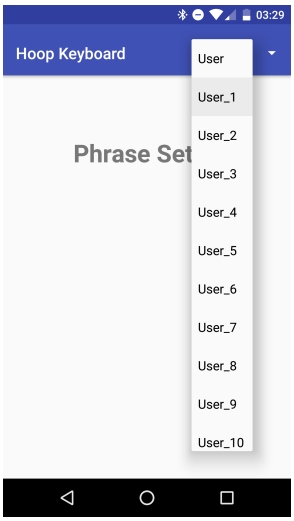


Figure M.1: The initial mock up storyboard of a simple task on the Hoop keyboard built in Balsamiq[1].

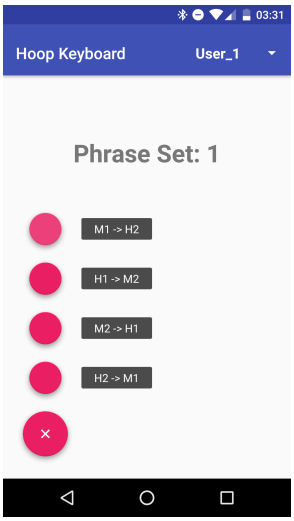
M.2 User Study Storyboard



(a) Step 1

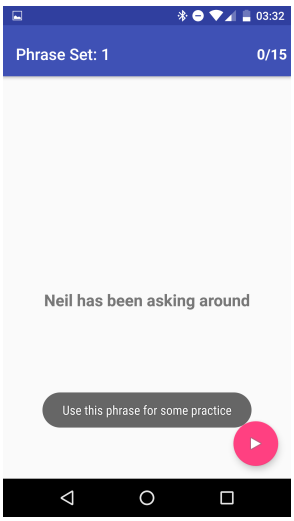


(b) Step 2

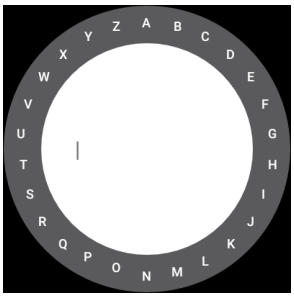


(c) Step 3

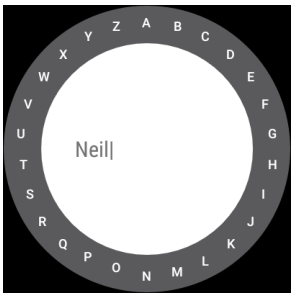
(d) Step 4



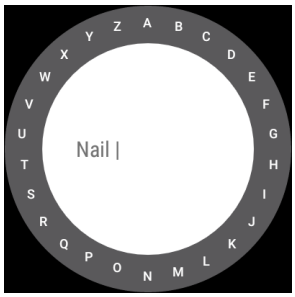
(e) Step 5



(f) Step 6

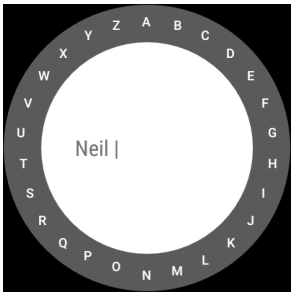


(g) Step 7



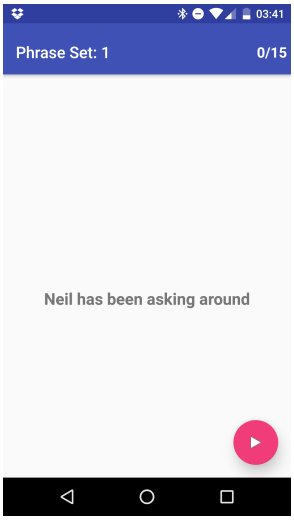
(h) Step 8

[1]Balsamiq Website - https://balsamiq.com/

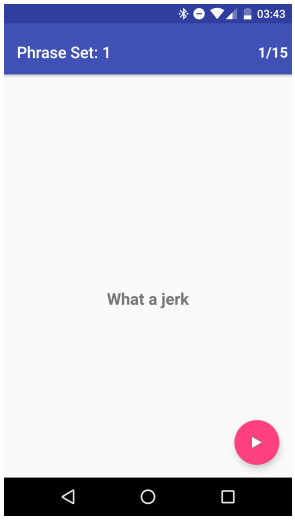


(i) Step 9

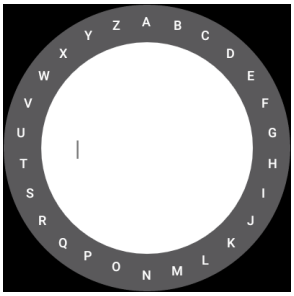
(j) Step 10



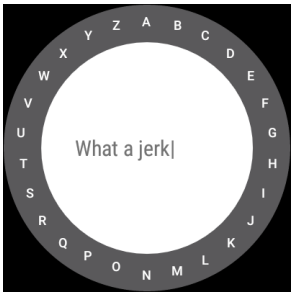
(k) Step 11



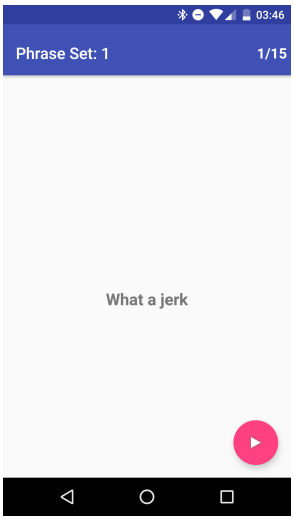
(l) Step 12



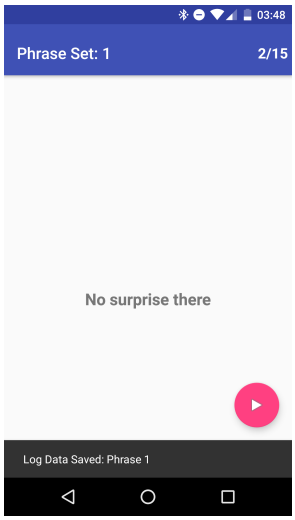
(m) Step 13



(n) Step 14



