MULTIMODAL INTERACTION TECHNIQUES FOR DISABLED DEVELOPERS

By

Bharat Paudyal

This dissertation is submitted in partial fulfilment

of the requirement for the degree of Doctor of Philosophy

March 2023

Digital Media Technology Lab

School of Computing and Digital Technology

Birmingham City University, UK

Dedications

To all the developers who put in so much effort to make the lives of others easier and more convenient but have developed physical impairments in the process.

DECLARATION

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

Bharat Paudyal

March 2023

ABSTRACT

Technologies such as speech recognition, eye tracking, and mechanical switches present alternative opportunities to make coding more accessible for people with physical impairments. These technologies have previously been explored independently to assess their potential for supporting development work, although each input method exhibits unique strengths and challenges. A multimodal approach utilising different combinations of these technologies holds significant potential to address the individual limitations of each technology. However, there has been a lack of research to date investigating how these input approaches can be combined and the extent to which they can support inclusive coding experiences for people with physical impairments.

To address the limited work in this area, three independent research studies were conducted investigating multimodal input approaches for disabled developers. The first study focused on developing a research prototype utilising a combination of speech, gaze and mechanical switches to support writing and editing syntax. A user evaluation with 29 non-disabled developers found that the system was perceived positively in terms of usability and facilitated the successful completion of common coding activities. A follow-up study with five developers who have physical impairments validated that this target audience could successfully utilise the multimodal approach to complete standard coding tasks.

The second study investigated the usability and feasibility of different multimodal voice coding approaches (i.e. natural language and fixed commands) in conjunction with a mechanical switch to support coding activities. A comparative study with 25 non-disabled developers found that both approaches demonstrated similar levels of efficacy and usability, although participants highlighted significant potential in terms of natural language coding. This approach was therefore developed further and evaluated within a multi-session study with five developers who have physical impairments. Results validated the feasibility of the multimodal natural language approach to support developers in successfully completing coding activities.

The final study investigated the efficacy of tailoring code navigation features commonly used within mainstream development environments (i.e. “Find by Reference”, “Go to Definition”, and “Find”) for multimodal voice and mechanical switch interaction. A user evaluation with 14 developers who have physical impairments highlighted that the code navigation approaches were efficient to utilise and demonstrated a high-level of usability.

The contributions presented in this thesis highlight how the combination of different alternative input methods can provide more inclusive coding experiences for developers with physical impairments.

ACKNOWLEDGEMENTS

I want to offer my profound gratitude to all the people who helped make completing this PhD possible, especially my supervisors, family, and friends. First, I would like to thank my supervisors: Prof. Chris Creed, Prof. Ian Williams and Dr Maite Frutos for their guidance, advice, extensive academic support and consistent encouragement throughout my research work. Chris, Ian and Maite, I am indebted for your meaningful feedback, words of inspiration and cooperation throughout my research. Special thanks to Chris for all the support he provided on both personal and professional levels during the research.

I would also like to thank Dr Sayan Sarcar and Dr Arthur Theil for their precious time and ideas during the last part of my research. I would further like to thank my peers and members of the Digital Media Technology (DMT) Lab I met during my PhD who have extended their support to make this research possible - Dr Mauadh Al-Kalbani and Alan Dollhasz. I would also like to thank my friends, Farkandah Aziz, Hsein Kew and Andreea Blaga for their support. I would also like to extend my most profound appreciation to all my research colleagues, who helped me in the initial evaluations of research prototypes on multiple occasions. Additionally, I wish to acknowledge the London RSI center and Talon community for supporting the recruitment of developers with physical impairments.

Nobody has been more important to me in the pursuit of this project than the members of my family. I would like to thank my uncle Dr Jagannath, aunt Laxmi, and cousin Dr Basu. They have encouraged and helped me at every stage of my personal and academic life and have wished to see this achievement come true. I would like to thank my family in Nepal and the UK for their support, well wishes, words of inspiration and appreciation throughout my research.

Most importantly, I wish to thank my father Chiranjibi, mother Bishnu, wife Binita, Sister Dr Pragya, brother-in-law Bishesh, daughter Adhbhika, nephew Aditya, father-in-law Chhabi Gyawali and mother-in-law Sushila Gyawali for their unfailing love, support and continuous encouragement, not only during the years that it has taken to produce this thesis, but also for their constant cooperation and motivation of my academic pursuits over the years.

CONTENTS

List of Tables

LIST OF FIGURES

1 INTRODUCTION

Software development encompasses multiple activities including designing, writing, understanding, testing, editing, and debugging code (Snell, 2000; LaToza et al., 2006; Minelli et al., 2015; Rosenblatt et al., 2018). Developers typically use Integrated Development Environments (IDEs) as a tool to support the writing and management of code (e.g. Visual Studio Code (2023), Brackets (2023) and Atom (2023)). These applications provide a wide range of standard features such as syntax highlighting, code navigation, autocompletion, refactoring, and versioning to facilitate developer workflows and manage large codebases (Albusays et al., 2017; Chi and Naik, 2019). These IDEs require the use of traditional input devices (i.e. a mouse and keyboard) to control applications and support interactions tailored for these tools. However, this can cause significant accessibility issues for people with physical disabilities who may have difficulty controlling these devices (Hubbell et al., 2006; Delimarschi et al., 2014). This can result in disabled people being excluded from development work and the opportunity to have technical careers in this area (Wagner and Gray, 2015; Rosenblatt et al., 2018; Marshall, 2022).

To address broader accessibility issues associated with controlling digital applications, developers typically require the use of different assistive technologies to control the interface (Creed, 2018; Wong, 2020). In particular, people with physical impairments commonly use tools such as mechanical switches (Anson, 1994; Lancioni et al., 2008; Di Ferrante and Bouchard, 2020; Elsahar et al., 2021), head tracking (Haque et al., 2021), voice control (Desilets, 2001; Wagner and Gray, 2015; Joshi and Bein, 2020; Aziz et al., 2022), and eye gaze interaction (Bednarik and Tukiainen, 2006; Bergstrom and Schall, 2014; Sharafi et al., 2020; Casarini et al., 2020). These tools can make interactive experiences accessible to varying levels, although each approach has different strengths and limitations

(Hu et al., 2011; Majaranta and Bulling, 2014; Borgestig et al., 2017). For instance, voice control can present a number of advantages over traditional input methods such as a more natural input paradigm (Gaikwad et al., 2010; Malik et al., 2021), a “hands-free” solution (Dai et al., 2003), and more efficient interactions across a range of scenarios (Martin, 1989; Hauptmann and Rudnicky, 1990; Ruan et al., 2016; Kim et al., 2019; Sims, 2022). However, it also presents a range of known challenges such as recognition accuracy (Wagner and Gray, 2015; Rosenblatt et al., 2018), performance issues in noisy environments (Suhm et al., 2001; Krishna et al., 2019), and poor mapping to standard interface designs optimised for mouse and keyboard input (e.g. in terms of being impractical for pointing actions (Rozado et al., 2016)).

Similarly, gaze input can present a number of accessibility benefits such as supporting complete application control via eye movements alone (Casarini et al., 2020; Lewien, 2021), as well as facilitating efficient target acquisition and selection of interface targets (Porta and Ravelli, 2009; Mott et al., 2017; Kumar et al., 2020). However, limitations also include challenges associated with selecting small targets (Miniotas et al., 2006; Biswas and Langdon, 2011) and the well-known Midas touch issue where each fixation on an interface target can activate selection of that object (leading to unintentional selections) (Jacob and Karn, 2003; Rajanna and Hammond, 2018). Furthermore, mechanical switches (e.g. large portable buttons) can make interfaces more accessible for people with severe motor impairments, although this can also be a tedious and time-consuming method for controlling digital systems (Kennedy et al., 2000; Krausz et al., 2003; Elsahar et al., 2019, 2021). In terms of IDEs, each of these assistive tools presents opportunities to make coding more accessible for people with physical impairments (Soto Munoz et al., 2019; Joshi and Bein, 2020). Whilst some initial work has explored the feasibility of these input devices in this domain (Begel and Graham, 2005; Wagner and Gray, 2015; Rosenblatt et al., 2018), we still lack a deeper understanding around the potential of these methods to support disabled developers.

In particular, a multimodal approach combining different types of tools could present interaction benefits through utilising the strengths of each input method (Beelders and Blignaut, 2010; Van Der Kamp and Sundstedt, 2011; Creed et al., 2020; Sengupta et al., 2020)– for instance, the use of speech input can help in addressing issues associated with the selection of small targets using gaze, while typing code via gaze can help in reducing speech misrecognition issues. Additionally, this approach provides developers with the ability to seamlessly transition between different modes according to their personal preferences (Rozado et al., 2016; Sengupta et al., 2018). However, there has again been a lack of work exploring the potential of this approach to support the writing, editing, and management of code. This is a particularly pertinent area given the rise of more hybrid and remote working practices resulting from the Covid-19 pandemic (Gratton, 2021; Ateeq, 2022). This therefore represents a crucial and timely area requiring further work to understand better the potential of assistive technologies and alternative input methods to make coding more accessible for people with physical impairments.

1.1 Research Questions

To investigate the potential of alternative methods to support disabled coders, this thesis initially explores the use of a multimodal approach (i.e. a combination of speech, gaze, and mechanical switches) to support the writing and editing of code. The rationale for initially focusing on these specific input devices is elaborated on further in Chapter 2, although crucial reasons relate to the increased availability and potential of these methods to support people with physical impairments. Results from this work identified some interaction challenges associated with eye gaze interaction in this context, so the research focus subsequently shifted to a multimodal voice and switch interaction technique. In particular, an emphasis is placed on utilising a multimodal voice and switch approach to compare natural language coding methods against using a fixed grammar comprised of predefined vocal commands. The motivation for focusing on these two approaches was driven by a lack of comparative work exploring the strengths and limitations of both approaches in the literature. Results from this work identified specific interaction challenges and opportunities associated with code navigation. This area forms the thesis's final focus, where a multimodal voice and switch approach is utilised to explore the feasibility of different code navigation features. The following research questions guided the focus of the work conducted:

RQ1: How can multimodal approaches (utilising speech, gaze, and mechanical switches) support people with physical impairments in writing, editing, navigating and selecting code?

Speech recognition and eye gaze interaction have improved significantly in recent years with both approaches being used independently to support disabled users with software development activities (Begel and Graham, 2005; Radevski et al., 2016; Rosenblatt et al., 2018; Shakil et al., 2019). Whilst each of these technologies can present accessibility benefits, both interaction approaches also present limitations (e.g. speech recognition issues (Begel and Graham, 2005; Wagner and Gray, 2015; Kim et al., 2019)) and challenges with the selection of small targets via gaze (Bates and Istance, 2002; Miniotas et al., 2006; Skovsgaard et al., 2010; Creed et al., 2020). Initial work has explored the combination of gaze and speech in different domains (Castellina et al., 2008; Van Der Kamp and Sundstedt, 2011; Rozado et al., 2016; Sengupta et al., 2018), which has confirmed that the limitations of each method can be minimised through the complementary strengths of the other input modality (Beelders and Blignaut, 2010; Elepfandt and Grund, 2012; Sengupta et al., 2020). However, it remains unclear whether the combination of these modalities can potentially present a viable approach to support development work for disabled developers. Furthermore, other assistive devices such as mechanical switches are also widely used by people with physical impairments (Lancioni et al., 2008; Folmer et al., 2011), yet it is unclear whether the addition of a further modality can also present interaction benefits. Additional work is therefore required to investigate the combination of speech, gaze, and mechanical switches within the software engineering field to support developers with physical impairments.

RQ2: To what extent can natural language and fixed grammar voice coding approaches support the writing, editing, navigation, and selection of code?

The majority of studies that have explored the feasibility of voice-controlled approaches to support coding activities have utilised a fixed grammar approach (i.e. a predefined set of vocal commands) (Wagner and Gray, 2015; Rosenblatt et al., 2018; Soto Munoz et al., 2019). Whilst this type of approach can make coding activities more accessible, it can also present a range of interaction challenges including issues around the cognitive load associated with the recall of voice commands (Good and Howland, 2017; Van Brummelen et al., 2020b). Alternatively, voice-controlled natural language approaches have recently been investigated and explored in domains such as task management (Sarmah et al., 2020), the creative sector (Srinivasan et al., 2019), and home automation systems (Pradhan et al., 2018; Lau et al., 2018; Ramadan et al., 2021). This approach can potentially address issues associated with the recall of commands, although it can present unique challenges around systems being able to understand and interpret a user's intent appropriately. Both approaches have been explored independently, yet it remains unclear which approach might be optimal for performing coding activities. Moreover, no work to date has examined the viability of natural language approaches with disabled developers, so it remains unclear whether this is feasible for this target audience. Further work is therefore required to better understand the strengths and limitations of fixed grammar and natural language methods, as well as disabled developers' preferences in relation to these two methods.

RQ3: How can a multimodal speech and switch interaction approach facilitate efficient code navigation?

Code navigation has been shown to constitute 35% of development time (Ko et al., 2006) with coders requiring efficient navigation tools to support syntax understanding, locate specific code snippets, and for debugging purposes (Bragdon et al., 2010; Piorkowski et al., 2013; Baker et al., 2015; Shakil et al., 2019). Initial research utilising voice-based coding has investigated simple inline code navigation features such as moving to specific line numbers and cursor positioning within a line (Begel and Graham, 2005; Rosenblatt et al., 2018). However, there are also a range of commonly used navigation features used within mainstream IDEs such as "find all references", "go to definition", and "find", which have not yet been explored in the context of interaction approaches utilising voice input. It remains unclear whether these approaches can be tailored for multimodal voice and switch control to support efficient code navigation via alternative methods. Further research is therefore required to investigate the potential of these features to support developers with physical impairments in navigating code.

1.2 Thesis Contributions

The primary contribution of this thesis is a more thorough understanding around the strengths and limitations of alternative interaction methods to support people with physical impairments in performing coding activities such as writing, editing, and navigation of syntax. The main contributions are detailed below:

- The development of a novel multimodal inclusive coding environment utilising a combination of voice control, eye gaze input, and mechanical switches for writing, editing, and navigating HTML and CSS code [Chapter 3].

- The first empirical investigation into the feasibility of a multimodal approach integrating speech, gaze and switch input to support people with physical impairments. Results found that non-disabled users could complete a series of standard coding tasks and perceived the multimodal input approach to be intuitive and simple to operate. A follow-up evaluation with developers who have physical impairments validated that they could also successfully complete a range of common coding activities via the prototype [Chapter 3].

- An exploratory study investigating the types of speech commands developers would prefer to use when writing, editing, and navigating code via speech input [Chapter 4].

- The development of two multimodal speech and switch interaction approaches to support developers with physical impairments in writing and editing code – one utilising a fixed grammar (i.e. predefined vocal commands) and another using natural language input [Chapter 4].

- The first empirical investigation comparing fixed grammar and natural language approaches for coding activities. Results highlighted a similar level of performance across both approaches with participants identifying natural language input as holding significant future potential. A multi-session follow-up study with disabled developers demonstrated that the natural language approach could facilitate the successful completion of common coding activities [Chapter 4].

- A first exploratory study with disabled developers investigating existing voice-based coding navigation strategies. A key theme to emerge was a current lack of voice support and control for standard mainstream IDE navigation features (i.e. "Find all Reference", "Go to Definition", and "Find") [Chapter 5].

- The development of standard code navigation techniques tailored for multimodal voice and switch input to support disabled developers (i.e. “Find all references”, “Go to definition”, and “Find”) [Chapter 5].

- The first empirical evaluation of code navigation techniques such as “Find all References”, “Go to Definition”, and “Find” operated via multimodal speech and switch interaction. Results found that the navigation approaches were efficient and usable and enabled developers with physical impairments to successfully complete a range of standard navigation tasks [Chapter 5].

1.3 Thesis Structure

Chapter 2 provides an introduction to disability and an overview of different forms of assistive technology used by people with physical impairments. Previous research focused around speech, gaze, and switch input is then highlighted with a particular emphasis on work that has investigated these methods for coding activities.

Chapter 3 presents research exploring the potential of a multimodal approach (i.e. a combination of speech, gaze, and switches) to support developers with physical impairments in writing code. Two user evaluations are presented highlighting the usability of this approach with non-disabled and disabled participants.

Chapter 4 investigates the efficacy of two multimodal speech and switch coding approaches – Fixed Grammar and Natural Language. An initial evaluation with non-disabled developers is detailed, as well as a follow-up study with disabled developers where their experience in using both methods is highlighted and discussed.

Chapter 5 presents research utilising a multimodal speech and switch interaction approach to support code navigation activities. An initial exploratory study investigating disabled developers’ existing code navigation approaches is explored. The results of this study informed the design of a research prototype supporting a range of navigation approaches (tailored for multimodal voice interaction). A study involving disabled developers to evaluate this prototype is outlined with key findings highlighted.

Chapter 6 summarises all key findings and insights obtained across all three studies detailed in Chapters 3-5. Limitations associated with the research conducted are also highlighted, alongside a discussion of important future areas requiring further investigation.

1.4 Published Work

The following papers have been published as part of this thesis.

- Paudyal, B., Creed, C., Frutos-Pascual, M. and Williams, I. (2020) Voiceye: A Multimodal Inclusive Development Environment. *In Proceedings of the 2020 ACM Designing Interactive Systems Conference* (pp. 21-33). **[Core A]**

- Paudyal, B. (2020) Assistive Interaction Techniques to Support Disabled Developers. *In Proceedings of the 33rd International BCS Human Computer Interaction Conference (BCS HCI 2020)* (pp. 67-69).

- Paudyal, B., Creed, C., Williams, I. and Frutos-Pascual, M. (2022) Inclusive Multimodal Voice Interaction for Code Navigation. *In Proceedings of the 2022 International Conference on Multimodal Interaction* (pp. 509-519). **[Core A] [Best Demo Award]**

- Paudyal, B., Creed, C., Williams, I., and Frutos-Pascual, M. Multimodal Voice Coding for Developers with Physical Impairments. *International Journal of Human-Computer Interaction* **[SJR Q1 (Human Factors and Ergonomics) – Impact Factor: 4.920] – In Submission**

1.5 Covid-19

This research was completed before and during the COVID-19 pandemic. Research studies and data collected during and after the pandemic followed COVID-19 safety guidelines.

2 Background and Related Work

This chapter reviews literature related to disability and assistive technology, emphasising speech, gaze, and multimodal input as interaction approaches to assist people with physical impairments. To establish the research context, disability is first discussed and defined, followed by an overview of the various technologies employed by people with physical impairments to address accessibility barriers. This is followed by an exploration of research investigating the efficacy of speech interaction in various domains with a particular emphasis on voice coding. Similarly, research examining the use of gaze interaction is then introduced with specific coverage of studies that have examined the use of this method to support coding activities. Research investigating the combination of speech and gaze as a multimodal interaction approach is then highlighted and discussed. Finally, the chapter concludes with a summary of key gaps in the literature where further research is required.

2.1 Disability (Physical Impairment)

Disability is regarded as a universal public health issue with approximately 15% of the global population experiencing some form of disability (Disability and Health, 2022). Recent studies estimate that 22% (14.6 million) of the United Kingdom is affected by some form of disability leading to challenges in their day-to-day activities (Gov UK, 2021). Common forms of disability include physical (Raghavan, 2015; Madaan and Gupta, 2021), visual (Morrison and McKenna, 2002; Hill-Briggs et al., 2007), hearing (Crow, 2008), and cognitive impairments (Oliver and Barnes, 2012; Sheehan and Hassiotis, 2017; Baker, 2022). The World Health Organisation's (WHO) International Classification of Functioning (WHO, 2013) defines disability as a "*dynamic interaction between a person's health condition, environmental factors, and personal factors*". Similarly, the Equality Act (2010) defines disability as *"a physical or mental impairment that has a 'substantial' and 'long-term' negative effect on the person's ability to do normal daily activities*".

The research detailed in this thesis focuses primarily on new assistive approaches to support people with physical impairments affecting the control and movement of their upper body limbs. Physical impairment can be defined as an impairment of body structures such as a significant deviation in the nervous system or structures related to movement (Raghavan, 2015). Physical impairments caused by a congenital condition (e.g. cerebral palsy), an illness or disease affecting the brain, nerves, or muscles (e.g. multiple sclerosis, amyotrophic lateral sclerosis), an injury to the brain, spine, or limb (e.g. spinal cord injury, stroke), or limb loss (e.g. amputation) can all have a negative impact on an individual's ability to perform daily tasks, leading to challenges associated with mobility, manual dexterity, and speech (Becker et al., 1997; Hill-Briggs et al., 2007). There is a broad spectrum of conditions that can result in physical impairments – for instance, cerebral palsy is a neurodevelopmental condition attributed to non-progressive issues in the developing fetal or infant brain and is associated with a group of disorders associated with movement, posture, and speech development (Bax et al., 2005). Symptoms can vary significantly (ranging from mild to severe) - for instance, some individuals have trouble walking and sitting, while others may have severe mobility and speech impairments, thereby affecting their communicative development (Evans et al., 1990; Geytenbeek et al., 2010; Ferreira et al., 2012).

Another common form of physical impairment can result from Repetitive Strain Injury (RSI) that develops gradually and includes symptoms such as burning, stiffness, pain, cramping, or numbness in the fingers, hand, or wrist (Necas, 1996; Yassi, 1997). For example, studies have highlighted how RSIs can develop from the prolonged use of a mouse and keyboard, exacerbated by downward extension or continuous pressure on the wrist (Arnold et al., 2000; Snell, 2000; Desilets, 2001; Begel and Graham, 2006). Carpal tunnel syndrome is also similar to RSI and includes symptoms such as numbness and tingling in the thumb, index finger, and middle finger (Yassi, 1997; Arnold et al., 2000). A further example of a condition resulting in physical impairments is motor neuron disease (MND) - a progressive disorder in which degeneration of upper and lower motor neurons leads to progressive weakness of the bulbar, limb, thoracic, and abdominal muscles (Leigh and Ray-Chaudhuri, 1994). MND includes symptoms such as the loss of speech communication, facial expression, and/or hand gestures, as well as lack of balance and difficulty walking due to muscle weakness (Miller et al., 2012; Mackenzie et al., 2016). There are also a wide range of other conditions that can affect physical abilities such as limb amputation (Jamieson et al., 2021; Zaheer et al., 2021), arthritis (Mease et al., 2018), epicondylitis (Lai et al., 2018; Lenoir et al., 2019), multiple sclerosis (Oh et al., 2018; Dobson and Giovannoni, 2019), Parkinson's disease (Adams et al., 2017; Blauwendraat et al., 2020), and muscular dystrophy (Yiu and Kornberg, 2015; Duan et al., 2021).

Disability can have significant societal implications - for instance, studies have highlighted that disabled people are more likely to experience social exclusion and poverty (Lindsay, 2011; Raja, 2016). This can make them highly reliant on others, hinder their personal development, and negatively impact their quality of life (Lancioni et al., 2008). Additionally, disabled people experience significant barriers and challenges in terms of employment opportunities and are under-represented in the workforce (Bell and Heitmueller, 2009; Whitehead et al., 2009; Lindsay, 2011; Hogan et al., 2012; Vornholt et al., 2018). Efforts have been undertaken to increasingly include disabled people in the labour market – for instance, the UK Government passed the Disability Discrimination Act in 1995, intending to end discrimination against disabled people by protecting employees and providing greater access to goods, facilities and services (Shaw and Coles, 2004; Pope and Bambra, 2005; Bell and Heitmueller, 2009). Furthermore, technological advancements (such as mobile applications and personal computers) enabled disabled people to enter employment and work professionally (Hogan et al., 2012; Macdonald and Clayton, 2013; Graham et al., 2018). However, despite these technological advancements increasing employment opportunities for disabled people, previous research has identified barriers associated with the "digital divide" when accessing work opportunities that can lead to exclusionary outcomes (Pieper et al., 2003; Macdonald and Clayton, 2013; Mavrou and Hoogerwerf, 2016).

Moreover, studies have highlighted that people with impairments are frequently overlooked during the design phase of systems due to the misconception that doing so requires additional effort, time, complexity, and expense. This results in tools that are not accessible to these user groups, thus presenting interaction barriers resulting in them being excluded from using mainstream applications (Mankoff et al., 2005; Beelders and Blignaut, 2010; Abualghaib et al., 2019). Common digital technologies such as computers, tablets, and other mobile devices typically require the dexterity of human hands to control them, which can present significant challenges for people with physical impairments (Desilets, 2001; Begel and Graham, 2005; Rosenblatt et al., 2018; De León Cordero et al., 2021). Similarly, these tools present substantial obstacles when performing actions such as clicking elements and focusing a mouse over small interface targets within digital experiences (e.g. buttons, icons, links) (Giakoumis et al., 2014; Valencia et al., 2017). Furthermore, prolonged use of input devices such as a mouse and keyboard is associated with an increased risk of RSI and carpal tunnel syndrome (CTS), which in turn can also present accessibility challenges (Arnold et al., 2000; Snell, 2000; Begel and Graham, 2006).

These factors can lead to the exclusion of disabled individuals from development work and the opportunity to pursue technical careers (Wagner and Gray, 2015; Rosenblatt et al., 2018). It is hard to quantify the exact number of developers with physical impairments, although the percentage of developers who self-reported having some form of physical impairment in the 2022 Stack Overflow survey (StackOverflow, 2022) was less than one percent (in relation to 70,000 responses from developers). According to the US National Bureau of Labour Statistics (Labour Statistics, 2021), only about 3% of workers in the computing and mathematical professions have a disability, compared to the US Census Bureau's estimate of approximately 20.6% of the general population. These figures suggest that developers with physical impairments remain underrepresented in the software development industry (Rosenblatt, 2017; Marshall, 2022).

Research has explored different approaches to make digital systems accessible for people with impairments resulting in the development of commercial products and research prototypes. For instance, research has explored interaction techniques such as eye trackers, speech recognition, motion tracking, and gesturing as alternative input methods, which can have a profoundly positive effect on the daily lives of disabled people (Hurst and Tobias, 2011; Creed, 2018; Petrie et al., 2018). The following section provides a general overview of assistive technologies commonly utilised by people with physical impairments.

2.2 Assistive Technologies

Assistive technology (AT) is typically defined as a device, product, or piece of equipment acquired commercially off the shelf, modified or customised to support an individual’s functional capabilities, thus enabling them to perform tasks they might not be able to do otherwise (Brown, 1992; Wilcox et al., 1999; Hurst and Tobias, 2011; Bennett et al., 2018;

Creed, 2018; Gatchalian, 2019). Assistive technologies for people with physical impairments can range from "low tech" physical solutions (e.g. switches, magnifiers, and walking sticks) to "high-tech" digital applications offering alternative methods to control systems (i.e. speech recognition, touch technology, eye gaze interaction) (Reichle, 2011; Iacono et al., 2013). The following subsections provide a high-level overview of key assistive technologies utilised by people with physical impairments (including benefits and limitations) before covering speech and gaze interaction in further detail (which forms the core focus of technology utilised in this thesis).

Alternative Keyboard and Mouse Devices: Researchers have explored adjustments to conventional keyboard designs to make them more adjustable, ergonomic and effective for supporting people with physical impairments (Brodwin et al., 2004; Henzen and Nohama, 2016; Nagendran et al., 2021). Several ergonomic keyboard designs, such as split and tented keyboards, negative slope devices, and keyboards with altered key positions, have been studied to reduce the physical demands of muscle tension during typing (Figure 2.1) (Smith et al., 1998; Marklin and Simoneau, 2001). Researchers have also investigated how virtual keyboards can be operated using additional input devices such as eye gaze, touch and head control (Henzen and Nohama, 2016). Studies have highlighted that these physical and virtual keyboards relieve the burden on shoulders and neck, provide more comfort, and decrease fatigue in arms and hands while typing (Lincoln et al., 2000; Gür et al., 2020; Nagendran et al., 2021). Similarly, studies have explored alternative mouse-pointing devices such as a trackball, touchpad, and vertical mouse as interaction approaches for people who may experience challenges using traditional designs (Figure 2.2) (Lee, 2005; Ziefle, 2019).

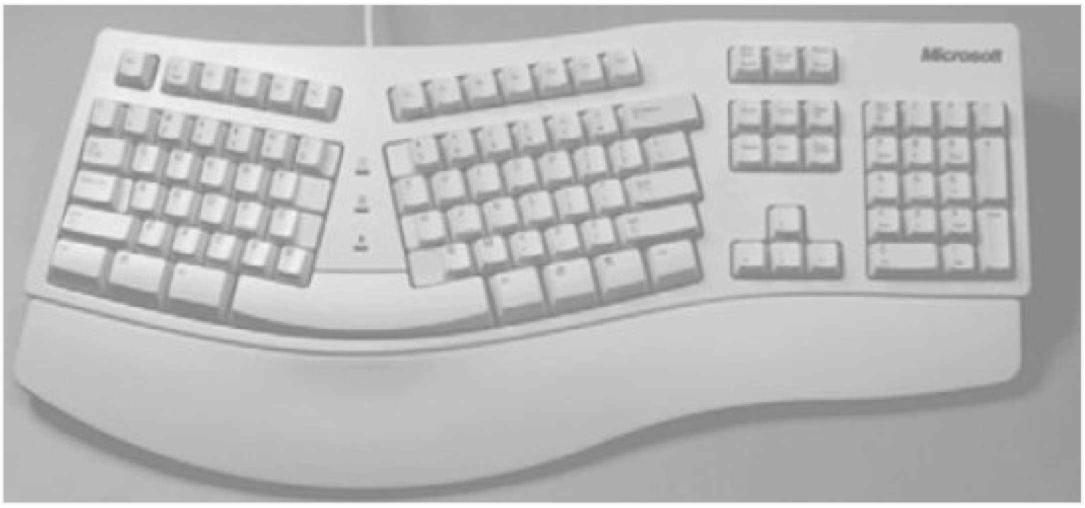


Figure 2.1: Microsoft Natural keyboard, an example of a Fixed-split keyboard. Image courtesy of McLoone et al. (2010)

Studies have highlighted that these devices decrease shoulder exertion, neck and shoulder muscle activity, and shoulder elevation (Bruno Garza and Young, 2015). Despite these benefits, research has highlighted several issues associated with the use of these devices - for instance, prolonged overuse of the tools and adopting a certain sitting posture can cause tiredness and can also lead to other types of injuries (Tiric-Campara et al., 2014; Mani, 2018).

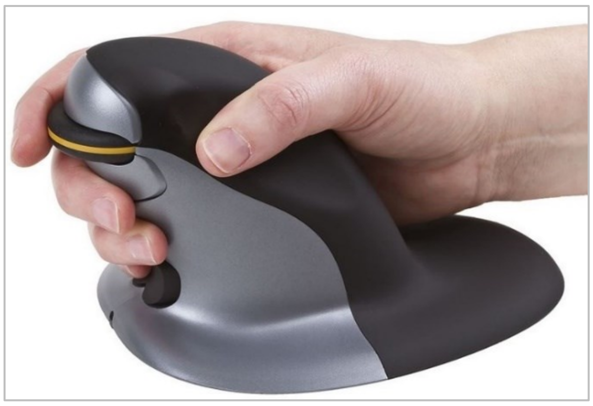


Figure 2.2: A vertical mouse from Penguin Ambidextrous. Image courtesy of Penguin (2021)

Touch Interaction: With the advancement of touch-enabled devices such as smartphones and tablets, touch has become an essential interaction approach as it can provide a natural input experience through the manipulation of onscreen objects without the direct use of any intermediary devices (Guerreiro et al., 2010; Mott et al., 2016; Gheran et al., 2018). The accessibility of this input method has also been investigated with users who have physical impairments (Kane et al., 2009; Anthony et al., 2013) - for instance, studies have highlighted that touch-enabled devices can facilitate faster interactions than a mouse (Anthony et al., 2013; Findlater et al., 2017). Additionally, digital buttons on a screen require less force to tap than physical buttons, resulting in less fatigue and pain (Irwin and Sesto, 2012; Naftali and Findlater, 2014). However, whilst touch can potentially make the interface easier to control, it also presents accessibility issues – for example, users typically have to suspend an arm, extend a finger and slide fingers across the screen (to perform swipes and strokes), which can be problematic for people with limited motor control (Mott et al., 2016; Gheran et al., 2018). Similarly, Findlater et al. (2017) highlighted that people with physical disabilities have a much higher error rate when making selections that involve touch, whilst Trewin et al. (2013) reported that people might encounter interaction challenges such as activating other features by accident when using touch (e.g. zooming instead of tapping).

Head Tracking: Researchers have also explored using head-tracking as an interaction method (Zapala and Balaj, 2012; Rodrigues et al., 2017). This approach has received significant attention over the past decade as detecting head movements can be easily captured through standard video cameras (Al-Rahayfeh and Faezipour, 2013; Sahadat et al., 2018; Elsahar et al., 2019). Numerous studies have examined the use of head tracking for controlling digital applications with a particular emphasis on cursor control (Machado et al., 2013; Rodriguez-Cartagena et al., 2015; Sahadat et al., 2018). Similarly, studies have also investigated the use of head tracking to facilitate navigation, obstacle recognition and automatic movement within a wheelchair (Manogna et al., 2010; Haque et al., 2021). However, whilst head tracking can make interface control somewhat accessible, there are also unique challenges associated with this technology, such as the inability to make continuous head movements or to exert sufficient control when fine movements are required (Ossmann et al., 2012). Similarly, people with physical disabilities may have limited neck motion and, as a result, a reduced ability to move the head in one or more directions (Karpov et al., 2004).

Mechanical Switches: Switches can range from simple physical buttons to head, foot or breath-controlled devices that can be activated more easily by people with limited motor control via applying pressure on a large button (e.g. using hands, fingers, arms, or feet) (Wilcox et al., 1999; Lancioni et al., 2008; Geytenbeek et al., 2010; Ntoa et al., 2014;

Standen et al., 2014). Switch-based interaction typically employs an indirect selection approach (also known as a scanning process) which involves intermediary steps for making selections within an interface. During the scanning process, items are highlighted in an application (for a predetermined amount of time), and the user can select their desired highlighted item by pressing the switch (Figure 2.3) (Biswas and Robinson, 2007; Folmer et al., 2011). Studies have highlighted that mechanical switches can provide a more robust, simple, and accessible input approach for people with physical impairments (Anson, 1994; Tai et al., 2008). More recently, mechanical switches have been combined with other modalities to make a multimodal interaction technique to support people with physical impairments in creative tasks (Creed et al., 2020; Aziz et al., 2021, Aziz et al.,2022). In their work, switches were combined with other input approaches such as speech and gaze to control the interface. However, whilst switches can make some applications more accessible, prolonged use has also been shown to cause mental and physical fatigue (Kennedy et al., 2000; Krausz et al., 2003).

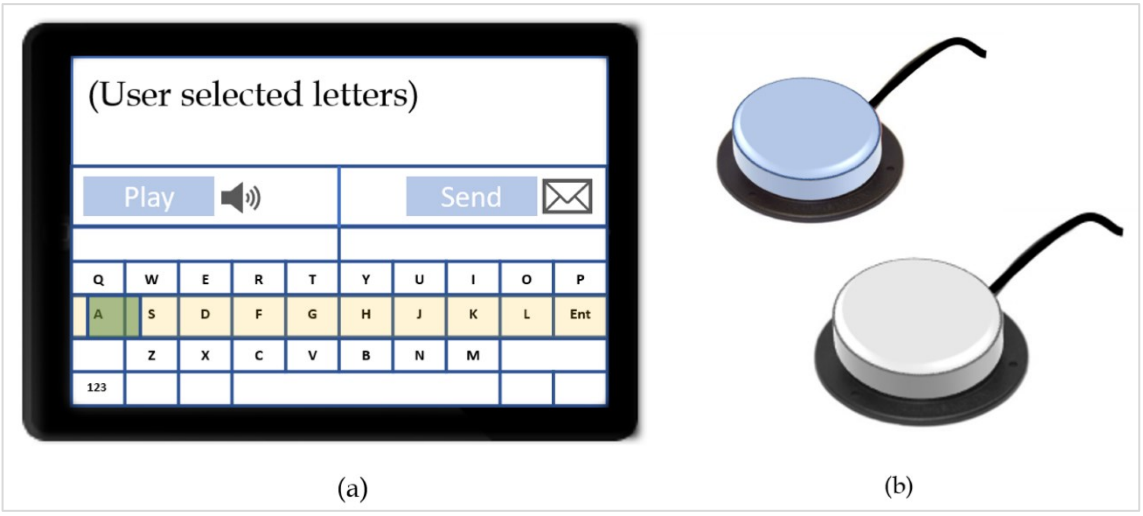


Figure 2.3: (a) A sample visual scanning interface activated via switch scanning. The yellow box moves vertically across the lines until a selection is made, followed by a gliding green box moving horizontally across the highlighted line until a letter is selected, (b): two switches. Image courtesy of Elsahar et al.(2019)

Two further technologies that have been explored extensively in terms of alternative interaction methods are speech and gaze interaction. Both have become more readily available in recent years and can support accessible interfaces for people with physical impairments. Furthermore, studies have started investigating the potential of these methods to support coding and development activities which can benefit disabled coders. A more detailed review of these specific technologies is provided below, alongside a review of research on multimodal interfaces investigating the potential of combining both of these technologies to provide more inclusive interactive systems.

2.3 Speech Interaction

2.3.1 Speech Recognition Technology

Research on automatic speech recognition (ASR) began in the 1950s when researchers at Bell Labs developed a speech recognition system (Audrey) that accurately identified 10 English digits (Meng et al., 2012). Speech recognition gradually began to evolve during the 1960s to convert speech input into character strings or commands (Karat et al., 1999; Feng and Sears, 2004; Gaikwad et al., 2010). Similarly, an initial investigation into continuous speech recognition was conducted using dynamic phoneme tracking during this period (Reddy, 1966). However, until the 1980s, ASR technology was expensive (Beelders, 2011), had poor recognition accuracy (Hu et al., 2011), and supported a limited vocabulary size (Schmandt et al., 1990).

The 1980s saw a flourish of activity in speech recognition technology with studies focusing on developing robust systems capable of recognising a fluently spoken string of concatenated words (Furui, 2010; Gulzar et al., 2014). A wide range of algorithms including two-level dynamic programming, one-pass methods, and statistical approaches such as the Hidden Markov Model (HMM) and N-gram were implemented to match concatenated patterns of individual words (Juang and Furui, 2000; Furui, 2010; Meng et al., 2012). ASR received increased public attention in the early 1990s, driven partly by a drop in the cost of speech recognition software (Danis and Karat, 1995). The advancement of digital signal processing, pattern matching and classification algorithms around this period also contributed to the mass development of applications supporting speech input (Karl et al., 1992).

Software such as Dragon Naturally Speaking and IBM Via Voice were available commercially on desktop computers in the mid-1990s, allowing users to write letters, compose emails, and work on spreadsheets by dictating text to the computer (Snell, 2000;

Begel and Graham, 2005; Hu et al., 2011). Continuous speech recognition was also developed around this period, allowing users to speak at their natural pace and rhythm (Koester, 2001). Since the 2000s, machine learning algorithms have been utilised to improve the accuracy and reduction of recognition errors, resulting in speech technology maturing considerably (Benkerzaz et al., 2019; Hannun, 2021; Malik et al., 2021). Today's systems (e.g. IBM Watson (2023), Google Speech (2023)) typically employ a standard procedure in which captured speech input is firstly digitised, then compared against a developed vocabulary, and finally translated and displayed (Freedman, 1995). This process comprises three key steps: pre-processing, recognition, and communication (Rosen and Yampolsky, 2000). During pre-processing, the analogue speech waveform obtained from the human speech input is transformed into a digital signal by sampling the signal at equally spaced points in time (Rosen and Yampolsky, 2000; Trivedi et al., 2018). In the recognition phase, algorithms such as Hidden Markov Models are utilised to identify the spoken input (Rosen and Yampolsky, 2000), whilst in the communication phase the corresponding spoken text is displayed back to the user (Figure 2.4).

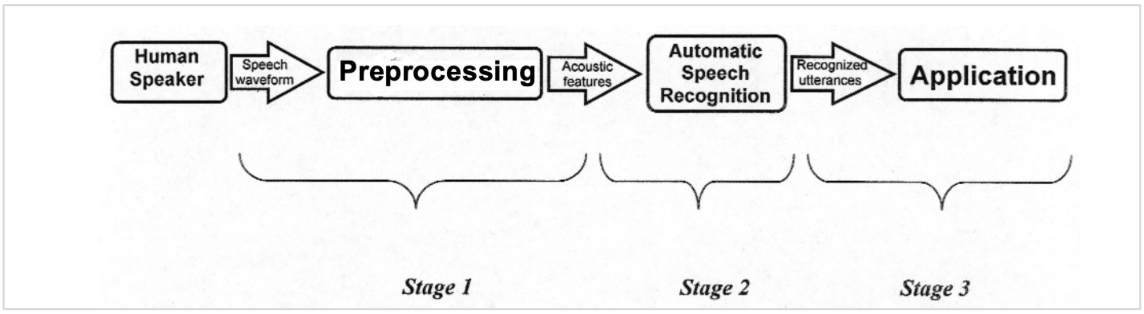


Figure 2.4: ASR Process. Image courtesy of Rosen and Yampolsky (2000)

Systems utilising voice interaction have further gained in popularity over recent years, culminating in the development of commercial voice-enabled international personal assistants such as Amazon Alexa (2023), Google Assistant (2023), and Apple's Siri (2023). These products enable users to interact with devices using natural language speech to perform a range of activities such as checking the weather, controlling home appliances or streaming music, among others (Pradhan et al., 2018; Lau et al., 2018; Ramadan et al., 2021). These significant advancements in speech recognition technology have also facilitated research exploring new methods of interaction in controlling applications for disabled people detailed in subsequent sections.

2.3.2 Speech Interaction Research

Speech technology has been widely used to support accessibility and has been a popular interaction approach for disabled people (Danis et al., 1994; Pradhan et al., 2018). In particular, in terms of people with physical impairments, speech input presents an alternative approach for users who may experience challenges in using traditional input devices such as a mouse and keyboard (Dai et al., 2003; Sears et al., 2003; Pradhan et al., 2018; Aziz et al., 2021, 2022). Research has focused on a range of key areas such as text entry via dictation (De La Paz, 1999; Feng and Sears, 2004; McCrocklin et al., 2019), supporting web browsing (Sarmah et al., 2020; Cambre et al., 2021), providing alternative cursor control methods (Dai et al., 2003; Harada et al., 2006), facilitating target selection (Sears et al., 2002; Zhu et al., 2009; Zhang et al., 2020), and for supporting creative work (Srinivasan et al., 2019; Kim et al., 2019; Aziz et al., 2022). Previous work exploring the use of voice interaction has primarily focused on three core approaches: grammar-based commands, natural language, and non-speech sounds (outlined in the subsections below).

2.3.2.1 Grammar-Based Commands

Grammar-based methods typically enable users to issue commands from a predefined list of possible commands in a strict and predefined way created through grammar rules (Rudžionis et al., 2013; Derboven et al., 2014; Kim et al., 2019). These rules specify the patterns and word sequences to be compared with vocal input (Wagner et al., 2012; Ferracani et al., 2017)– for instance, the user has to utter the exact phrase from the predefined list (e.g. if the command for moving a position right is “move right”, the voice inputs “right move”, “right” will not trigger any actions). Researchers have implemented the use of grammar-based or fixed commands in a wide range of applications – for instance, prior work on cursor movement and target selection used fixed commands such as "left", "right", "up", "down", "click", "stop" for controlling the motion of mouse (Karimullah and Sears, 2002; Sears et al., 2003; Dai et al., 2003). Similarly, Aziz et al. (2021) utilised predefined voice commands such as “left”, “right”, “up”, among others to support the positioning of graphical objects within a design canvas. Sears et al. (2003) explored fixed commands for dictating and editing text – their system utilised commands such as “select x” (for selecting words) and “move right” to move the cursor one position right. Aziz et al. (2022) developed voice-based approaches to manipulate the size of graphical objects within a digital canvas. Users could issue fixed voice commands such as “big” and “small” to resize objects from different directions based on transformation handles assigned to each object's boundary. Elepfandt and Grund (2012) also investigated the use of short voice commands such as “select” and “drop” for object manipulation and highlighted that participants prefer fixed voice commands with fewer words and syllables.

Kim et al. (2019) developed an extension of Adobe Photoshop that utilises short vocal commands and touch input for selecting the tools and menus, changing parameters and manipulating the document layers. For example, users are able to click the button on their stylus pen and issue vocal shortcuts such as "watercolour brush" to select the relevant brush and again can issue a command such as "size 50" or "50" to change the brush size. Hu et al. (2011) used predefined commands (such as “select book” and “move up five lines”) for interacting to perform tasks such as editing text and web browsing. Moreover, Peixoto et al. (2013) used predefined commands to control a powered wheelchair through commands such as “right”, “left”, “forward”, and “reverse”.

Studies have explored the possibilities of creating predefined grammars tailored to a disabled individual’s specific needs– for instance, Cavalcante and Lorens (2015) designed a system that enables users to customise their own grammars for triggering actions such as “I am in the park”, “up”, “down”. The approach was evaluated using a person with physical impairments. Similarly, Talon (2023) allows users to customise grammar for performing a range of activities, including coding - for example, to type the word "dad", the user can utilise chained custom commands such as “drum air drum”.

Overall, studies have highlighted that using predefined commands has higher recognition accuracy as they are easy to train due to the limited vocabulary and grammar (Rayner et al., 2005; Rudžionis et al., 2013; Cavalcante and Lorens, 2015). Whilst a predefined grammar can potentially support user interactions, studies have highlighted some negative aspects such as users having to learn and recall commands (thus increasing cognitive load) and challenges in recognising different accents resulting in users having to repeat the same command on multiple occasions (Wagner and Gray, 2015; Good and Howland, 2017; Kim et al., 2019; Van Brummelen et al., 2020b).

2.3.2.2 Natural Language Interaction

Research has also started investigating whether natural language commands can provide a more intuitive and effective approach for interacting with digital applications (Srinivasan et al., 2019; Van Brummelen et al., 2020b; Cambre et al., 2021). Researchers have highlighted that natural language commands are more flexible and often provide users with a higher degree of freedom in issuing commands to a speech engine (Rudžionis et al., 2013; Sarmah et al., 2020). In particular, recent significant improvements in natural language understanding and speech recognition have increased the development of voice-enabled technologies such as chatbots, search engines and home automation platforms (Porcheron et al., 2018; Van Brummelen et al., 2020b; Song et al., 2022). Voice-enabled intelligent personal assistants such as Amazon Alexa (2023), Google Assistant (2023), and Apple Siri (2023) are also widely available within smart devices and facilitate more natural conversations between users and intelligent systems (Cambre and Kulkarni, 2019; Clark et al., 2019). Similarly, Stahl and Laub (2017) developed a voice-enabled system that enables users to control the lights, telephone, television, computer and front door within their home environment. An evaluation of the system with participants who have physical impairments found the system to be accessible and usable, as well as empowering users to manage their daily activities. Similarly, Kowalski et al. (2019) explored the potential of natural language voice interactions with older adults in the context of smart home technology. Results highlighted positive perceptions from participants as the natural language approach enabled participants to issue commands with less training.

Sarmah et al. (2020) developed a tool that allows users to control different web applications via natural language voice input through facilitating the adding and modifying of calendar events (i.e. “move this [event] to next week”, “shift Group Meeting to Friday”) and controlling music players (e.g. “adding all of these to the playlist”). Participants in a user evaluation perceived the usability of the system positively as they found it was able to accurately understand their natural language input, thus resulting in it being simple and easy to learn. Similarly, Cambre et al. (2021) investigated the use of natural language voice commands to allow users to navigate the web (e.g. “page scrolling”, “moving backward and forward”) and browser utilities (e.g. “opening email/calendar”, “bookmarking pages”) through the Firefox Voice browser extension. During the initial deployment phase (July 2020), more than 12,000 users installed the tool in their browser, although usage declined, and it was decommissioned in February 2021. More widely, Jung et al. (2019) designed a voice-enabled interactive educational programming game (TurtleTalk) for children, allowing them to use natural voice input to move the turtle to a target location. While moving the turtle, TurtleTalk utilised programming concepts such as sequencing (e.g. "move forward and then turn left") and iteration (e.g. "move forward six times"). Initial exploratory research with children highlighted that voice interaction with the tool led children to be more immersed in the experience and also found the interaction to be easy and useful.

Whilst studies have started to highlight the potential of natural language voice interaction between users and intelligent systems, researchers have also identified challenges where the use of natural commands (without providing context) can lead to users "guessing" commands, thus leading to interpretation errors and frustration (Rudžionis et al., 2013; Srinivasan et al., 2019). Additionally, some of the voice commands issued can be personal (i.e. user asking the system to tell their name, their age) and complicated, which may pose system comprehension challenges, resulting in user disappointment and dissatisfaction (Luger and Sellen, 2016; Bentley et al., 2018; Zargham et al., 2022). Studies have suggested various approaches to limit these issues – for instance, Srinivasan et al. (2019) implemented a hybrid approach (i.e. combination of natural language and fixed grammar), where users were displayed with contextually relevant examples to make them aware of the commands they can issue within an experience. Results highlighted positive responses from participants, although issues such as speech misrecognition and misinterpretation resulted in interaction challenges. Similarly, studies have explored error-handling approaches where the system provides an appropriate response if a user command is not recognised (Bohus and Rudnicky, 2008; Pearl, 2016). Zargham et al. (2022) also implemented an anticipatory error handling approach within a speech-controlled video game, where the system picks the best available command match if it is unable to recognize the speech commands issued by a user.

2.3.2.3 Non-Verbal Commands

Studies have also explored non-verbal speech commands as an interaction approach for controlling digital applications. For instance, Igarashi and Hughes (2001) examined several approaches such as "control by continuous voice" (where the user produces continuous non-verbal sounds to trigger an action - e.g. the command "volume up, ahhhhhh" will continue to increase the volume as long as “ahhh” continues), “control by pitch” (where a change in pitch of a user’s voice can alter a particular control – e.g. "move up, ahhhh" for increasing scrolling speeds) and “control by tonguing” (where discrete peaks in sound signals are detected to form particular actions - e.g. “volume up ta ta ta” will increase the volume by three points). No user study was conducted in this research, so the usability and reliability of this specific approach remain unclear. Similarly, Mihara et al. (2005) combined both discrete verbal commands (e.g. "move right" to move the cursor to the right) and non-verbal commands (e.g. "ahhhh" to move the cursor continuously) for positioning a cursor to a particular location. Whilst this facilitated control of the cursor, results from a user evaluation highlighted that the participants felt tired when making continuous non-verbal sounds. Furthermore, Bilmes et al. (2005) and Harada et al. (2006) both utilised vowel sounds (e.g. “a” for up, “i” for down) for mouse control, while Sporka et al. (2006) implemented humming ("mmm") sounds for controlling a Tetris game where tones with a falling pitch triggered “left” movement actions and rising tones resulted in “right” movements. As part of this work, the authors conducted a comparative study between non-speech and standard speech commands with results highlighting that non-verbal commands facilitated more efficient and accurate responses to support gameplay. Lea et al. (2022) explored non-verbal mouth sounds such as "pop" and "click" to perform actions such as selecting items or navigating applications on mobile devices. The study initially used the recordings of more than 700 people with different accents (e.g. British, Chinese, Spanish) to develop and train a model to detect non-verbal sounds. The system was tested with 28 people with speech differences (such as cerebral palsy, muscular dystrophy, and motor-speech disorder) and received positive feedback from participants in a user evaluation. Similarly, Jaddoh et al. (2021) examined non-verbal commands such as “Ah” to play the news and “/u:/” (pronounced as “oo”) to lower the volume when interacting with virtual home assistants for people with speech impairments. Whilst this approach supported more accessible interactions, an initial evaluation identified challenges in terms of cognitive load associated with having to memorise and issue commands. Previous work has also highlighted that the use of non-verbal commands can address recognition issues in fixed grammar and natural language approaches through reducing the requirement to use short single-syllable words and homophones (Igarashi and Hughes, 2001; Cambre et al., 2021; Lea et al., 2022; Wei et al., 2022).

2.3.3 Speech Interaction for Coding

One key area where researchers have explored the efficacy of speech interaction is within the domain of voice coding. This specific area can present some unique challenges – for example, programming languages utilise punctuation symbols, as well as the composition of words and abbreviations which may not be natively or accurately handled by speech recognisers. Research has therefore explored a variety of approaches to support efficient and productive approaches for supporting the writing, editing, and navigation of code via voice interaction.

For instance, a voice-based code editing plugin for the Eclipse environment (VASDE - Voice-Activated Syntax-Directed Editor) was developed for creating Java Syntax code. To support with writing syntax, the plugin presents a dialogue box with predefined code that users can select via voice commands, as well as additional commands for controlling the editor interface (Hubbell et al., 2006). Similarly, Shaik et al. (2003) developed SpeechClipse, a plugin embedded in the Eclipse IDE to allow users to use voice commands to invoke keyboard actions that allow users to control menus and dialogue boxes, although the main coding editor was inaccessible via speech. VoiceGrip, a code generation tool developed by Desilets, (2001), enables users to verbally state pseudo-code, which is then automatically translated into syntax for a specified programming language. However, the tool was unable to distinguish between homophonic words (e.g. "for" with "four", “to” with “two”), making it highly error-prone and frustrating for users. The tool was further iterated upon and a new tool (VoiceCode - (Désilets et al., 2006)) was developed, capable of generating code through high-level spoken syntax. For example, a vocal command such as “*if current record number is less than max offset then*” would generate the following syntax:

```
if(currRecNum < maxOffset){

}
```

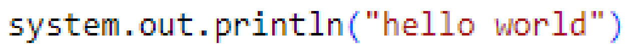
Begel and Graham (2005) also explored how programmers write code via voice input and highlighted that developers use high-level abstraction and shorthand notations rather than speaking each character in turn (i.e. “i >10” is verbalised as “i greater than 10”). Their tool SPEED (SPEech EDitor), an Eclipse plugin, allows developers to verbalise Java code in general spoken language. Table 2.1 displays a verbalised command along with its generated code:

Command	**Code**

For int, i equals zero i less than ten i plus plus
X gets math dot cosine x
End for loop
	for(int i=0; i<10; i++){
	  x=math.cos(x)
	}

Table 2.1: Example voice command and corresponding code generated via the SPEED tool by Begel and Graham (2005)

While the application was intended for developers with physical impairments, it was evaluated with experienced developers without physical impairments who perceived speech-based programming as less efficient than typing (primarily due to speech misrecognition issues). Ayub and Saleem (2012) developed a system that allowed users to verbalise a programming language's syntax to generate code. Their system compares voice input against the reserved keywords (such as “cin”, “cout”, “for”, “loop”) of the C++ programming language and returns the best possible matched syntax. For instance, if the user states the command “see out”, the system generates the C++ code “cout <<” as it closely matches the C++ syntax (“cout”). A similar approach was implemented by Patel and Patel (2014) using Java, enabling a user to generate code through verbalisation of the language’s syntax and semantics. For example, “*hello world string*” will output the following code:



```
system.out.println("hello world")
```

Rudd (2013) built a Python-based dictation tool to allow developers to write code using non-natural voice commands mapped to different actions. For instance, the command “slap” is used to trigger the enter key, while “sup” is used for searching tasks. Talon (2023) is a tool widely used by developers with physical impairments (Nowogdorzki, 2018; Nowrin et al., 2022) that also employs a similar approach (e.g. the user has to say “slap” to move the cursor to the end of the line). Talon can be integrated into mainstream coding IDEs such as Visual Studio Code and Atoms, although it can be challenging for novice users to use as it requires learning custom voice commands in addition to programming syntax (De León Cordero et al., 2021; Nowrin et al., 2022).

In another approach, Wagner and Gray (2015) integrated speech interaction to trigger features such as “click”, “drag”, and “delete” with Scratch (a block-based programming tool). A user evaluation was conducted with individuals who have physical impairments, although the tool suffered from significant speech misrecognition issues which negatively influenced participants' perceptions. Other studies have also explored speech for coding purposes – for instance, Rosenblatt et al. (2018) developed a voice-based coding tool (VoiceIDE) that supports people with upper motor impairments in writing code. The tool was informed through a Wizard of Oz study, which provided insights into the commands developers prefer to use when performing coding activities. The system allowed users to perform a number of coding activities such as writing (e.g. "make a for loop" to generate a for loop), navigation (e.g. "go to line x" to move the cursor to line x), deletion (e.g. "delete" to remove a character), and selection of code (e.g. "select line" to select the line on the current cursor position). They also introduced a context colour editing feature where syntax on different lines is highlighted with different colours, thus allowing users to specify corresponding colour names in associated with different actions (e.g. "delete red" removes the character or syntax highlighted in red). Whilst the evaluation with users who have physical impairments demonstrated that navigation and selection tasks could be performed more effectively using VocalIDE, the system was limited by speech accuracy challenges. Okafor and Ludi (2022) proposed a speech-based interaction approach that enables people with upper-body motor impairments to code using a block-based coding tool. A preliminary investigation into the viability of voice commands revealed the importance of ensuring that short vocal commands are utilised to reduce the vocal effort required (e.g. "up" instead of "move up" and "delete" instead of "delete block").

The majority of studies highlighted have utilised fixed or constrained commands, where the user has to verbalise a specific command to initiate different coding activities. Researchers have also started to explore the use of unconstrained natural language in coding environments. For instance, Gordon (2013) developed a new programming language that allows developers to dictate code via natural language phrases to create high-level programming language constructs such as declaring variables and creating loops. For example, to create a variable, the user can issue commands such as "set Y to 2 * X" and "Set Y to 2X"). An initial evaluation with participants produced mixed results – participants with prior coding experience successfully completed all tasks, although participants with less development experience experienced challenges such as forgetting the commands, and speech misrecognition issues. De León Cordero et al. (2021) developed a generic programming vocabulary utilising natural language commands (issued via voice input) to allow users to write basic syntaxes such as declaring variables and inserting conditional expressions and loops. For example, to create a variable, commands such as “define integer a equals 5”, “define integer b 5”, and “assign number 5 to variable b” can be used. Whilst the system targets developers with limited mobility, no user evaluations have been conducted around the application to date. Additionally, the focus is only on writing code, as opposed to other coding activities such as syntax deletion, navigation and selection.

Chadha et al. (2018) and Modak et al. (2016) utilised natural language voice commands to generate HTML and CSS code (e.g. the command “create a table with five rows and three columns” would generate the corresponding HTML code). Whilst this approach helps users quickly build web pages, the system is incapable of supporting key actions such as assigning properties to an element (i.e. id and class). Van Brummelen et al. (2020, 2020b) introduced CONVO, a conversational programming tool that allows users to interact with a conversational agent using natural voice input to perform programming activities. In this application, users can have intelligent conversations with the system – for example, a user can issue a command such as “create a variable”, which triggers the system to ask for the variable name. Whilst the tool was perceived positively in terms of accessibility and usability, the system suffered from speech recognition accuracy issues which presented significant challenges for participants. Similarly, studies have explored the potential of natural language voice commands for enabling users to query databases (Obaido et al., 2020) – for instance, TalkSQL translates natural language input into an executable Structured Query Language (SQL) query (e.g. the command “amend the student name to x whose id is x” would update the table as per the query, i.e. “update student set name =’x’ where id=x”). Whilst the tool received positive feedback from the users, however, was unable to handle complex queries such as nested statements.

Serenade (2023) is an open-source voice coding system that allows users to insert, navigate, delete and edit code in any programming language via natural language speech. The tool includes features such as writing code (e.g. “add class page” generates “class page { }”) and editing (“rename function to x” changes the function name to “x”). Table 2.2 displays some of the vocal commands and the generated output in Serenade.

Command	**Generated Output/Action**
Add function factorial	Function factorial(){ }
Delete lines three to four	Delete multiple lines
Add import numpy as np	import numpy as np
Add return say of string hello	return say (“hello”)
Add while i not equal zero	while(i!=0){ }

Table 2.2: Verbal commands and the generated output in Serenade

Whilst the system is readily available as an extension for applications such as Visual Studio Code and Chrome, the usability and efficacy of the system remain unclear as no evaluations have been reported within the literature to date.

Another key area associated with voice coding is the navigation of syntax - the majority of speech based coding studies highlighted have employed common inline navigational commands such as left/right/up/down. Désilets et al. (2006) enabled users to move the cursor to different code blocks through voice using navigation commands such as “next comma” (i.e. to move the cursor after the comma) and “after clients array at index zero” (i.e. to move the cursor after the particular code snippet). Similarly, Begel and Graham (2005) created a context-sensitive mouse grid structure with labels on the screen where users could verbalise grid numbers to move the cursor to the corresponding position with the code. Additionally, they used commands such as “jump to constructor linked list” (i.e. to move the cursor to LinkedList constructor) and “move down 2 lines” (i.e. to move two lines below the current cursor position). Moreover, Rosenblatt et al. (2018) utilised vocal commands such as “go to line” and “up X lines” to navigate between lines of code.

Whilst researchers have highlighted that speech recognition has matured to an acceptable level due to its successful integration into commercial products (Errattahi et al., 2018; Filippidou and Moussiades, 2020), voice interaction comes with inherent limitations around recognition issues and accuracy (Suhm et al., 2001; Wagner and Gray, 2015; Rosenblatt et al., 2018). Additionally, speech is associated with poor performance in noisy environments

(Suhm et al., 2001) and impracticality for pointing and selecting tasks (Rozado et al., 2016). Furthermore, while studies have investigated the use of voice input to support developers with physical impairments in coding activities, there remain several areas where little or no work has been conducted to date. For instance, it remains unclear which voice coding approaches are most suitable for supporting developers with physical impairments. Moreover, no comparative analysis has been conducted to date to gauge the efficacy and usability of fixed grammar, and natural language approaches when performing coding activities.

2.4 Eye Gaze Interaction

2.4.1 Eye Gaze Tracking Technology

The use of eye gaze tracking technology has also been explored as an alternative means for enabling people with physical impairments to interact with digital systems (Sibert et al., 2001; Jacob and Karn, 2003; Duchowski, 2018; Creed et al., 2020). Studies around this area began in the nineteenth century to observe and understand eye movements when reading (Rayner, 1998; Poole and Ball, 2004), although it was not until the 1970s and 1980s that it started to flourish as a technology. This was driven through improvements in the accuracy and precision of eye-tracking technologies (Rayner, 1998; Nivala et al., 2016; Sharafi et al., 2020), as well as increased interest in physiological studies within the scientific community, linking eye-tracking data with cognitive processes (Jacob and Karn, 2003). The majority of these studies focused on analysing eye movement data for physiological observation and were confined to controlled laboratory environments (Rayner, 1998; Morimoto and Mimica, 2005; Majaranta and Bulling, 2014; Hemmingsson and Borgestig, 2020).

More recently, studies began to explore the possibilities of eye tracking within the HCI field (Rayner, 1998; Jacob and Karn, 2003; Poole and Ball, 2004). This initiated the demand for using eye gaze tracking in research studies to solve usability issues (Schiessl et al., 2003; Li et al., 2005) and for interacting with a graphical interface (Sibert et al., 2001; Jacob and Karn, 2003). In turn, this led to consumer level eye trackers becoming more widely available and accepted within the HCI community as a part of empirical research studies (Dalmaijer, 2014; Duchowski, 2018). Three types of eye tracking techniques have predominantly been used - (1) videooculography (video-based tracking using a head-mounted or remote visible light video camera) (Päivi, 2011; Duchowski, 2018), (2) infrared oculography (utilising infrared to recognise user pupils for tracking eye movements) (Jacob and Karn, 2003), and (3) electrooculography (where electrodes are placed to the right and left of the eyes to track movement) (Morimoto and Mimica, 2005; Majaranta and Bulling, 2014; Müller et al., 2016; Duchowski, 2018). Gaze interaction typically requires an eye-tracking device (such as an infrared sensor) which captures images of an individual's eyes when fixating on a point of interest. The cost of commercial eye-tracking devices has reduced significantly in the last decade as major technology companies, and the gaming industry has demonstrated increased interest in embedding eye-tracking within their products (Majaranta and Bulling, 2014; Duchowski, 2018). For example, commercial products from companies such as Tobii (2023) now retail for around £230 (e.g. in terms of the Tobii 5 (2023)). Sensors must be calibrated for each user to facilitate more accurate gaze interaction – this is commonly achieved through showing a number of calibration points on a screen and asking the user to consecutively fixate on these points (Jacob and Karn, 2003; Majaranta and Bulling, 2014; Duchowski, 2018).

The majority of commercial devices provide an average accuracy of 0.5° to 1° of visual angle, which relates to between 16-33px in size within a display utilising a screen resolution of 1280x1024px (Miniotas et al., 2006; Kumar et al., 2007). Interface targets below 1° visual angle are considered too small for comfortable gaze selection, thus presenting a significant limitation of gaze interaction (Miniotas et al., 2006; Skovsgaard et al., 2010; Creed et al., 2020). Increasing the size of interface objects can address this issue, although it can present constraints around the number of elements visible on the screen at any time (Bates and Istance, 2002; Jacob and Karn, 2003). Another issue is calibration drift, which refers to the accumulating deterioration in eye tracker accuracy after an initial calibration (often caused due changes in the position of users or sensors) (Kytö et al., 2018; Müller et al., 2019). The Midas touch issue a further key challenge associated with gaze interaction, where fixations on interface targets can activate unintentional selections (Jacob and Karn, 2003; Meena et al., 2017; Rajanna and Hammond, 2018). Whilst there are challenges associated with eye tracking, a wide range of research has been completed investigating the efficacy of this technology as an alternative or supplementary interaction approach.

2.4.2 Eye Gaze Interaction Research

Research on eye gaze interaction has received significant interest from researchers (Ware and Mikaelian, 1986; Jacob, 1990; Zhai et al., 1999; Sibert et al., 2001; Schiessl et al., 2003; Morimoto and Mimica, 2005). A key focus of this work has explored the feasibility of the technology to support people with physical impairments in controlling digital systems (Hutchinson et al., 1989; Borgestig et al., 2017; Creed et al., 2020). In particular, research has examined novel approaches for selecting small targets (Miniotas et al., 2006; Skovsgaard et al., 2010), investigating solutions for the Midas touch issue (Jacob and Karn, 2003; Shakil et al., 2019), and using gaze as part of multimodal interaction approaches (Van Der Kamp and Sundstedt, 2011; Beelders, 2011; Creed et al., 2020). Most of these research studies have been conducted in the context of eye typing (Majaranta et al., 2009; Kristensson and Vertanen, 2012; Mott et al., 2017), although other work has focused on domains such as design (Hornof and Cavender, 2005; Santella et al., 2006; Heikkilä, 2013; Lewien, 2021), web browsing (Porta and Ravelli, 2009; Kumar et al., 2017; Casarini et al., 2020), gaming (Dorr et al., 2007; Istance et al., 2009; Nacke et al., 2010) and object selection (Huckauf and Urbina, 2008; Lutteroth et al., 2015). Key methods for selecting targets such as dwell time, dwell-free, gaze gestures, and zooming are described in further detail below.

2.4.2.1 Dwell Time

The majority of research on eye gaze selection of interface elements has focused on dwell time approaches where users fixate on a desired interface target and wait for a specific amount of time before a selection is completed. Studies have highlighted that dwell time is a convenient and natural selection method as it does not require significant training and can prevent false selections (Jacob, 1990; Sibert et al., 2001; Majaranta and Räihä, 2007; Beelders, 2011). Typically, dwell times range from 400-1000 milliseconds (Jacob, 1990; Špakov and Miniotas, 2004), although studies have highlighted that long fixation times can be tiring for an individual's eyes, as well as making the interaction slower (Wobbrock et al., 2008; Majaranta et al., 2009). A number of approaches have been implemented to enhance the use of dwell time - for instance, Majaranta et al. (2009) explored an adjustable dwell time approach where users can vary dwell time duration based on their requirements. A 10-session longitudinal study was conducted with participants (who had little experience with gaze experience) where they were asked to type using gaze and were informed to adjust the dwell time if they felt comfortable. After using gaze typing for ten sessions, the dwell time decreased from an average of 876ms on the first session to 282ms in the last session – thus increasing the average typing speed from 6.9 WPM (Word Per Minute) to 16.2 WPM.

Mott et al. (2017) implemented a cascading dwell time approach which uses a language model to calculate the probability of the next letter and then autonomously adjust dwell time. This approach was evaluated with both disabled and non-disabled participants with results highlighting that the cascading technique was significantly faster than a static dwell time, enabling participants to achieve typing speeds up to 12.39 WPM. Pi et al. (2020) utilised a similar cascading approach based on the likelihood of the next selected key with results also presenting positive feedback. Similarly, Špakov and Miniotas (2004) implemented an algorithm to adjust the dwell time automatically based on exits time (i.e. the time interval between the moment a key was selected and the moment the gaze left the key). Results found considerable variation in exit times (300ms – 1100ms) – with the average error rate being 2.3% while the average typing speed was 12.1 WPM. Furthermore, studies have confirmed that the use of dynamically adjusted dwell times can speed up gaze typing and make key selection easier by predicting the probability of the next keys, thus maintaining typing rhythm (Mott et al., 2017; Majaranta et al., 2019; Pi et al., 2020).

2.4.2.2 Dwell-Free

Whilst dwell time is a common selection approach within the field, research studies have also explored methods to mitigate the delays caused by dwell selections. Studies have explored the potential of dwell-free approaches, where instead of fixating on a key for a predetermined amount of time, the user can briefly gaze at the target before moving to the next one. For instance, Kristensson and Vertanen (2012) examined the potential of a dwell-free approach where users can gaze in the proximity of the desired letters in their target word, and the system would then recognise the sequences of the word and would type the word automatically. This method was investigated to gauge the impact on typing speed, with results highlighting an entry rate of 46 WPM, indicating that dwell-free eye-typing may be more than twice as fast as the current state-of-the-art methods. In another approach, Sarcar et al. (2013) utilised a dwell-free interaction by moving the eye pointer through an interface element in an inside-outside-inside sequence. To select a target, users have to position their eye pointer at the target onscreen key, fixate their gaze away from the key, and finally return to the same key area. Results highlighted that the tool achieved an average of 15% higher text entry rates over existing dwell-time based approaches. Similarly, GazeTry implemented a swipe approach, which predicts possible words as users gaze through the letters of the word (Liu et al., 2015). Results from a user evaluation highlighted that the approach provided higher accuracy and resilience to text entry errors. EyeSwipe (Kurauchi et al., 2016) utilised a similar approach allowing users to type using a "reverse crossing mechanism" that involves initially selecting the first character of a word from an onscreen keyboard, then glancing in the approximate vicinity of the next characters until the final character is reached. Results from a user study highlighted that EyeSwipe produced significantly higher typing speeds than standard dwell-time based approaches.

Overall, studies have highlighted that dwell-free approaches have the potential to improve gaze typing through removing the need for dwell time (Kurauchi et al., 2016; Sarcar et al., 2013; Kristensson and Vertanen, 2012). However, despite having the potential to make interactions more efficient, dwell-free systems also include limitations and issues. For instance, dwell-free gaze typing can result in fatigue and users requiring numerous breaks when working on tasks (Mott et al., 2017). As the user will gaze briefly at the intended key, there is a high chance of fixation landing on the neighbour key, thus reducing the accuracy (Majaranta et al., 2019).

2.4.2.3 Gaze Gestures

Researchers have explored the concept of gaze gestures as an alternative method to reduce unintentional selections and overcome the limited accuracy associated with eye trackers (Hyrskykari et al., 2012). Gaze gestures are sequences of eye movements (typically strokes) used to initiate commands in a sequential order (Drewes and Schmidt, 2007; Vickers, 2011). Initial gaze gesture work was conducted by Drewes and Schmidt (2007), who evaluated gaze movement around a dialog window in a clockwise direction to respond "yes" and in an anti-clockwise direction for "no". A key benefit of this approach is that no calibration is required as users initiate commands through making a sequence of "eye strokes". Porta and Ravelli (2009) implemented a gaze gesturing approach for selecting buttons and hyperlinks within a browser where users first look at the hyperlink target, move their gaze in an upward direction, and then look back again at the target to trigger the selection. Results from a user study found that most participants could successfully complete common selection tasks using this type of approach. Wobbrock et al. (2008) utilised alphabetic gaze gestures to support text entry through their EyeWrite interface, where four corners and the centre of a small square are used for entering characters. Users can write characters though fixating their eyes in a sequential order around the corners of the window - for example, to type the letter “t”, the user has to move the gaze point from the top left corner to the top right corner, and then to the bottom right corner. Results found that this approach was perceived faster, less error-prone and resulted in less eye fatigue than on-screen QWERTY keyboard approaches utilizing a dwell time method for character selections. Møllenbach et al. (2010) used single gaze gesture in which the user initially looks at a target and moves their gaze away to perform the selection. Similarly, a range of studies have investigated the combination of eye movement and blinks to provide people with impairments with an effective means to communicate and control their environment (Park and Lee, 1996; Norris and Wilson, 1997; Borgestig et al., 2017). User evaluations of this type of approach have highlighted that blink interaction could offer a feasible and efficient approach in interaction, although issues such as fatigue due to prolonged use, accidental/unconscious blinks, and accuracy challenges have been identified (Park and Lee, 1996; Ohno et al., 2003; Chakraborty et al., 2019).

Whilst the studies highlighted have demonstrated that gestures can support enhanced performance over dwell selections in some scenarios (Hyrskykari et al., 2012; Lutteroth et al., 2015), they also possess some limitations. In particular, gaze gestures can be very slow and cause eye fatigue when used for prolonged periods (Porta and Ravelli, 2009), as well as adding cognitive load in terms of recalling required gesture movements (Heikkilä and Räihä, 2010; Penkar, 2014).

2.4.2.4 Zooming

The use of interface zooming is an additional alternative approach that has been explored for supporting target selection via eye gaze. Zooming techniques enable users to adjust the size of the area being viewed to see more or less detail (Skovsgaard et al., 2010; Vickers, 2011). Early work on eye controlled zooming interaction typically utilised a two-step approach – where the target area magnifies on dwelling, and users can use a mouse to select targets (Lankford, 2000; Bates and Istance, 2002). For example, the ERICA system allows users to dwell on a target region to magnify the size, select a target with a second dwell, and then utilise a mouse to perform actions such as a left or right click (Lankford, 2000). A similar approach was by Bates and Istance (2002), where the whole screen magnifies by dwelling on the area of interest, and users can then select a target using a mouse. The authors conducted a comparative study between the head mouse, a standard eye mouse, and the developed zoom eye mouse with results highlighting that the zoom eye mouse was easier to use and reduced cognitive workload. Skovsgaard et al. (2010) used different zooming levels to select interface objects, including a two-step zoom method and a continuous zoom approach (in which the area surrounding the target would continue to zoom as long as the user fixates their gaze, thus enabling a user to select the target). Results from a user evaluation found the continuous zooming approach to be more accurate and faster than the two-step method. Similarly, Ashmore et al. (2005) utilised a fisheye lens and a video-based eye tracker to locally magnify the display at the point of the user’s gaze without losing the content outside the magnified region. A user evaluation found that the fisheye lens approach performed better in terms of efficiency and usability than a traditional zoom-based approach. In another approach, Bubble Cursor (Grossman and Balakrishnan, 2005) dynamically resizes the activation area (where a user is fixating their gaze) so that only one target is selectable at any time. Results demonstrated that the bubble cursor outperformed other pointing techniques in 1D and 2D target acquisition tasks. Whilst different zooming approaches can present some enhancements over other methods (in some scenarios), studies have also highlighted a range of interaction issues. For example, zooming over targets can lead to visual distortions, such as the content around the periphery of a zoomed area becoming obscured when the zoom level increases (Lankford, 2000). Similarly, the zoomed-in portion of the screen may move disproportionately to other content, thus making it challenging to read text or view objects (Bates and Istance, 2002; Skovsgaard et al., 2010).

In addition to the aforementioned work, studies have also explored the potential of multimodal approaches (combining gaze with another input modality) for performing selections - for instance, Creed et al. (2020) utilised mechanical switches instead of dwell time as a selection method, which was perceived positively by participants. Similarly, Eyepoint (Kumar et al., 2007) used a keyboard to complete selections of targets that users are currently fixating on, while PressTapFlick (Rajanna et al., 2022) adopted foot input for selection. Studies have also utilised webcams for eye tracking to facilitate the triggering of mouse actions – right/left eye wink to trigger left/right clicks and squinting eyes for activating scrolling (Papoutsaki et al., 2017; Meena et al., 2020; Anantha Prabha et al., 2022).

2.4.3 Eye Gaze Interaction for Coding

Coding via eye gaze input was first investigated in 1990 when Crosby and Stelovsky (1990) explored code comprehension techniques using an eye tracker. However, this type of approach was not widely adopted in the programming field until 2006 due to the high costs of the technology and issues associated with tracker accuracy (Bednarik and Tukiainen, 2006; Sharafi et al., 2015). In the coding environment, eye tracking studies to date have focused on two key areas: (1) studying the behaviour and working practice of programmers (including visual analysis of code reading patterns, code debugging, traceability and collaborative programming) (Crosby and Stelovsky, 1990; Busjahn et al., 2011; Shaffer et al., 2015; Clark and Sharif, 2017; Begel and Vrzakova, 2018) and (2) enhancing the functionality of IDEs via the use of gaze as a supplementary input tool (Glücker et al., 2014; Shakil et al., 2019; Santos, 2021).

The first approach has attempted to understand the psychology of programming by exploring how eye movement can inform methods used for code comprehension, debugging, collaborative interaction, and traceability (Begel and Vrzakova, 2018; Sharafi et al., 2020). For example, Busjahn et al. (2015) conducted a comparative study to compare the eye movements of novice and expert programmers in relation to identifying the reading patterns of natural language text and source. Results found that novice and expert programmers read code less linearly than natural language text, as they tend to skip words and lines when reading code. Similarly, Jessup et al. (2021) compared the number of fixations and fixation durations during code comprehension processes between novice and expert programmers - where they were asked to read two pieces of code, rate the code on various attributes, and describe the functionality of code. The results highlighted that experts and novices significantly differ in their fixation counts made during the task - with experts having more fixations than a novice, indicating that they focused more and used more cognitive skills. Other studies have also explored the potential of eye tracking within program comprehension scenarios to understand how developers read code and the effort required to comprehend syntax. (Aschwanden and Crosby, 2006; Bednarik and Tukiainen, 2006; Busjahn et al., 2011; Mondal et al., 2022).

Furthermore, research has investigated eye tracking in relation to code debugging – for instance, Lin et al. (2016) examined the eye movements of expert and novice programmers to better understand cognitive processes associated with debugging activities. Results highlighted that experts traced the program more logically, whereas novice coders tended to stick to a line-by-line sequence and often jumped to random lines to identify issues. Similarly, D'Angelo and Begel (2017) investigated collaborative coding processes in remote pair programming exercises where a novel gaze visualisation was designed to display where in the code their partner is currently looking. A user evaluation highlighted that pairs spent a significant portion of their time concurrently looking at the exact code locations when the visualisation graph was turned on – suggesting that they understood what their partner was referring to. Prior literature has also investigated the use of an eye tracker in traceability to keep track of syntax changes – for instance, Shaffer et al. (2015) developed an open-source Eclipse IDE-based plugin, iTrace, which captures developers' eye movements while working on change tasks by automatically mapping gaze movements to source code elements such as if statements and methods.

Another approach involves using an eye tracker as an alternative approach to support code navigation. For instance, Shakil et al. (2019) developed a code navigation prototype (CodeGazer) that allows users to perform navigation actions such as "Go to Definition" and "Find all Usages" using gaze interaction. To make a selection, the user must dwell at the corresponding identifiers (e.g. classname, function name) in the syntax, which are highlighted in different colours, and to confirm the selection, they must again dwell at the corresponding coloured button on the left and right sides of the screen. Whilst this approach can support the selection of small targets within a coding context (e.g. identifiers such as variable and function names), it also adds an additional step to the interaction process as the users have to perform dwell selections twice. Results from a comparative study found the gaze-based code navigation approach to be comparable with a keyboard, although slower than mouse-based navigation.

Similarly, EyeNav (Radevski et al., 2016) utilised a combination of gaze interaction and keyboard shortcuts to support code navigation functions such as left and right mouse clicks for selecting code and horizontal and vertical page scrolling. For example, users use the ALT+F keyboard shortcut to execute the page scrolling feature and then look in a specific direction (up or down). Whilst EyeNav is available as a plugin in Brackets (2023), it has not been formally evaluated within the literature to date. Another gaze based tool is EyeDe (Glücker et al., 2014) which provides code navigation features such as switching between files, navigating different methods, reviewing code documentation and providing a minimap view for scrolling to syntax positions. The tool required users to gaze on the target code for a dwell time duration to trigger a context menu, then gaze on the context menu icon to confirm the selection, which increased selection time and slowed the interaction, however after a pilot study, the authors implemented a combination of a keyboard and dwell time for selection. Whilst the system was perceived positively by participants overall, they highlighted issues around the selection of small targets and the accuracy of the tracking device. Additionally, Santos (2021) developed a code editor where the cursor can be controlled via gaze input. The editor allowed users to dwell on their desired location and quickly jump to that position by pressing a key on either the mouse or keyboard. No user evaluation was conducted, so how developers with physical impairments might perceive this approach remains unclear.

In summary, while existing literature has focused on utilising eye gaze interaction for navigation, debugging and traceability of code, no prior work has been conducted to date investigating the feasibility of eye gaze for writing or editing code. Further research is therefore required to understand how gaze interaction can best be utilised to support these essential coding activities. Furthermore, prior gaze-based coding tools have only been evaluated with non-disabled developers, so additional work is also required to explore gaze as an interaction approach with developers who have physical impairments.

2.5 Multimodal Speech and Gaze Interaction

Research on multimodal interface design (which involves processing of two or more modalities such as speech, pen, touch, gaze) can be traced back to the 1980s where the focus was on developing new approaches that support flexible and efficient interaction methods (Bolt, 1980; Oviatt, 1997; Ruiz et al., 2009; Seizov and Wildfeuer, 2017; Oviatt, 2017; Oviatt and Cohen, 2022). Studies on multimodality have examined how multimodal interface design can be informed through cognitive theories such as Gestalt theory, Working Memory theory, and Cognitive Load theory to help users minimise cognitive load and hence improve performance (Dumas et al., 2009; Kopp and Bergmann, 2017; Oviatt, 2017). Gestalt theory emphasise perceiving and interpreting information as a whole rather than focusing on isolated components with a focus on highlighting how people perceive visual and spatial elements, extending to the comprehension of sounds, touch, and other types of modalities (Oviatt et al., 2003; Trujillo and Holler, 2023). Similarly, Cognitive Load theory pertains to the regulation of mental exertion or "load" placed on an individual's working memory while engaging with different modalities, such as speech, gestures, or visuals to enhance experiences through reducing cognitive burden, enabling users to efficiently process information (Sweller et al., 1990; Dumas et al., 2009). Furthermore, Working Memory theory focuses on improving users' processing and retention abilities – for example, the combination of visual and auditory inputs can make it easier for users to understand and remember content,(e.g., when compared with reading), thus reducing cognitive load on working memory (Baddeley, 1992). Oviatt et al. (2000) believes that the successful development of multimodal systems requires tackling important research challenges such as creating cognitive theories to guide system design and improve natural language processing, dialogue processing, and error-handling techniques. However, whilst there has been substantial work around multimodal interactions and experiences, there remains a lack of work in relation to improving tools for supporting multimodal applications and interfaces so they can become mainstream, especially since combining multimodal inputs (e.g., gaze, speech, and switches) across a range of domains is seen as a promising way to achieve universal access (Dumas et al., 2009).