(o) Step 15

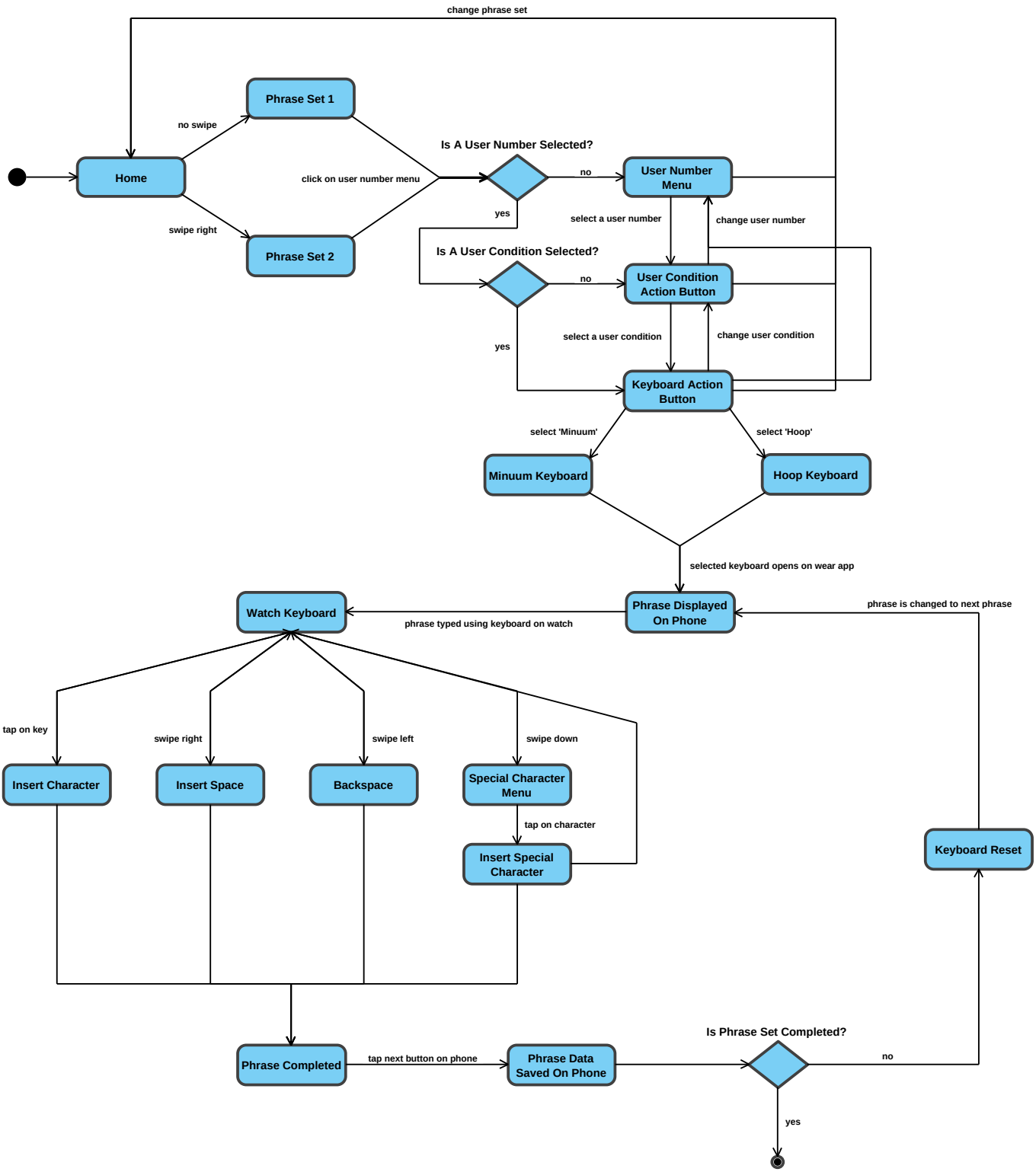


(p) Step 16

Appendix N

State Transition Network (STN)

Below is the state transition network for the Hoop Keyboard.


change phrase set
Phrase Set 1
no swipe
Home
click on user number menu
swipe right
Phrase Set 2
Is A User Number Selected?
no
User Number Menu
yes
select a user number
change user number
Is A User Condition Selected?
no
User Condition Action Button
yes
select a user condition
change user condition
Keyboard Action Button
select 'Minuum'
select 'Hoop'
Minuum Keyboard
Hoop Keyboard
selected keyboard opens on wear app
Watch Keyboard
phrase typed using keyboard on watch
Phrase Displayed On Phone
phrase is changed to next phrase
tap on key
swipe right
swipe left
swipe down
Insert Character
Insert Space
Backspace
Special Character Menu
tap on character
Insert Special Character
Keyboard Reset
Phrase Completed
tap next button on phone
Phrase Data Saved On Phone
Is Phrase Set Completed?
no
yes

Appendix O

Analysis Software

For the Analysis Software built for the project, please refer to the fourth_year_project.zip file under the directory "Java Projects/User Data Analyser" or the following url:

https://github.com/camerondrennan/Circular-Watch-Text/tree/master/Analysis\%20Software

Appendix P

Trello Boards

For the Trello boards used for the project, please refer to the following url:

`https://trello.com/circularwatchtext`

Appendix Q

Log Book

For the log book kept for the project, please refer to the following url:

`http://circularwatchtext.camerondrennan.com/`

Appendix R

GitHub Repository

For the GitHub repository for the project, please refer to the following url:

`https://github.com/camerondrennan/Circular-Watch-Text`

Appendix S

User Guide

Below is the user guide for the Hoop Keyboard.