2.5.1 Multimodal Speech and Gaze Rationale

Previous research has identified that unimodal interaction approaches can possess limitations and issues – for instance, gaze interaction can present challenges associated with accuracy (Miniotas et al., 2006; Beelders and Blignaut, 2011), selection of small targets (Skovsgaard et al., 2010; Creed et al., 2020), the Midas Touch (Van Der Kamp and Sundstedt, 2011; Rajanna et al., 2022), and calibration errors (Heikkilä, 2013). Similarly, speech interaction also has a range of limitations in relation to accuracy (Rozado et al., 2016) and misrecognition of vocal input (Wagner et al., 2012; Rosenblatt et al., 2018). To address these challenges, researchers have investigated the potential of combining both input approaches to enhance interaction experiences and address the limitations of each modality (Miniotas et al., 2006; Elepfandt and Grund, 2012). Additionally, this type of approach can provide a “fall-back option” - for example, if an input method is not functioning correctly, the user can continue working with the alternative modality or as a unimodal interaction approach where the user has the option to fluidly choose between input methods (Beelders and Blignaut, 2010; Sengupta et al., 2018). Additionally, studies have highlighted that implementing multimodal approaches can require significantly less cognitive load and present usability benefits over unimodal approaches (Kaiser et al., 2003; Zimmerer et al., 2022).

2.5.2 Multimodal Speech and Gaze Research

Studies have also explored various multimodal approaches associated with both gaze and speech for interaction purposes. For instance, researchers have explored combining gaze input with a mouse (Lankford, 2000; Bates and Istance, 2002; Hansen et al., 2003), keyboard (Wang et al., 2001; Kumar et al., 2007), gestures (Ghosh Sanjay et al., 2013; Zhang et al., 2020), mechanical switches (Creed et al., 2020), and touch (Khamis et al., 2016; Pfeuffer and Gellersen, 2016). Similarly, research studies have explored the use of speech with gestures (Bolt, 1980; Wang, 1995; Oviatt, 1997; Cohen et al., 1997; Ghosh et al., 2020; Kang et al., 2019), touch (Srinivasan et al., 2019), and mechanical switches (Aziz et al., 2021, 2022).

Research has also explored the direct combination of speech and gaze modalities in different domains – for instance, Wang (1995) proposed an application integrating eye gaze, voice and gestures for manipulating 2D and 3D objects within a window-based environment. Miniotas et al. (2006) developed a gaze and speech-based multimodal pointing technique for selecting small, closely spaced targets. The study found that the combination of speech and gaze interaction outperformed gaze-only interaction. Studies have also explored multimodal interaction techniques to support creative design – for instance, Van Der Kamp and Sundstedt (2011) combined gaze and speech to produce graphical work. The application used gaze to control the cursor while speech input triggered actions such as drawing and opening the colour palette. Whilst the results validated the combination of multimodal speech and gaze interaction for an efficient drawing process and was perceived positively by participants, several issues such as speech misrecognition and gaze accuracy were reported. Sengupta et al. (2018) developed a multimodal browser that can be operated using gaze and voice interaction where users can combine input methods to work together or utilise one modality based on their preferences and context. The authors highlighted that multimodal interaction helped overcome accuracy limitations and ambiguity issues associated with the individual use of gaze and speech modalities. A pilot test with five users found that the multimodal approach was more efficient and accurate across a range of tasks when compared with a single modality.

Eye gaze and speech have also been combined to form a multimodal system for controlling desktop environments - for instance, Rozado et al. (2016) utilised gaze to dwell on a target position within a desktop environment and used voice to trigger actions such as a left mouse click. Whilst the authors highlighted that the combination of both input methods could outperform the unimodal approach (i.e. speech only and gaze only interaction), no evaluation was conducted, so it remains unclear how the participants would have perceived the approach. A similar approach was also proposed by Castellina et al. (2008), who highlighted that using speech can remove the requirement for dwell time as a selection approach, thus supporting more efficient interactions. Beelders and Blignaut (2011) incorporated speech and gaze as additional input modalities within a word processor to provide alternative interaction opportunities for controlling the application. Users were provided with the freedom to select either the combination of gaze and speech or a single modality to work based on their suitability. Speech was used for text dictation and providing commands to control the interface (e.g. formatting text, cursor control), while various eye tracking options such as blinking, user-defined dwell time, and look-and-shoot approach (using the keyboard’s Enter key) were used to trigger selections. Results from a user evaluation confirmed that a combination of speech and eye gaze significantly improved the accessibility of the word processor.

Speech and gaze have also been utilised for game control – for instance, in the game “The Revenge of Killer Penguins” (Wilcox et al., 2008), users select objects within the game by using either voice commands such as “look” and “pickup” or gaze selection approaches such as blinking and winking. No user evaluation was conducted, so it is unclear how participants would have perceived this approach. Uludağli and Acartürk (2018) explored a multimodal gaming approach combining gaze and voice inputs – where users are able to move from one position to another position through gaze while using voice commands to trigger the actions on the objects. Results from a user study highlighted that combining the two modalities can be an acceptable interface control method for mobile devices.

The use of multimodal interaction approaches to support developers in performing coding activities has received little attention to date. Delimarschi et al. (2014) explored the combination of gesture and voice for interacting with an existing coding IDE interface. Their application KinectCommander used speech commands to trigger actions such as “find and replace”, “add new files”, and “activate menus”, whereas gestures were used for moving and resizing specific sub-windows and scrolling through code within the Microsoft Visual Studio IDE (2023). Whilst the system has the potential to minimise the use of a keyboard and mouse, the authors highlighted accuracy and usability issues with the system. For example, it typically took multiple attempts for users to complete gesture commands, while it took an average of two attempts to successfully trigger a voice command.

2.6 Summary

This chapter presented and discussed disability and common barriers encountered by people with physical impairments, as well as some of the common assistive tools that are utilised to make interactions with digital systems more accessible. Voice input and gaze interaction were highlighted as two technologies that hold significant potential to support people with physical impairments due to increased availability, cost reductions, and increased efficacy of the technologies. Furthermore, it was highlighted that previous work has identified how combining these technologies in a multimodal system can provide potential interaction benefits (e.g. the strengths of each modality can address the limitations of the corresponding input approach). A particular emphasis of the review was placed on research investigating the potential of these technologies to support developers with physical impairments in writing, editing, navigating and managing code. In particular, whilst previous studies have explored the combination of voice and gaze in non-programming fields (Van Der Kamp and Sundstedt, 2011; Beelders and Blignaut, 2011; Elepfandt and Grund, 2012; Sengupta et al., 2018), it remains unclear what the optimal complimentary roles are for each input method when utilised to support coding activities. Moreover, the use of additional controls (e.g. mechanical switches commonly used by people with physical impairments) could also further support a combined gaze and speech approach, although we have little understanding around the feasibility of a system integrating multiple methods of input in this context. Further work is therefore required to investigate to what extent gaze, speech and other input tools can be combined to make coding accessible for people with physical impairments.

Whilst studies have explored voice input to support developers with physical impairments with coding activities, there remain several areas where less or no work has been conducted to date – for instance, no comparative studies have been conducted to measure the efficacy and usability of fixed grammar and natural language coding approaches. Similarly, no evaluations using natural language approaches have been conducted with developers with physical impairments, so it remains unclear whether these coding methods are viable for this target audience. Similarly, little work has been focused on implementing code navigation features such as call graphs to navigate code blocks or scrolling to a different section of the codebase using multimodal approaches. This represents a significant gap in the literature where further work is required to understand the potential impact of these interaction approaches in this context. To address the lack of research around different combinations of gaze, speech, and other input approaches to support coding, the following three chapters present a series of studies focused on multimodal interaction approaches for supporting developers with physical impairments.

3 MULTIMODAL CODING VIA SPEECH, GAZE AND SWITCHES

This work has been published in the proceedings of the 2020 ACM Designing Interactive Systems Conference (DIS '20') as "Voiceye: A Multimodal Inclusive Development Environment" (Paudyal et al., 2020)

3.1 Introduction

This chapter presents an investigation into the use of a multimodal prototype ("Voiceye"), enabling developers with physical impairments to write and edit code using a combination of speech, gaze, and mechanical switches. Whilst some initial work has investigated the use of speech as an interaction paradigm to support writing code (Begel and Graham, 2005; Wagner and Gray, 2015; Maloku and Pllana, 2016; Rosenblatt et al., 2018), studies have highlighted that this method of input presents known challenges around voice recognition accuracy (Oviatt and VanGent, 1996; Suhm et al., 2001). Similarly, gaze interaction has been utilised to support different coding activities (such as code navigation (Radevski et al., 2016; Shakil et al., 2019; Santos, 2021)), although studies have also highlighted that this technology exhibits inherent limitations such as issues around the selection of small targets (Miniotas et al., 2006; Skovsgaard et al., 2010), slow typing speeds (Majaranta and Räihä, 2002; Bafna, 2018), and well-known challenges associated with the Midas touch issue (Jacob and Karn, 2003; Glücker et al., 2014; Meena et al., 2017).

Research has started to explore how speech and gaze can be combined for making interactions more accessible for people with physical impairments in applications such as web browsing (Sengupta et al., 2018), operating desktop interfaces (Pireddu, 2007; Rozado et al., 2016), and word processors (Beelders, 2011), although there has been a lack of work to date exploring this approach to support coding activities. It also remains unclear the extent to which the use of additional input modalities (e.g. mechanical switches) can potentially present more inclusive and accessible interaction methods for disabled developers. To address this gap in the literature, this chapter initially highlights the design rationale behind a new multimodal prototype (Voiceye), along with details around a user evaluation with non-disabled developers investigating the feasibility of the prototype to support development tasks. Iterative updates are then highlighted based on feedback from participants, with a further evaluation then detailed involving disabled developers to better understand their perceptions and performance when utilising the Voiceye prototype for coding. This chapter therefore presents three key contributions:

(1) a novel multimodal coding environment enabling people with physical impairments to write and edit code.

(2) a user study demonstrating the usability of the system and new insights into this form of multimodal interaction.

(3) validation of this approach through a further evaluation demonstrating that people with physical impairments can effectively write code.

3.2 Prototype Design

To address the lack of research exploring the potential of multimodal interaction approaches to facilitate coding activities for disabled users, a code editor (Voiceye) was developed that can be operated using the combination of gaze, speech input, and mechanical switches. Voiceye is a desktop-based application developed using ElectronJS (2023) that enables writing and editing HTML/CSS code via gaze interaction, whilst voice input is used for dictating long text (e.g. comments) and performing commands such as selecting, deleting, and navigating code. It is built on top of CodeMirror (2023) – an open-source JavaScript-based code editor which provides standard features such as syntax highlighting, search, replace, and code indentation.

Voiceye consists of four components: an automatic speech recogniser (using Microsoft's Azure Speech service (Microsoft, 2023)), a rule-based syntax grammar, an onscreen keyboard (controlled via gaze input), and two mechanical switches – one for performing selections on buttons within the interface (e.g. a virtual keyboard key) and the other for toggling activation of the voice recognition system (Figure 3.1). As the user speaks, the prototype converts speech to text and performs a check against the rule-based syntax grammar listed in Table 3.1. Appropriate actions are then triggered based on the spoken command. The onscreen keyboard allows users to type code via fixating on characters of interest and then completing the selection by pressing one of the designated mechanical switches.

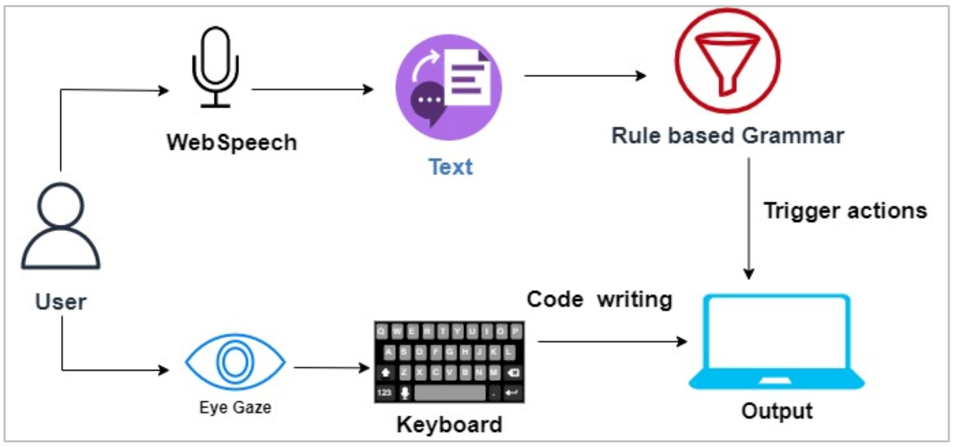
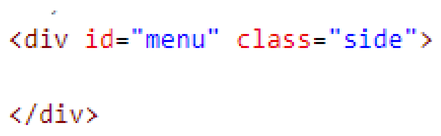


Figure 3.1: A system diagram of the Voiceye application

The main interface was informed through the design of existing mainstream development environments (e.g. Visual Studio Code (2023), Atom (2023), Brackets (2023)). These applications typically display line numbers by default on the left side of the interface with the code included in the main interface area (with features such as syntax formatting enabled). A similar interface layout was adopted, along with a theme similar to the default one used in Visual Studio Code (Figure 3.2). The QWERTY keyboard layout is the most common approach used in eye typing studies (Kristensson and Vertanen, 2012; Mott et al., 2017). This layout was therefore integrated into the application with keys that are 110x110px in size to support more comfortable selection via gaze. The keyboard included standard delete, backspace, space, enter, shift/capslock, and tab keys. A “char” key was also provided (“123?”) to toggle between letters, numbers, and special characters. Arrow keys are included that allow users to navigate through single characters of text (speech provides navigation around lines and words). Early usability testing found that placing the keyboard at the bottom border of the screen resulted in some keys being problematic to select via gaze – 100px of padding was therefore applied to the bottom of the keyboard to make all keys easier to select.

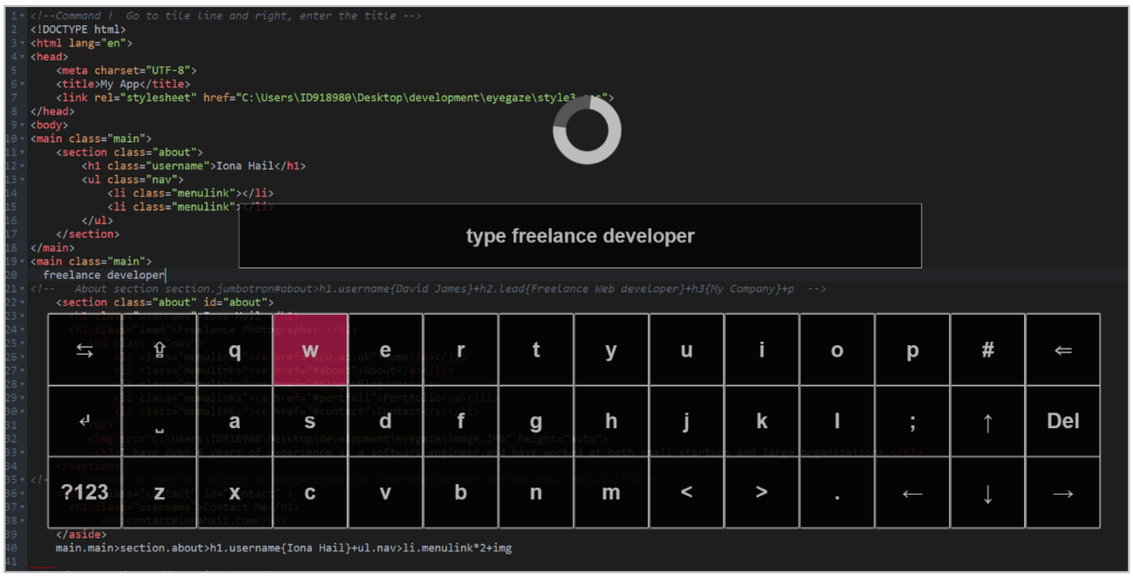
To address issues of slow typing speeds via gaze, Emmet (2023) was integrated into the system allowing users to write HTML and CSS code using a shorthand notation that is then expanded to full syntax. For example, users can type "*div#menu.side*" to generate the following code:



```
<div id="menu" class="side">

</div>
```

A number of approaches were also considered for performing selections (i.e. dwell time, speech commands, gaze gestures), but a decision was made to opt for mechanical switches commonly used by people with physical impairments (Lancioni et al., 2008). There has been a lack of studies investigating the use of switches as a selection approach in a multimodal context, so this approach presents an opportunity to explore users' perceptions of this type of interaction method.

Speech input can be activated using the designated switch – upon selection, a semi-transparent text area is displayed above the keyboard, along with an animated spinner. When the user speaks, the text recognised by the system is displayed to the user to provide instant feedback. An animated spinner icon also provides a visual cue that the system is currently processing speech input. These interface elements were added after initial usability testing on an earlier version, where users expressed frustration at not knowing if the system had detected their input correctly. This typically resulted in a delay in seeing if the system performed the correct action or whether the user had to repeat the command.



```
<!--Command !  Go to tile line and right, enter the title -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My App</title>
    <link rel="stylesheet" href="C:\Users\ID918980\Desktop\development\eyegaze\style3">
</head>
<body>
<main class="main">
    <section class="about">
        <h1 class="username">Iona Hail</h1>
        <ul class="nav">
            <li class="menulink"></li>
            <li class="menulink"></li>
        </ul>
    </section>
</main>
<main class="main">
  freelance developer
<!--   About section section.jumbotron#about>h1.username{David James}+h2.lead{Freelance Web developer}+h3{My Company}+p  -->
    <section class="about" id="about">
    </aside>
    main.main>section.about>h1.username{Iona Hail}+ul.nav>li.menulink*2+img
```

type freelance developer

Figure 3.2: Screenshot of the Voiceye interface when the speech recogniser has been activated. The spinner indicates that the recogniser is listening for speech input. The user has issued the text “type freelance developer” in this example. The user’s gaze also hovers over the “w” key (resulting in visual feedback through the red background).

The choice of vocal commands (Table 3.1) were informed through Rosenblatt et al. (2018)’s Wizard of Oz (WOz) study with coders and their final choice of commands integrated into the VocalIDE system. The WOz was conducted with ten non-disabled participants to gain an understanding of the commonly used vocal commands when writing, editing, selecting and deleting code. During this WOz study, participants were provided with two blocks of code - one containing errors and another without any errors. They were asked to provide verbal instructions to correct each syntax error to a researcher, who operated a text editor according to a predetermined protocol. Additional commands were also included, such as “open/close keyboard” (to trigger the keyboard), “clear” (to clear any highlighted words), and “undo/redo” (to redo/undo the previous command).

Tasks	**Commands**	**Description**	**Example Code / Utterances**	**Example Output**
Code Entry	"Type"	Entering new text.	<p>■</p> *Speech*: "Type welcome"	<p>welcome■</p>
Navigation	"Go to {line number}"	Navigating between the lines.	*Speech*: "Go to 17"	Cursor placed at the start of line 17.
	"Left/Right"	Navigate one position left/right.	<main■class="main"> *Speech*: "left"	<■main class="main">
	* "Left/Right {number}"	To navigate x positions left/right.	padding: 4px ■ 2px 3px 4px; *Speech*: "Left 2"	■padding: 4px 2px 3px 4px;
	"Up/Down"	Navigate one line up/down.	8. <div class="main"> 9. ■<h1>User</h1> *Speech*: "Up"	8. ■<div class="main"> 9.<h1>User</h1>
	"End of Line"	Navigate to the end of the current line.	■<h1>User</h1> *Speech*: "End of line"	<h1>User</h1>■
Selection	"Select"	Select one /multiple elements of code.	padding: 4px■2px 3px 4px; *Speech*: "Select Select"	padding: 4px 2px 3px ■ 4px;
	"Select {line number}"	Select a single line.	8. ■<nav> 9. <div>header </div> 10. </nav> *Speech*: "Select 9"	8. <nav> 9. <div>header </div>■ 10. </nav>
	"Select {line number} to {line number}"	Select multiple lines.	12. ■<div> 13. … 20. </div> *Speech*: "Select 12 to 20"	12. <div> 13. … 20. </div>■
	"Select {property}"	Select attributes and values.	<img■src="abc.png" height="20px"> *Speech*: "Select property"	<img src="abc.png"■ height="20px">
Deletion	"Delete line {line number}"	Delete a single line.	12. ■<nav> 13. <div>header</div> 14. </nav> *Speech*: "Delete line 13"	12. <nav> 13. ■</nav>
	"Delete line {line number} to {line number}"	Delete multiple lines.	11. <section> 12. ■<div> 13. … 20. </div> 21. </section> *Speech*:"Delete line 12 to 20"	11. <section> 12. ■</section>
	"Delete selected"	Delete highlighted code.	padding:4px 2px 3px 4px; *Speech*: "Delete selected"	padding: 4px 2px 3px■;
Add Comment	"Comment"	Add single line comment.	■padding: 4px 2px 3px 4px; *Speech* "comment"	/*padding: 4px 2px 3px 4px;*/ ■

Table 3.1: List of main vocal commands used in Voiceye. The "Vocal Commands" column contains key words/terms that users actually speak. Underlined text in "Example Code / Utterances" column includes example code to demonstrate how each speech command functions. The action performed through each speech command is visualized in the "Example Output" column. ■ denotes the position of the cursor. * denotes new commands added after the first user evaluation.

3.3 User Study 1

To explore the feasibility and usability of the Voiceye prototype an initial evaluation was conducted with non-disabled developers. A key requirement for using the application was that all users could utilise a combination of speech, gaze, and switches for controlling the interface (regardless of whether they have a physical impairment). It was therefore felt that a first study with non-disabled participants would provide an important and relevant insight into using this multimodal approach for coding purposes. It also provided an opportunity for identifying areas where future improvements could be made through iterative development work (prior to conducting evaluations with developers who have physical impairments).

3.3.1 Research Question

The research question below was developed to address the limited work completed around the potential of multimodal interaction to support development tasks:

RQ: How can multimodal approaches (utilising speech, gaze, and mechanical switches) support people with physical impairment in writing, editing, navigating and selecting code?

The broad focus of this question facilitated an exploratory research approach to help develop a deeper understanding of the challenges and opportunities associated with this method of interaction within a coding context.

3.3.2 Participants

29 participants (two female) were recruited from a population of university students and social media platforms with ages ranging from 19 to 45 years (M=27.9; SD=7.87). 13 participants wore corrective lenses (10 glasses). Participants were asked to self-assess their web development skills, as well as experience in using alternative input methods for interaction with computers. Eight participants had prior experience using an eye tracker, while 20 participants reported prior experience with using a speech recognition system. Nine were native English speakers, five were native Urdu speakers, and three participants' native language was Bengali. Similarly, other native languages included Nepali, Malay, Romanian,

French, Farsi and Polish. 25 participants had experience with web development including HTML and CSS coding with an average of 1.7 years of previous experience (SD=4.02).

3.3.3 Apparatus

Voiceye was installed on a Windows laptop (Intel® Core (TM) i3-7100U CUP and 8GB RAM) with an external 23-inch LCD monitor of 1920x1080px resolution. The Eye Tribe device (Eye Tribe, 2023) was used to support gaze tracking, providing an average accuracy of 0.5 to 1° of visual angle and an operating range between 45-75cm. The Eye Tribe was placed in front of the monitor on a tripod at a distance of approximately 60cm away from the participant's eyes. The laptop's built-in microphone was used for capturing the speech input. Two 65mm Jelly Bean switches were placed in front of the monitor and eye tracker (Figure 3.3) – the one on the left was used to trigger keyboard selections, and the one on the right for activating the speech recognition system.

3.3.4 Procedure

Pre-Test: Institutional Review Board (IRB) approval was obtained for the study. Participants were provided with an information sheet containing details about the study and asked to provide informed consent. They were also asked to complete a pre-test questionnaire (Appendix A.1) to collect demographic information, as well as details on their level of experience with software development and the use of alternative input methods. Participants were given a demonstration of the prototype and encouraged to ask any clarifying questions. A nine-point calibration process was then performed using the Eye Tribe sensor. After successful calibration, participants were asked to practice with the application for 5-10 minutes to ensure they could comfortably control the interface. During the practice session, they were asked to write HTML and CSS code, navigate to different line numbers, select and delete code, and edit syntax errors.

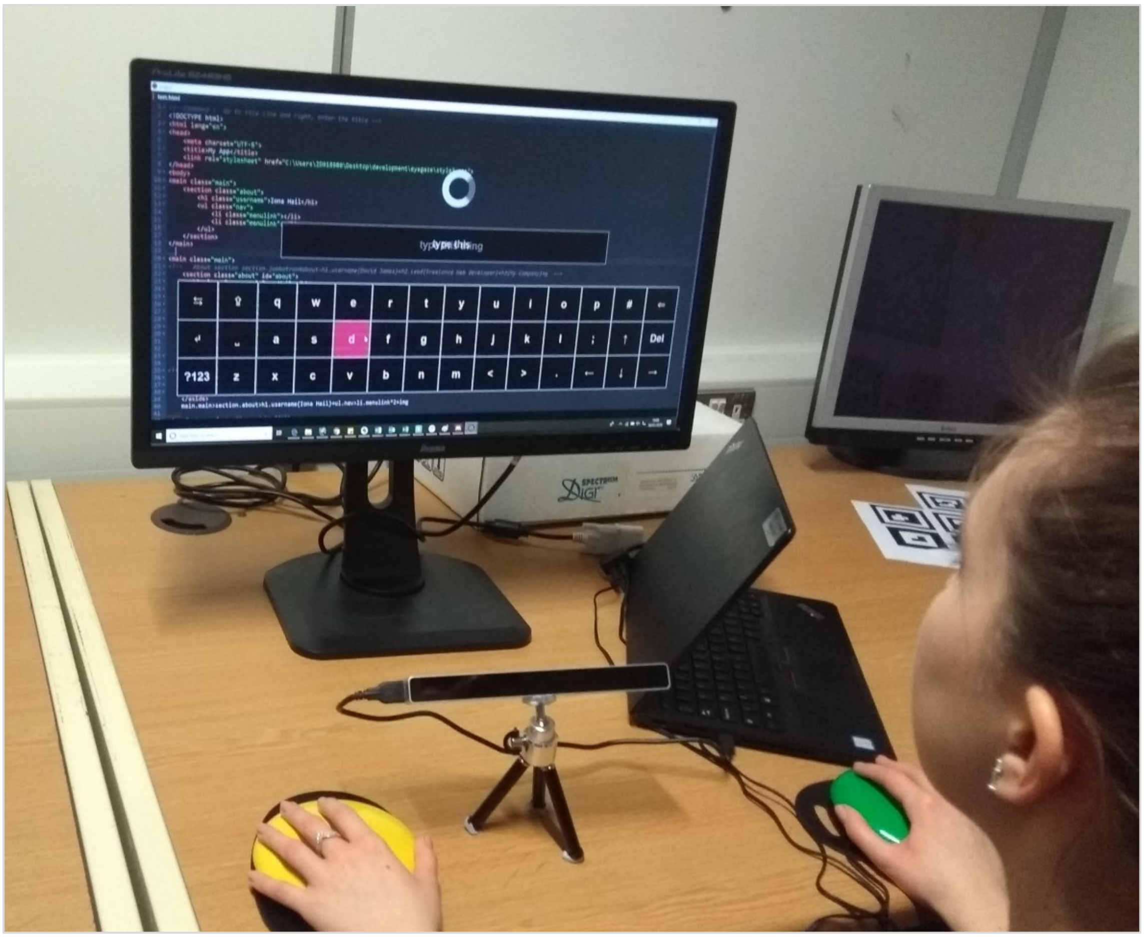


Figure 3.3: A participant performing the usability evaluation.

Main Test: After the practice session, participants started working through the main tasks. The tasks chosen were designed and categorised based on the development scenarios utilised by Rosenblatt et al. (2018). The main task categories included Adding Code [ADD], Selecting Code [SELECT], Deleting Code [DELETE], and Editing Code [EDIT]. Each category consisted of 16 tasks of which 8 were related to HTML and the remaining 8 were focused around CSS (64 in total) (Appendix A.2). The EDIT tasks were informed by the highest occurring HTML/CSS syntax errors made by computing majors (Park et al., 2015). The ADD tasks for HTML included adding a new element (e.g. <h1></h1>), a new element with single or multiple properties (e.g. <div id="articles"></div>), creating elements with child elements, writing HTML manually without using Emmet, adding some paragraph text (e.g. <p>Welcome to my Portfolio</p>), and creating a freeform comment. ADD tasks for CSS included adding new empty styles for classes (e.g. ".div {}"), styles with single and multiple properties (e.g. "color: #000; background: #FFF"), styles targeting children of an element (e.g. ".div>span"), writing CSS manually without Emmet, and standard freeform comments. The SELECT tasks for both HTML and CSS involved selecting elements/styles, lines, properties of elements/styles, blocks of code (i.e. spanning multiple lines), and specific words within a block of text. DELETE tasks required removing elements/styles, lines, element/style properties, a block of code, specific words within a block of text, and a comment. Finally, EDIT tasks included fixing typographical mistakes, addressing unclosed element pairs, correcting comment syntax, and fixing misidentified or "confused" constructs (e.g. declaring <h1> instead of <title> within a header). A list of all tasks is provided in Appendix A.2.

Participants were initially shown the "starting" code on paper that would be seen upon starting the task. The final completed code snippet was also shown to participants (on paper) in order to help them clearly understand what code or updates needed to be made. The researcher would outline the actions required to complete the task – voice commands and Emmet code related to the task categories were also provided to participants. Once they made the necessary updates and verbally stated that they had completed the task, the researcher used a keyboard shortcut to move onto the next task. Once all the tasks for a particular category were completed (e.g. ADD tasks), participants moved onto the next category, and the process was repeated until all tasks were completed. The task categories and tasks within each category were randomised to reduce the potential impact of order effects. The start time, end time, vocal commands, and gaze actions (i.e. buttons clicked, position of gaze) were logged for later analysis. SUS (Brooke, 1996; Bangor et al., 2009) and NASA-TLX (Rubio et al., 2004) were also administered after all tasks had been completed, as well as an online survey (Appendix A.3) to obtain subjective perceptions of the application. The aim of designing the study was not to create a highly controlled evaluation but instead to provide structured tasks that encouraged participants to gain experience in using the application.

3.3.5 Measures

3.3.5.1 Task Completion time

Task completion time was measured to gain an indication around the length of time required to complete tasks using the multimodal approach. Task completion time was measured in milliseconds from when participants started each task (i.e. after the researcher initiated the task via a keyboard shortcut) until the task had been completed.

3.3.5.2 Usability and Cognitive Workload

SUS has been widely used as a standard measure to assess the subjective usability of systems including software, web applications, and hardware (Shakil et al., 2019; Sengupta et al., 2020; Creed et al., 2020). It is comprised of 10 items associated with usability and can efficiently be administered to study participants (Brooke, 1996; Bangor et al., 2009). SUS will therefore be utilised to measure the perceptions of usability at the end of the study. Similarly, NASA-TLX (Rubio et al., 2004) has been widely used in research studies to measure perceptions of workload and will also be administered to participants after completion of the study tasks.

3.3.5.3 Speech and Gaze Performance

Speech and gaze interaction errors were categorised into four themes: *Speech Recognition* – where the recogniser was inaccurate (e.g. “go to line 10” is recognised as “go two nine 10”); *Unrecognised commands* – where users issued commands that were not defined (e.g. “remove line 6” instead of “delete line 6”)*;* *Gaze Typing* – where an incorrect character is selected when writing text (e.g. when required to select “d” as a first character, a user incorrectly selects “s”)*;* *System error* – where a user performs the correct action (i.e. correct voice command or selection via eye gaze), but the system did not perform the appropriate action (due to latency issues).

3.3.5.4 Post-Test Questionnaire

To obtain qualitative feedback, participants were presented with an online survey with questions focused around their perceptions of using a multimodal approach for writing code, their experiences in using speech for issuing commands, how they found the use of gaze and Emmet for writing syntax and any suggestions for future updates (Appendix A.3).

3.3.6 Results

3.3.6.1 Task Completion Time

Task completion times ranged between 28:02 minutes to 76:01 minutes with an average time of 41.10 minutes (SD=12.04 minutes). The ADD tasks for both HTML (7:15 min – SD=2.10 min) and CSS (8:11 min – SD=4:41 min) and EDIT HTML tasks (7:48 min – SD=2:57 min) took the longest to complete. The average completion times for the other categories ranged from 2:30 min (SD=0.46 min) for SELECT CSS to 4:58 min (SD=2:06 min) for the DELETE HTML category.

3.3.6.2 Usability and Workload

The Voiceye application received an average SUS score of 68.1 (SD=20.8). According to Bangor et al. (2009), the score can be labelled as OK. An average NASA-TLX score of 42.0 (SD=20.1) was received for the application indicating the prototype has a "somewhat high workload".

3.3.6.3 Speech and Gaze Performance

All vocal and gaze actions were captured and analysed in terms of the errors categorised in Section 3.3.5.3. A total of 2493 errors were recorded (HTML: 1355; CSS: 1138) – in terms of HTML, 350 errors were related to Unrecognised Commands (e.g. a user issuing commands such as "remove line 100" and "go center", which were not part of the defined vocal grammar), 686 were associated with Speech Recognition (e.g. the command "right 10" which was often misrecognised as "write 10", 270 were related to Gaze Typing (e.g. a user selecting the wrong character when typing code), and 49 errors were related to System Error (e.g. the system not responding to a command issued by the user). For CSS, 307 errors were related to Unrecognised Commands (e.g. user uttering "cut this" – which was not part of the command), 565 were related to Speech Recognition (e.g. the command "delete selected" was misrecognised as "delete elected"), 218 were associated with Gaze Typing (e.g. user accidentally pressing the characters twice), while 48 were System Errors.

3.3.6.4 Qualitative analysis

Qualitative data obtained through the semi-structured interviews and observations were retrieved for analysis. A thematic analysis (Braun and Clarke, 2012) was implemented to identify overarching themes and concepts by deriving codes from the retrieved qualitative data. These themes and concepts were then iteratively analysed and grouped into the five key themes as listed below:

Speech Interaction

20 participants provided positive responses around the use of voice for controlling the system - comments focused around the voice being accurate, quick and easy to use:

"*… the speech controlling system was efficient and easy to use*" (P15).

Seven participants also highlighted that they preferred speech over gaze as an interaction approach:

"*it was much quicker, more reliable and allowed for shortcuts (such as end-of-line etc.) ... with a few extra commands I would not need to rely on the gaze-based input as much*" (P28).

However, two participants explicitly commented that they found voice interaction challenging to use:

"*…your eye would be focused on the editor, not the prompt coming back so it would take time to work out it had misheard you, then break your focus look at what happened and repeat*" (P14).

Furthermore, seven participants highlighted that some of the listed commands were not always recognised and that they sometimes had to repeat commands on multiple occasions.

Gaze Typing

16 participants provided positive comments about typing code via gaze stating that it was easy and intuitive to use:

"*…it was reasonably quick and the keyboard layout was intuitive*" (P28).

Five participants commented that they initially found it challenging but that they found it easier over time:

"*…it was harder in the beginning, but I got used to it pretty quickly*" (P22).

Seven participants commented that typing via gaze was "hard" or "difficult", including three participants who stated accuracy and calibration issues made the experience "frustrating" (P27, P29, P30). 20 participants commented positively on using Emmet when writing code via gaze. These positive statements tended to emphasise that the approach was fast and simple to use:

"*…it was really easy as it did not take any time to write the code manually*" (P4).

Six participants again commented that calibration and accuracy issues influenced their coding experience via gaze:

"*it was a good way of speeding up the workflow, but the eye tracker's low accuracy made it tricky to be consistent*" (P28).

Overall Multimodal Approach

19 participants provided positive responses around their experience of using a multimodal approach in a coding environment. These participants described the approach as "simple" and "easy" to use:

"*my overall experience was good, it was easy to use and the functions were simple…*" (P15).

Further comments re-emphasised participants' overall positive perceptions of the prototype and ability to complete tasks – for example, P17 commented that:

"*…there is potential for this tool to aid programmers in general; programmers using multiple screen at home may benefit particularly from talking to one screen and type-code on another screen – allowing programmers to sift through data files/code*".

Observation Insights

It was observed that Voiceye was unable to distinguish between different homophones – for instance, when attempting to delete lines 2 to 5, participants often said "*delete line 2 to 5*", although the recogniser would often interpret this as "*delete line 225*". Voiceye would therefore attempt to delete line 225 instead of deleting lines 2, 3, 4 and 5. Similarly, the recogniser also commonly misinterpreted the context when participants stated numbers – for example, "2" was often recognised as "to" and "4" as "for". Additionally, participants occasionally forgot to trigger the mechanical switch for initiating the recogniser, thus resulting in them having to repeat commands once they realised voice input was not being detected. It was also observed that participants triggered the speech recogniser key instead of the gaze selection key to perform the selection of keyboard input. In a small number of cases (*N*=34), participants also used voice commands to write the code in addition to gaze input (e.g. instead of typing "*color: red;*" via gaze, participants would say "*type colour red*" and then would use the keyboard to type the ":").

Future Improvements

Four participants suggested including more voice commands to navigate around syntax:

"*… the 'right' and 'left' options were very tedious to use. It might be useful to have some sort of feature which allows you to move several spaces in the left and right direction at a time*" (P6).

Similarly, P28 commented that a more extensive range of commands would be useful:

"*… allowing for typing individual letters, allowing jumping to specific columns within a line and simplifying repetitive navigation tasks (such as moving to words in the middle of lines)*".

Eight participants also emphasised the importance of ensuring the gaze system is accurate and comfortable to use.

3.3.7 Study Summary

The main objective of this study was to examine the feasibility and usability of multiple input modalities in performing coding activities. The majority of participants had no prior experience of using speech or gaze in the development environment and despite practicing for a short period (approximately 10 minutes), all participants were able to use gaze to type, voice to control the interface, and switches for selection. The results of the study were positive and validated the feasibility of combining these methods of interaction for performing coding activities. Whilst participants found the approach to be intuitive, simple and easy to use, they also highlighted some limitations around the accuracy of speech recogniser and eye tracking. In terms of gaze interaction, this can potentially be addressed through utilising an eye-tracking device with a higher level of accuracy and precision. The issues around speech misrecognition can also be minimised further through the inclusion of additional vocal commands (e.g. the command "right" was often misrecognised as "write", so mapping these commands to perform the same action can help to address this issue). The insights obtained from this evaluation provided opportunities for further iterative development (detailed in the following section) prior to testing the feasibility of the multimodal approach with the primary target audience (i.e. developers with physical impairments).

3.4 User Study 2

Informed through the feedback obtained from the exploratory study, iterative updates were made to Voiceye, and a follow-up study was conducted with five users who have physical impairments to explore their experiences in writing HTML and CSS syntax using the system.

3.4.1 Voiceye (Version 2)

To address feedback from the first study around enhancing syntax navigation, a new vocal command enabling users to jump multiple positions within a line was added (e.g. Left/Right x - “Left 5” would move the cursor five positions to the left from the current position). Additionally, several participants during the initial study commented on the system’s misrecognition of some commonly used voice commands like “delete” and “select”, so similar words were integrated into the speech recogniser and then mapped to the correct command (e.g. the system was often misrecognising a “delete” command as “de’lite” – this “incorrect” term was therefore added to the grammar). Moreover, Wagner and Gray (2015) previously highlighted that non-native English participants could experience more pronunciation errors, so the system was updated to support the selection of different English accents (i.e. British English, American English, Indian English, Australian English accent) and offered through Microsoft’s Azure Speech service (to help further enhance recognition accuracy) (Microsoft, 2023). Whilst most users in the first study were able to control the system via the Eye Tribe device effectively, it was decided to integrate the Tobii 4C sensor (Tobii, 2023) which provides a higher level of accuracy and has a more significant operating distance to address accuracy issues reported by some users.

3.4.2 Participants

Five participants were recruited through the support of the London RSI group (London RSI, 2023) and existing links within the research team. Participants' ages ranged from 20 to 38 years. Participants had a mean age of 23.6 years (SD=7.3 years), with four being diagnosed with RSI and one with Cerebral Palsy. Participants were asked to self-assess their experience with web development skills and alternative input methods (Table 3.2). Participants had 2.4 years (SD=1.5 years) of web development experience, three participants had previously used speech recognition technology for development work, whilst none had prior experience using eye gaze as an alternative interaction approach. One participant was a native-English speaker – the other native languages included Nepali, Lithuanian, Polish and Spanish. Two participants wore corrective lenses (all glasses).

3.4.3 Apparatus

The study was conducted on a Windows 10 laptop (Intel® Core (TM) i3-7100U CPU and 8GB RAM) with the screen resolution set to 1920x1080px. The Tobii 4C eye tracker was used and attached to the bottom of the screen via a magnetic connection. This sensor provides an average accuracy of 0.4 to 0.9° of visual angle and an operating range between 50-95cm. Two 65mm Jellybean switches were again used for completing gaze-based selections and for triggering the speech recogniser. The equipment was set up on a table and tailored to participants' needs (e.g. in terms of where the switches were placed).

3.4.4 Procedure

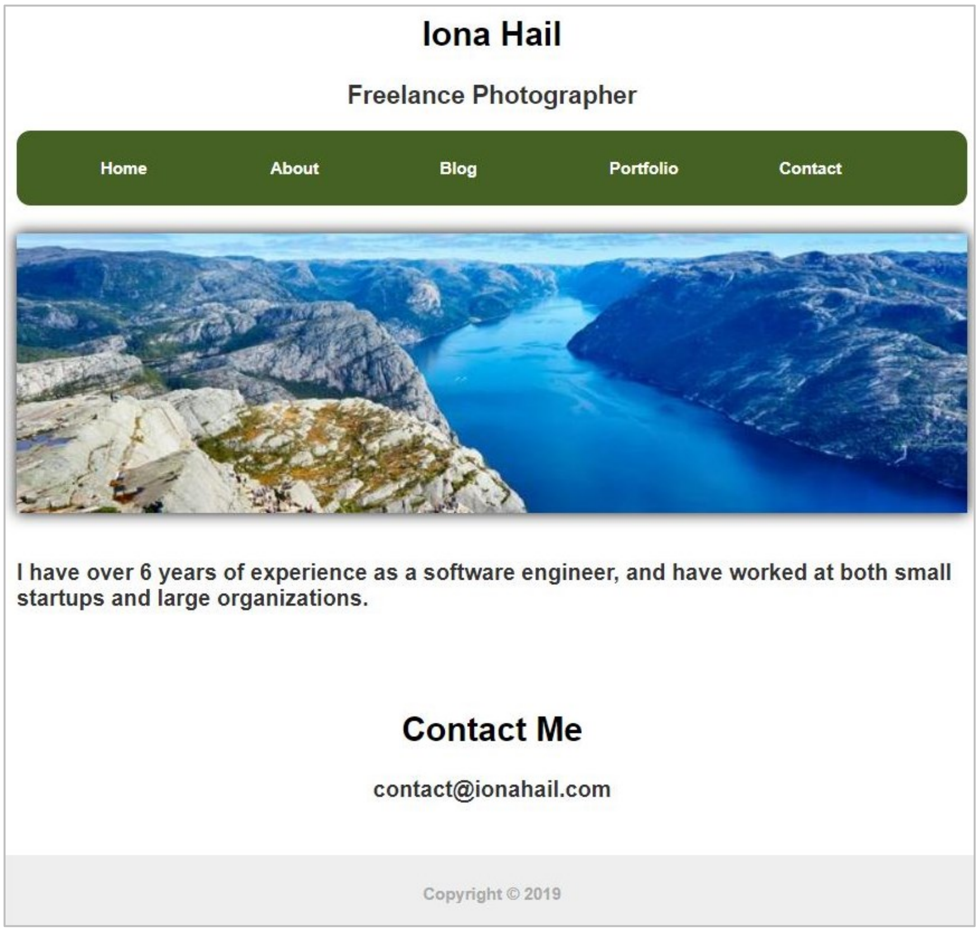
Institutional Review Board (IRB) approval was obtained for the study. Evaluations were conducted at participants' home or work environments (P1, P2, P4 and P5) and the DMT Research Lab (P3). Participants were initially given an information sheet and asked to provide informed consent. A survey (Appendix A.4) was then administered to collect demographic information, details of impairments, and information about their web development skills. Upon completion of the survey, participants were given a guided demonstration of the prototype by the researcher. The eye tracker calibration process was then completed, and participants were asked to practice using the system for around 5-10 minutes.

ID	Age	Impairments	Technical Experience	Challenges
P1	21 (M)	RSI (since 2017). Burning sensation in wrists and forearms.	WD: 2 years; SRT: 1 year; EG: None; IDE: Visual Studio; AT: Uses VoiceCode and vertical mouse.	Limited HTML/CSS support in VoiceCode; misrecognition of speech input.
P2	38 (M)	RSI (Since 2018); Pain in the wrist and fingertips.	WD: 5 years; SRT: 1 year; EG: None; IDE: Web Storm; AT: Used Nuance Dragon once – uses mouse/keyboard.	Issues with speech recognition accuracy; experiencing severe pain when using keyboard/mouse.
P3	20 (M)	Epidermolysis bullosa (since birth); gets blisters from friction and extreme heat.	WD: 1 year; SRT: None; EG: None; IDE: Dreamweaver; AT: Previously used IntelliKeys – currently uses a standard keyboard and vertical mouse.	Experiences pain when typing via keyboards (both standard and IntelliKeys).
P4	18 (M)	Mild Cerebral Palsy (since birth); affected mobility; stiff limbs; difficulty in typing.	WD: 1 year; SRT: None; EG: None; IDE: Dreamweaver; AT: None.	Difficulty typing on a keyboard and using a mouse to control software (due to mobility impairments).
P5	21 (M)	RSI (Since 2019); Pain in the left shoulder, numbness in left arm and fingers.	WD: 3 years; SRT: 1 year; EG: None; IDE: VSCode, Atom; AT: Uses TalonVoice.	No standardized voiced commands with TalonVoice; voice recognition accuracy issues.

Table 3.2: Participant Details - WD =Web development; SRT =Speech Recognition Tools; EG =Eye Gaze; IDE =Integrated Development Environment; AT =Assistive Technology; VSCode = Visual Studio Code

The main task was designed around a real-world development scenario in which a website had to be coded using HTML and CSS. Participants were provided with a screenshot of the website on paper to provide some context around the task (Figure 3.4). They were also given a sheet with the available voice commands and examples of Emmet commands that could be used to complete the task to avoid bias due to learning effects. The Voiceye application was then started with a blank HTML and CSS document opened by default. Participants were asked to start working towards coding up the design using the multimodal interaction approach. Moreover, upon completing the activity, they were asked to complete the main task separately using their existing assistive technology and code editor of choice (for comparative purposes). Three participants (P1, P2, P3) were able to complete this additional element of the study, whilst the other two preferred to leave this due to the potential discomfort it might cause them.

This study design aimed to give participants a realistic coding activity that encouraged them to explore and openly evaluate the application and interaction approach (as opposed to conducting a highly controlled study). There was no time limit set for the activity, although participants were informed that they could take a break or withdraw at any time if they experienced any fatigue, tiredness, or discomfort. Sessions lasted between 65-130 minutes, and participants were then asked to complete a SUS form to assess the overall usability of Voiceye. They then completed an online survey (Appendix A.5) with questions focused around exploring their experiences of using the system. Finally, an informal interview was conducted to further investigate their perceptions of the application and the multimodal coding approach used.



Iona Hail

Freelance Photographer

I have over 6 years of experience as a software engineer, and have worked at both small startups and large organizations.

Contact Me

contact@ionahail.com

Copyright © 2019

Figure 3.4: The design used for the follow-up study

3.4.5 Measures

3.4.5.1 Usability

Perceptions of usability were measured using SUS (Brooke, 1996; Bangor et al., 2009) administered at the end of the study. The NASA TLX was not employed for this study to ensure participants were not overwhelmed after completing all tasks, surveys, and interview questions.

3.4.5.2 Voiceye Experience

Participants completed an online survey and took part in a semi-structured interview to obtain their overall impressions of using a multimodal approach for coding activities (Appendix A.5).

3.4.5.3 Speech and Gaze Performance

Similar to the first study, all gaze (selection commands) and voice (verbal commands actions were collated, and errors were classified into the same four themes: **Speech Recognition, Unrecognised Speech commands, Gaze Typing, and System Error.**

3.4.6 Results

3.4.6.1 Usability (SUS)

All the participants were able to utilise the features within Voiceye to perform coding activities from a blank file. Voiceye received an average SUS score of 74.0 (SD =4.6), which can be labelled as exhibiting a “Good” level of usability (Bangor et al., 2009). Four participants (P2, P3, P4 and P5) provided scores of 70 or over, whilst a lower score was provided by P1 (67.5), which can be labelled as “OK” in terms of usability.

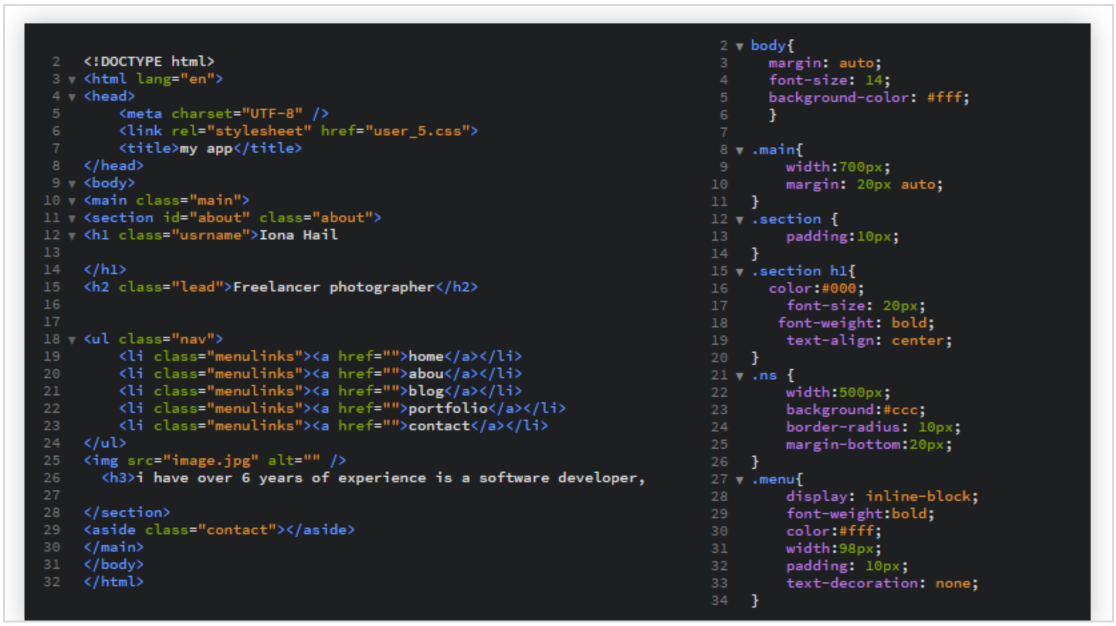
3.4.6.2 Speech and Gaze Performance

A total of 2180 voice commands and gaze selections were issued by participants during the evaluations (HTML: 1335; CSS: 845) - 1290 were gaze-related actions, while 890 were speech-related commands. A total of 527 errors were recorded across all five evaluations - 243 errors were related to Gaze Typing, 58 with Unrecognised Speech, 34 to System Error, and 192 with Speech Recognition.

3.4.6.3 Qualitative Feedback

Coding with Existing Tools: P1 was able to write most of the HTML code using his current assistive tools, as well as create multiple new CSS styles. However, P1 commented that he had to "stress" his voice when using VoiceCode (Voicecode, 2023), often due to issues associated with speech misrecognition. He was also only able to use one hand with a physical keyboard resulting in a slower typing rate. P2 was able to write the key components of an HTML document and create some new CSS styles, although he regularly required breaks due to experiencing pain when typing. It was also observed that both P1 and P2 had issues when attempting to edit code with each commenting that they found editing incorrect syntax to be particularly cumbersome using their existing tools. P3 was able to produce markup for the basic components of an HTML document, although only wrote a single CSS style. This was influenced through P3's comment that he found it challenging when having to use multiple keys simultaneously (i.e. typing characters such as "<" where the use of the shift key is required).

Coding with Voiceye: All participants were able to utilise the features within Voiceye to write HTML and CSS code starting from a blank file. P1 and P5 were able to add more HTML elements (i.e. images, navigation, sections), whilst P2 spent more time creating styles for CSS classes to work on the presentation of the HTML (i.e. the code on the right side of Figure 3.5). The code on the left side of Figure 3.5 shows a screenshot of the HTML written by P1, including common page elements such as a header, navigation bar with a menu, images, and some longer text blocks. P1, P2, and P5 provided positive comments about the application in that it was simple, intuitive, and easy to use. Participants P3 and P4 were also able to utilise Voiceye to write code, although their final output typically incorporated fewer HTML and CSS elements. Time spent on the coding activity ranged from 15 minutes (P3) – 46 minutes (P2).



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="user_5.css">
    <title>my app</title>
</head>
<body>
<main class="main">
<section id="about" class="about">
<h1 class="usrname">Iona Hail

</h1>
<h2 class="lead">Freelancer photographer</h2>


<ul class="nav">
    <li class="menulinks"><a href="">home</a></li>
    <li class="menulinks"><a href="">abou</a></li>
    <li class="menulinks"><a href="">blog</a></li>
    <li class="menulinks"><a href="">portfolio</a></li>
    <li class="menulinks"><a href="">contact</a></li>
</ul>
<img src="image.jpg" alt="" />
  <h3>i have over 6 years of experience is a software developer,

</section>
<aside class="contact"></aside>
</main>
</body>
</html>
```

```css
body{
  margin: auto;
  font-size: 14;
  background-color: #fff;
  }

.main{
    width:700px;
    margin: 20px auto;
}
.section {
    padding:10px;
}
.section h1{
  color:#000;
    font-size: 20px;
   font-weight: bold;
    text-align: center;
}
.ns {
    width:500px;
    background:#ccc;
    border-radius: 10px;
    margin-bottom:20px;
}
.menu{
    display: inline-block;
    font-weight:bold;
    color:#fff;
    width:98px;
    padding: 10px;
    text-decoration: none;
}
```

Figure 3.5: The HTML script on the left is written by P1, while P2 coded the CSS script on the right

Voiceye vs Existing Assistive Technology: P1, P2 and P5 reported that Voiceye was better than their existing coding tool and provided positive comments about the multimodal approach. In particular, P1 and P5 reported that the use of eye gaze and voice allowed them to select between the different input methods to fix errors when typing code, which was not present in their current voice-based tool (i.e. TalonVoice and VoiceCode). Three participants (P1, P2 and P3) highlighted that Voiceye helped remove the burden of using their hands when pressing keys when using a keyboard and mouse interaction approach. P4 stated that he had never previously been exposed to any assistive tools but commented that Voiceye could be a viable approach to support him with coding. Three participants (P1, P2 and P3) also used their existing tools to complete the task and believed that the inclusion of additional features such as switching to different tabs and customisable voice commands would make the prototype a viable solution for future coding.

Speech Control: All five participants utilised voice to perform coding activities with their comments focusing on the approach being natural and easy to use, as well as commands being easily memorable. However, each participant still experienced some minor issues with the accuracy of the recogniser at times:

"*…I know it will not recognise the programming languages, however the idea of using it as a command is really good approach. It worked well with slight issues (not recognising...)*" (P5).

P4 reported that his stutter worsened when trying to type longer text but worked effectively with the main commands. Two participants (P2, P4) commented that they would like to see more voice commands (e.g. find/replace and select/delete for selecting and deleting specific word(s) within a line rather than using left/right commands).

Typing Syntax via Gaze: Three participants (P3, P4, P5) commented that gaze was a fast and easy approach for writing code:

"*… frankly, I thought it would be slow to type HTML … but with emit [sic] code, it was faster…*" (P5).

Two participants (P1, P2) suggested that they would like to see improvements in the accuracy of the gaze interaction, although both were able to effectively use the system to write code. It was observed that all the participants tended to look away from the keyboard screen while releasing the switch button, thus resulting in them selecting the wrong character. This added a significant interaction cost as the character would then need to be deleted. P2 and P4 highlighted that using the switches required some effort and could result in their hands becoming tired over prolonged periods of interaction. Three participants (P1, P2 and P5) also suggested incorporating other languages (e.g. JavaScript) to support more comprehensive coding activities.

3.5 Conclusions

This chapter has addressed the research question highlighted in Section 3.3.1 through the development and evaluation of a multimodal system to support people with physical impairments with writing and editing code. Previous studies have independently explored the potential of gaze and speech input for writing code (Begel and Graham, 2005; Bednarik and Tukiainen, 2006; Glücker et al., 2014; Rosenblatt et al., 2018; Shakil et al., 2019), although both approaches have been found to have strengths and limitations (e.g. challenges around the selection of small targets via gaze and speech recognition issues). Researchers have previously highlighted that the combination of speech and gaze could present a complimentary and more effective method of interaction ((Beelders, 2011; Van Der Kamp and Sundstedt, 2011; Sengupta et al., 2018)), although there has been a lack of research to date investigating the potential of this approach. This chapter has addressed the limited work in this area through presenting a new system (Voiceye) that integrates these methods of interaction (in addition to mechanical switches) to enable developers with physical impairments to write code.

The results from the two user evaluations were positive and demonstrated the feasibility of combining speech, gaze, and mechanical switch interaction for supporting development work. The majority of participants from the first study found coding via multimodal input controls to be an intuitive and simple interaction approach. Coders with physical impairments from the second study also made similar positive comments, with several stating that this approach provided advantages over their existing assistive interaction methods. This combination of speech and gaze interaction (supported with switch controls) demonstrates how they can be utilised together to help overcome some of the limitations of each technology.

In particular, writing code via gaze interaction (using a common shorthand notation) helps to address some of the misrecognition issues around producing syntax via speech recognition. Similarly, the use of speech input (using a fixed set of vocal commands) for enabling actions such as the navigation, selection, and deletion of code helps to overcome the issues of activating small targets via eye gaze. This new multimodal approach, therefore, provides key insights on the potential of combining speech, gaze, and mechanical switches for development work, as well as highlighting future challenges to be addressed.

From a wider perspective, this research highlights how multimodal interaction can support the control of systems, although there is a lack of work examining the combination of additional input methods to develop new assistive solutions (e.g. mid-air gesturing, head tracking, facial expression control). Also, as highlighted by P17 (during the first study), exploring the interplay of these new multimodal approaches with different external displays and configurations could hold much potential for both disabled and non-disabled coders (e.g. in terms of one display being optimised primarily for speech controlled activities, whilst another could support different tasks aligned to an alternative input method).

Whilst the evaluations highlighted that the multimodal approach is viable and was perceived positively by participants, one issue observed was the efficacy and usability of gaze input.

In particular, some participants across both studies encountered calibration and accuracy issues on occasions when attempting to select interface targets. While steps were taken to mitigate the impact of this known challenge (e.g. through using large selection targets (Miniotas et al., 2006; Biswas and Langdon, 2011)), this still caused issues and frustration for some participants. Furthermore, whilst the use of a standard shorthand coding notation (i.e. Emmet) was used to support more efficient syntax generation via gaze interaction, this can still potentially be a slow process. It was observed during the studies that participants found speech a viable approach for writing longer comments as part of the coding activities, so one potential alternative approach is to explore further the use of voice input along with another input (e.g. mechanical switch) for improving the efficiency of writing code.

Another key area suggested by participants was to investigate the use of high-level programming languages (e.g. JavaScript) via alternative input methods. These languages present additional challenges in that a wider range of syntax and coding constructs need to be supported. This may lead to issues around the recall of voice commands that could impact on the feasibility of voice coding (Joshi and Bein, 2020), although other approaches, such as natural language voice input, may potentially provide some interaction benefits (Good and Howland, 2017; Barmpoutis, 2018). However, there has been a lack of work exploring the viability of these methods and whether they can support more inclusive coding experiences. This, therefore, forms the focus of the next chapter, where two different multimodal voice coding approaches are compared in the context of writing syntax for a high-level language.

4 Multimodal Fixed Grammar and Natural Language Voice Coding

4.1 Introduction

This chapter explores the development and evaluation of a research prototype to enable developers with physical impairments to write and edit JavaScript code utilising a multimodal speech and mechanical switch approach. A particular emphasis is on investigating and comparing development activities using a fixed grammar and natural language voice coding approach. Speech as an interaction approach has developed significantly in recent years, with researchers starting to explore the feasibility of this approach to support coding activities - for instance, studies have investigated the efficiency of rule-based grammar in coding environments where specific commands can be used to write new syntax and initiate different features (Wagner and Gray, 2015; Nowogdorzki, 2018; Rosenblatt et al., 2018). Whilst this type of voice coding approach presents new opportunities for developers, it can also give rise to some interaction challenges – for example, users have to learn the supported vocal commands, which can add significant cognitive load during development work (Good and Howland, 2017; Van Brummelen et al., 2020b). Similarly, the recognition system may be unable to distinguish between homophonic words (i.e. “for” with “four”, “to” with “two”) and may encounter difficulties in recognising commands issued by non-English speakers (Wagner and Gray, 2015; Rosenblatt et al., 2018). Commands may also not be recognised properly, requiring users to repeat the same command on multiple occasions, resulting in inefficient development experiences (Desilets, 2001; Nowogdorzki, 2018; De León Cordero et al., 2021).

An alternative approach that can address these limitations is the use of natural language (i.e. unconstrained commands) to write syntax. This has become an increasingly viable approach in recent years and has been investigated in different domains (e.g. creative fields (Srinivasan et al., 2019), task management (Sarmah et al., 2020), home automation system (Pradhan et al., 2018; Lau et al., 2018; Ramadan et al., 2021), and conversational coding (Jung et al., 2019; Van Brummelen et al., 2020b)). This method could provide developers with more flexibility around the commands they use to write code, thus facilitating a more natural user experience (Delimarschi et al., 2014; Good and Howland, 2017; Mazumder et al., 2019). Conversely, the unfamiliarity of this method could create uncertainty around which commands to use and potentially lead to perceptions of an unnatural user experience (Srinivasan et al., 2019; Van Brummelen et al., 2020b). Both fixed grammar and natural language voice coding approaches have been explored independently, although no comparative studies have been conducted to date to measure the efficacy and usability of these approaches. Moreover, no evaluations using natural language approaches have been conducted with developers who have physical impairments, so it remains unclear whether this method of coding is a viable approach for this target audience.

This chapter initially explores design requirements for vocal programming through detailing an exploratory study investigating the types of voice commands developers would prefer to use for completing a range of coding activities. The findings of this study informed the design of a novel research prototype utilising both natural language and fixed grammar approaches to facilitate the writing, editing, and navigation of code. A comparative study of the two approaches is then highlighted, where the strengths and limitations of both methods are identified. A rationale is then provided for conducting further research on the natural language approach, where a second version of the prototype is evaluated in a multi-session study with developers who have physical impairments. This chapter, therefore, presents four key contributions:

(1) results from an exploratory study highlighting the voice commands developers prefer to use while performing coding activities.

(2) a novel multimodal speech and mechanical switch-enabled coding environment enabling people with physical impairments to write, edit, select, remove and navigate JavaScript code via a fixed grammar and natural language approach.

(3) a user study comparing fixed grammar and natural language approaches with results demonstrating similar levels of performance and efficacy across both coding approaches.

(4) a multi-session study with developers who have physical impairments validating the feasibility of writing and editing code via multimodal natural language speech input

4.2 Exploratory Study

Limited work has been conducted to understand the types of commands preferred by developers when performing coding tasks such as writing, navigating, and editing code. It was therefore important to conduct an initial exploratory study with developers to understand more deeply the types of speech commands that might be used to facilitate different coding activities.

4.2.1 Participants

12 university students (all male, ages 20-25, M=22.2, SD=1.7) with at least a year of coding experience (M=2.1; SD=1.2) volunteered for the study. Seven participants were native English speakers, whilst other native languages included Polish, Italian, Punjabi, Nepali and Romanian. Participants were initially asked to self-assess their experience using alternative input methods for interacting with computers. Eight participants had prior experience using voice for controlling a home environment, whilst two had previously utilised voice interaction for performing coding activities.

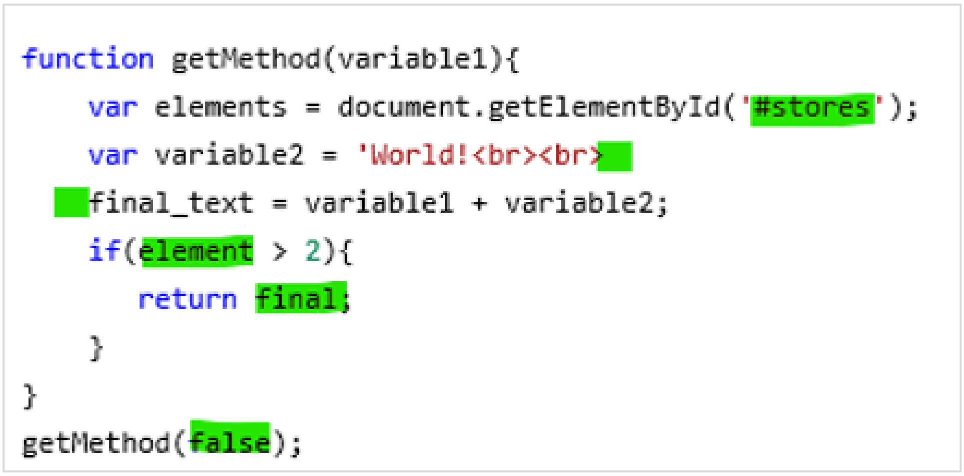
4.2.2 Apparatus

The study was conducted on a Windows laptop (Intel® Core(TM) i3-7100U CUP and 8GB RAM) machine. The laptop's built-in microphone was used for capturing speech input which was recorded using Microsoft's default sound recorder software.

4.2.3 Procedure

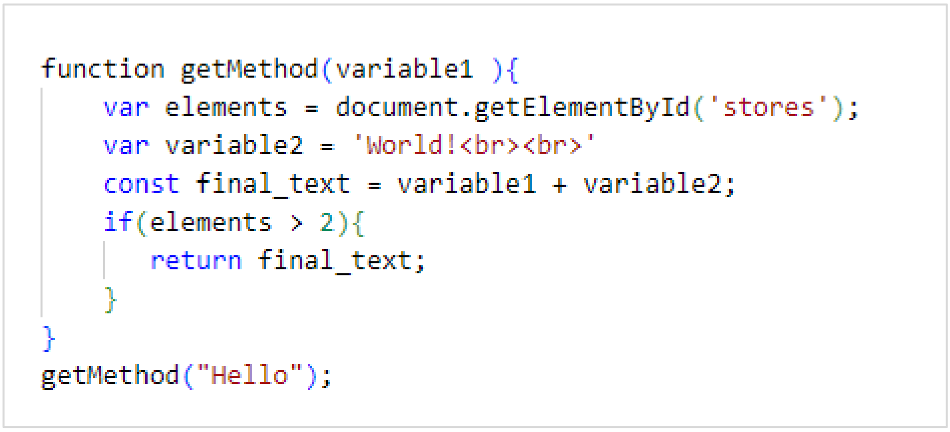
Institutional Review Board (IRB) approval was obtained prior to the study. Participants were initially briefed about the motivation of the research and provided informed consent. Participants were then handed two code snippets on paper: one containing six syntax errors (Figure 4.1) and another with the same code, but where the errors were corrected (Figure

4.2). The syntax issues incorporated into the code were informed through the common JavaScript errors identified by Ocariza et al. (2013) and Hanam et al. (2016) to create a realistic coding scenario. The errors were also clearly marked through highlighting the relevant code to ensure developers were aware of the syntax issues regardless of their coding experience. Once participants had viewed both snippets and were clear on the updates required, they were asked to verbalise the voice commands they would use to correct each syntax error to match the correct code. This involved them having to issue commands in relation to navigating, selecting, deleting, and writing code.



```
function getMethod(variable1){
    var elements = document.getElementById('#stores');
    var variable2 = 'World!<br><br>
   final_text = variable1 + variable2;
    if(element > 2){
       return final;
    }
}
getMethod(false);
```

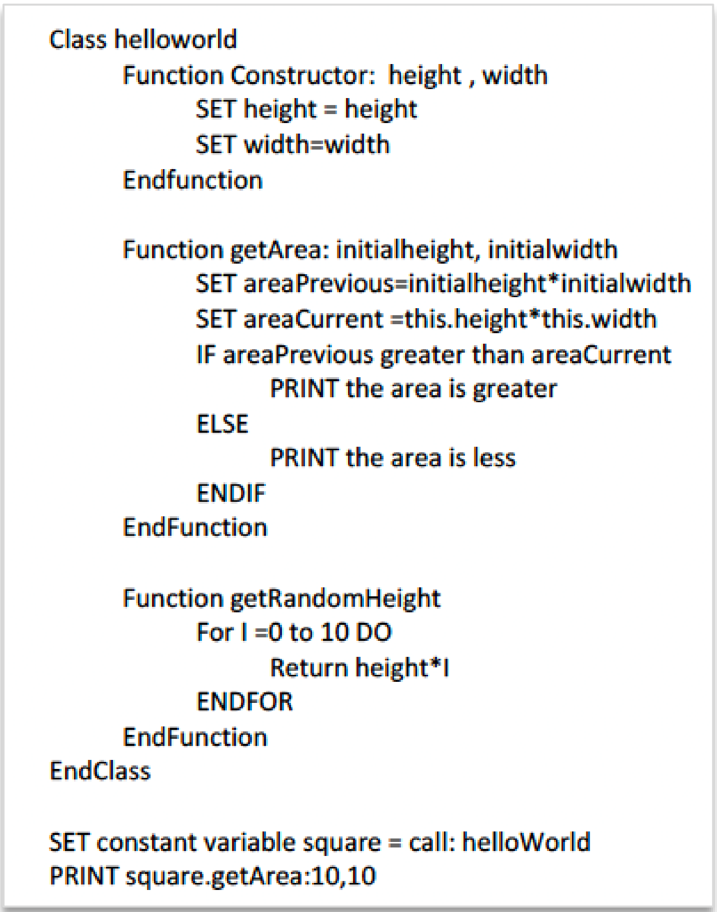
Figure 4.1: JavaScript code snippet used for the exploratory study containing syntax errors (green highlighted text denotes where errors are located).



```
function getMethod(variable1 ){
    var elements = document.getElementById('stores');
    var variable2 = 'World!<br><br>'
    const final_text = variable1 + variable2;
    if(elements > 2){
       return final_text;
    }
}
getMethod("Hello");
```

Figure 4.2: JavaScript code snippet used for the exploratory study where the errors are not present.

After correcting the syntax errors, participants were given an additional task around writing code (as opposed to editing), where they were initially given some pseudo code on paper containing logic for creating a class, two methods, variables, a loop, and conditional statements (Figure 4.3). Once participants had familiarised themselves with this code, they were asked to verbalise the commands they would use to generate the relevant syntax. Each session was recorded, and participants' verbal commands were transcribed into text (all participants completed the study individually).



```
Class helloworld
        Function Constructor:  height , width
                SET height = height
                SET width=width
        Endfunction

        Function getArea: initialheight, initialwidth
                SET areaPrevious=initialheight*initialwidth
                SET areaCurrent =this.height*this.width
                IF areaPrevious greater than areaCurrent
                        PRINT the area is greater
                ELSE
                        PRINT the area is less
                ENDIF
        EndFunction

        Function getRandomHeight
                For I =0 to 10 DO
                        Return height*I
                ENDFOR
        EndFunction
EndClass

SET constant variable square = call: helloWorld
PRINT square.getArea:10,10
```

Figure 4.3: Pseudocode used in the exploratory study

Participants issued a total number of 417 voice commands – 263 were associated with writing code and 154 for editing syntax (Table 4.1).

Task	Voice Commands	Occurances
Declare a class	Create Class X	10
	Charlie Lima Alpha Sierra Sierra space X (NATO Alphabet)	2
Declare a function	Create function X	18
	Create function X with Y and Z	8
	Create function X that takes Y and Z	5
	Fox Uniform November Charlie Tango India Oscar November space X	5
Assign Variable	Set X to Y	32
	Create variable X	17
	Assign X to Y	20
Condition	If X greater than Y	4
	Check If X greater than Y	3
Loop	For X equals 0 X greater than Y and X++	5
	For loop I from 0 to Y	4
	For open bracket var I equals 0 I less than or equals to Y and increment I close bracket	3
Print	Console log X	20
	Print X	12
Navigate to line	Go to line	28
	Move to line X	19
Navigate around Position	Move x position Left/Right/Up/Down	24
	Up x, Down X, Left X, Right Y	21
	Go X Left/Right	10
Delete/	Delete X	31
	Delete line x	7
Select	Select Line	4
	Select word X	7

Table 4.1: Frequently used commands during the exploratory study

A thematic analysis (Braun and Clarke, 2012) was performed to understand the common themes associated with voice commands for performing coding activities. The first stage involved developing initial categories around the commands issued across all participants, followed by multiple rounds of iteratively refining key themes capturing the insights from participants. The resulting five themes are detailed below:

Voice Command Variation: All participants used a variation of commands with the majority (*N=10*) utilising high-level phrases and abstractions when issuing voice commands to write code, while two participants dictated each letter using the NATO alphabet. For example, a condition such as "*if (a > 5 { })*" was pronounced as "*if a greater than 5*" by four participants. Similarly, three participants used "*check if a is greater than 5*", while "*if condition a greater than 5*" was used by three participants, and two participants dictated the code using the NATO alphabet (i.e. "*India Fox Open Bracket Alpha greater than five close brackets open curly bracket close curly bracket*"). In some cases, participants verbalised code which was syntactically incorrect, but where the semantics of the code was correct. For instance, when creating a for loop, four participants used the command "*for loop i from 0 to 10*".

Symbol Omission: Eight participants omitted common symbols (i.e. (,), ==, ", ', ++, --) when coding (e.g. *for (var i =0; i <=10, i++)*, was verbalised as "*create for loop variable i from 0 to 10*". Similarly, "*function getarea(height, width)*" was spoken as "*create function getarea with height width*". However, two participants who dictated each letter when writing code used symbols for opening brackets and including operators. In the above example, the for loop was verbalised as "*for open bracket var i equals 0 i less than or equals to 10 and increment i close bracket*".

Code Navigation: 11 participants used terms such as "*new line*", "*enter*", "*next line*", and "*down*" after each command to move to the next statement. Additionally, participants used the terms "*up x position*" and "*go x position down*" to move to different lines. Similarly, for navigating to different positions, seven participants used multiple individual steps to move to their desired location (i.e. "*go to line number x*", "*move x position right*"). However, five participants used a single command (combining multiple steps) to navigate to a specific position (i.e. "*go to x position of line x*"). Five participants also varied the commands they issued for performing similar actions – for instance, when moving to a line number, P7 used the command "*go to line x*" on three occasions and "*line x*" twice.

Selecting and Deleting Code: Eight participants used multiple steps when selecting and deleting code (i.e. navigating to a particular line and then issuing a relevant command). For example, to delete the word "*defined*" on line 7, participants typically stated, "*go to line 9*" and then "*delete defined*". When selecting and deleting whole lines, participants used generic terms such as "*delete line number x*" or "*select line number x*". Symbols were not skipped when deleting or making selections. For instance, to select the term "*this.height*" participants would commonly state, "*select this dot height*".

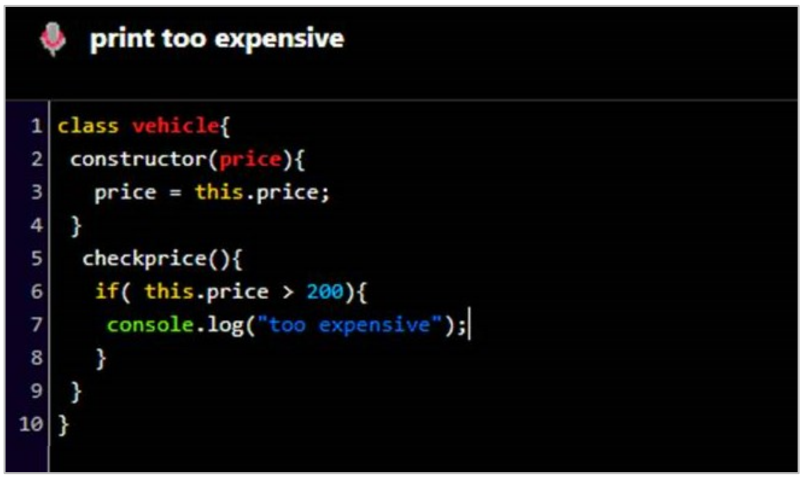
Common Commands: Nine participants typically used the word "create" when declaring a class and functions ("*create class helloworld"*). Similarly, participants typically used the term "*go to line*" when navigating to a specific line of code. Moreover, the "*delete*" command was commonly used by seven participants when removing lines of syntax, the "*select*" command was widely used to select lines of code, and "*end of line*" was regularly used by eight participants to manipulate the cursor. To delete and edit code, participants would first delete the incorrect code or line (e.g. "*delete line x*", "*delete word*") by navigating to the correct position and would then type the correct code. This is in contrast to previous studies in this area (Rosenblatt et al., 2018; Begel and Graham, 2006) where participants would use commands such as "*change to*" and "*replace*" to edit code. Finally, seven participants commonly used the command "*print*" to output any text or values (e.g. "*print area is less*").

This exploratory study provided insight into the types of commands that developers prefer to use when writing and editing syntax. This work was used to inform the design of a research prototype detailed in the following section that supports multimodal voice coding via both fixed grammar and natural language interaction.

4.3 System Design

The prototype from Chapter 3 was adapted for the purposes of the study discussed in Chapter 4 – the main interface design (Figure 4.4) was again informed through the design and layout of existing mainstream development environments (e.g. Visual Studio Code (2023), Brackets (2023)). In particular, the prototype was designed to run in fullscreen mode with a minimum width of 1100px and used a theme similar to the default one used in Visual Studio Code. A microphone icon was also displayed at the top of the interface, with the speech recogniser being disabled by default.

The architecture of the prototype consists of three major components: (1) a speech input interface to convert speech into text (using the WebSpeech API (2023), (2) a command interpreter to parse a user's intent and commands, and (3) an execution method to process coding actions based on the commands issued. The recognition system can be initiated via the "space" key (or a different switch input), which results in the microphone icon at the top of the interface subtly pulsating to highlight that the system is "listening" for input. Once a user issues a command the latest speech input recognised by the system is positioned next to the icon to provide feedback to users. An error message is also displayed below the speech input if the voice system fails to recognise the command issued. For example, if a user fails to provide a class name when creating a class, the prototype will display an error highlighting that a class name is expected. Two different speech-based coding methods were developed based on this underlying architecture – fixed grammar (FG) and natural language (NL).



print too expensive

```
class vehicle{
 constructor(price){
   price = this.price;
 }
  checkprice(){
   if( this.price > 200){
    console.log("too expensive");
   }
 }
}
```

Figure 4.4: A screenshot of the code editor - text displayed next to the microphone icon highlights the latest recognised speech command issued to the system

4.3.1 Fixed Grammar (FG)

The FG approach provides users with fixed commands for writing, editing, and navigating code (Table 4.2). In this approach, voice input is initially converted to text and then compared against the defined grammar (using a rule-based parser) – appropriate coding actions are then performed if a match is detected. For example, if a user states, "*create function area with height width*", the system will output

```
function area (height, width){

}
```

Tasks	Speech Commands	Description	Example Code / Utterances	Generated Output
Code Entry	“Create class *classname*”	To create class	*Speech*: “create class car”	Class car { ■}
	“Create function *functionname*”	To create function	*Speech*: “create function check”	check (){ ■}
	“Create function *functionname* with *parameters*”	To create function with parameters	*Speech*: “create function area with height width”	area (height, width) { ■ }
	“Assign variable name to *value*”	To assign a variable.	*Speech*: “assign let height to 10”	let height =10■;
	“If variable operator final value”	To create an if condition	*Speech*: “if let area greater than 10”	If (let area > 10){ ■}
	“Loop variable from initial value to final value	To initialize a for loop	*Speech*: “loop variable i from 0 to 5”	for (var i =0; i < 5; i++){ ■}
	“Print value”	To print	*Speech:* “print test”	console.log (“test”)■;
	“Type value”	To write text	*Speech*: “type welcome”	Welcome■
Navigate	“Go to number”	Navigating between the lines of code.	*Speech*: “Go to 10”	Cursor placed at start of line 10.
	“Left/Right/ Up/Down position”	Navigate x position left/right /up/down.	var height=■price; *Speech*: “right 5”	var height=price■;
	“End of Line”	Navigate to the end of current line.	■throw ‘intent is not’ *Speech*: “End of line”	throw ‘intent is not’■

Select	“Select line *line number*”	Select a single line	5. ■Class car {
			6. getcar (price, brand){
			Speech: “Select line 6”
				5. Class car {
				6. getcar (price,brand){ ■

	“Select line from line number to line number”	Select multiple lines	1. ■class area { 2……. 9 } *Speech:* “Select line from 1 to 9	1. class area { 2……. 9 }■
	“Next number”	Select x words right to the cursor.	■throw ‘intent is not’ *Speech*: “next 2”	throw ‘intent■ is not’
	“Previous number”	Select x words left to cursor.	Let this.brand =■brand *Speech*: “previous 3”	Let ■this.brand =brand

Delete	“Delete line *line number*”	Delete a single line	5. ■Class car {
			6. getcar (price, brand){
			7. this.price =price;
			Speech: “delete line 6”
				5. Class car {
				6. ■this.price =price;

	Delete line from line number to line number	Delete multiple lines	5. ■class area {
			6. Constructor(height){
			7 ………
			10 }
			Speech “delete line from 6 to 9”
				5. class area {
				6 ■}

	“Delete”	Delete highlighted code; deletes one character if not highlighted	■throw ‘intent is not valid’ *speech*: “delete”	■‘intent is not valid’
	“Back”	Remove character left	Let area =10■; *speech*: “back”	Let area =1■;

Table 4.2: List of main vocal commands used for fixed command grammar. “Vocal Commands” column contains keywords/terms that are spoken by users. Underline text in the “Speech Commands” column denotes the optional values, whereas the bold text denotes the compulsory value. The action performed through each speech command is visualized in the “Example Output” column. ■ denotes the position of the cursor.

Any deviation from the defined structure results in an error message being displayed. The available commands are hidden by default within the editor and can be displayed on the right side of the editor using the verbal command "*help*". The list of commands defined for writing code were informed through a combination of the most common terms stated by participants in the exploratory study and a previous study in the field where commands were categorised into four key areas: code entry, navigation, selection, and deletion (Rosenblatt et al., 2018).

4.3.2 Natural Language (NL)

The NL method provides users with the freedom to write and edit code using unconstrained natural speech input, as opposed to being constrained by specific commands that must be used to control the system. To facilitate this approach, Wit.ai (Wit, 2023), an open-source natural language interface, was used to train 182 utterances obtained from the initial exploratory study conducted. Wit.ai implements a natural language understanding model to extract the user's intent and associated parameters (such as entities and traits) from vocal commands that have been issued to the system.

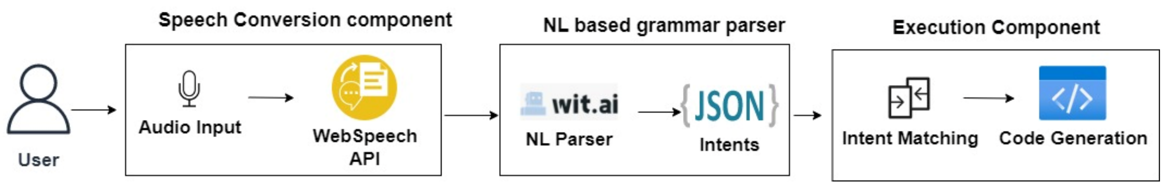


Figure 4.5: Architecture of the Natural Language System; The user voice input is converted into text and is passed to Wit.ai, which generates Intents and entities in a JSON format and is passed to the prototype. The prototype then checks the intent to trigger appropriate coding actions.

When coding via NL, speech input is initially converted to text and passed to the Wit.ai model using HTTP request. The model processes the request and returns the corresponding intent and entities in JSON format. The system then checks the intent against the rule-based interpreter to perform appropriate actions (Figure 4.5). For instance, if the user stated, "*create a new function check with two parameters brand and price*", the system would infer the intent as "*create function*" and recognise two parameters (brand and price). These results are passed onto the execution component to generate the following code within the editor.