CS408 User Guide

Circular Watch Text

Class
CS408 Individual Project

Course
MEng Computer Science

Project Supervisor
Mark Dunlop

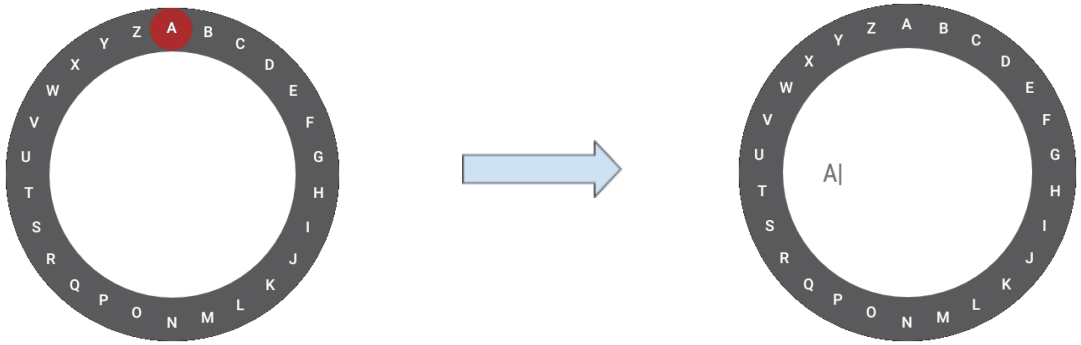
Student
Cameron Drennan

Reg No.
201126086

Typing On the Hoop Keyboard

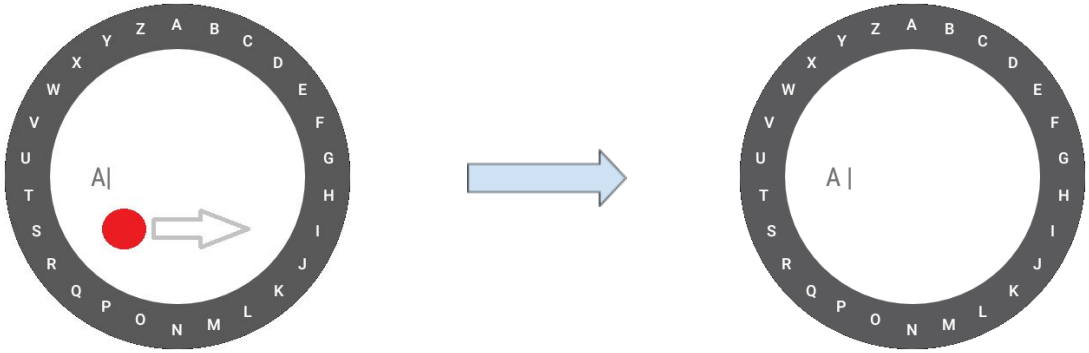
Inserting a character

To type on the Hoop Keyboard simply tap on a character and the character will be inserted into the inner text box as shown in the image below:



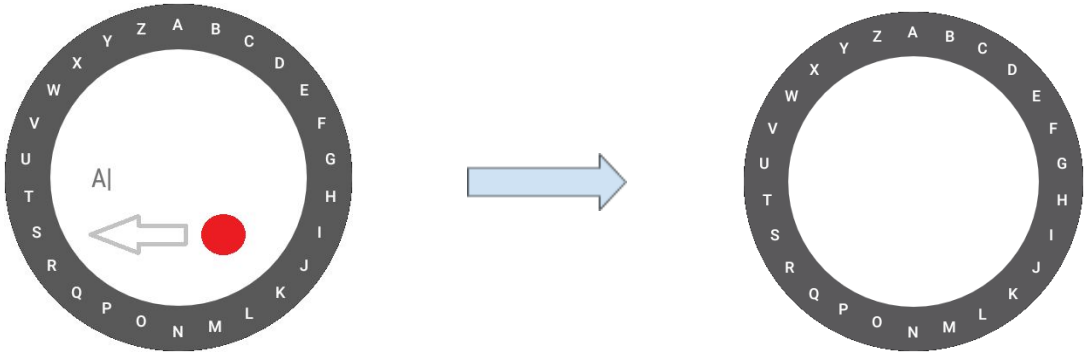
Inserting a space

To insert a space on the Hoop Keyboard simply swipe to the right as shown in the image below:



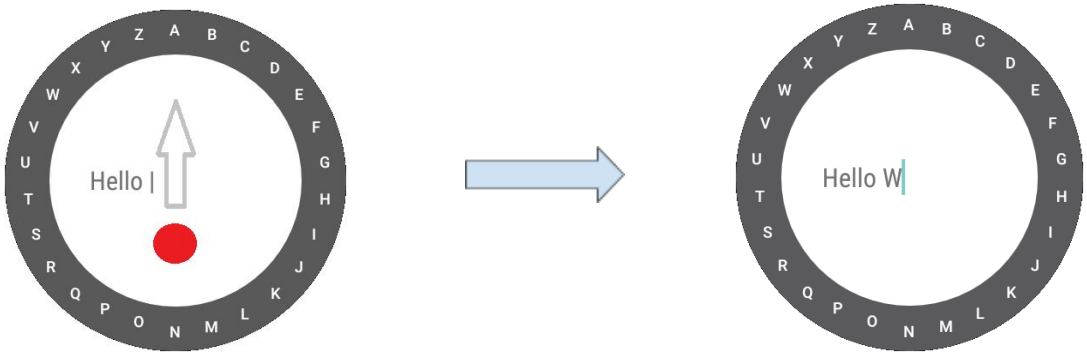
Deleting a character

To delete a character on the Hoop Keyboard simply swipe to the left as shown in the image below:



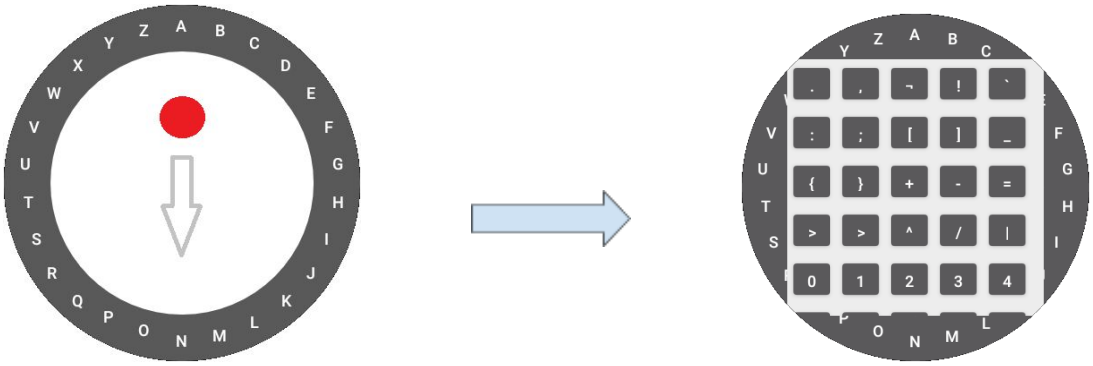
Changing the case of text

To change the case of the text simply swipe upwards as shown in the image below (Note: feature is disabled in testing mode):



Open the special character menu

To access the special character menu simply swipe downwards as shown in the image below (Note: feature is disabled in testing mode):



Appendix T

Documentation

For the documentation for the Hoop Keyboard, please refer to the fourth_year_project.zip file under the directory "/JavaDoc" or the following url:

`http://hoopkeyboard.camerondrennan.com/docs/`

Appendix U

Hoop Keyboard APK

For the apk for the Hoop Keyboard, please refer to the fourth_year_project.zip file under the directory "Android APK/hoopkeyboard.apk" or the following url:

`http://hoopkeyboard.camerondrennan.com/apk/hoopkeyboard.zip`