```
function check (brand, price){

}
```

The list of intents used in the NL model are listed in Table 4.3.

Tasks	Intents	Example Utterances
Code Entry	Create new class/function.	"create a new class car", "create a new function called getarea which takes initialwidth"
	Create/assign new variable.	"create a new variable called area which takes initialheight times initialwidth"
	Create for loop.	"loop I between 0 to 10", "for loop I equal to 0 until 10 increment I"
	Create if statement.	"if height greater than width", "if true", "if not equals to false"
	Console/Non-code	"console log test", "type hello world", "print test"
	Add parameters to function*	"add hello in get method", "add parameter height to the function getarea"
Navigation	Navigate to lines of code.	"go 10", "move to line 3", "line 3", "go to line 10"
	Navigate code.	"go up 10", "left 30", "move right 10", "go down 20"
	Navigate to start/end of line.	"start of line", "end of line", "line end"
	Copy/paste code/text*	"copy", "copy my code", "paste", "paste code"
	Undo/redo previous action.	"undo", "undo up", "redo", "do redo"
Select	Select code.	"select 10 to 20", "select car", "select line 2"
Delete	Delete code.	"delete line 7", "delete constant height", "back"

Table 4.3: List of NL intents and example utterances used to train the NL model. * indicates the new commands added after the first evaluation.

4.4 Study 1: Fixed Grammar versus Natural Language

An evaluation was conducted with non-disabled participants to assess the performance and usability of the two voice coding interaction approaches developed (FG and NL). Whilst the core focus of this work is on exploring solutions for disabled developers, it was important to explore the efficacy of both approaches prior to conducting evaluations with developers who have physical impairments to ensure the system was usable. Given that the key interaction requirement for controlling the prototype is the use of voice input (regardless of whether participants have physical impairments), it was felt that working with non-disabled participants would provide essential insights. This methodology also aligns with other related studies focused around investigating assistive systems (Rosenblatt et al., 2018; Shakil et al., 2019; Aziz et al., 2021)

4.4.1 Research Question

The following research question was developed to address the limited work around the potential of different multimodal speech coding approaches (i.e. fixed grammar and natural language) to support development work:

RQ: To what extent can natural language and fixed grammar voice coding approaches support the writing, editing, navigation, and selection of code?

The broad focus of this question facilitated an exploratory research approach to help develop a deeper understanding around the challenges and opportunities associated with this method of interaction within a coding context.

4.4.2 Participants

25 participants (4 female) were recruited from a population of university students and social media platforms with ages ranging from 18 to 50 years (M=26.2; SD=8.4). Only two participants from the previous study (Chapter 3) took part in this study. All participants were fluent in English with 14 native English speakers, whilst other native languages included Nepali, French, Hindi, Bengali, Polish and Dutch. Participants were asked to self-assess their level of coding experience - 24 participants reported an average level of coding experience (M=4.48 years; SD=3.6), whilst one participant reported being a novice developer. All participants had at least one year of JavaScript coding experience. Four participants had prior experience using assistive tools to support coding (three had previously used speech, whereas one had experience with eye gaze interaction).

4.4.3 Apparatus

All study sessions were conducted remotely via Zoom (2023) and Microsoft Teams (2023). Participants were required to use their microphones during the study - 21 participants used their computer's built-in audio input device, while four participants used an external microphone. Participants were also required to use the Google Chrome browser as the Web speech API was incompatible with other mainstream browsers.

4.4.4 Procedure

Pre-Test: Institutional Review Board (IRB) approval was obtained prior to the study. The participant and researcher initially met online via Teams (*N=23*) or Zoom *(N=2)* (whichever was most suitable for participants) at an agreed time. The research prototype was hosted on a university server, and the link was then shared with participants. After a brief introduction about the prototype, participants were asked to navigate to a project information page and were requested to provide informed consent. They were then redirected to the pre-test questionnaire (Appendix B.1) to collect demographic information, as well as details around their coding and assistive technology experience. A within participants design was utilised, with participants randomly assigned to use either the FG or NL approach. They were then given a demonstration of the prototype and asked to practice with the approach they had been assigned (e.g. through creating classes and functions, navigating to different lines of code, selecting and deleting code) for at least 10 minutes to ensure they were comfortable with the coding methods.

Main Test: Following the practice session, participants started to work on the main experimental tasks. The tasks were designed and categorised into four key areas: CREATE, DELETE, SELECT, and EDIT. Each category consisted of six separate tasks (24 tasks in total) - CREATE tasks were informed through the evaluation tasks utilised by Soto Munoz et al. (2019) and Van Brummelen et al. (2020) and required participants to write new syntax within a blank document (e.g. create a class, a function with multiple attributes, an if condition to check two variables, and a for loop). EDIT tasks included common JavaScript errors, which were informed through the work of Ocarize et al. (2013) and Hanam et al. (2016), who examined and classified common JavaScript errors across different code repositories. Participants were therefore required to edit existing code to fix issues such as syntax errors, typographical mistakes, and undefined methods. SELECT tasks included selecting single and multiple lines of code, single and multiple characters and words, and specific syntax within a snippet of code. DELETE tasks involved removing lines, characters, snippets of code, and specific syntax within a snippet of code. The task categories and tasks within each category were randomised across all participants to reduce the potential impact of order effects. The list of tasks can be found in Appendix B.2.

Prior to starting a task, participants were initially shown the starting code they would see on the screen once the task started (e.g. a code snippet that contained a syntax error), along with the final code they needed to write (e.g. the same code where the syntax error has been corrected). Once participants had familiarised themselves with what the task entailed, they started the task via selecting the "Start Task" button. They then worked on the required task and selected the "Completed Task" button once they had finished making the necessary updates. Once all the tasks for a particular category were completed, participants would automatically move on to the next category, and the process was repeated until all the category tasks were finished. Participants were then administered the SUS survey (Brooke, 1996; Bangor et al., 2009). Participants then moved on to the next coding approach (FG or NL) and followed the same process. At the end of the evaluation, participants were presented with some open-ended questions (Appendix B.4) focusing on their subjective perceptions around the two different coding methods. All sessions were recorded for later analysis with participants' consent.

4.4.5 Measures

Task Completion Time: As stated in Chapter 3, it was important to measure task completion time to gain an understanding around the length of time required to complete tasks and to assess the efficiency of the interaction approach. Task completion time was measured in milliseconds from when participants started each task (i.e. after they selected the "Start Task" button) until the task had been completed.

Usability: Perceptions of usability were measured via SUS (Brooke, 1996), which was administered after all tasks had been completed for a coding approach.

Speech Performance: Errors were categorised into four themes: <u>Speech Misrecognition</u> – where the recogniser was inaccurate (e.g. "*delete line 10*" misread as "*delete nine*"); <u>System Error</u> – where a user command was correctly recognised, but the system did not perform the appropriate action (due to latency issues with the Web Speech API and Wit.ai); <u>Unrecognised Commands</u> – where users stated commands that were not part of the defined FG grammar; <u>Model Misunderstanding</u> – where the NL model misinterpreted the intention behind user voice commands. Speech misrecognition and system errors are applicable to both FG and NL - unrecognised commands are only relevant for FG, whilst model misunderstanding is only applicable to NL.

Post-Test Questionnaire: To obtain qualitative feedback, participants were presented with an online survey with questions focused on their perceptions and experiences of each voice coding approach (i.e. NL and FG), as well as any suggestions for future iterative development (Appendix B.3).

4.4.6 Results

A total of 1200 coding tasks were completed by participants across the study (24 tasks * 2 conditions* 25 participants). Shapiro-Wilk's test for normality (Rochon et al., 2012) found that task completion time, speech recognition accuracy, and usability were not normally distributed. A Wilcoxon test (Divine et al., 2013) was therefore used to analyse the data with an alpha of 0.5%.

4.4.6.1 Task Completion Time

No statistically significant differences (Z =0.895; p>0.05) were observed for overall task completion time between NL (M=16.7, SD=4) and FG (M=16.4, SD=4.4). The times ranged between 08:54 minutes to 24:08 minutes for FG and 09:12 minutes to 24:16 min for NL. To gain a more detailed perspective of task completion times for each task across the two different conditions, a taskwise analysis was conducted. The EDIT tasks for both NL (6:33 min, SD=1.87 min) and FG (5:44 min, SD=1.85 min) took the longest to complete, followed by CREATE tasks (NL - 4:23, SD=1.9 min; FG - 3:58 min, SD =1.5 min), SELECT tasks (NL - 3:0 min, SD=0.96 min; FG - 3:10 min, SD =1.50 min), and DELETE tasks (NL - M=2:26 min, SD =0.75; FG - M=3:08 min, SD=1.3 min).

4.4.6.2 Usability

NL obtained an average SUS score of 74.5 (SD =14.7), while FG received 73.9 (SD=16.7), where scores can be labelled as "Good" (Bangor et al., 2009). No statistically significant difference was found between conditions (Z=0.78, *p*>0.05).

4.4.6.3 Speech Performance

A total of 4996 vocal commands were issued (FG: 2868; NL: 2128). In terms of NL, 346 commands were related to speech misrecognition (16.2%), 202 were related to model misunderstanding (9.5%), and 69 (3.4%) were associated with a system error. For FG, 563 commands were related to speech misrecognition (19.6%), 299 to unrecognised commands (10.4%) and 125 to system error (4.3%). No significant differences were observed for speech misrecognition (Z=0.300, p>0.05) between NL and FG. No statistically significant differences were also observed between FG and NL in terms of system error (Z=0.19, p>0.05).

4.4.6.4 Participant Feedback

Natural Language Coding*:* 11 participants stated a preference for NL where positive comments emphasised that the approach was easier to learn, more intuitive, and less constrained:

"... I prefer natural language speech as it was fast and responsive to ... change code" (P20).

"*I found that commands that were clearly interpreted by the NPL mostly did what I asked for. The commands were intuitive and allowed me to do the actions required*" (P11).

Six participants highlighted that they struggled with NL at the beginning, but found it easier after further exposure:

"*after a few attempts it became very easy and fluid to create code ...*" (P18).

Eight participants perceived NL to be harder and more challenging to use than FG – in particular, it was highlighted that the system did not always interpret the participant's intention to write code (despite the voice input being recognised correctly):

"*…Fixed command was easier to write code as the Natural Language could not recognise what I was trying to say*" (P4).

This was likely influenced by the NL model being trained with a limited dataset (182 utterances from the exploratory study). However, 18 participants explicitly reported they felt that NL could be a better approach than FG due to potentially having fewer constraints around the commands that could be issued to the system.

It was observed during the NL evaluation that in most cases, participants provided single instructions at a time (e.g., "*Create a class car").* However, in a smaller number of occurrences (24), participants tried to provide multiple actions joined together (e.g. "*Create a class car and create a function check and add parameters brand and price in the function"*). In 16 cases, participants stated a single command without providing any attributes or values when creating functions, classes, declaring if conditions, and printing values (e.g. "*define class*"). Whilst in this scenario, the system would output an error message, six participants stated they wanted to see the structure of the code with relevant placeholders (e.g. when stating "*create class*", the code "*class [name] { ... }*" would be displayed within the editor – where "*[name]*" is a placeholder that can be edited).

Fixed Grammar: 13 participants preferred FG, with comments emphasising that the inclusion of specific commands made it easier and more structured to use:

"*… (having) defined keywords and rules to follow made it feel more like programming because it was a 'syntax' to follow*" (P2).

"Fixed commands ... seemed to be more reliable as commands were easily interpreted and executed " (P13).

The majority of the participants (N=10) highlighted that the use of example commands helped them to recall the available commands:

"*I found the fixed commands easy to use when I had the help screen open to assist me"* (P5).

"*Made it much easier to have a help portal with the commands to use. For example, I had troubles before creating a loop, but found it easier with the fixed command*" (P12).

However, 12 participants felt that FG was less intuitive and harder to use than NL because of the constrained commands

"*…with the fixed command, I feel it was a lot more constricted with what you can say*" (P21).

It was observed that some of the commands were not recognised by the system, thus participants had to repeat the same command multiple times, which created frustration:

"This process was a bit difficult as sometimes it would not select the line, so i would have to say "up" 3 time or "left " 30 times" (P25).

Six participants suggested including more variations in the commands to make the system more flexible and reliable:

"*… the commands worked as they should but having a variety of commands would make it easier*." (P7).

Speech Recognition Issues: In both NL and FG the recogniser had issues in distinguishing between homophones (similar to Study 1 – Section 3.2.6). To overcome these issues, participants typically used alternative strategies to complete the tasks (e.g. navigating to lines using commands such as "up" and "down"). Whilst this facilitated the completion of tasks, it typically required the use of additional vocal commands. Participants had to use the exact vocal command to trigger any coding functionality in FG, which occasionally led to challenges as they would have to repeat commands multiple times when there were recognition issues. However, NL was more frequently able to interpret the command correctly despite any speech misrecognition. For example, to create a class named vehicle, "*glass vehicle*" would create the class in NL, although for FG, participants had to issue the correct command "create class vehicle". This, therefore, contributed towards the overall number of commands issued in FG being higher than for the NL approach.

Additional Coding Commands: Participants also attempted to use commands such as "copy", "cut", and "paste" on 18 occasions, although the system was not trained to handle these features. Seven participants suggested that incorporating this functionality would support them when writing code. Moreover, when using NL, seventeen participants used commands associated with navigating to the start of the line (i.e. "*go to beginning of line*"), although this was also unsupported by the system. Additionally, nine participants suggested including functionality for removing or selecting a line without specifying the line number (i.e. delete/select the line where the cursor is currently positioned) to minimise numeric value misinterpretation in this context.

4.4.7 Study Summary

Overall, the results from this study found no significant differences between NL and FG in terms of performance and accuracy. However, whilst issues were highlighted around NL (i.e. around the accurate interpretation of user intentions), 18 participants indicated that this method held the significant potential to better support development work if the model could be developed further to become more robust. For instance, it was highlighted that having fewer constraints around the language and commands that can be issued to the system could present significant benefits over FG. Given the current lack of work around natural language coding approaches (in contrast to fixed grammar methods), it was decided to take this approach forward for iterative development and evaluation with developers who have physical impairments to explore further its potential to support coding activities.

4.5 Study 2: Evaluation with Disabled Coders

A multi-session follow-up study was conducted with five developers with physical impairments to investigate their experiences using NL for writing JavaScript code.

4.5.1 Version 2

1832 unique utterances provided by participants during the first evaluation (i.e. commands not previously used for training) were used to further develop the Wit.ai model. Several iterative updates were also made to the prototype based on participants' feedback. In particular, support was added for copying, cutting, and pasting individual words or syntax and single or multiple lines simultaneously. Functionality was also incorporated where a code skeleton is displayed if users omit key properties from a command. For instance, if a user states "create class" without providing a class name, the system would display "*class [name] { ... }*" within the editor (as opposed to an error message).

4.5.2 Participants

Five participants (all male) with ages ranging from 21 to 42 years old (M=28, SD=8.7) were recruited through existing links with the research team. For clarification, none of the participants from the studies detailed in Chapter 3 were involved in this study. Four participants were diagnosed with RSI, and one with Tendonitis/Carpal Tunnel. All participants were native English speakers and had at least a year of coding experience (M=9.5 years, SD=9.8), as well as experience in using speech technologies in their coding environment. Table 4.4 highlights participant demographics and technical background.

4.5.3 Apparatus

The evaluation was conducted remotely online via Zoom (*N=4*) and Microsoft Teams (*N=1*) (depending on the user's preference). All participants used an external microphone and were again asked to use Google Chrome during the evaluation to ensure compatibility with the Web Speech API. Participants also had the choice around which type of switch (i.e. mechanical switch, eye tracker, head tracker, foot pedal) to use to control the speech recogniser, with all of them opting to use the keyboard spacebar (which has been used to control speech recognition in previous studies (Aziz et al., 2021, 2022)).

ID	Age	Impairments	Technical Experience	SA/ SUS Score
P1	23(M)	RSI. Had a week-long rest at some points.	CD: 2.5 years; AT: remapped keyboard to be more ergonomic; Ergonomic keyboard; JE: Expert; CL: TypeScript, Kotlin, Python; IDE: Vim, VSCode, IntelliJ	3 (37.5)
P2	42 (M)	Tendonitis/Carpal Tunnel; difficulty in typing	CD: 24 years; AT: Talon Voice (recently), Kinesis Advantage keyboard; JE: Novice; CL: Clojure and ClojureScript; IDE: Spacemacs.	3(87.5)
P3	21(M)	RSI. Pain in fingertips	CD: 1 year; AT: Specialized mouse and keyboard, speech detecting; JE: Novice; CL: Python; IDE: Brackets, Spyder.	3 (72.5)
P4	23 (M)	RSI, numbness in left arm and fingers	CD: 5 years; AT: Talon; JE: Intermediate; CL: Java, TypeScript, Ruby on Rails; IDE: Neovim	3 (42.5)
P5	31 (M)	RSI due to typing	CD: 15 years; AT: Talon Voice; JE: Novice; CL: Haskell, Racket, Python; IDE: Emacs	3 (45)

Table 4.4: Participant Details - CD=Coding Experience; AT =Assistive Technology; CL =Coding Languages; JE =JavaScript Experience; IDE =Integrated Development Environment; SA=Session Attended; VSCode = Visual Studio Code

4.5.4 Procedure

Institutional Review Board (IRB) approval was obtained prior to the study. Participants were informed that they would need to be involved with three different sessions over the period of (approximately) a week (as well as being encouraged to use the application outside of these sessions). The evaluation consisted of different types of tasks for each session based on difficulty level (as utilised by Rosenblatt et al. (2018) and Van Brummelen et al. (2020)). During the first session, participants were provided with the evaluation link and asked to complete a consent form. They were then administered a survey to collect demographic information, details of their impairments, and information about their technical skills. Participants were then given a brief introduction to the prototype and provided with the practice link. After the practice task, they were provided with the evaluation link and asked to share their screen before proceeding with the tasks.

Tasks for the first session focused on “easy” coding activities that could be completed using 1-2 voice commands (e.g. creating a new class, moving the cursor to a specific position, selecting and deleting single lines or words, adding a single word, and assigning a value to a variable). The second session included more challenging tasks such as selecting and deleting multiple lines or words within a line, creating a new function with multiple parameters, creating an if statement, and rearranging code using cut, copy, and paste features. For the first two sessions, participants were initially shown starting code that would be seen upon starting the task along with the final code. The researcher outlined the actions required to complete the task, and participants clicked a “Start” button to begin. Once the necessary updates were made, they clicked the “Complete” button and moved on to the next task. In the third session, participants were shown a code snippet that included a class, two functions (one with multiple parameters), and multiple variables with different values assigned. They were then asked to write this code on a blank document within the editor. The tasks utilised in the evaluation can be found in Appendix B.5.

After completing the session, participants were provided with a link to freely interact with the prototype outside the timetabled sessions. In the second and third sessions, participants were initially asked about any issues they encountered when using the prototype during their own time. After the completion of each session, a semi-structured interview was conducted with participants to understand their perceptions of the prototype. Moreover, after completing the third session, a SUS form was administered to assess the prototype's usability.

4.5.5 Results

All participants attended the required sessions and successfully completed the evaluation tasks, with sessions ranging from 12-40 minutes. Two participants (P3 and P4) reported using the system on five occasions outside of the main sessions, whilst the other participants only took part in the scheduled sessions.

4.5.5.1 Usability

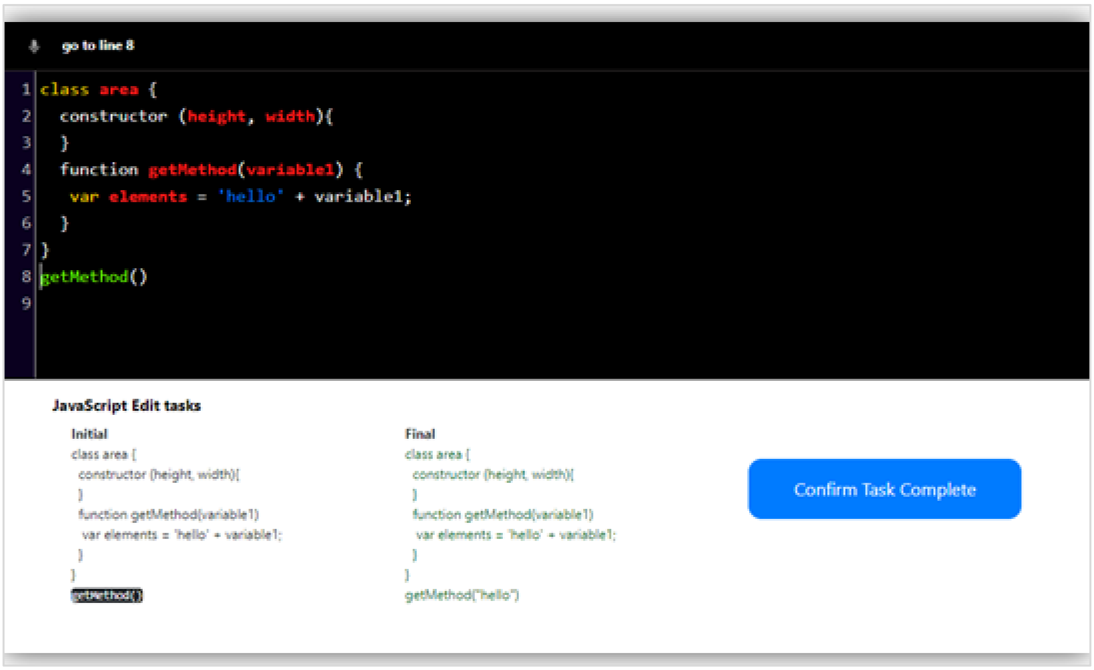
The system received an average SUS score of 57 (SD= 19.51), which can be labelled as "Poor" in terms of usability. P2 and P3 provided SUS scores that can be labelled as "good", whereas P1, P4, and P5 rated the prototype as having a poor level of usability (Bangor et al., 2009).

4.5.5.2 Speech Performance

A total of 682 voice commands were issued by participants across all sessions. 195 commands were related to Speech Misrecognition (28.6%), 90 commands to Model Misunderstanding (13.2%), and 73 (10.7%) were associated with System Error. Whilst a number of commands were related to misrecognition and misunderstanding, the system accurately determined the user's intention in 95% of cases. However, the majority of misunderstanding errors occurred when the model did not accurately interpret the role of entities within an utterance. For instance, if the user stated, "delete line 10" and the system recognised "delete nine 10", the model would accurately identify the intent as "delete line" but would recognise 9 and 10 as entities (thus resulting in lines nine and ten being deleted).

4.5.5.3 Qualitative Feedback

All the qualitative data obtained during the semi-structured interviews and observations were retrieved for analysis. A thematic analysis (Braun and Clarke, 2012) was employed where codes were derived from qualitative data to identify initial themes and concepts. These themes were then iteratively refined and grouped into the five themes detailed below.



```
class area {
  constructor (height, width){
  }
  function getMethod(variable1) {
   var elements = 'hello' + variable1;
  }
}
getMethod()
```

JavaScript Edit tasks

Initial

```
class area {
  constructor (height, width){
  }
  function getMethod(variable1)
   var elements = 'hello' + variable1;
  }
}
getMethod()
```

Final

```
class area {
  constructor (height, width){
  }
  function getMethod(variable1)
   var elements = 'hello' + variable1;
  }
}
getMethod("hello")
```

Figure 4.6: A screenshot of the task interface utilised in the study – text displayed next to the microphone icon highlighted the latest recognised speech command issued to the system. The code used for tasks is displayed in the main coding area, while the area below demonstrated how study tasks were presented to users.

***Overall Experience**:* All participants stated they were impressed with the prototype and were able to utilise the features available to perform coding activities. In particular, two participants (P2 and P3) with novice speech coding experience reported the prototype to be easy and useful to use, as well as emphasising that they would like to continue using the prototype:

> *"…this will help my RSI, so looking forward to incorporating in my editor" (P5).*

Other participants who use existing speech coding tools like Talon Voice (P1, P4, and P5) reported that the tool worked well in the majority of cases, although highlighted that they would like to see improvements in the accuracy of the recognizer, which on occasions could cause frustration (thus likely contributing to some of the lower SUS scores).

Voice Commands: Participants primarily used high-level phrases and abstractions to write code, although on six occasions they dictated specific words representing different syntax constructs (e.g. *"type function constructor open bracket close bracket open curly bracket*."). In contrast to the first study, all participants issued single commands at a time to achieve particular actions (e.g. "*go to line 30*", "*end of the line*", "*backspace*") as opposed to chaining multiple actions together (e.g. "*go to end of line 30 and backspace*"). There were also occasions where participants used different variations of commands to those observed in the previous evaluation – for example, three participants (P1, P4, and P5) used the command "*delete that*" without specifying which line or word to delete. This was likely due to participants' previous experience using Talon Voice which provides support for this particular command. The coding skeletons incorporated into the second prototype iteration were used on eight occasions by participants – copy and paste commands were also widely utilised to support the completion of tasks.

Speech Recognition Initialisation: No significant issues were highlighted in terms of using the spacebar to activate the recogniser. However, participants occasionally forgot to initialise the recogniser, thus resulting in them having to repeat commands once they realised voice input was not being detected. Providing further feedback within the interface regarding the recogniser's status could help to address this issue (e.g. displaying a message if a user is talking, but the recogniser has not been initiated).

System Limitations: All participants occasionally experienced challenges around the system misrecognising terms. Whilst the system was capable of correctly understanding a user's primary intention via misrecognised commands, inaccurate identification of entities (e.g. custom variables or class names) could cause issues and result in users having to repeat commands. To address the issue of recognition challenges when using custom names it may be beneficial to allow users to phonetically verbalise key terms (De León Cordero et al., 2021; Kim et al., 2019). Moreover, due to some latency in relation to the recogniser, participants would sometimes repeat commands when the system was still processing their original command (also resulting in commands being issued on multiple occasions). However, it was observed that participants typically utilised the "undo" command to quickly address any unintentional commands that were actioned by the system.

Code Navigation: Whilst participants were able to write, navigate, delete and select code using the system, a key theme to emerge was a desire from participants to have access to additional navigational features commonly found in mainstream IDEs (such as find, reference navigation, and jumping to definitions). In particular, participants emphasised that existing voice coding tools (e.g. Talon) do not currently provide these features and thus suggested incorporating voice commands to support these syntax navigation approaches.

4.6 Conclusions

This chapter investigated the usability and feasibility of different voice coding approaches (i.e. fixed grammar and natural language commands) to make coding activities accessible for developers with physical impairments. Two user studies were conducted – the first study with 25 non-disabled developers to compare the usability and feasibility of the two approaches and a second multi-session follow-up study utilising natural language commands involving five developers with physical impairments to facilitate the writing and editing of JavaScript code. The first comparative evaluation with developers found that both approaches were perceived positively and enabled participants to complete common coding tasks. Both methods were also found to perform at similar levels in terms of usability and task completion time, although the results indicated that some developers prefer vocal coding to be more like standard programming (with fixed constraints – i.e. the FG version) whilst others prefer the benefits associated with a more natural language approach (e.g. not having to recall specific syntax). However, the majority of participants felt the NL method had good potential if the model could be developed further to become more robust. Hence, the natural language approach was taken forward for further evaluation with developers who have physical impairments.

Results from the second multi-session study (utilising an updated version of the NL prototype) highlighted that the system enabled all participants to complete common coding tasks of varying difficulty. However, the results were mixed around perceptions of usability, with some participants providing higher scores, whilst others were lower. Two participants who provided higher scores both self-identified as being new to voice coding – this potentially suggests that the NL approach can benefit developers who are less familiar with existing voice-based coding tools, although further evaluation work will be required to validate this point. The lower scores can be linked with issues around speech recognition accuracy that was highlighted by participants across both studies and have been noted as a common issue in other related work (Wagner and Gray, 2015; Rosenblatt et al., 2018; Van Brummelen et al., 2020b) In particular, around half of the commands issued by participants resulted in either speech misrecognition, model misunderstanding, or a system error.

To address this point, the training sample would benefit from additional diversity (e.g. input from female developers) to develop it further and improve system recognition of intention

(Stumpf et al., 2020). The prototype also currently uses the Web Speech API (an online automatic speech recogniser), which can present potential challenges around recognition accuracy and delays in obtaining speech-to-text responses (due to latency rates). Offline speech conversion tools such as Mozilla Speech (2023) could help to address this point in future work. Another limitation is that directed tasks were used in the studies to facilitate a controlled comparison across the two approaches, although, in a real-world coding environment, it will not always be clear for developers which step is required next. Further work is therefore needed to explore and validate the findings presented in real-world scenarios over extended periods.

Whilst the natural language prototype was unconstrained in the commands that could be issued, participants commonly stated that they were unsure about which commands to use. In particular, several participants from the second evaluation emphasised that they wanted some "contextual" command suggestions within the interface to help support them. As highlighted by Srinivasan et al. (2019), the use of suggestions can make users aware of the operations to be performed and can help to address the issue of coders potentially guessing vocal commands. The way in which these suggestions are visualised and presented to users is an interesting and underexplored area that holds significant potential to further support developers in this context. Another key area for further work is around investigating how natural language can be used to support other common coding activities (debugging, document navigation, autocompletion). This chapter primarily focuses on how JavaScript code can be written and edited via speech commands, so these wider areas are crucial to explore further to help facilitate the development of more inclusive programming environments.

Overall, this chapter presents new insights around the strengths and limitations of both fixed grammar and natural language multimodal voice coding approaches, thus developing our understanding around the potential of these methods. However, another key theme highlighted by participants was focused around the need for additional navigational features (tailored for voice control) that are commonly found in mainstream IDEs to support efficient code navigation (e.g. find, find all references, and go to definition). The ability to navigate code effectively is a fundamental requirement for facilitating productive workflows, although there has been no work to date investigating how these navigation approaches can be adapted for multimodal voice control and whether they provide any benefits for developers with physical impairments. This area, therefore, forms the focus of the next chapter, where a range of new voice-controlled navigation approaches are developed and evaluated with developers who have physical impairments.

5 MULTIMODAL VOICE INTERACTION FOR CODE NAVIGATION

This work has been published in the Proceedings of the 2022 ACM International Conference on Multimodal Interaction (ICMI '22) as "Inclusive Multimodal Voice Interaction for Code Navigation" (Paudyal et al., 2022)

5.1 Introduction

The ability to navigate code efficiently is an essential activity for common software development activities such as inspecting code, debugging and correcting syntax errors, and code refactoring (LaToza et al., 2006; Baker et al., 2015; Smith et al., 2017; Albusays et al., 2017; Shakil et al., 2019). Research has highlighted that developers spend a significant portion of their time navigating code to support comprehension and maintenance (Ko et al., 2006; Piorkowski et al., 2013; Radevski et al., 2016). Moreover, studies have demonstrated that effective code navigation is crucial to support developers in building a cognitive model of a codebase that can facilitate an understanding of code flow and the location of different code blocks (Begel and Kariv, 2002; Krämer et al., 2013; Henley and Fleming, 2014). Multiple code navigation approaches have been developed to support the efficient location of syntax within a codebase including the use of call graphs (e.g. displaying a tree view of different methods or functions) (Karrer et al., 2011; Krämer et al., 2012), canvas-based navigation (e.g. visualising all methods in a 2D plane) (Bragdon et al., 2010; DeLine and Rowan, 2010; Henley and Fleming, 2014), and a structural navigation approach (e.g. displaying all usages of a method when clicked (Smith et al., 2017) or jumping to a function definition (Shakil et al., 2019)). These navigation approaches are commonly integrated within mainstream development environments (e.g. Visual Studio Code (2023)) where the syntax is typically navigated via a mouse and keyboard.

However, this can present significant barriers for people with physical impairments who may experience challenges using traditional input devices to support coding activities. The use of voice control in conjunction with other input modalities (e.g. mechanical switches and eye gaze, among others) presents an alternative interaction method for enabling disabled developers to navigate code (Begel and Graham, 2005; Rosenblatt et al., 2018), although there has been a lack of work around this area to date. Initial studies have explored simple inline navigation approaches via speech (e.g. moving the cursor to a specific position on a line (Wagner and Gray, 2015; Rosenblatt et al., 2018)) and page scrolling through utilising a grid visualisation (Begel and Kariv, 2002), but there has been less work on other crucial features such as the use of a call graph to navigate and locate specific code blocks (Karrer et al., 2011; Krämer et al., 2012, 2013; Smith et al., 2017). Multimodal voice-controlled methods developed to date are also yet to be empirically evaluated with developers who have physical impairments, thus contributing to a current lack of understanding around the potential of alternative methods to support code navigation.

To address the limited work in this area, this chapter explores the potential of different code navigation approaches optimised for multimodal voice control such as finding references of user-defined code constructs (e.g. variables), jumping to function definitions, and conducting a search for specific syntax. Interviews with five developers who have physical impairments are initially presented to further understand their current methods for navigating code within voice-controlled editors. Findings from this exploratory study were used to inform the development of an editor containing different code navigation approaches that can be operated via multimodal speech input. The navigation techniques were evaluated in a user study with developers who have physical impairments (*N*=14), where it was found that they supported all users in successfully completing a range of standard code navigation tasks. Results also highlighted that the approaches developed were efficient and intuitive to use and demonstrated a high-level of usability. This chapter, therefore, presents three key contributions:

(1) insights from interviews with five disabled developers focused around current code navigation challenges using their existing assistive tools.

(2) a multimodal voice prototype utilising standard code navigation approaches tailored for voice and switch control.

(3) a user study with disabled coders demonstrating the usability and validity of the code navigation approaches developed.

5.2 Exploratory Study

Due to the limited work around voice-based code navigation approaches, it is unclear how developers with physical impairments currently perform code navigation activities when programming. Understanding their current code navigation approaches and challenges is essential to identify intuitive and appropriate code navigation approaches prior to developing new methods. Semi-structured interviews with developers who have physical impairments were therefore conducted to obtain a deeper understanding around their current code navigation strategies.

5.2.1 Participants

Five participants (all male) with ages ranging from 24 to 42 years (M=31.2, SD=6.3) were recruited through existing links with the research team. Four participants were diagnosed with RSI and one with Tendonitis/Carpal Tunnel (Table 5.1). All participants were fluent in English and had at least a year of coding experience (M=10.2, SD=8.7) – they also currently use voice as an interaction approach to perform coding activities.

ID	Age	Impairments	Technical Experience
P1	31 (M)	RSI due to typing	AT: Talon Voice; CE: Novice; CL: Haskell, Racket, Python; IDE: Emacs
P2	42 (M)	Tendonitis/Capral Tunnel	CD: 24 years; AT: Talon Voice (recently), Kinesis Advantage keyboard; CE: Novice; CL: Clojure and ClojureScript; IDE: Spacemacs.
P3	24 (M)	RSI is recovering, difficulty in typing	AT: Talon Voice, Aenea/Dragon; CE: Intermediate; CL: Ellixir, Js, Bash, TypeScript; IDE: Terminal, Tmux, Neovim; TE: 2 years
P4	33 (M)	RSI; Hands and Shoulders	AT: Talon Voice; CE: Intermediate; CL: Java, Python, JS, Haskell, PureScript; IDE: Vim
P5	26 (M)	Occasional RSI flare-ups	AT: Dragon, Mechanical Keyboard, Logitech MX Ergo trackball; CE: Intermediate; CL: JS, TypeScript; IDE: VSCode

Table 5.1: Participant information for the exploratory study- AT =Assistive Technology; CL =Coding Languages; CE =Coding Experience; IDE =Integrated Development Environment; RSI=Repetitive Strain Injury; VSCode = Visual Studio Code

5.2.2 Procedure

Institutional Review Board (IRB) approval was obtained prior to the study. After a brief introduction about the project, participants were asked to navigate to a project information page and were asked to provide informed consent. The study consisted of an initial questionnaire to capture demographic information, coding experience, and the nature of participants' impairments (Appendix C.1). Semi-structured interviews were then conducted on Slack (2023) (a team collaboration tool) and focused on exploring challenges participants experienced in relation to navigating code, any strategies used to overcome barriers, and future features they felt could support them further (Appendix C.2). All responses were captured and recorded (with participants' consent) for later analysis.

5.2.3 Findings

Responses from all participants were retrieved for analysis. A thematic analysis was employed, where codes were derived from retrieved data to identify initial themes and concepts based on the concepts. These themes were then analysed iteratively and were grouped into the three key themes detailed below.

Navigation Challenges: All participants highlighted that they use speech-based tools (e.g. Talon (2023), Dragon (2023)) for development work and emphasised that their existing tools provide limited navigation features. For instance, these tools only provide commands for inline navigation, thus resulting in the majority of participants relying on keyboard shortcuts (e.g. "*:x*" to move the cursor to line x, *"/ word*" to search for the term "word"):

"*…I normally code in Vim, so I use tags to jump around a bit, but it's not really as nice as "go to definition" features in IDEs like IntelliJ*" (P4).

Additionally, three participants highlighted that custom voice commands for code navigation with speech-based tools are not intuitive and can be challenging to learn (e.g. "*slap*" will move the cursor to the end of a line, "*command up*" will move the cursor to line 1). Whilst tools such as Talon support customisation of commands, participants highlighted that this process can be time-consuming and requires a broader understanding of technical skills (such as configuring a microphone and coding environment) and can be a frustrating experience:

"For Talon it is hard. If you don't have technical expertise, it is hard to configure your own customized commands..." (P2).

Navigation Workarounds: All participants reported that they use additional tools (e.g. an eye tracker, keyboard) in conjunction with speech interaction to help overcome navigational barriers:

"I have an eye tracker which lets me do things like ctrl-click to follow definitions and such..." (P2).

Similarly, P3 stated:

"*. . .I use relative line numbers in vim, so if I want to jump to a particular line, I can say 26 down easily. I can also type fx or say fine plex to jump to the next x character … for scrolling I might just use the trackpad 2 finger scroll*".

Moreover, two participants stated that they use key bindings (e.g. "*gd*" to go to definition, "*gg*" to navigate to the first line) to map with their IDE navigation features to overcome barriers:

"*. . . with Talon I use the Spacemacs key bindings which would be gd to move to definition*" (P5).

Future Enhancements: All participants expressed a desire for navigation approaches that were less reliant on additional modalities such as a keyboard:

"*…if I can use just voice to search or see the list of all search results without keys, it will save lot of time*" (P1).

Four participants highlighted a requirement for voice optimised approaches that facilitate using common navigation features found in traditional development environments, such as "*navigating to a definition*" and "*listing all references*" of a specified search item. Participants also emphasised that any new approaches developed using solely voice as the primary method of interaction should be simple and intuitive "out-of-the-box" instead of requiring additional configuration (which can potentially be tedious and time-consuming).

The findings from this initial study provided a solid starting foundation for the project and informed the focus of new multimodal voice code navigation approaches that support a wider range of features (such as navigating to definitions and listing references) that have not been explored in the existing literature to date.

5.3 Research Prototype

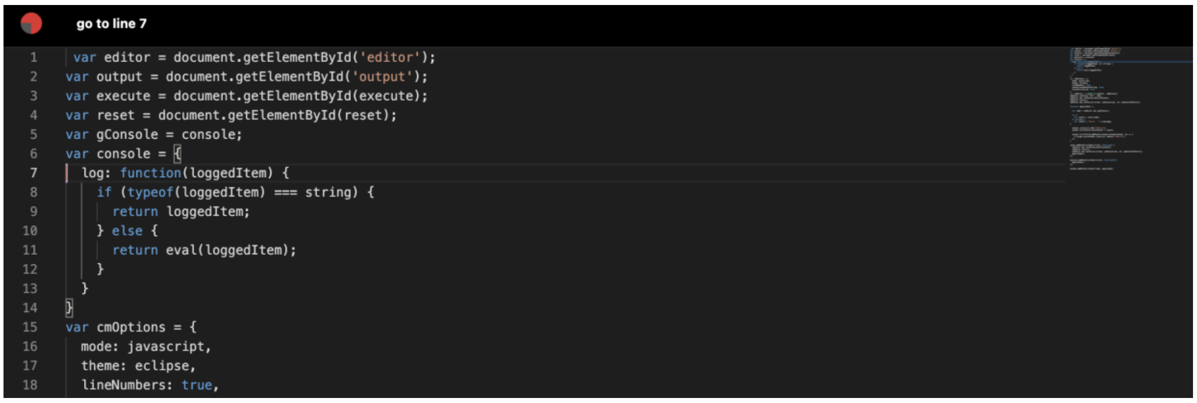
Drawing from prior research and insights collected from the exploratory study with developers, a research prototype integrating different code navigation approaches tailored for multimodal voice interaction was developed.

5.3.1 Prototype Design

The prototype was built-upon an open-source JavaScript based editor (Monaco Editor (2023)) that provides a range of standard coding features. Similar to the prototypes outlined in Chapters 3 and 4, the system architecture consists of three components: (1) a speech recognition interface to convert speech into text (using the WebSpeech (2023)), (2) a command interpreter to parse a user's intent and command (using Wit.ai (2023)) and (3) an execution method to process code navigation actions based on a user's vocal commands. The system was trained using navigation commands highlighted in previous work (Rosenblatt et al., 2018), as well as variations of the common names of features available in widely-used IDEs (such as "*Find all References*" and "*Go to Definition*" in Visual Studio

Code and "*Find Usages*" in JetBrains, along with some alternatives – e.g. "*find reference*", "*all reference*" and "*definition*").

The interface design resembled the layout of existing mainstream development environments (e.g. Atom (2023), Brackets (2023), Visual Studio Code (2023)) and utilised a similar theme to the default one used in Visual Studio Code (Figure 5.1). The header area within the interface contains a microphone icon by default and provides feedback in relation to the speech commands that the user has issued. The recognition system can be initiated through using an external mechanical switch (or the space key on a standard keyboard), thus resulting in the microphone icon subtly pulsating to highlight that the system is "listening" for input. Once a user completes an utterance, the latest speech input recognised by the system is positioned next to the icon to provide feedback to users. An error message is displayed below the speech input if the voice recogniser fails to understand the command. The area below the header includes the main editor, where syntax can be written, navigated, and edited. A list of example commands is also available and can be displayed on the right side of the editor via the "help" vocal command.


go to line 7

```
1   var editor = document.getElementById('editor');
2   var output = document.getElementById('output');
3   var execute = document.getElementById(execute);
4   var reset = document.getElementById(reset);
5   var gConsole = console;
6   var console = {
7     log: function(loggedItem) {
8       if (typeof(loggedItem) === string) {
9         return loggedItem;
10      } else {
11        return eval(loggedItem);
12      }
13    }
14  }
15  var cmOptions = {
16    mode: javascript,
17    theme: eclipse,
18    lineNumbers: true,
```

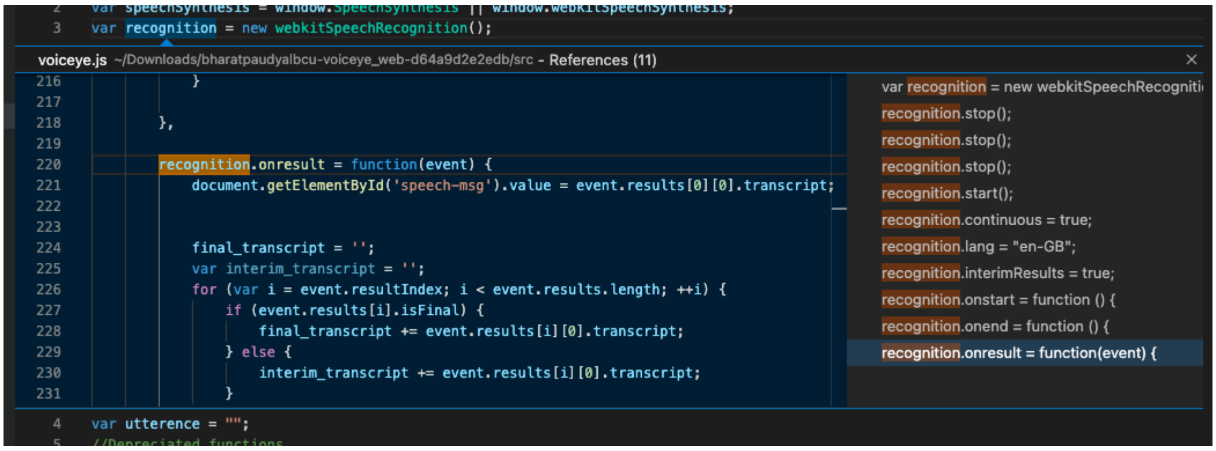
Figure 5.1: The editor interface – at the top of the interface, a red spinner icon indicates that the recogniser is listening. The "go to line 7" voice command has been issued by the user, and the corresponding action has been performed.

Common code navigation features found in mainstream development environments were also integrated into the interface and tailored for speech interaction. This includes standard approaches such as navigating to specific lines (e.g. "*go to line x*", "*down x*"), positioning the cursor position within a line of code (e.g. "*left x*", "*right x*"), and jumping to the start or end of a line. Additional navigation approaches such as "*go to definition*", "*find all references*", and "*find*" were also incorporated - these are widely used features within mainstream editors (Visual Studio Code, 2023; Brackets, 2023), although no work to date has explored how they can be adapted for speech interaction or whether this can present any interaction benefits for developers with physical impairments. The following sub-sections provide further detail on the design of each of these key features and how they were tailored for voice control (a list of all navigational speech commands can be found in Table 5.2).

5.3.1.1 Find all References

In modern IDEs, the "Find all References" feature typically displays the list of all relevant references in relation to an identifier specified by the user via keyboard input (e.g. a variable, function). Figure 5.2 demonstrates how this feature operates within Visual Studio Code – in this scenario, a user has placed the cursor around their desired identifier "recognition" (at line 3) and initiated the "Find all References" feature through either a keyboard shortcut (Shift + F12) or via the context menu (triggered through a right mouse click). This results in a list of references being overlaid on the main editor that can be navigated through either directional keys on the keyboard or via direct mouse selection. Once a reference has been selected from the list, a preview of the relevant code is displayed in the main editor window, and a user can then navigate to their desired location.

To tailor this approach for speech interaction, users can initially position the cursor (via the voice commands available) to a specific identifier within the code (e.g. a variable, function name). They can then issue a command to activate the "Find all References" feature (e.g. "*reference*", "*find reference*", among others), and a list of references is displayed on the right side of the interface (Figure 5.3). To remove the requirement for mouse and keyboard input to navigate the list, each list item is mapped with a unique numeric value which users can verbalise to select their desired reference (e.g. if "11" is issued as a vocal command, the 11th item will be selected). This results in the relevant code being previewed in the main editor (similar to Visual Studio Code) – users can then issue a "select" vocal command to complete the navigation and reposition the cursor at the appropriate reference point (within the editor area).



```
var speechSynthesis = window.speechSynthesis || window.webkitSpeechSynthesis;
var recognition = new webkitSpeechRecognition();
```

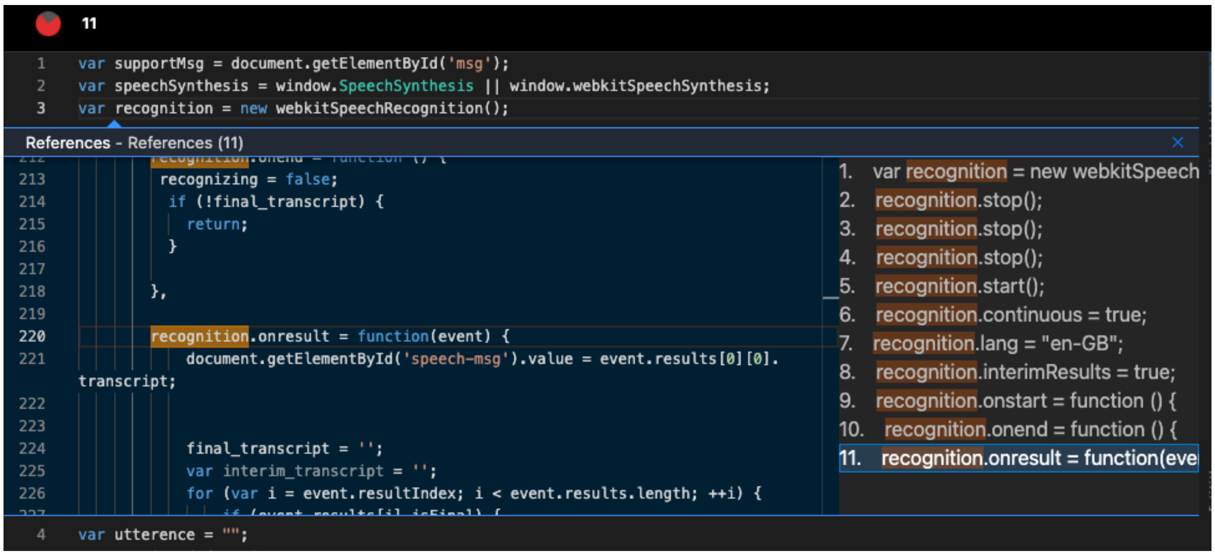
voiceye.js ~/Downloads/bharatpaudyalbcu-voiceye_web-d64a9d2e2edb/src - References (11)

```
            }

        },

        recognition.onresult = function(event) {
            document.getElementById('speech-msg').value = event.results[0][0].transcript;


            final_transcript = '';
            var interim_transcript = '';
            for (var i = event.resultIndex; i < event.results.length; ++i) {
                if (event.results[i].isFinal) {
                    final_transcript += event.results[i][0].transcript;
                } else {
                    interim_transcript += event.results[i][0].transcript;
                }
```

```
var recognition = new webkitSpeechRecogniti
recognition.stop();
recognition.stop();
recognition.stop();
recognition.start();
recognition.continuous = true;
recognition.lang = "en-GB";
recognition.interimResults = true;
recognition.onstart = function () {
recognition.onend = function () {
recognition.onresult = function(event) {
```

```
var utterence = "";
//Depreciated functions
```

Figure 5.2: Find all references utilised in Visual Studio Code



11

```
var supportMsg = document.getElementById('msg');
var speechSynthesis = window.SpeechSynthesis || window.webkitSpeechSynthesis;
var recognition = new webkitSpeechRecognition();
```

References - References (11)

```
        recognition.onend = function () {
            recognizing = false;
            if (!final_transcript) {
                return;
            }

        },

        recognition.onresult = function(event) {
            document.getElementById('speech-msg').value = event.results[0][0].
transcript;


            final_transcript = '';
            var interim_transcript = '';
            for (var i = event.resultIndex; i < event.results.length; ++i) {
                if (event.results[i].isFinal) {
```

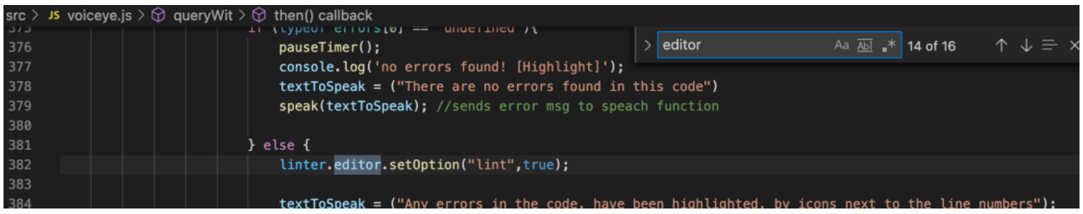
```
1.  var recognition = new webkitSpeech
2.  recognition.stop();
3.  recognition.stop();
4.  recognition.stop();
5.  recognition.start();
6.  recognition.continuous = true;
7.  recognition.lang = "en-GB";
8.  recognition.interimResults = true;
9.  recognition.onstart = function () {
10. recognition.onend = function () {
11. recognition.onresult = function(eve
```

```
var utterence = "";
```

Figure 5.3: Find all references tailored for multimodal voice control

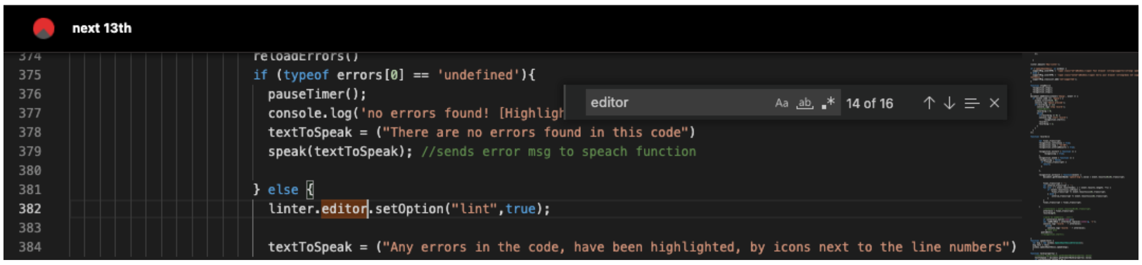
5.3.1.2 Find / Search

A “Find” feature is also a common component of mainstream IDEs and enables users to search for code through providing a search term (via the keyboard). For example, in Visual Studio Code, developers can initiate the tool either through a menu item or via a keyboard shortcut (CTRL/CMD + F). Users can then enter their search terms (e.g. “*editor*”) into a text field and press the enter key to navigate to the first instance of any search results. If there are multiple instances of the search term, users can navigate them through either selecting the enter key to move to the next result or using the mouse to directly select the arrows within the “Find” area located towards the top of the interface (Figure 5.4). This feature was adapted for speech interaction through enabling the user to issue an associated verbal command (e.g. "*find x*", "*search x*") to search for the desired term. The search would then be performed, and a similar "Find" box would be displayed towards the top of the interface. Users can then navigate through search results (where there are multiple occurrences of the search term) through stating the number of instances they would like to select. Figure 5.5 shows the implementation of the "*Find/Search*" feature in the research prototype – the user has issued the command "find editor", which has performed a search for the keyword "editor" and returned 16 results. The user then issued the command "next 13th", which resulted in navigating to the 14th search result.



```
        if (typeof errors[0] == 'undefined'){
            pauseTimer();
            console.log('no errors found! [Highlight]');
            textToSpeak = ("There are no errors found in this code")
            speak(textToSpeak); //sends error msg to speach function

        } else {
            linter.editor.setOption("lint",true);

            textToSpeak = ("Any errors in the code, have been highlighted, by icons next to the line numbers");
```

Figure 5.4: The 'find/search' function in Visual Studio Code. The user has entered the term 'editor' where 16 occurrences are found - the user has to press the 'enter' key to move to the desired position.

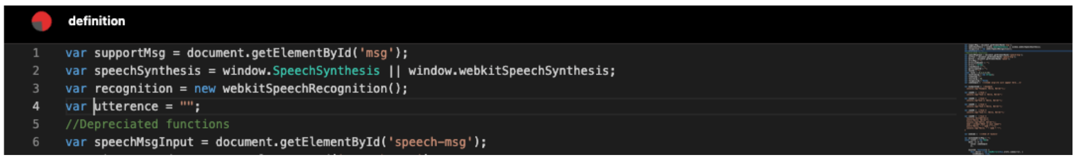


```
        reloadErrors()
        if (typeof errors[0] == 'undefined'){
            pauseTimer();
            console.log('no errors found! [Highligh
            textToSpeak = ("There are no errors found in this code")
            speak(textToSpeak); //sends error msg to speach function

        } else {
            linter.editor.setOption("lint",true);

            textToSpeak = ("Any errors in the code, have been highlighted, by icons next to the line numbers")
```

Figure 5.5: Implementation of the 'Find' feature for multimodal voice interaction.

5.3.1.3 Go to Definition

The "Go to Definition" feature within mainstream IDEs enables developers to navigate directly to the definition of an identifier (e.g., a variable, function). To use this feature, the user typically has to place the cursor around the identifier and use a keyboard shortcut (e.g. F12 in Visual Studio Code) or select the tool via the context menu (accessed via right mouse click). To tailor this for voice interaction, the user can place the cursor around an identifier (via the available voice commands) and then issue a command to initiate the "Go to Definition" feature (e.g. "definition", "define"). The cursor then jumps to the appropriate definition of the user-selected identifier, and the navigation is completed (Figure 5.6).



definition

```
1  var supportMsg = document.getElementById('msg');
2  var speechSynthesis = window.SpeechSynthesis || window.webkitSpeechSynthesis;
3  var recognition = new webkitSpeechRecognition();
4  var utterence = "";
5  //Depreciated functions
6  var speechMsgInput = document.getElementById('speech-msg');
```

Figure 5.6: Implementation of the 'go to definition' feature. The cursor was initially at line 152 and placed over the variable "utterance" - the verbal command 'definition' has been issued resulting in the cursor jumping to line 4.

Whilst adopting these common IDE features for voice interaction potentially presents new opportunities for code navigation, an important next step was to formally evaluate them with developers who have physical impairments to explore the viability of the approaches developed.

Features	Usage	Example utterances
Go to Definition	To navigate to the definition of Identifier (function/class/variable)	'define', 'go to definition', 'definition'
Find all references	To display all the possibilities of a selected Identifier	'reference', 'get reference', 'refer'
Search	To find/search a word using a caret	'find getarea', 'search function hello'
Select Next Item	To move forward to the next item	'next', 'next 12', 'next 5th '
Select Previous Item	To go back to the previous item	'previous', 'move previous', 'previous 2nd'
Left/Right/Up/Down	To navigate to the desired position across the code caret. Defaulted to one position if the position is not provided	'left', 'right 20', 'up 10', 'move 10 position left', 'go 20 position down'
Navigate to specific line	To navigate between the lines of code	'go to 10', 'go to line 20'
End of line	Navigate to the end of the current line	'end of line', 'end line', 'line end'

Table 5.2: List of features along with example vocal commands used for navigation.

5.4 User Study

A user evaluation was conducted to investigate the potential of the multimodal voice-controlled code navigation techniques developed to support developers with physical impairments.

5.4.1 Research Question:

The research question below was developed to address the limited work completed around the potential of multimodal speech and switch interaction to facilitate the code navigation process:

RQ: How can a multimodal speech and switch interaction approach facilitate efficient code navigation?

The broad focus of this question facilitated an exploratory research approach to help develop a deeper understanding around the challenges and opportunities associated with this method of interaction within a coding context.

5.4.2 Participants

14 participants with ages ranging from 21 to 55 years old (M=33.3, SD=10.9) were recruited via online advertisements posted within disability groups and through existing links with the research team. 12 participants were diagnosed with RSI, one with Cerebral Palsy, and another with Tendonitis/Carpal Tunnel. All participants had at least a year of coding experience (M=13.0, SD=12.1), whilst 12 participants had existing experience in using speech interaction to support coding activities (Table 5.3). For clarification, two participants from the second study of Chapter 4 were involved in this study. 10 Participants were native English speakers – the other native languages included Hebrew, Dutch, Tamil and Hindi.

5.4.3 Apparatus

All study sessions were conducted remotely via Zoom (2023) and Microsoft Teams (2023). Participants were required to use the Google Chrome browser to ensure compatibility with the Web Speech API. Participants were also required to use their own microphones for voice input, as well as an additional device of their choice (e.g. keyboard, mechanical switch) to trigger the speech recogniser.

5.4.4 Procedure

Institutional Review Board (IRB) approval was obtained for the study. The researcher initially met online with participants (via Zoom or Microsoft teams - whichever was most suitable for participants) and provided a link to the prototype, along with a brief introduction to the project and a demonstration of the prototype. All participants were informed that they could use their existing assistive input devices to control through the research prototype (e.g. selecting buttons for starting and completing tasks) and initiate the speech recogniser. However, they were still required to use the speech commands available within the prototype for completing the navigation tasks (via their microphone). Four participants opted to use an eye tracker, while the remainder confirmed they would use a keyboard as an additional modality. They were then requested to complete a consent form and were redirected to the pre-test questionnaire to collect some demographic information, as well as details of their impairments and technical skills (Appendix C.3). Participants then completed training tasks (e.g. moving to different lines/positions, finding references and moving to a specified position) for approximately 10 minutes to ensure familiarity with the voice-controlled navigation approaches.

Following the practice session, participants started to work on the main experimental tasks. The study tasks were designed and categorised based on real case debugging scenarios utilised in previous studies (Baker et al., 2015; Rosenblatt et al., 2018). Similar to Shakil et al. (2019), participants were presented with three code snippets of varying lengths (i.e. "short" – 71 lines, "medium" – 535 lines, and "long" – 2092 lines) where each snippet consisted of 5-7 syntax errors requiring 12 navigational to uncover the bug for each code snippet (36 in total). The code snippets were taken from an online (open source) code repository (Chhekur, 2023) with common JavaScript syntax errors integrated (i.e. undefined methods, incorrect method parameters, typographic errors) (Ocariza et al., 2013; Hanam et al., 2016). The tasks utilised in the study are presented in Appendix C.4. This approach was adopted to present realistic navigation scenarios where manipulation of the onscreen cursor was required over shorter and longer distances within different code snippets.

Participants were initially presented with one of these snippets and then required to navigate to each error embedded within the file. To eliminate code comprehension time impacting task completion, participants were provided with the exact steps required for navigating to errors (thus utilising the same approach used in previous related studies) (Shakil et al., 2019; Sengupta et al., 2020). For instance, at the start of each task, participants were initially shown a screenshot of the "starting" position of the cursor, alongside a second screenshot highlighting the location of a syntax error and the required final cursor position. The tasks associated with each code snippet were developed to encourage using the different navigation features integrated within the prototype. In particular, there were two instances in each code snippet which required participants to use the "*Find All References*" feature (e.g. "*Find all the references of the method* taskShuffle and select the 9th Instance"), two instances associated with "*Go to Definition*" (e.g. "*Navigate to the definition of CreateMarker*") in each code snippet, and two instances where the use of the "Find" tool was required to successfully complete the task (e.g. "*Find the variable editor*"). Whilst an emphasis was placed on these three features, tasks were also designed in a way where further navigation commands were required to complete them (i.e., *"go to line", "left", "right", "up", "down"* and "*end of the line*").

Once participants had familiarised themselves with a task, they were then required to work on the task using the commands available and selected a "Complete Task" button once they had completed the necessary navigation step. Once all 12 navigation tasks for a code snippet were completed, participants would move to the next snippet, and the process was repeated until all tasks were finished. The order of code snippets was counterbalanced to reduce the potential impact of order effects. Participants were then administered the SUS survey (Bangor et al., 2009) followed by an online open-ended questionnaire exploring their experiences of using the speech-based code navigation features, as well as suggestions for any improvements. A follow-up semi-structured interview was also conducted to discuss any key points further (Appendix C.5).

ID	Age	Impairments	Technical Experience	SUS
P1	42 (M)	Mild RSI- flares up under stress or excessive typing	CD: 25 years; AT: Talon Voice; CE: Expert; CL: iOS, Swift, Unity, C#, Python, SQL; IDE: XCode, PyCharm, VSCode	62.5
P2	21 (M)	RSI, Pain in fingertips	CD: 1 year; AT: Specialized mouse and keyboard, speech detecting; CE: Novice; CL: Python; IDE: Brackets, Spyder.	80
P3	20 (M)	Cerebral Palsy	CD: 4 years; AT: Speech, Ergonomic keyboard; CE: Intermediate; CL: HTML, JAVA; IDE: Dreamweaver, Android Studio	62.5
P4	33 (M)	Tennis elbow in the right arm, finger tendonitis in most fingers of the left hand.	CD: 17 years; AT: Talon Voice with Eye-tracking; CE: Expert; CL: Java, Python, JS, Haskell, PureScript; IDE: Vim	87.5
P5	45 (M)	RSI, Both hands	CD: 6 years; AT: None; CE: Intermediate; CL: Python, Scala, Clojure, JS; IDE: PyCharm	87.5
P6	40 (M)	RSI, pain in wrists from prolonged use of mouse	CD: 20 years; AT: Vertical Mouse; CE: Expert; CL:JS, TypeScript; IDE: WebStorm	90
P7	28 (M)	Hand and shoulder RSIs.	CD: 1 year; AT: Talon Voice, Tobii Eye Tracking; CE: Novice; CL: JS, HTML, CSS; IDE: VSCode	92.5
P8	55 (M)	RSI. Both Hands, Dyslexia	CD: 50 years; AT: MS Dictation, Immersive Reader; CE: Intermediate; CL: C#, Windows; IDE: Visual Studio.	100
P9	21 (M)	RSI Wrist, hypermobility, scoliosis. Chronic inflammation in joints, shoulder, back, arms	CD: 4 years; AT: Talon Voice, Kinesis Advantage Keyboard; CE: Intermediate; CL: C#, Java, Python, C++; IDE: IntelliJ, Notepad++	92.5
P10	43 (M)	Tendonitis/Carpal Tunnel; difficulty typing	CD: 24 years; AT: Talon Voice (recently), Kinesis Advantage keyboard; CE: Novice; CL: Clojure and ClojureScript; IDE: Spacemacs.	95
P11	26 (M)	Limited hand function and motor impairments	CD: 5 years; AT: Speech Recognition; CE: Intermediate; CL: Julia, MATLAB, Java, C, JS; IDE: VSCode	97.5
P12	26 (M)	Occasional RSI flare-ups	CD: 5 years; AT: Mechanical Keyboard, Logitech MX Ergo trackball; CE: Intermediate; CL: JS, TypeScript, NodeJS, Python; IDE: VSCode	70
P13	26 (M)	RSI, Ulnar Nerve Pain	CD: 5 years; AT: Talon Voice; CE: Intermediate; CL: HTML, CSS, JS; IDE: VSCode	57.5
P14	40 (M)	RSI, Wrist and elbow tendon issues	CD: 15 years; AT: Talon/Dragon Voice, Eye Tracker; CE: Expert; CL: Java, Angular; IDE: IntelliJ	92.5

Table 5.3: Participant Details - CD=Coding Experience; AT =Assistive Technology; CL =Coding Languages; CE =Coding Experience; IDE =Integrated Development Environment; RSI=Repetitive Strain Injury; SUS=System Usability Score; VSCode = Visual Studio Code

5.4.5 Measures

Task Completion: Task completion time was measured to gain an indication around the efficiency of the interaction approach developed. Task completion times were measured in milliseconds from when participants started each task (i.e. after selecting the "Start Task" button) until the task had been completed.

Usability*:* Perceptions of usability were measured using SUS (Brooke, 1996; Bangor et al., 2009), which was administered after all navigation tasks had been completed.

Speech Recognition Accuracy: Errors were categorised into two themes: *Speech Recognition* – where the recogniser was inaccurate (e.g. "Go to line 160" misrecognised as "Go to wine 160"), and *Unrecognised Commands* – where the recogniser misinterpreted the intention behind the user voice commands or the command was not recognised (e.g. "write down").

5.4.6 Results

5.4.6.1 Task Completion Time

All participants were able to complete all the tasks and utilise the navigation features within the prototype. Task completion times ranged between 6:24 minutes to 17:16 minutes with an average time of 11.9 minutes (SD=3.5 minutes). Each snippet took under 5 minutes on average to complete, with navigation tasks associated with the short code snippet taking an average time of 4.7 minutes (SD =1.8 min), tasks for medium length code requiring an average time of 3.6 minutes (SD=1.45 min), and navigation steps for the long code snippet taking 3.6 minutes on average (SD=1.2 min).

5.4.6.2 Usability

The navigation features also obtained an average SUS score of 83.4 (SD=14.3), which can be labelled "Excellent" (Brooke, 1996; Bangor et al., 2009).

5.4.6.3 Speech Performance and Analysis

A total of 1173 vocal commands were issued (Short: 429; Medium: 382; Long: 362). 230 commands were related to speech misrecognition (19.6%), while 150 commands (12.8%) were associated with unrecognised commands. Some of the common speech misrecognition commands included "*write*" instead of "*right*", "*done*" instead of "*down*", and "*to*" instead of "*two*". Despite misrecognition in a small number of cases (*N*=30), the model could still identify the correct intention of the command issued by the user. For instance, in multiple cases, "*right*" was misrecognised as "*write*", although the model would accurately identify the intent as "right" and would perform the appropriate action.

5.4.6.4 Subjective Feedback

All qualitative data obtained during the semi-structured interviews and observations were retrieved for analysis. A thematic analysis (Braun and Clarke, 2012) was employed where initial groupings where developed based on data collected and then iteratively refined into the following themes.

Perceptions of "Find All References"

All participants provided positive feedback in relation to the "Find all References" navigation approach, with comments emphasising that the approach was intuitive, easy to use, and time-saving:

"*The navigation for references and definitions is very, very straightforward . . . and saves a tremendous amount of time*" (P8).

Eight participants highlighted that this feature is not available in other voice coding editors, with four participants (P1, P5, P6, P14) explicitly stating that they would like the feature integrated into their current IDE:

". . .*I will have to wait for this to come to PyCharm. I think that would be remarkable*!" (P5).

Seven participants commented that the ability to verbalise numeric values to support the selection of references was a useful feature - in particular, the option to traverse the item position via absolute numbers (i.e. verbalising the exact numeric value, e.g. "5", "9th") and relative numbers (i.e., verbalising the number with respect to the currently selected item position, e.g. "next 6th", "previous 5") was useful. An issue highlighted by three participants was in relation to speech recognition (i.e. in terms of some of the commands not being appropriately recognised), with participants suggesting the possibility of incorporating customisable vocabulary where commands can be tailored depending on user preferences:

". . .*In Talon I can customise my own thing and has less recognising bugs, so including custom grammar can improve the accuracy of recogniser can be boost the performance*" (P13).

Perceptions of "Find"

Eleven participants provided positive comments in relation to the "Find" feature, with comments focusing on simplicity and ease of use. Four participants stated that the tool was particularly useful in terms of searching for multiple terms within a single search query - for instance, P5 was positive about the "Find" feature, although they felt it might present challenges in some scenarios:

". . .*the voice search worked remarkably well in finding variable/function names which consist of multiple concatenated words . . . but maybe it will be less so for more complex word combinations…*" (P5).

P9 re-iterated this point and highlighted that the tool could be improved further through providing users with the ability to enter individual characters via speech to address challenges in pronouncing custom identifiers:

". . .*would be nice to have an option to manually type letter by letter for find - since not all keywords lend themselves to easily identified pronunciation*" (P9).

Similar to the "Find all References" tool, five participants highlighted the option to select a specific instance of a search result (e.g. "5", "5th") in addition to relative commands (e.g. "previous 2", which would select the instance two places before the currently selected result) to be particularly useful and easy to use:

". . .*The option of moving forward/backward and directly jumping to a number makes it easier . . . this is not present in Talon*" (P13).

Two participants suggested that commands such as "go back" and "go forward" would also be useful additions to support the efficient navigation of search results.

Perceptions of "Go to Definition"

Nine participants provided positive feedback on the use of the "Go to Definition" feature, with comments highlighting that it was useful, intuitive, and easy to use:

"*…I found those features to be useful. I have not seen them in any other packages…*" (P11).

Three participants suggested incorporating similar features (e.g. "navigating to local declaration", "navigating to inner block within curly braces"), which exist in other coding tools (i.e. Vim) to further enhance accessibility:

"*Use the commands of vim like gd to navigate to local declaration, fx to find next occurrence*" (P10).

Two participants suggested integrating a feature for multiple file navigation to enable jumping directly to the definition of the main identifier located in a different file.

Suggestions for iterative development

All participants stated they were impressed with the navigation approaches and were able to successfully utilise them to complete the tasks. Four participants suggested potential improvements around the accuracy of the recogniser through incorporating a local speech recogniser. Five participants highlighted that the inclusion of a scrolling feature via voice would also help them effectively navigate to different sections of a code listing. Three participants suggested including a feature to navigate to a specific position of a line (e.g., "center of line x", "end of line x"). Three participants also felt a "page up/down" feature would be beneficial to support navigation within a code listing.

5.5 Conclusions

This chapter has addressed the research question highlighted in Section 5.4.1 through the development and evaluation of a multimodal system to support people with physical impairments with navigating syntax. Prior research on voice-based coding approaches has primarily focused on writing and editing code, as opposed to different code navigation techniques. The small number of research prototypes that do allow for voice-controlled navigation typically only support simple functionality such as jumping to a specific line (e.g. "line x") or inline navigation (i.e. "left" to move the cursor one position to the left) (Begel and Graham, 2005; Wagner and Gray, 2015; Rosenblatt et al., 2018). No previous studies have explored the potential of additional commonly used code navigation techniques with mainstream IDEs (such as "Find All References", "Go to Definition", and "Find" (Smith et al., 2017; Shakil et al., 2019)) within a voice coding context. Furthermore, no research has investigated or evaluated the efficacy of voice-based source code navigation for developers with physical impairments. This chapter addresses the lack of work in this area to date through presenting a novel prototype that integrates widely used code navigation approaches that have been tailored and optimised for multimodal speech interaction.

The results from a user evaluation with developers who have physical impairments found the code navigation approaches intuitive and easy to use, with SUS scores indicating an excellent level of usability. These findings build on other related work in the field – for instance, Rosenblatt et al. (2018) used navigation commands such as "go to line" and "go x left" in their VocalIDE application (which was also received positively by participants), although their work did not include the common navigation features investigated in this study. Moreover, whilst navigation features such as "Find all References" and "Go to Definition" have been explored and evaluated within systems supporting eye gaze interaction (Shakil et al., 2019), no previous work has investigated these types of features in relation to voice coding. This work also confirms that multimodal voice is a feasible approach in facilitating a wide range of code navigation approaches and can help developers with physical impairments to reduce their dependency on traditional input devices (i.e. a keyboard and mouse). Furthermore, the majority of the participants from the study *(N=11)* who use Talon as their primary voice coding tool expressed a desire to integrate the approaches into existing mainstream IDEs (such as Visual Studio Code and Talon) to support their development workflow. This work, therefore, addresses key challenges associated with the navigation of code via voice interaction through presenting fully functional techniques that can be utilised by developers with physical impairments.

One limitation of the work is the accuracy of speech recognition (which is a known issue within the field) (Rosenblatt et al., 2018; Kim et al., 2019; Van Brummelen et al., 2020b) - the system currently utilises a cloud-based speech recogniser which can present potential challenges around accuracy and recognition delays due to network and latency issues. As suggested by participants, the system could be integrated with the local Talon recogniser (stored on the client-side) to help further address these issues. Scrolling through code via voice control is another important area where there has been a lack of work completed to date. Moreover, the model is currently trained with a limited range of samples and could be developed further through integrating the terms collected during the evaluation, as well as through additional samples from a broader and more diverse user base (e.g. female developers, non-English speakers).

The next chapter will present the key contributions and conclusions of the research conducted across the thesis, as well as a discussion around limitations associated with the work completed and future important research areas that now require further investigation.

6 Discussion and Limitations

The first chapter highlighted a lack of work on multimodal coding approaches to support people with physical impairments in writing code, as well as the potential benefits such an approach could provide to present more inclusive development experiences. This chapter expands on the findings from the user studies presented in Chapters 3-5 by delving into transferable findings and their implications for the wider disability community. In this chapter, we first discuss the main contributions of the thesis with a particular focus on the research questions stated in Chapter 1. The chapter concludes with limitations that are based on the findings in the three user studies.

6.1 Main Contributions

This thesis focused on exploring different multimodal approaches (i.e. different combinations of speech, gaze, and mechanical switches) to support inclusive development interactions. Research studies were focused around the three research questions highlighted in the introductory chapter. These questions are highlighted again below along with details of how the research completed has developed our understanding of these key areas.

RQ1: How can speech recognition, eye gaze tracking and mechanical switches be combined as a multimodal interaction approach to support coding activities for people with physical impairments?

Research in Chapter 3 investigated this question through the development of a multimodal coding approach combining speech, gaze, and mechanical switches. An exploratory study was conducted with 29 non-disabled developers to measure the feasibility and usability of the system where the results found that all participants were able to complete a series of tasks and the prototype was rated as "OK" in terms of usability. Subjective feedback also highlighted the intuitiveness and usability of the approach with the majority of participants providing positive feedback and comments. Similarly, results from a follow-up study with five developers who have physical impairments found coding via multimodal input control to be simple and easy to use with SUS scores indicating that the system has a good level of usability score, as well as highlighting that the approach provided advantages over their existing interaction approaches. This combined speech and gaze interaction with switch controls demonstrates how they can work together to support individuals with physical impairments for coding purposes. Gaze interaction can address speech recognition problems, while, on the other hand, speech input can manage selection, navigation, and deletion of syntax, addressing difficulties in eye gaze activation of small targets. Additionally, it was noticed on several occasions that participants were using the gaze-based onscreen keyboard to control the interface rather than using speech and also issued vocal command to fix syntax and typographical errors as opposed to using gaze. This highlights the necessity for adaptable interaction, enabling developers to effortlessly switch between gaze and speech based on individual preferences, mood, or coding needs.

This research therefore highlights the potential of using a multimodal approach to support coders with physical impairments, as well as findings around the challenges and issues this type of interaction method presents in this context.

RQ2: To what extent can natural language and fixed grammar voice coding approaches support the writing, editing, navigation, and selection of code?

The research highlighted in Chapter 4 addressed this question through the development of two coding approaches – one that utilised a fixed grammar and another using a natural language method. An exploratory study was initially conducted to gain an understanding of typical commands that developers would prefer to use when performing coding activities via voice. Informed by the findings from this study, a novel coding system providing users with the ability to perform coding activities via a fixed grammar and natural language approach was developed. The first comparative evaluation with non-disabled developers found that both approaches were perceived positively and enabled participants to successfully complete common coding tasks. Both methods were also found to perform at similar levels in terms of usability and task completion times. However, the results indicate that the majority of participants felt that the natural language method had good potential if the model could become more robust and also highlighted that this approach was intuitive and easy to learn.

Taking this into consideration, it was therefore decided to take the natural language approach forward for further development and evaluation within a multi-session evaluation with five developers who have physical impairments. Results from this study emphasised that the system enabled all participants to successfully complete common coding tasks of varying difficulty. However, the results were mixed around perceptions of usability with some participants providing higher scores, whilst others were lower. Two participants who provided higher scores both self-identified as being new to voice coding – this potentially suggests that the natural language approach can provide benefits for developers who are less familiar with existing voice-based coding tools, although further evaluation work will be required to validate this point. Overall, results from the user study validated that natural language can be a feasible approach in performing coding activities for this target audience, in addition to identifying some additional challenges that require further work moving forward (e.g. training the natural language model with the dataset obtained during the first evaluation, the addition of coding features such as "copy", "cut", "paste", and the generation of a skeleton code if an identifier is not provided).

The research conducted in these studies demonstrates that both fixed grammar and natural language voice coding can facilitate standard coding activities. Furthermore, the work highlights the viability of natural language coding via a multimodal voice approach, as well as highlighting new insights around the challenges of this approach.

RQ3: How can a multimodal speech and switch interaction approach facilitate efficient code navigation?

The research presented in Chapter 5 addresses this question through presenting a novel prototype that explores how commonly used code navigation approaches (e.g. jumping to function definitions, conducting a search for specific syntax, scrolling techniques) can be optimised for multimodal voice interaction. The system design was initially informed through an exploratory study with five physically impaired developers to elicit insights around their experiences in navigating code within existing voice-controlled development environments. This exploratory study informed the design of a code editor integrating different navigation features tailored for multimodal speech input. Results from a user evaluation with 14 developers who have physical impairments found that the code navigation techniques were intuitive to use and rated as "Excellent" in terms of usability, as well as being perceived positively overall. Several participants expressed their desire to integrate these features within their existing coding tools, which demonstrates the importance and novelty of this work. Participants also highlighted key challenges that require further work, such as voice-controlled scrolling features and navigating to the identifier located in multiple files. This research, therefore, presents new contributions and knowledge on how commonly used code navigation approaches can be tailored for multimodal voice and switch input, as well as the viability of these approaches for developers with physical impairments. Overall, this work contributes valuable insights into enhancing accessibility and usability for developers with physical impairments in the coding domain.

6.2 Limitations

Whilst the research conducted presents multiple original contributions that have developed our understanding in this area, there are also a range of limitations associated with the work completed. For instance, participants across all studies experienced challenges on occasions with the accuracy of speech recognition, thus resulting in them having to repeat commands multiple times. Furthermore, the research prototypes developed utilised a cloud-based speech recogniser, which can occasionally pose issues in terms of delays in processing (due to latency rates and network issues). These are known challenges in the field (Wagner and Gray, 2015; Rosenblatt et al., 2018; Soto Munoz et al., 2019; Van Brummelen et al., 2020b), although it is important to note that they were not highlighted as major concerns for participants across all studies (who were able to successfully complete all experimental tasks). An alternative approach could involve the integration of offline conversion tools (e.g. Mozilla Speech (2023), Wav2letter (2023)), which could potentially help to address these issues (particularly in relation to latency), although further testing is required to examine whether this presents any interaction benefits.

Similar limitations were also observed in relation to the eye tracker when evaluating the multimodal prototype detailed in Chapter 3. In particular, calibration drifts were observed during this study as participants moved their heads and seating position over the duration of the study (these challenges have also been highlighted in previous work (Miniotas et al., 2006; Biswas and Langdon, 2011; Creed et al., 2020)). The use of sensors with higher levels of accuracy (e.g. Tobii 5 (Tobii, 2023)) could help to address some of the challenges identified, although there is still scope for calibration drift to influence user experiences. It will therefore be important for future work to investigate customisable interfaces that cater for the unique needs of individual users to help mitigate this issue (e.g. adjusting the size of interaction targets (Skovsgaard et al., 2010; Casarini et al., 2020)).

Another general limitation of the research is that all studies (aside from the final evaluation in Chapter 4) were conducted over a single testing session (based in the lab or remotely within a participant's environment). Whilst the approach taken in the thesis still presents important new insights and contributions around the feasibility of multimodal coding approaches for people with physical impairments, it will be crucial moving forward to also conduct longitudinal studies to investigate the efficacy of the approaches developed. For example, this could present important new insights around how developers adapt their coding practice and workflow around any limitations associated with voice coding approaches (e.g. to work around accuracy challenges).

Another limitation of the research is that the user evaluations conducted in Chapters 4 and 5 were conducted remotely in either a participant's home or work environment. Participants were therefore using their microphones to support voice input which led to a lack of consistency in the technology used within the evaluation sessions. A lack of control of external environmental factors (e.g. background noise) could also have influenced experiences in using the voice-controlled elements of the prototypes (Krishna et al., 2019). However, it is important to note that participants did not raise voice recognition as a major concern across both studies, and all participants successfully completed evaluation tasks. A remote evaluation approach can also present some benefits in that the prototypes were evaluated in more realistic scenarios where external factors can influence interaction experiences. Whilst this testing method, therefore, presents some limitations, it also helps to enhance the validity of the findings reported.

A further limitation is that the research prototypes developed only supported web-based markup and scripting languages (i.e. HTML, CSS, and JavaScript), as opposed to other popular languages such as Python, C#, and C++. These languages all have their coding conventions and standards, which may mean that adaptations are required to support the efficient writing and manipulation of syntax via multimodal coding approaches. For instance, using Emmett (Emmet, 2023) helped to support the more rapid production of

HTML and CSS code, although this approach would not be compatible with other languages. Further work is therefore needed to explore the nuances associated with other languages and new approaches to support efficient workflows via multimodal interaction methods.

Another limitation is associated with the multimodal approaches explored during this thesis – this included a combination of speech, gaze, and multiple switches for the first research prototype (Chapter 3) and the use of speech and a switch for the studies conducted in Chapters 4 and 5. Whilst there is a lack of work exploring the combination of these technologies within a coding context (as highlighted in Chapters 1 and 2), and the work conducted presents new insights, there are also other modalities that could have been explored. These include mid-air gesturing (Morrison and McKenna, 2002; Groenewald et al., 2016), head tracking (Haque et al., 2021; Kabir et al., 2022), alternative mice and keyboards (Smith et al., 1998; McLoone et al., 2010; Aigner et al., 2016; Henzen and Nohama, 2016), Brain Computer Interfaces (BCIs) (Vallabhaneni et al., 2005; Allison et al., 2007) and touch (Kane et al., 2009; Mott et al., 2016; Gheran et al., 2018). This represents an important future research area where different combinations of input modalities may help to support developers with specific requirements and forms of impairments.

In terms of the research conducted in Chapter 4, the natural language approach was taken forward for further development and evaluation work (after the first evaluation), although the fixed grammar approach also received positive feedback. An argument could have been made to also take this approach forward for further research within a multi-testing session study to gain further insights around the viability of this method. Whilst this still represents an important area requiring further investigation, participant feedback highlighted a particular interest in the potential of natural language input to support coding activities. It was therefore decided that this area held potential to present new insights and contributions for the field, although it is important to acknowledge that the fixed grammar approach was generally perceived positively and can present benefits for developers.

Finally, in relation to the research detailed in Chapter 5 around code navigation, the study focused on new inclusive approaches for navigating syntax within a single document. This work did not cover other scenarios, such as searching for specific code over multiple source files associated with a development project. Whilst this is also an important area requiring further investigation, it was felt that an important initial step was first to investigate whether common navigation features such as “Find”, “Find all References”, and “Go to Definition” could be tailored for multimodal voice and switch control within a single document. It will now be important to explore the wider application of the approaches developed to understand whether any further updates are required to these approaches to support additional code navigation scenarios.

A wider important point relates to the recruitment disabled and non-disabled participants for the research studies. The research detailed in Chapters 3 and 4 employed a two-phase methodology, consisting of an initial investigation involving non-disabled developers, followed by a subsequent investigation involving developers with physical impairments. Chapter 5 broadened this scope by incorporating a more extensive cohort of disabled developers, consisting of 14 participants. As mentioned earlier in Chapters 3 and 4, a key requirement of this research is that all users were able to use assistive tools to control the system. We therefore felt that conducting a first study with non-disabled participants provided a relevant insight into the use of the system. It also enabled us to identify areas where further improvements were required prior to conducting an evaluation with disabled participants.

The recruitment of disabled participants was predominantly carried out via online advertising, with the objective of targeting a wide range of individuals with diverse impairments. This recruitment strategy was designed to mitigate the influence of bias that may arise from exclusively relying on the research lab's personal contacts. It also offered an opportunity for individuals who may not have direct affiliations with the research team to take part in the study.

This chapter provided a discussion of findings and limitations from all three user studies presented in this thesis (Chapter 3-5). The main contributions of this work were presented, as well as the potential benefits to the research community. The key conclusions drawn from the work in this thesis will be presented in Chapter 7.

7 CONCLUSIONS AND FUTURE WORK

This thesis explored the integration of different assistive technologies such as eye gaze, speech, mechanical switches to facilitate usable, efficient and feasible approaches for supporting developers with physical impairments to perform coding activities. The thesis initially investigated the combination of speech, gaze, and a mechanical switch as an alternative input approach for writing, editing, deleting, and selecting HTML and CSS code (Chapter 3). The developed coding environment allowed developers to type code using eye gaze, control the interface via speech, and trigger actions through mechanical switches, thus leveraging the strengths of each modality to present a more viable coding approach. The feasibility of this prototype was assessed through evaluations involving both non-disabled (N=29) and disabled (N=5) participants. The design and evaluation processes primarily focused on writing, deleting, and selecting HTML and CSS code. Two user evaluations yielded positive results, showcasing the viability of integrating gaze, speech, and mechanical switches for supporting development tasks.

Drawing from the feedback and insights gained from Chapter 3, the next chapter explored the feasibility of different voice inputs (i.e., natural language and fixed grammar) along with a mechanical switch to perform coding activities (Chapter 4). Prior work on voice-based coding predominantly focused on using fixed and constrained commands, and none were evaluated with disabled developers. The initial design of the system was informed via an exploratory study involving twelve developers, aimed at gaining insights into the language employed while utilizing voice commands for code composition. This exploratory research informed the development of a multimodal inclusive coding environment using two voice coding approaches: Fixed Grammar and Natural Language. A comparative study involving non-disabled developers (N=25) produced a similar level of efficiency and usability between both approaches when writing JavaScript code, with each method receiving positive feedback and enabling the successful completion of development tasks. A multi-session follow-up study with disabled developers (N=5) demonstrated that coding through natural language speech facilitates effective code writing, editing, and navigation.

Disabled developers from the previous study (Chapter 4) emphasised a need for accessible common code navigation features that can be utilised via speech interaction. To investigate this further, an exploratory study with five disabled developers was conducted to gain insights into their experience with code navigation within their existing voice-based development environment. The findings from this study informed the design of a code editing prototype that integrated common code navigation techniques (e.g., "Find," "Find reference," and "Go to Definition") controlled via speech and switch input. A research study was conducted with 14 developers with physical impairments, with results demonstrating that the approach was a usable and efficient method for code navigation.

This thesis involved the exploration and implementation of novel multimodal input modalities to assist developers with physical impairments in performing coding activities. It examined the development and validation of multimodal interaction approaches within the coding environment through evaluations with both disabled and non-disabled developers, thus providing a comprehensive understanding of the strengths and limitations associated with the use of multimodal interaction approaches in a coding context. Additionally, the research explored natural language and fixed grammar voice coding methods, confirming both approaches as usable and feasible for coding purposes. Further investigations into the natural language approach were conducted through multi-session studies, offering a deeper understanding of its strengths and limitations in supporting developers during coding activities. Moreover, the research investigated mainstream code navigation approaches and identified voice and switch as feasible interaction methods to support developers with physical impairments in performing code navigation. Table 7.1 presents a summary of the key contributions.

Study	User Type	Participants	Categories/Tasks	Conditions	Number of tasks	Utilized tools	SUS
Study 1 (Multimodal Interaction)	• NDD	29 participants with an age range of 19 to 45 (M=27.9; SD=7.87)	• HTML ADD • HTML SELECT • HTML DELETE • HTML EDIT • CSS ADD • CSS SELECT • CSS DELETE • CSS EDIT	One	64 (8 categorie s*8 tasks)	• **Speech** (Controlling interface) • **Gaze**(Typing) • Mechanical Switches (Selection)	68.1 (SD =20.8)
Study 2 (Multimodal Interaction)	• DPI	5 participants with an average age of 23.6 years (SD=7.3); (4 with RSI, 1 with CP)	• Creating a single webpage from a blank document	One		• **Speech** (Controlling interface) • **Gaze**(Typing) • Mechanical Switches (Selection)	74.0 (SD =4.6)
Study 1 (Multimodal Speech Interaction)	• NDD	25 with ages ranging from 18 to 50 years (M=25.8; SD=7.85)	• Create • Select • Delete • Edit	• Natural language (NL) • Fixed Gramma r(FG)	48 (2 condition s*4 categorie s*6 tasks)	• **Speech** (performing the tasks) • **Mechanical switch** (triggering the recognizer)	NL =75.8 (SD =16.0); FG=75.4 (SD =17.4).
Study 2 (Multimodal Speech Interaction) (Multi-session)	• DPI	6 (M =26.7, SD =8.4); Four with RSI, one with CP and one with Tendonitis/Carpal Tunnel	• Create • Select • Delete • Edit	One		• **Speech** (performing the tasks) • **Mechanical switch** (triggering the recognizer)	59.2 (SD=18. 5)
Study 3 (Multimodal Code Navigation)	• DPI	14 (M=33.3, SD=10.9); 12 with RSI; 1 CP and one with Tendonitis/Carpal Tunnel	• Short, • Medium • Long	One	36 (12 tasks*3 categorie s)	• **Speech** (performing the tasks) • **Mechanical switch** (triggering the recognizer)	83.4 (SD=14. 3)

Table 7.1: A summary of user studies across each study; RSI=Repetitive Strain Injury; CP: Cerebral Palsy; NDD =Non-disabled developers; DPI=Developers with Physical Impairments

7.1 Future Work

Whilst the thesis presents new insights and contributions around multimodal methods to support people with physical impairments in writing code, it also gives rise to a range of important research areas requiring further investigation. For instance, a key emphasis of the work detailed in this thesis focused on specific coding activities such as writing, editing, navigating, selecting, and deleting code. Whilst these are all fundamental areas where further work was required, future studies also need to concentrate on other key development activities such as debugging, code versioning, and general management of a codebase. These are all areas where there has been a lack of work to date, and it remains unclear how they can be facilitated via multimodal interaction approaches. For instance, in terms of debugging, this is a crucial activity that will likely require custom commands to control different aspects of debugging, such as setting flags, stepping through code, and controlling the compilation process. There may also be scope for natural language approaches to support this process (e.g. "set a flag at line x and step through", "find the function x and add a breakpoint"). No work has been conducted in this area to date, although it is a crucial element of the application development process where further research is now required.

Whilst the multimodal system presented in Chapter 3 highlights some interaction issues in relation to eye gaze accuracy, it will be important to conduct additional work in this area. For instance, it will be important to work with more accurate sensors (e.g. Tobii 5) and examine whether this further supports the interaction approach. The study also focused on using gaze primarily for typing code and speech for other actions (e.g. code navigation). However, it would also be interesting to swap the primary roles of these approaches (e.g. speech for writing code and gaze for supporting navigation, and selection of targets). Furthermore, the ability to customise input methods and their mapping to different tasks is an important area that requires further work, as well as ensuring that the process for configuring these mappings is inclusive for people with physical impairments.

Another area where further research may prove fruitful is in relation to the customisation of vocal commands to support coding. For instance, one of the limitations of a fixed grammar approach is that users have to utilise a fixed set of commands. In some cases, the system misrecognised some of the commands issued by participants due to their accents (e.g. "delete line 10", often misread as "delete nine nine"), thus resulting in them having to repeat commands on multiple occasions. Similarly, whilst voice-based coding extensions such as Cursorless (2023) are available for platforms such as Visual Studio Code (e.g. to support with code navigation via voice input), they can provide some interaction challenges - for example, developers have to learn the supported voice commands (e.g. “chuck line near” to delete lines between two highlighted words), thus presenting additional cognitive load. Future work, therefore, needs to investigate the potential of having a customisable approach, where users can select and customise commands depending on their own preferences. Similar to the customisation of input methods (highlighted above), the process for defining these custom commands will also need to be accessible to ensure this does not present developers with additional barriers.

Novel innovations around natural language text input to generate code have been shown to present efficiency benefits for developers. For example, GitHub has introduced GitHub Copilot (Nguyen and Nadi, 2023), which can suggest large code snippets of different programming languages (e.g. if a user types “function farenheitTocelcius”, the relevant code snippet will be auto typed within the function). This has the potential to present significant benefits for coders with impairments, but further work is required to empirically evaluate the strengths and limitations of this method. In particular, this approach could reduce the amount of syntax that needs to be manually written, which could present benefits, although it may also introduce issues if the generated code contains any issues that does not reflect the developer’s desired outcome. For instance, a significant overhead could be added if users have to correct the code, which may potentially be a longer process compared with manually writing the syntax (without the support of auto-generation via natural language text input). However, further research is required to empirically evaluate and understand the opportunities and challenges associated with this coding approach.

Code completion is another feature that is commonplace in mainstream IDEs and has been shown to present productivity benefits for non-disabled developers as it lowers the number of keys needed to type the code by displaying relevant code snippets (Asaduzzaman et al., 2016; Amlekar et al., 2018). This also holds significant potential to make coding more efficient for people with physical impairments who are utilising alternative input methods, although there has been limited work exploring how this feature can be tailored for multimodal interaction. For example, this feature could be tailored to work in largely the same way where numbers are associated with different autocompletion options which can then be executed via an associated voice command (e.g. "complete 5"). However, there are potentially scenarios where there can be significant numbers of autocomplete options - how these completion options can be navigated via voice commands then becomes an important consideration. Simple navigation commands could be utilised (e.g. "select xth item", "scroll list and get xth item"), although further work is required to understand the types of interaction challenges this may present.

Another area where there has been a lack of work to date is around collaborative coding experiences. Tools such as Slack (2023), CodePen (2023) facilitates collaborative coding sessions (e.g. remote pair programming), although no work has explored experiences of developers working within mixed ability teams, where teammates may be utilising alternative input methods. This raises questions around the usability of such systems and whether they can efficiently facilitate collaborative sessions, although it also presents social questions around how developers feel when working collaboratively with disabled and non-disabled colleagues. For example, if disabled developers take longer to write or navigate code, it is not clear how this might make them feel within shared coding scenarios. Similarly, it is unclear how non-disabled colleagues feel in these scenarios and how it might impact their behaviour and working processes. These are important and timely questions, although they currently remain underexplored.

More widely, a key element of coding is referring to other sources online to help address technical issues and bugs (Slack, 2023; Stack Overflow, 2023). There has been significant work on navigating the web via voice and gaze interaction independently (Porta and Ravelli, 2009; Kumar et al., 2017; Sinha and Dasgupta, 2021), although less work has focused on multimodal solutions facilitating this type of activity within a coding context. For instance, the ability to easily copy and paste code from a web browser and then integrate this syntax within an IDE may present barriers for developers with physical impairments. There may also be challenges with viewing educational videos demonstrating coding concepts and examples whilst simultaneously attempting to apply the knowledge within an IDE. Similarly, the ability to rapidly switch between different applications (e.g. a browser and IDE) and different tabs within a web browser could also integrate challenges into the code writing process. Further research is therefore required to understand the types of barriers that may exist for developers in this context, as well as potential solutions to make these coding scenarios more accessible and efficient.

The use of plugins is also a commonly used resource by developers where the wider community develop additional features or approaches to support development productivity. For instance, within the Visual Studio Code Community there are over 40000 plugins available that can support developers Visual Studio Code Extensions (2023). However, no work has examined how disabled developers with physical impairments can initially navigate these plugins and then install them (i.e. this process may present some accessibility barriers). It is also not clear whether the additional features presented by a plugin may present accessibility challenges for developers that can exclude them from important tools that their colleagues might be utilising. Further research is therefore required to understand the accessibility of the wider ecosystem associated with development platforms to ensure they are not presenting barriers for disabled developers.

Finally, in terms of wider impact activities, many of the developers with physical impairments from the code navigation research studies (Chapter 5) expressed a desire for the navigation features developed (i.e. "Go to Definition", "Find all References") to be integrated into mainstream IDEs (e.g. Visual Studio Code and JetBrains). In particular, they indicated that these features could be useful for the wider development community to make code navigation more efficient. Similarly, it will be important to incorporate the findings from the other studies conducted (e.g. commands for writing, selecting, and editing code via voice) into commonly used development environments to widen the impact of the research completed. Further work is therefore required to explore the impact of integrating these features into mainstream IDEs and the extent to which they can support enhance code development for both disabled and non-disabled users.

7.2 Conclusion

This thesis highlights new insights and contributions into how multimodal interaction approaches can support people with physical impairments with writing, editing, and navigating code. The work presented highlights how the input methods developed can make coding more inclusive and accessible for disabled developers, as well as confirming that combinations of input modalities such as speech, gaze, and switches are a viable approach to support inclusive development experiences. There remain significant opportunities to further enhance the accessibility of development environments and platforms that can remove barriers and obstacles for people with physical impairments. Additional work is now essential across the wider community to investigate the areas and gaps identified where there is still limited work and understanding. This future work is crucial to ensure that developers with physical impairments are not excluded from the software engineering and development field and that they are able to work productively within professional mixed-ability development teams.

8 APPENDICES

Appendix A

A.1 Pre-test Questionnaires – Multimodal Coding Environment

Participant ID: Date:

1. Age……………
2. Gender: ………….
3. Do you wear any corrective lenses (e.g. glasses or contact lens)?

   ☐Yes ☐No
4. Is English your first language? If not, then what is your first language?

   …………………………………..
5. Do you have any experience of web development work?

   ☐Yes ☐No

   If Yes, number of year:

   ☐Less than a year ☐1-3 years ☐4-5 years ☐5+ years

   What is your experience in HTML Coding?

   ☐Beginner ☐Intermediate ☐Advanced

   What is your experience in CSS Coding?

   ☐Beginner ☐Intermediate ☐Advanced
6. Do you use any coding IDE for development work?

   ☐Yes ☐No

   If Yes, please specify the tool which you use?

   ……………………………………………………………………………………………

7. Do you have any experience of using speech recognition to control system or devices?

   ☐Yes ☐No

   If Yes, which devices ……………………………………………………………..

   If yes, frequency

   ☐Daily ☐Weekly ☐Monthly ☐One-off

8. Do you have any experience of using eye tracker to control the cursor or any other purposes?

   ☐Yes ☐No

   If Yes, which devices ……………………………………………………………..

   If yes, frequency

   ☐Daily ☐Weekly ☐Monthly ☐One-off

9. Do you have any experience of using Emmet to write code?

   ☐Yes ☐No

   If Yes, number of year:

   ☐Less than a year ☐1-3 years ☐4-5 years ☐5+ years

A.2 Tasks for the exploratory Study

SN	Task	Type	Initial Code	Final Code
1	Add a new element	HTML ADD	<body> </body>	<body> <h1></h1> </body>
2	Add a new element with a property	HTML ADD	<body> </body>	<body> <div id="articles"></div> </body>
3	Add a new element with two properties	HTML ADD	<body> </body>	<body> <div id="article1"class="article"> </div> </body>
4	Add a new element with children	HTML ADD	<body> </body>	<body> <ul> <li>Home</li> <li>About</li> <li>Article</li> <li>Contact</li> </ul> </body>
5	Add a new element (with an ID) with children that have properties [adding properties at different levels]	HTML ADD	<body> </body>	<body> <ul> <li class="menulink"></li> <li class="menulink"></li> <li class="menulink"></li> <li class="menulink"></li> <li class="menulink"></li>

				`</ul>` `</body>`
6	Manually add some HTML code	HTML ADD	`<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>`	`<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>` `</ul>`
7	Add free text	HTML ADD	`<h1>Latest Articles</h1>`	`<h1>Latest Articles</h1>` `<p>Please find our most recent published articles listed </p>`
8	Add a comment	HTML ADD	`<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>` `</ul>`	`<!-- The main menu for website -->` `<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>` `</ul>`
9	Select a line	HTML SELECT	`<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>` `</ul>`	`<ul id="menu">` `<li class="menu-item">Home</li>` `<li class="menu-item">About</li>` `<li class="menu-item">Articles</li>` `<li class="menu-item">Contact</li>` `</ul>`

10	Select an element	HTML SELECT	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
11	Select first property of an element	HTML SELECT	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
12	Select second property of element	HTML SELECT	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item" id=”menulink”>Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
13	Select block of code	HTML SELECT	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
14	Select a word in free text	HTML SELECT	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>

15	Select multiple words from a free text	HTML SELECT	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>
16	Select comment	HTML SELECT	<!-- This is a menu --> <ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<!-- This is a menu --> <ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
17	Remove a line	HTML REMOVE	<div class="article"></div> <div class="article"></div> <div class="article"></div> <div class="article"></div>	<div class="article"></div> <div class="article"></div> <div class="article"></div>
18	Remove an element	HTML REMOVE	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> </ul>
19	Remove first property of an element	HTML REMOVE	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
20	Remove nth property of element	HTML REMOVE	<ul id="menu"> <li class="menu-item" id="menulink">Home</li>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li>

			<li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
21	Remove block of code	HTML REMOVE	<body> <ul id="menu"> <li class="menu-item" id="menulink">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul> </body>	<body> </body>
22	Remove a comment	HTML REMOVE	<ul id="menu"> <!-- the main menu for website --> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>	<ul id="menu"> <li class="menu-item">Home</li> <li class="menu-item">About</li> <li class="menu-item">Articles</li> <li class="menu-item">Contact</li> </ul>
23	Remove a word from free text	HTML REMOVE	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>	<h1>Latest Articles</h1> <p>Please find our recent published articles listed </p>
24	Remove multiple words from free text	HTML REMOVE	<h1>Latest Articles</h1> <p>Please find our most recent published articles listed </p>	<h1>Latest Articles</h1> <p>Please find our most listed </p>
25	Typographical Error	HTML EDIT	<div id="article" calss="article"> <h1>Article Title</h1> <p>Article Content</p>	<div id="article" class="article"> <h1>Article Title</h1> <p>Article Content</p>

			</div>	</div>
26	Unclosed pair	HTML EDIT	<body> <h1>Article Title</h1> <p>Article Content </body>	<body> <h1>Article Title</h1> <p>Article Content </p> </body>
27	Confused similar construct (using incorrect syntax)	HTML EDIT	<title>Article Title</title> <p>Article Content</p>	<h1>Article Title</h1> <p>Article Content</p>
28	Mixed mode (using CSS syntax)	HTML EDIT	<h1 class: "red">Article Title</h1> <p>Article Content</p>	<h1 class="red">Article Title</h1> <p>Article Content</p>
29	Misidentified Construct	HTML EDIT	<input button-type="submit" value="submit">	<input type="button" value="submit">
30	Invalid lists and list items	HTML EDIT	<ul > <p>Item one </p> <p>Item two</p> </ul>	<ul > <li>Item one </li> <li>Item two</li> </ul>
31	Invalid comment syntax	HTML EDIT	<body> <ul > // list items </ul> </body>	<body> <ul > <!-- list items --> </ul> </body>
32	HTML Foundations	HTML EDIT	<h1 class= "red">Article Title</h1> <align="right">Sidebar</align>	<h1 class= "red">Article Title</h1> <div class="right">Sidebar</div>
33	Add a new element	CSS ADD		.div{ }
34	Add a new element with a property	CSS ADD	.div { }	.div { color: #000; }

35	Add a new element with two properties	CSS ADD	.div {
			
			}
				.div {
				color: #000;
				background-color: #fff;
				}

36	Add a new element with children	CSS ADD	.div{
			
			}
				.div > span{
				
				}

37	Add a new element (with an ID) with children that have properties	CSS ADD	.div{
			
			}
				.div#article{
				color: #000;
				background: #000;
				font-weight: bold;
				}

38	Manually add some CSS code	CSS ADD	.div {
			color:#fff;
			background:#ccc;
			text-align:
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				}

39	Add a free text	CSS ADD	footer{
			color:red;
			}
				footer[data-type='copy']{
				color:red;
				}

40	Add a comment	CSS ADD	.div {
			color:#fff;
			background:#ccc;
			}
				/* This will be div */
				.div {
				color:#fff;
				background:#ccc;
				}

41	Select a line	CSS SELECT	.div {
			color:#fff;
			background:#ccc;
				.div {
				color:#fff;
				background:#ccc;

			text-align:center;
			position:absolute;
			}
				text-align:center;
				position:absolute;
				}

42	Select an element	CSS SELECT	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				}

43	Select first property of an element	CSS SELECT	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				}

44	Select second property of element	CSS SELECT	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			border: 2px solid red;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				border: 2px solid red;
				}

45	Select block of code	CSS SELECT	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			border: 2px solid red;
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				border: 2px solid red;

			}	}

46	Select a word in free text	CSS SELECT	footer[data-type='copy']{
			color:red;
			}
				footer[data-type='copy']{
				color:red;
				}

47	Select multiple words in free text	CSS SELECT	footer[data-type='copy']{
			color:red;
			}
				footer[data-type='copy']{
				color:red;
				}

48	Select comment	CSS SELECT	/* This will have some comment */
			.div {
			color:#fff;
			background:#ccc;
			}
				/* This will have some comment */
				.div {
				color:#fff;
				background:#ccc;
				}

49	Remove a line	CSS REMOVE	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			}
				.div {
				color:#fff;
				text-align:center;
				position:absolute;
				}

50	Remove an element	CSS REMOVE	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			}
			.div2 {
			margin:2px;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				}

51	Remove first element of an element	CSS REMOVE	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			padding: 2px 4px 5px 3px;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				padding: 4px 5px 3px;
				}

52	Remove nth property from element	CSS REMOVE	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			padding: 2px 4px 5px 3px;
			}
				.div {
				color:#fff;
				background:#ccc;
				text-align:center;
				position:absolute;
				padding: 2px 4px 3px;
				}

53	Remove a block of code	CSS REMOVE	.div {
			color:#fff;
			background:#ccc;
			text-align:center;
			position:absolute;
			padding: 2px 4px 5px 3px;
			}
				.div {
				color:#fff;
				}

54	Remove a word from free text	CSS REMOVE	footer[data-type='copy']{
			color:red;
			}
				footer[data-type=]{
				color:red;
				}

55	Remove multiple words from free text	CSS REMOVE	footer[data-type='copy']{
			color:red;
			}
				footer{
				color:red;
				}

56	Remove a comment	CSS REMOVE	/* This will have some comment */
			.div {
			color:#fff;
			background:#ccc;
			}
				.div {
				color:#fff;
				background:#ccc;
				}

57	Misidentified Construct	CSS EDIT	.div{
			font-color:#fff;
			background:#000;
			}
				.div{
				color:#fff;
				background:#000;
				}

58	Unclosed pair	CSS EDIT	.div{
			font-color:#fff;
			padding:20;
			background:#000;
			}
				.div{
				font-color:#fff;
				padding:20px;
				background:#000;
				}

59	CSS Selector	CSS EDIT	.div > #article {
			color:#000;
			background:#fff;
			font-weight:bold;
			}
				.div#article {
				color:#000;
				background:#fff;
				font-weight:bold;
				}

60	CSS Foundation Error	CSS EDIT	.div: color: #fff;	.div {
				color: #fff;
				}

61	Typographic Error	CSS EDIT	.div{
			font-color:#fff;
			padding:20;
			bckagruond:#000;
			}
				.div{
				font-color:#fff;
				padding:20;
				background:#000;
				}

62	Comment Syntax	CSS EDIT	//image	/* image */

			.div{
			font-color:#fff;
			background:#000;
			}
				.div{
				font-color:#fff;
				background:#000;
				}

63	Unclosed Element X	CSS
		EDIT
			.div {
			color:#000;
			background:#fff;
			font-weight:bold;
			.div {
			color: #fff;
			}
				.div {
				color:#000;
				background:#fff;
				font-weight:bold;
				}
				.div {
				color: #fff;
				}

64	Invalid Construct	CSS
		EDIT
			.div{
			font-color=#fff;
			padding=20;
			background=#000;
			}
				.div{
				font-color:#fff;
				padding:20;
				background:#000;
				}

A.3 Post-test Questionnaire

Participant ID: Date:

1. What was your overall experience of using eye gaze and speech recognition in development tasks?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………..
2. How did you find writing code via gaze?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………
3. How did you find speech for controlling the system?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………
4. How did you find the use of Emmet via eye gaze as an approach in coding??
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………
5. Do you have suggestions or feedback for the improvement of the tool?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………

A.4 Pre-test Questionnaire – Formal Study

Participant ID: Date:

1. Age……………………………………………………………………………
2. Gender: ………….
3. Do you wear any corrective lenses (e.g. glasses or contact lens)?

   ………………………..
4. Is English your first language? If not, then what is your first language?

   …………………………………..
5. Please specify the form of disability you have? Please provide detail about your impairments

   ……………………………………………………………………………………
   ………………………………………………………………………………………
   …………………………………………………………………………………
6. Do you have any experience of web development work?

   ☐Yes ☐No

   If Yes, number of years:

   ☐Less than a year ☐1-3 years ☐4-5 years ☐5+ years
7. Do you have any experience of using speech recognition to control systems or devices?

   ☐Yes ☐No

   If Yes, which devices (e.g. Amazon echo, Siri)

   …………………………………………………………………..

   Frequency

   ☐Daily ☐Weekly ☐Monthly ☐One-off
8. Do you have any experience of using eye tracking technologies to control systems or devices?

   ☐Yes ☐No

   If Yes, please give details

   ……………………….…………………………………………………………

A.5 Post-test Questionnaire – Formal Study

Participant ID: Date:

1. What was your overall experience of using a combination of eye gaze and speech recognition in development tasks?
   …………………………………………………………………………………………
   ……………………………………………………………………………………….
2. How did you find eye gaze tracker for writing code?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
3. What is your experience of using Voice?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
4. How do you compare your current approach of programming in compare to this prototype?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
5. Do you think this approach of coding can be viable for future coding??
   …………………………………………………………………………………………
   …………………………………………………………………………………………
6. Do you have any suggestions for the improvement of this system?
   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………………

Appendix B

B.1 Pre-test Questionnaires – Multimodal Voice

Participant ID: Date:

1. Age……………
2. Gender: ………….
3. Is English your first language? If not, then what is your first language?
   …………………………………..
4. Please specify the years of coding experience ? (in numbers)
   ……………
5. Please specify the coding IDE you are using for your development work?
   ……………………………………………………………………………….
6. Please specify any assistive technologies (e.g. speech recognition, eye gaze tracking, etc.) you use for development work?
   ……………………………………………………………………………….
7. Please specify the audio device you will be using (e.g. microphone, inbuilt-audio, etc)?
   ……………………………………………………………………………….

B.2 Tasks for exploratory study

SN	Task	Type	Initial Code	Final Code

1	Create a class	ADD		class car {
				}

2	Create a function	ADD		check(brand, price){
				}

3	Create a variable	ADD		var userprice = document.getElementById(“price”).value

4	Create a condition	ADD		If(userprice >=this.price) {
				}

5	Create a print script	ADD		console.log(“out of my range”);

6	Create a for loop	ADD		for(var i=0; i<10; i++){
				}

7	Select the word ‘brand’	SELECT	const brand=this.brand;	const brand = this.brand;
8	Select “this.height “	SELECT	var this.height = price	var this.height = price

9	Select a single line	SELECT	class abc {
			constructor(height){
			this.height=height;
			}
			}
				class abc {
				constructor(height){
				this.height=height;
				}
				}

10	Select multiple lines	SELECT	class abc {
			constructor(height){
			this.height=height;
				class abc {
				constructor(height){
				this.height=height;

			}
			 }
				}
				 }

11	Select a single word	SELECT	throw "the intent is not defined";	throw "the intent is not defined";

12	Select multipke words	SELECT	class abc {
			 constructor(height){
			 }
			 }
				class abc {
				 constructor(height){
				 }
				 }

13	Incorrect constraint	EDIT	var elements = document.getElementById("#stores ");	var elements = document.getElementById("stores");
14	Missing syntax	EDIT	var variable2 = 'world!

	var variable2 = 'world!

';
15	Misisng Identifier	EDIT	final_text = variable1+variable2;	var final_text = variable1+variable2;

16	Typographical error	EDIT	var elements = documents.getElementById('stores') ;
			 if(elemnets > 2){
			 return final;
			 }
				var elements = documents.getElementById('stores');
				 if(elements > 2){
				 return final;
				 }

17	Incorrect syntax	EDIT	if(elements >2)
			 {
			 return final;
			 }
				if(elements >2)
				 {
				 return final_text;
				 }

18	Missing identifier	EDIT	class area {
			 function getMethod(variable){
			 var elements = "hello"+variable;
			 }
			 }
			 getMethod()
				class area {
				 function getMethod(variable){
				 var elements = "hello"+variable;
				 }
				 }
				 getMethod("world")

19	Delete comma	DELETE	const brand=this.brand;,	const brand = this.brand;
20	Delete the word 'the .'	DELETE	var this.height = price	Var height = price

21	Delete 3rd line	DELETE	class abc {
			  constructor(height){
			    this.height=height;
			  }
			}
				class abc {
				  constructor(height){
				  }
				}

22	Delete line 2, 3 and 4	DELETE	class abc {
			 constructor(height){
			    this.height=height;
			  }
			}
				class abc {
				}

23	Delete Multiple words	DELETE	throw "the intent is not defined";	throw "the intent";
24	Delete the comment	DELETE	// the code goes here var elements = documents.getElementById('price');	var elements = documents.getElementById('price');

B.3 Post-test Questionnaire

1. How did you find the process of writing, editing, selecting and deleting the code using Natural Language Speech?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ……………………………………………………………………………………..

2. How did you find the process of writing, editing, selecting and deleting the code using Fixed Commands?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………………

3. Did you have any preference between the two versions? If so, please explain why?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………………

4. Do you have any suggestions for Improving the Natural Language Approach?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………………

5. Do you have any suggestions for Improving the Natural Language Approach?

   …………………………………………………………………………………………
   …………………………………………………………………………………………
   ………………………………………………………………………………………

B.4 Pre-test questionnaire for Formal study

Participant ID: Date:

1. Age………………………………………………………………………………… ………………………
2. Gender: ………….
3. Is English your first language? If not, then what is your first language? …………………………………..
4. Please specify the form of disability you have? Please provide detail about your impairments ……………………………………………………………………………………… ………………………………………………………………………………………… ……………………………………………………………………………………
5. What Assistive tools do you use ? ………………………………………………………………………………………… ………………………………………………………………………………………… ….
6. How do your rate your JavaScript skill?
   ☐Novice ☐Intermediate ☐Expert
7. What Programming language/ tools do you use ? ………………………………………………………………………………………… ………………………………………………………………………………
8. Please specify the audio device you will be using (e.g. microphone, inbuilt-audio, etc)?

B.5 Tasks for Formal Study

First Session

SN	Task	Type	Initial Code	Final Code

1	Create a class	Add		class car {
				}

2	Create a function	Add		check(){
				
				}

3	Create a variable and assign value	Add		var height=5;
4	Move the cursor to 9 position right	Navigate	var this height = price;	var this█height = price;

5	Delete single line	Delete	constructor(height){
			this.height=height;
			}
				constructor(height){
				}

6	Delete multiple words	Delete	throw "the intent is not defined ";	throw "the intent is";
7	Select a word	Select	const height=25;	const height = 25;

8	Select single line	select	constructor(height){
			this.height=height;
			}
				constructor(height){
				this.height=height;
				}

9	Missing syntax	Edit	var variable2 = 'world!

	var variable2 = 'world!

’;
10	Misisng Identifier	Edit	final_text = variable1+variable2;	var final_text = variable1+variable2;

11	Incorrect construct	Edit	var elements = documents.getElementById('stores') ;
			if(elemnets > 2){
			return final;
			}
				var elements = documents.getElementById('stores');
				if(elements > 2){
				return final;
				}

12	Incorrect	Edit	if(elements >2)
			{
			return final;
			}
				if(elements >2)
				{
				return final_text;
				}

Second Session

SN	Task	Type	Initial Code	Final Code

1	Create a function with parameters	ADD		checkarea (height, width){
				}

2	If condition	ADD		If(userprice >=this.price) {
				}

3	Loop	ADD		for(var i=0; i<10; i++){
				}

4	Delete multiple lines	DELETE	constructor(height, width){
			this.height=height;
			this.width=width;
			}
				constructor(){
				}

5	Delete multiple words	DELETE	const height=this height	this height;
6	Select multiple words	SELECT	throw "the intent is not defined ";	throw "the intent is not defined";

7	Select multiple lines	SELECT	constructor(height, width){
			this.height=height;
			this.width=width;
			}
				constructor(height, width){
				this.height=height;
				this.width=width;
				}

8	Incorrect syntax	EDIT	var elements =
			documents.getElementById('stores');
			if(eelmenst > 2){
			return final;
			}
				var elements =
				documents.getElementById('stores');
				if(elements > 2){
				return final;
				}

9	Missing syntax	EDIT	class area {
			function getMethod(variable){
			var elements = "hello"+variable;
			}
			}
			getMethod(false)
				class area {
				function getMethod(variable){
				var elements = "hello"+variable;
				}
				}
				getMethod(“world”)

10	Copy code and paste	SELECT	class car {
			this.price=price;
			constructor(brand, price){
			this.brand=brand;
			}
			}
				class car {
				constructor(brand, price){
				this.brand=brand;
				this.price=price;
				}
				}

Third Session

```
class area {
  constructor(height, width){
    this.width=width;
    this.height=height;
  }
  checkarea(){
    let initialheight = document.getElementById("height").value;
    let initialwidth = document.getElementById("width").value;
    let previousarea = initialheight*initialwidth;
    let currentarea = this.height*this.width;
    if(previousarea >=currentarea){
      console.log("The area is greater")
    }
    else{
      console.log("The area is smaller")
    }
  }
}
```

B.5 Post -test Questionnaire

1. How did you find the process of writing new syntax?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ……………………………………………………………………………..

2. How did you find the process of editing existing code?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ………………………………………………………………………………

3. How did you find the process of deleting and selecting code?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ………………………………………………………………………………

4. How might the use of voice influence your future coding practice?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ………………………………………………………………………………

5. Do you have any suggestions for Improving the prototype?

   ……………………………………………………………………………………

   ……………………………………………………………………………………

   ………………………………………………………………………………

Appendix C

C.1 Pre-test questionnaire for Exploratory Study

Participant ID: Date:

1. Age………………………………………………………………………………
2. Gender: ………….
3. How do your rate your Coding experience?
   ☐Novice ☐Intermediate ☐Expert
4. Please specify the form of disability you have? Please provide detail about your impairments
   ……………………………………………………………………………………
   ………………………………………………………………………………………
   …………………………………………………………………………………
5. What Assistive tools do you use ?
   ………………………………………………………………………………………
6. What coding IDE do you use?
   ……………………………………………………………………………………
7. What coding Language do you use?
   ………………………………………………………………………………………

C.2 Semi-Structured Questionnaire

Participant ID: Date:

1. Do you use any code navigation features when coding - if so, please elaborate on how you typically use them (e.g. in terms of selection via any assistive tools you use)
   ………………………………………………………………………………………
   ………………………………………………………………………………
2. Which kind of issues do you experience when performing code navigation using the assistive tools?
   ………………………………………………………………………………………
   ………………………………………………………………………………………
3. How do you overcome those issues?
   ………………………………………………………………………………………
   ………………………………………………………………………………………
4. Do you have any suggestions for making code navigation more accessible using your assistive tools?

C.3 Pre-test Questionnaire

Participant ID: Date:

1. Age………………………………………………………………………………
2. Gender: ………….
3. Is English your first language? If not, please specify your native?

   …………………………………..
4. Please specify the years of coding experience (in number)

   ……………………………………………………………………………..
5. Please specify your Programming experience?

   ☐Novice

   ☐Intermediate

   ☐Expert
6. Please specify platforms/languages you commonly used for programming?

   ………………………………………………………………………………

   ……………………………………………………………………………..
7. Please specify the form of disability you have? Please provide detail about your impairments

   ……………………………………………………………………………
8. Please specify the coding IDE you are using for your development work?

   ………………………………………………………………………………
9. Please specify any assistive technologies (e.g. speech recognition, eye gazing tracking) you use for development work?

   ………………………………………………………………………………
10. Please specify the audio device you will be using (e.g. microphone, inbuilt-audio, etc)?

    ……………………………………………………………………………

C.4 Tasks for First Study

Short Tasks

1. Search for the word **editor** and move forward to **4th instance**
2. Place the cursor after **“var”** on the same line of code
3. Find all references of the variable **cmEditor** and select the **7th instance** from the reference list
4. Move the cursor **6 lines down**
5. Navigate to the definition of the function **applyCode**
6. Move the cursor to **7 lines up**
7. Move the cursor to the **end of the line**
8. Find the variable **console** and move forward to **4th instance**
9. Place the cursor before “**output**” on the same line of code
10. Find the reference of variable **output** and select the **6th instance** from reference list
11. Navigate to the definition of the variable **output**
12. Navigate to line **41**

Medium Tasks

1. Search for the word “**recognition**” and move forward to the **6th position**
2. Move the cursor **8 lines down**
3. Place the cursor after “**re**” on the same line of code
4. Find all reference of the function **record** and select the **4th instance** from the reference
5. Move the cursor **3 lines up**
6. Find all references of the variable **interval** and select the **8th instance** from the reference list
7. Navigate to the line **160**
8. Navigate to the definition of the variable **linter**
9. Search for the variable **recording** and move forward to the **6th instance**
10. Navigate to **1 line up**
11. Navigate to definition of the variable **final_script**
12. Navigate to the **end of line**

Long Task

1. Navigate to **line 102**
2. Place the cursor after "**a**" on the same line of code
3. Find all references of the function **startRecognising** and select the **5th instance** from the reference list
4. Move the cursor **66 lines down**
5. Navigate to the definition of the array **punctLists**
6. Search for the variable **taskshuffle** and move forward to the **9th item**
7. Move the cursor to **14 lines up**
8. Find all references of the function **stopRecognising** and select the **1st instance** from the reference list
9. Move the cursor to **5 lines up**
10. Navigate to the definition of the function **insertintotext**
11. Search for the variable **firstword** and select the **11th instance**
12. Navigate to the **end of line**

C.5 Post-test Questionnaire

1. How did you find the process of code navigation using speech?

   …………………………………………………………………………………………

   …………………………………………………………………………………………

   ……………………………………………………………………………………..
2. How did you find code navigation features such as ‘find all references’ and ‘go to definition’ using speech ?

   …………………………………………………………………………………………

   …………………………………………………………………………………………

   ………………………………………………………………………………………
3. How might the use of voice might influence your future coding practice ?

   …………………………………………………………………………………………

   …………………………………………………………………………………………

   ………………………………………………………………………………………
4. Do you have suggestions for improving the prototype?

   …………………………………………………………………………………………

   …………………………………………………………………………………………

   ………………………………………………………………………………………

9 REFERENCES

Abualghaib, O., Groce, N., Simeu, N., et al. (2019) Making visible the invisible: why disability-disaggregated data is vital to “leave no-one behind.” *Sustainability*, 11 (11): 3091.

Adams, J.L., Dinesh, K., Xiong, M., et al. (2017) Multiple Wearable Sensors in Parkinson and Huntington Disease Individuals: A Pilot Study in Clinic and at Home. *Digital Biomarkers*, 1 (1): 52–63. doi:10.1159/000479018.

Aigner, B., David, V., Deinhofer, M., et al. (2016) “FLipMouse: a Flexible Alternative Input Solution for People with Severe Motor Restrictions.” In *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. DSAI 2016 New York, NY, USA, 1 December 2016. Association for Computing Machinery. pp. 25–32. doi:10.1145/3019943.3019948.

Albusays, K., Ludi, S. and Huenerfauth, M. (2017) “Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges.” In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS ’17 New York, NY, USA, 19 October 2017. Association for Computing Machinery. pp. 91–100. doi:10.1145/3132525.3132550.

Alexa (2023) *Amazon Alexa*. Available at: https://alexa.amazon.com/ (Accessed: 24 March 2023).

Allison, B.Z., Wolpaw, E.W. and Wolpaw, J.R. (2007) Brain–computer interface systems: progress and prospects. *Expert Review of Medical Devices*, 4 (4): 463–474. doi:10.1586/17434440.4.4.463.

Al-Rahayfeh, A. and Faezipour, M. (2013) Eye Tracking and Head Movement Detection: A State-of-Art Survey. *IEEE Journal of Translational Engineering in Health and Medicine*, 1: 2100212–2100212. doi:10.1109/JTEHM.2013.2289879.

Amlekar, R., Gamboa, A.F.R., Gallaba, K., et al. (2018) “Do software engineers use autocompletion features differently than other developers?” In *Proceedings - International Conference on Software Engineering*. 2018. pp. 86–89. doi:10.1145/3196398.3196471.

Anantha Prabha, P., Srinivash, K., Vigneshwar, S., et al. (2022) “Mouse Assistance for Motor-Disabled People Using Computer Vision.” In Mahapatra, R.P., Peddoju, S.K., Roy, S., et al. (eds.). *Proceedings of International Conference on Recent Trends in Computing*. Lecture Notes in Networks and Systems Singapore, 2022. Springer Nature. pp. 403–413. doi:10.1007/978-981-16-7118-0_35.

Anson, D. (1994) Finding Your Way in the Maze of Computer Access Technology. *The American Journal of Occupational Therapy*, 48 (2): 121–129. doi:10.5014/ajot.48.2.121.

Anthony, L., Kim, Y. and Findlater, L. (2013) “Analyzing user-generated youtube videos to understand touchscreen use by people with motor impairments.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’13 New York, NY, USA, 27 April 2013. Association for Computing Machinery. pp. 1223–1232. doi:10.1145/2470654.2466158.

Arnold, S.C., Mark, L. and Goldthwaite, J. (2000) “Programming by voice, VocalProgramming.” In *Proceedings of the fourth international ACM conference on Assistive technologies - Assets ’00*. 2000. pp. 149–155. doi:10.1145/354324.354362.

Asaduzzaman, M., Roy, C.K., Schneider, K.A., et al. (2016) A Simple, Efficient, Context-sensitive Approach for Code Completion. *Journal of Software: Evolution and Process*, 28 (7): 512–541. doi:10.1002/smr.1791.

Aschwanden, C. and Crosby, M. (2006) “Code scanning patterns in program comprehension.” In *Proceedings of the 39th hawaii international conference on system sciences*. 2006. Citeseer.

Ateeq, K. (2022) “Hybrid Working Method: An Integrative Review.” In *2022 International Conference on Business Analytics for Technology and Security (ICBATS)*. February 2022. pp. 1–8. doi:10.1109/ICBATS54253.2022.9759041.

Atom (2023). Available at: https://atom.io/.

Ayub, M. and Saleem, M.A. (2012) *A Speech Recognition based Approach for Development in C ++.*, 12 (10): 110–114.

Aziz, F., Creed, C., Frutos-Pascual, M., et al. (2021) “Inclusive Voice Interaction Techniques for Creative Object Positioning.” In *Proceedings of the 2021 International Conference on Multimodal Interaction*. ICMI ’21 New York, NY, USA, 18 October 2021. Association for Computing Machinery. pp. 461–469. doi:10.1145/3462244.3479937.

Aziz, F., Creed, C., Sarcar, S., et al. (2022) “Voice Snapping: Inclusive Speech Interaction Techniques for Creative Object Manipulation.” In *Designing Interactive Systems Conference*. Virtual Event Australia, 13 June 2022. ACM. pp. 1486–1496. doi:10.1145/3532106.3533452.

Baddeley, A. (1992) Working Memory. *Science*, 255 (5044): 556–559.

Bafna, T. (2018) "Gaze typing using multi-key selection technique." In *ASSETS 2018 - Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. New York, New York, USA, October 2018. Association for Computing Machinery, Inc. pp. 477–479. doi:10.1145/3234695.3240992.

Baker, C. (2022) *Mental health statistics: prevalence, services and funding in England*. Available at: https://commonslibrary.parliament.uk/research-briefings/sn06988/ (Accessed: 15 November 2022).

Baker, C.M., Milne, L.R. and Ladner, R.E. (2015) "Struct jumper: A tool to help blind programmers navigate and understand the structure of code." In *Conference on Human Factors in Computing Systems - Proceedings*. New York, New York, USA, April 2015. Association for Computing Machinery. pp. 3043–3052. doi:10.1145/2702123.2702589.

Bangor, A., Kortum, P. and Miller, J. (2009) Determining what individual SUS scores mean: adding an adjective rating scale. *Journal of Usability Studies*, 4 (3): 114–123.

Barmpoutis, A. (2018) "Learning programming languages as shortcuts to natural language token replacements." In *ACM International Conference Proceeding Series*. New York, NY, USA, November 2018. Association for Computing Machinery. pp. 1–10. doi:10.1145/3279720.3279721.

Bates, R. and Istance, H. (2002) *Zooming interfaces!* In January 2002. pp. 119–119. doi:10.1145/638249.638272.

Bax, M., Goldstein, M., Rosenbaum, P., et al. (2005) Proposed definition and classification of cerebral palsy, April 2005. *Developmental Medicine & Child Neurology*, 47 (8): 571–576. doi:10.1017/S001216220500112X.

Becker, H., Stuifbergen, A. and Tinkle, M. (1997) Reproductive health care experiences of women with physical disabilities: a qualitative study. *Archives of physical medicine and rehabilitation*, 78 (12): S26–S33.

Bednarik, R. and Tukiainen, M. (2006) "An eye-tracking methodology for characterizing program comprehension processes." In *Proceedings of the 2006 symposium on Eye tracking research & applications - ETRA '06*. 2006. pp. 125–125. doi:10.1145/1117309.1117356.

Beelders, T.R. (2011) *Enhancing the user experience for a word processor application through vision and voice By*. Available at: https://scholar.ufs.ac.za/bitstream/handle/11660/809/BeeldersTR.pdf?sequence=1&isAllowed=y.

Beelders, T.R. and Blignaut, P.J. (2010) "Using vision and voice to create a multimodal interface for Microsoft Word 2007." In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications - ETRA '10*. ETRA '10 New York, NY, USA, 2010. ACM. pp. 173–173. doi:10.1145/1743666.1743709.

Beelders, T.R. and Blignaut, P.J. (2011) *The Usability of Speech and Eye Gaze as a Multimodal Interface for a Word Processor*. IntechOpen. doi:10.5772/16604.

Begel, A. and Graham, S.L. (2005) “Spoken programs.” In *Proceedings - 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. 2005. IEEE. pp. 99–106. doi:10.1109/VLHCC.2005.58.

Begel, A. and Graham, S.L. (2006) “An assessment of a speech-based programming environment.” In *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006*. 2006. pp. 116–120. doi:10.1109/VLHCC.2006.9.

Begel, A. and Kariv, Z. (2002) *SpeedNav: Document Navigation By Voice*.

Begel, A. and Vrzakova, H. (2018) “Eye movements in code review.” In *Proceedings of the Workshop on Eye Movements in Programming - EMIP ’18*. New York, New York, USA, 2018. ACM Press. pp. 1–5. doi:10.1145/3216723.3216727.

Bell, D. and Heitmueller, A. (2009) The Disability Discrimination Act in the UK: Helping or hindering employment among the disabled? *Journal of Health Economics*, 28 (2): 465–480. doi:10.1016/j.jhealeco.2008.10.006.

Benkerzaz, S., Elmir, Y. and Dennai, A. (2019) A study on automatic speech recognition. *Journal of Information Technology Review*, 10 (3): 80–83.

Bennett, C.L., Brady, E. and Branham, S.M. (2018) “Interdependence as a Frame for Assistive Technology Research and Design.” In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS ’18 New York, NY, USA, 8 October 2018. Association for Computing Machinery. pp. 161–173. doi:10.1145/3234695.3236348.

Bentley, F., Luvogt, C., Silverman, M., et al. (2018) Understanding the Long-Term Use of Smart Speaker Assistants. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2 (3): 91:1-91:24. doi:10.1145/3264901.

Bergstrom, J.R. and Schall, A. (2014) *Eye tracking in user experience design*. Elsevier.

Bilmes, J.A., Li, X., Malkin, J., et al. (2005) “The vocal joystick: a voice-based human-computer interface for individuals with motor impairments.” In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. HLT ’05 USA, 6 October 2005. Association for Computational Linguistics. pp. 995–1002. doi:10.3115/1220575.1220700.

Biswas, P. and Langdon, P. (2011) A new input system for disabled users involving eye gaze tracker and scanning interface. *Journal of Assistive Technologies*, 5 (2): 58–66. doi:10.1108/17549451111149269.

Biswas, P. and Robinson, P. (2007) “Simulation to predict performance of assistive interfaces.” In *Proceedings of the 9th international ACM SIGACCESS conference on*

Computers and accessibility. Assets ’07 New York, NY, USA, 15 October 2007. Association for Computing Machinery. pp. 227–228. doi:10.1145/1296843.1296885.

Blauwendraat, C., Nalls, M.A. and Singleton, A.B. (2020) The genetic architecture of Parkinson’s disease. *The Lancet Neurology*, 19 (2): 170–178. doi:10.1016/S1474-4422(19)30287-X.

Bohus, D. and Rudnicky, A.I. (2008) “Sorry, I Didn’t Catch That!” In Dybkjær, L. and Minker, W. (eds.) *Recent Trends in Discourse and Dialogue*. Text, Speech and Language Technology. Dordrecht: Springer Netherlands. pp. 123–154. doi:10.1007/978-1-4020-6821-8_6.

Bolt, R.A. (1980) ““Put-that-there”: Voice and gesture at the graphics interface.” In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1980*. SIGGRAPH ’80 New York, NY, USA, 1980. ACM. pp. 262–270. doi:10.1145/800250.807503.

Borgestig, M., Rytterström, P. and Hemmingsson, H. (2017) Gaze-based assistive technology used in daily life by children with severe physical impairments–parents’ experiences. *Developmental Neurorehabilitation*, 20 (5): 301–308. doi:10.1080/17518423.2016.1211769.

Brackets (2023) *Brackets*. Available at: http://brackets.io/.

Bragdon, A., Zeleznik, R., Reiss, S.P., et al. (2010) “Code bubbles.” In *Proceedings of the 28th international conference on Human factors in computing systems - CHI ’10*. ICSE ’10 New York, NY, USA, 2010. ACM. pp. 2503–2503. doi:10.1145/1753326.1753706.

Braun, V. and Clarke, V. (2012) “Thematic analysis.” In *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. APA handbooks in psychology®. Washington, DC, US: American Psychological Association. pp. 57–71. doi:10.1037/13620-004.

Brodwin, M.G., Star, T. and Cardoso, E. (2004) Computer assistive technology for people who have disabilities: Computer adaptations and modifications. *Journal of rehabilitation*, 70 (3).

Brooke, J. (1996) Sus: a “quick and dirty’usability. *Usability evaluation in industry*, 189 (3): 189–194.

Brown, C. (1992) Assistive technology computers and persons with disabilities. *Communications of the ACM*, 35 (5): 36–45. doi:10.1145/129875.129877.

Van Brummelen, J., Weng, K., Lin, P., et al. (2020a) “CONVO: What does conversational programming need?” In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*. 16 July 2020. Institute of Electrical and Electronics Engineers (IEEE). pp. 1–5. doi:10.1109/VL/HCC50065.2020.9127277.

Bruno Garza, J.L. and Young, J.G. (2015) A literature review of the effects of computer input device design on biomechanical loading and musculoskeletal outcomes during computer work. *Work*, 52 (2): 217–230. doi:10.3233/WOR-152161.

Busjahn, T., Bednarik, R., Begel, A., et al. (2015) "Eye Movements in Code Reading: Relaxing the Linear Order." In *2015 IEEE 23rd International Conference on Program Comprehension*. 2015. pp. 255–265. doi:10.1109/ICPC.2015.36.

Busjahn, T., Schulte, C. and Busjahn, A. (2011) "Analysis of Code Reading to Gain More Insight in Program Comprehension." In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. Koli Calling '11 New York, NY, USA, 2011. ACM. pp. 1–9. doi:10.1145/2094131.2094133.

Cambre, J. and Kulkarni, C. (2019) One Voice Fits All? Social Implications and Research Challenges of Designing Voices for Smart Devices. *Proceedings of the ACM on Human-Computer Interaction*, 3 (CSCW): 223:1-223:19. doi:10.1145/3359325.

Cambre, J., Williams, A.C., Razi, A., et al. (2021) "Firefox Voice: An Open and Extensible Voice Assistant Built Upon the Web." In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21 New York, NY, USA, 6 May 2021. Association for Computing Machinery. pp. 1–18. doi:10.1145/3411764.3445409.

Casarini, M., Porta, M. and Dondi, P. (2020) "A Gaze-Based Web Browser with Multiple Methods for Link Selection." In *Symposium on Eye Tracking Research and Applications*. Stuttgart Germany, 2 June 2020. ACM. pp. 1–8. doi:10.1145/3379157.3388929.

Castellina, E., Corno, F. and Pellegrino, P. (2008) "Integrated speech and gaze control for realistic desktop environments." In *Proceedings of the 2008 symposium on Eye tracking research & applications - ETRA '08*. ETRA '08 New York, NY, USA, 2008. ACM. pp. 79–79. doi:10.1145/1344471.1344492.

Cavalcante, A.B. and Lorens, L. (2015) "Use case: a mobile speech assistant for people with speech disorders." In *Proceedings of the 7th Language & Technology Conference*. 2015. pp. 192–197.

Chadha, H., Mhatre, S., Ganatra, U., et al. (2018) "HTML Voice." In *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*. August 2018. IEEE. pp. 1–4. doi:10.1109/ICCUBEA.2018.8697733.

Chakraborty, P., Roy, D., Rahman, M.Z., et al. (2019) Eye gaze controlled virtual keyboard. *International Journal of Recent Technology and Engineering*, 8 (4): 3264–3269.

Chhekur, H. (2023) *Colon IDE*. Available at: https://github.com/Chhekur/colon-ide (Accessed: 11 May 2022).

Chi, S. and Naik, A.K. (2019) *An Empirical Study of How Developers Use Autocompletion*.

Clark, B. and Sharif, B. (2017) “iTraceVis: Visualizing Eye Movement Data Within Eclipse.” In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. 2017. pp. 22–32. doi:10.1109/VISSOFT.2017.30.

Clark, L., Doyle, P., Garaialde, D., et al. (2019) The State of Speech in HCI: Trends, Themes and Challenges. *Interacting with Computers*, 31 (4): 349–371. doi:10.1093/iwc/iwz016.

CodeMirror (2023) *CodeMirror*. Available at: https://codemirror.net/.

CodePen (2023). Available at: https://codepen.io/ (Accessed: 7 December 2022).

Cohen, P.R., Johnston, M., McGee, D., et al. (1997) “QuickSet: multimodal interaction for distributed applications.” In *Proceedings of the fifth ACM international conference on Multimedia - MULTIMEDIA ’97*. Seattle, Washington, United States, 1997. ACM Press. pp. 31–40. doi:10.1145/266180.266328.

Creed, C. (2018) Assistive technology for disabled visual artists: exploring the impact of digital technologies on artistic practice. *Disability and Society*, pp. 1–17. doi:10.1080/09687599.2018.1469400.

Creed, C., Frutos-Pascual, M. and Williams, I. (2020) “Multimodal Gaze Interaction for Creative Design.” In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20 New York, NY, USA, 23 April 2020. Association for Computing Machinery. pp. 1–13. doi:10.1145/3313831.3376196.

Crosby, M.E. and Stelovsky, J. (1990) How do we read algorithms? A case study. *Computer*, 23 (1): 25–35. doi:10.1109/2.48797.

Crow, K.L. (2008) Four Types of Disabilities: Their Impact on Online Learning. *TechTrends*, 52 (1): 51–55.

Cursorless (2023). Available at: https://marketplace.visualstudio.com/items?itemName=pokey.cursorless (Accessed: 26 March 2022).

Dai, L., Goldman, R., Sears, A., et al. (2003) Speech-based cursor control: a study of grid-based solutions. *ACM SIGACCESS Accessibility and Computing*, (77–78): 94–101. doi:10.1145/1029014.1028648.

Dalmaijer, E. (2014) *Is the low-cost EyeTribe eye tracker any good for research?* PeerJ PrePrints.

Danis, C., Comerford, L., Janke, E., et al. (1994) “Storywriter: a speech oriented editor.” In *Conference Companion on Human Factors in Computing Systems*. CHI ’94 New York, NY, USA, 28 April 1994. Association for Computing Machinery. pp. 277–278. doi:10.1145/259963.260490.

Danis, C. and Karat, J. (1995) “Technology-driven design of speech recognition systems.” In *Proceedings of the conference on Designing interactive systems processes, practices, methods, & techniques - DIS ’95*. Ann Arbor, Michigan, United States, 1995. ACM Press. pp. 17–24. doi:10.1145/225434.225437.

De La Paz, S. (1999) Composing via Dictation and Speech Recognition Systems: Compensatory Technology for Students with Learning Disabilities. *Learning Disability Quarterly*, 22 (3): 173–182. doi:10.2307/1511284.

De León Cordero, D., Ayala, C. and Ordóñez, P. (2021) “Kavita Project: Voice Programming for People with Motor Disabilities.” In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS ’21 New York, NY, USA, 17 October 2021. Association for Computing Machinery. pp. 1–3. doi:10.1145/3441852.3476516.

Delimarschi, D., Swartzendruber, G. and Kagdi, H. (2014) “Enabling integrated development environments with natural user interface interactions.” In *Proceedings of the 22Nd International Conference on Program Comprehension*. ICPC 2014 New York, NY, USA, 2014. ACM. pp. 126–129. doi:10.1145/2597008.2597791.

DeLine, R. and Rowan, K. (2010) “Code canvas.” In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE ’10*. ICSE ’10 New York, NY, USA, 2010. ACM. pp. 207–207. doi:10.1145/1810295.1810331.

Derboven, J., Huyghe, J. and De Grooff, D. (2014) “Designing voice interaction for people with physical and speech impairments.” In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*. NordiCHI ’14 New York, NY, USA, 2014. ACM. pp. 217–226. doi:10.1145/2639189.2639252.

Desilets, A. (2001) VoiceGrip: A tool for programming-by-voice. *International Journal of Speech Technology*, 4 (2): 103–116. doi:10.1023/A:1011323308477.

Désilets, A., Fox, D.C. and Norton, S. (2006) “VoiceCode: An Innovative Speech Interface for Programming-by-voice.” In *CHI ’06 extended abstracts on Human factors in computing systems - CHI EA ’06*. New York, 2006. ACM. pp. 239–242. doi:10.1145/1125451.1125502.

Di Ferrante, L. and Bouchard, J. (2020) The Nature and Function of Vocalizations in Atypical Communication. *Current Developmental Disorders Reports*, 7 (1): 23–27. doi:10.1007/s40474-020-00186-x.

Disability and health (2023). Available at: https://www.who.int/news-room/fact-sheets/detail/disability-and-health (Accessed: 17 July 2022).

Divine, G., Norton, H.J., Hunt, R., et al. (2013) A Review of Analysis and Sample Size Calculation Considerations for Wilcoxon Tests. *Anesthesia & Analgesia*, 117 (3): 699. doi:10.1213/ANE.0b013e31827f53d7.

Dobson, R. and Giovannoni, G. (2019) Multiple sclerosis – a review. *European Journal of Neurology*, 26 (1): 27–40. doi:10.1111/ene.13819.

Dorr, M., Böhme, M., Martinetz, T., et al. (2007) *Gaze beats mouse: a case study*., p. 4.

Dragon (2023) *Dragon Speech Recognition - Get More Done by Voice | Nuance*. Available at: https://www.nuance.com/dragon.html.

Drewes, H. and Schmidt, A. (2007) "Interacting with the Computer Using Gaze Gestures." In Baranauskas, C., Palanque, P., Abascal, J., et al. (eds.). *Human-Computer Interaction – INTERACT 2007*. Lecture Notes in Computer Science Berlin, Heidelberg, 2007. Springer. pp. 475–488. doi:10.1007/978-3-540-74800-7_43.

Duan, D., Goemans, N., Takeda, S., et al. (2021) Duchenne muscular dystrophy. *Nature Reviews Disease Primers*, 7 (1): 1–19. doi:10.1038/s41572-021-00248-3.

Duchowski, A.T. (2018) Gaze-based interaction: A 30 year retrospective. *Computers and Graphics (Pergamon)*, 73: 59–69. doi:10.1016/j.cag.2018.04.002.

Dumas, B., Lalanne, D. and Oviatt, S. (2009) "Multimodal Interfaces: A Survey of Principles, Models and Frameworks." In Lalanne, D. and Kohlas, J. (eds.) *Human Machine Interaction: Research Results of the MMI Program*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. pp. 3–26. doi:10.1007/978-3-642-00437-7_1.

Electronjs (2023) Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS. *Electron*. Available at: https://www.electronjs.org/.

Elepfandt, M. and Grund, M. (2012) "Move it there, or not?: The design of voice commands for gaze with speech." In *Proceedings of the 4th Workshop on Eye Gaze in Intelligent Human Machine Interaction, Gaze-In 2012*. Gaze-In '12 New York, NY, USA, 2012. ACM. pp. 1–3. doi:10.1145/2401836.2401848.

Elsahar, Y., Hu, S., Bouazza-Marouf, K., et al. (2019) Augmentative and Alternative Communication (AAC) Advances: A Review of Configurations for Individuals with a Speech Disability. *Sensors*, 19 (8): 1911. doi:10.3390/s19081911.

Elsahar, Y., Hu, S., Bouazza-Marouf, K., et al. (2021) A study of decodable breathing patterns for augmentative and alternative communication. *Biomedical Signal Processing and Control*, 65: 102303. doi:10.1016/j.bspc.2020.102303.

Emmet (2023). *NA*. Available at: https://docs.emmet.io/.

Equality Act (2023). Available at: https://www.gov.uk/definition-of-disability-under-equality-act-2010 (Accessed: 28 June 2022).

Errattahi, R., El Hannani, A. and Ouahmane, H. (2018) Automatic Speech Recognition Errors Detection and Correction: A Review. *Procedia Computer Science*, 128: 32–37. doi:10.1016/j.procs.2018.03.005.

Evans, P.M., Evans, S.J. and Alberman, E. (1990) Cerebral palsy: why we must plan for survival. *Archives of Disease in Childhood*, 65 (12): 1329–1333. doi:10.1136/adc.65.12.1329.

Eye Tribe (2023) The Eye Tribe. *copenhagen, denmark*. Available at: https://theeyetribe.com/theeyetribe.com/about/index.html.

Feng, J. and Sears, A. (2004) Using confidence scores to improve hands-free speech based navigation in continuous dictation systems. *ACM Transactions on Computer-Human Interaction*, 11 (4): 329–356. doi:10.1145/1035575.1035576.

Ferracani, A., Faustino, M., Giannini, G.X., et al. (2017) "Natural Experiences in Museums through Virtual Reality and Voice Commands." In *Proceedings of the 25th ACM international conference on Multimedia*. MM '17 New York, NY, USA, 19 October 2017. Association for Computing Machinery. pp. 1233–1234. doi:10.1145/3123266.3127916.

Ferreira, A.O., Ferreira, S.B.L. and da Silveira, D.S. (2012) Accessibility for People with Cerebral Palsy: The use of Blogs as an Agent of Social Inclusion. *Procedia Computer Science*, 14: 245–253. doi:10.1016/j.procs.2012.10.028.

Filippidou, F. and Moussiades, L. (2020) "A Benchmarking of IBM, Google and Wit Automatic Speech Recognition Systems." In Maglogiannis, I., Iliadis, L. and Pimenidis, E. (eds.). *Artificial Intelligence Applications and Innovations*. IFIP Advances in Information and Communication Technology Cham, 2020. Springer International Publishing. pp. 73–82. doi:10.1007/978-3-030-49161-1_7.

Findlater, L., Moffatt, K., Froehlich, J.E., et al. (2017) "Comparing Touchscreen and Mouse Input Performance by People With and Without Upper Body Motor Impairments." In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. New York, New York, USA, 2017. ACM Press. pp. 6056–6061. doi:10.1145/3025453.3025603.

Folmer, E., Liu, F. and Ellis, B. (2011) "Navigating a 3D avatar using a single switch." In *Proceedings of the 6th International Conference on Foundations of Digital Games*. FDG '11 New York, NY, USA, 29 June 2011. Association for Computing Machinery. pp. 154–160. doi:10.1145/2159365.2159386.

Freedman, A. (1995) *The computer glossary (7th ed.): the complete illustrated dictionary*. USA: American Management Assoc., Inc.

Furui, S. (2010) "History and Development of Speech Recognition." In Chen, F. and Jokinen, K. (eds.) *Speech Technology: Theory and Applications*. New York, NY: Springer US. pp. 1–18. doi:10.1007/978-0-387-73819-2_1.

Gaikwad, S.K., Gawali, B.W. and Yannawar, P. (2010) A review on speech recognition technique. *International Journal of Computer Applications*, 10 (3): 16–24.

Gatchalian, C. (2019) *Assistive Technologies in the 21st Century*. Available at: https://pressbooks.pub/techandcurr2019/chapter/21st-century-assistive-tech/ (Accessed: 5 November 2022).

Geytenbeek, J.J., Heim, M.M., Vermeulen, R.J., et al. (2010) Assessing comprehension of spoken language in nonspeaking children with cerebral palsy: application of a newly developed computer-based instrument. *Augmentative and alternative communication*, 26 (2): 97–107.

Gheran, B.-F., Ungurean, O.-C. and Vatavu, R.-D. (2018) “Toward Smart Rings as Assistive Devices for People with Motor Impairments: A Position Paper.” In *RoCHI*. 2018. pp. 99–106.

Ghosh, D., Foong, P.S., Zhao, S., et al. (2020) “EYEditor: Towards On-the-Go Heads-Up Text Editing Using Voice and Manual Input.” In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20 New York, NY, USA, 21 April 2020. Association for Computing Machinery. pp. 1–13. doi:10.1145/3313831.3376173.

Ghosh Sanjay, and Joshi, A., and and Tripathi Sanjay (2013) “Empirical Evaluation of Multimodal Input Interactions.” In Yamamoto, S. (ed.). *Human Interface and the Management of Information. Information and Interaction Design*. Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. pp. 37–47.

Giakoumis, D., Kaklanis, N., Votis, K., et al. (2014) Enabling user interface developers to experience accessibility limitations through visual, hearing, physical and cognitive impairment simulation. *Universal Access in the Information Society*, 13 (2): 227–248. doi:10.1007/s10209-013-0309-0.

Glücker, H., Raab, F., Echtler, F., et al. (2014) “EyeDE: Gaze-enhanced Software Development Environments.” In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*. CHI EA ’14 New York, NY, USA, 2014. ACM. pp. 1555–1560. doi:10.1145/2559206.2581217.

Good, J. and Howland, K. (2017) Programming language, natural language? Supporting the diverse computational activities of novice programmers. *Journal of Visual Languages & Computing*, 39: 78–92. doi:10.1016/j.jvlc.2016.10.008.

Google Assistant (2023) *Google Assistant, your own personal Google*. Available at: https://assistant.google.com/intl/en_uk/ (Accessed: 24 March 2023).

Google Speech (2023) *Speech-to-Text: Automatic Speech Recognition*. Available at: https://cloud.google.com/speech-to-text (Accessed: 8 January 2023).

Gordon, B.M. (2013) *Improving Spoken Programming Through Language Design and the Incorporation of Dynamic Context*. Available at: https://digitalrepository.unm.edu/cs_etds.https://digitalrepository.unm.edu/cs_etds/28.

Gov UK (2021) *Family Resources Survey: financial year 2020 to 2021*. Available at: https://www.gov.uk/government/statistics/family-resources-survey-financial-year-2020-to-2021/family-resources-survey-financial-year-2020-to-2021 (Accessed: 28 June 2022).

Graham, C.W., Inge, K.J., Wehman, P., et al. (2018) Barriers and facilitators to employment as reported by people with physical disabilities: An across disability type analysis Inge, K.J. and Wehman, P. (eds.). *Journal of Vocational Rehabilitation*, 48 (2): 207–218. doi:10.3233/JVR-180929.

Gratton, L. (2021) Four Principles to Ensure Hybrid Work Is Productive Work. *MIT Sloan Management Review*, 62 (2): 11A-16A.

Groenewald, C., Anslow, C., Islam, J., et al. (2016) *Understanding 3D mid-air hand gestures with interactive surfaces and displays: a systematic literature review*.

Grossman, T. and Balakrishnan, R. (2005) “The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor’s activation area.” In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2005. pp. 281–290.

Guerreiro, T., Nicolau, H., Jorge, J., et al. (2010) “Towards accessible touch interfaces.” In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. ASSETS ’10 New York, NY, USA, 25 October 2010. Association for Computing Machinery. pp. 19–26. doi:10.1145/1878803.1878809.

Gulzar, T., Singh, A., Kumar, D., et al. (2014) *A Systematic Analysis of Automatic Speech Recognition: An Overview*., 4.

Gür, D., Schäfer, N., Kupnik, M., et al. (2020) A Human–Computer Interface Replacing Mouse and Keyboard for Individuals with Limited Upper Limb Mobility. *Multimodal Technologies and Interaction*, 4 (4): 84. doi:10.3390/mti4040084.

Hanam, Q., Brito, F.S.D.M. and Mesbah, A. (2016) “Discovering bug patterns in Javascript.” In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. November 2016. Association for Computing Machinery. pp. 144–156. doi:10.1145/2950290.2950308.

Hannun, A. (2021) *The History of Speech Recognition to the Year 2030*. doi:10.48550/arXiv.2108.00084.

Hansen, J.P., Johansen, A.S., Hansen, D.W., et al. (2003) “Command Without a Click: Dwell Time Typing by Mouse and Gaze Selections.” In *Interact*. 2003. Citeseer. pp. 121–128.

Haque, F.B., Shuvo, T.H. and Khan, R. (2021) “Head Motion Controlled Wheelchair for Physically Disabled People.” In *2021 Second International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. December 2021. pp. 1–6. doi:10.1109/ICSTCEE54422.2021.9708577.

Harada, S., Landay, J.A., Malkin, J., et al. (2006) “The vocal joystick: evaluation of voice-based cursor control techniques.” In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*. Assets ’06 New York, NY, USA, 23 October 2006. Association for Computing Machinery. pp. 197–204. doi:10.1145/1168987.1169021.

Hauptmann, A.G. and Rudnicky, A.I. (1990) “A comparison of speech and typed input.” In *Proceedings of the workshop on Speech and Natural Language*. HLT ’90 USA, 24 June 1990. Association for Computational Linguistics. pp. 219–224. doi:10.3115/116580.116652.

Heikkilä, H. (2013) “EyeSketch: A drawing application for gaze control.” In *ACM International Conference Proceeding Series*. ETSA ’13 New York, NY, USA, 2013. ACM. pp. 71–74. doi:10.1145/2509315.2509332.

Heikkilä, H. and Räihä, K.-J. (2010) Speed and Accuracy of Gaze Gestures. *Journal of Eye Movement Research*, 3 (2). doi:10.16910/jemr.3.2.1.

Hemmingsson, H. and Borgestig, M. (2020) Usability of Eye-Gaze Controlled Computers in Sweden: A Total Population Survey. *International Journal of Environmental Research and Public Health*, 17 (5): 1639. doi:10.3390/ijerph17051639.

Henley, A.Z. and Fleming, S.D. (2014) “The patchworks code editor: toward faster navigation with less code arranging and fewer navigation mistakes.” In *Proc. CHI*. CHI ’14 New York, NY, USA, 2014. ACM. pp. 2511–2520. doi:10.1145/2556288.2557073.

Henzen, A. and Nohama, P. (2016) “Adaptable virtual keyboard and mouse for people with special needs.” In *2016 Future Technologies Conference (FTC)*. December 2016. pp. 1357–1360. doi:10.1109/FTC.2016.7821782.

Hill-Briggs, F., Dial, J.G., Morere, D.A., et al. (2007) Neuropsychological assessment of persons with physical disability, visual impairment or blindness, and hearing impairment or deafness. *Archives of Clinical Neuropsychology*, 22 (3): 389–404. doi:10.1016/j.acn.2007.01.013.

Hogan, A., Kyaw-Myint, S.M., Harris, D., et al. (2012) Workforce Participation Barriers for People With Disability. *International Journal of Disability Management*, 7: 1–9. doi:10.1017/idm.2012.1.

Hornof, A.J. and Cavender, A. (2005) “EyeDraw.” In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI ’05*. CHI ’05 New York, NY, USA, 2005. ACM. pp. 161–161. doi:10.1145/1054972.1054995.

Hu, R., Zhu, S., Feng, J., et al. (2011) “Use of Speech Technology in Real Life Environment.” In Stephanidis, C. (ed.). *Universal Access in Human-Computer Interaction. Applications and Services*. Lecture Notes in Computer Science Berlin, Heidelberg, 2011. Springer. pp. 62–71. doi:10.1007/978-3-642-21657-2_7.

Hubbell, T.J., Langan, D.D. and Hain, T.F. (2006) “A voice-activated syntax-directed editor for manually disabled programmers.” In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility - Assets ’06*. 2006. pp. 205–205. doi:10.1145/1168987.1169022.

Huckauf, A. and Urbina, M.H. (2008) On object selection in gaze controlled environments. *Journal of Eye Movement Research*, 2 (4). doi:10.16910/jemr.2.4.4.

Hurst, A. and Tobias, J. (2011) “Empowering individuals with do-it-yourself assistive technology.” In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. ASSETS ’11 New York, NY, USA, 24 October 2011. Association for Computing Machinery. pp. 11–18. doi:10.1145/2049536.2049541.

Hutchinson, T.E., White, K.P., Martin, W.N., et al. (1989) Human-computer interaction using eye-gaze input. *IEEE Transactions on Systems, Man, and Cybernetics*, 19 (6): 1527–1534. doi:10.1109/21.44068.

Hyrskykari, A., Istance, H. and Vickers, S. (2012) “Gaze gestures or dwell-based interaction?” In *Proceedings of the Symposium on Eye Tracking Research and Applications*. Santa Barbara California, 28 March 2012. ACM. pp. 229–232. doi:10.1145/2168556.2168602.

Iacono, T., Lyon, K., Johnson, H., et al. (2013) Experiences of adults with complex communication needs receiving and using low tech AAC: an Australian context. *Disability and Rehabilitation: Assistive Technology*, 8 (5): 392–401. doi:10.3109/17483107.2013.769122.

IBM Watson (2023) *IBM Watson Speech to Text - Speech to Text*. Available at: https://www.ibm.com/uk-en/cloud/watson-speech-to-text (Accessed: 8 January 2023).

Igarashi, T. and Hughes, J.F. (2001) “Voice as sound: using non-verbal voice input for interactive control.” In *Proceedings of the 14th annual ACM symposium on User interface software and technology*. UIST ’01 New York, NY, USA, 11 November 2001. Association for Computing Machinery. pp. 155–156. doi:10.1145/502348.502372.

Irwin, C.B. and Sesto, M.E. (2012) Performance and touch characteristics of disabled and non-disabled participants during a reciprocal tapping task using touch screen technology. *Applied Ergonomics*, 43 (6): 1038–1043. doi:10.1016/j.apergo.2012.03.003.

Istance, H., Vickers, S. and Hyrskykari, A. (2009) “Gaze-based interaction with massively multiplayer on-line games.” In *CHI ’09 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’09 New York, NY, USA, 4 April 2009. Association for Computing Machinery. pp. 4381–4386. doi:10.1145/1520340.1520670.

Jacob, R.J. (1990) “What you look at is what you get: eye movement-based interaction techniques.” In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1990. pp. 11–18.

Jacob, R.J. and Karn, K.S. (2003) “Eye tracking in human-computer interaction and usability research: Ready to deliver the promises.” In *The mind’s eye*. Elsevier. pp. 573–605.

Jaddoh, A., Loizides, F. and Rana, O. (2021) Non-verbal interaction with virtual home assistants for people with dysarthria. *Journal on Technology & Persons with Disabilities*, 9 (13): 71–84.

Jamieson, A., Murray, L., Stankovic, L., et al. (2021) Human Activity Recognition of Individuals with Lower Limb Amputation in Free-Living Conditions: A Pilot Study. *Sensors*, 21 (24): 8377. doi:10.3390/s21248377.

Jessup, S., Willis, S.M., Alarcon, G., et al. (2021) *Using Eye-Tracking Data to Compare Differences in Code Comprehension and Code Perceptions between Expert and Novice Programmers*. Available at: http://hdl.handle.net/10125/70624 (Downloaded: 20 January 2023).

Joshi, P. and Bein, D. (2020) “Audible Code, a Voice-Enabled Programming Extension of Visual Studio Code.” In *Advances in Intelligent Systems and Computing*. 2020. Springer. pp. 335–341. doi:10.1007/978-3-030-43020-7_44.

Juang, B.-H. and Furui, S. (2000) Automatic recognition and understanding of spoken language - a first step toward natural human-machine communication. *Proceedings of the IEEE*, 88 (8): 1142–1165. doi:10.1109/5.880077.

Jung, H., So, S., Oh, C., et al. (2019) “TurtleTalk: An educational programming game for children with voice user interface.” In *Conference on Human Factors in Computing Systems - Proceedings*. New York, NY, USA, May 2019. Association for Computing Machinery. pp. 1–6. doi:10.1145/3290607.3312773.

Kabir, M.R., Abedin, M.I., Ahmed, R., et al. (2022) *Auxilio: A Sensor-Based Wireless Head-Mounted Mouse for People with Upper Limb Disability*. doi:10.48550/arXiv.2210.04483.

Kaiser, E., Olwal, A., McGee, D., et al. (2003) “Mutual disambiguation of 3D multimodal interaction in augmented and virtual reality.” In *Proceedings of the 5th international conference on Multimodal interfaces*. ICMI ’03 New York, NY, USA, 5 November 2003. Association for Computing Machinery. pp. 12–19. doi:10.1145/958432.958438.

Kane, S.K., Jayant, C., Wobbrock, J.O., et al. (2009) “Freedom to roam: a study of mobile device adoption and accessibility for people with visual and motor disabilities.” In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*. Assets ’09 New York, NY, USA, 25 October 2009. Association for Computing Machinery. pp. 115–122. doi:10.1145/1639642.1639663.

Kang, R., Guo, A., Laput, G., et al. (2019) “Minuet: Multimodal Interaction with an Internet of Things.” In *Symposium on Spatial User Interaction*. New Orleans LA USA, 19 October 2019. ACM. pp. 1–10. doi:10.1145/3357251.3357581.

Karat, C.-M., Halverson, C., Horn, D., et al. (1999) “Patterns of entry and correction in large vocabulary continuous speech recognition systems.” In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. CHI ’99 New York, NY, USA, 1 May 1999. Association for Computing Machinery. pp. 568–575. doi:10.1145/302979.303160.

Karimullah, A.S. and Sears, A. (2002) “Speech-based cursor control.” In *Proceedings of the fifth international ACM conference on Assistive technologies*. Assets ’02 New York, NY, USA, 8 July 2002. Association for Computing Machinery. pp. 178–185. doi:10.1145/638249.638282.

Karl, L.R., Pettey, M. and Shneiderman, B. (1992) Speech-Activated Versus Mouse-Activated Commands for Word Processing Applications: An Empirical Evaluation. *undefined*. Available at: https://www.semanticscholar.org/paper/Speech-Activated-Versus-Mouse-Activated-Commands-An-Karl-Pettey/12c5940edd3dfafc0d2dea56fe9b1db0bd9a2fa8 (Accessed: 13 August 2022).

Karpov, A.A., Ronzhin, A.L., Nechaev, A.I., et al. (2004) “Assistive multimodal system based on speech recognition and head tracking.” In *9th Conference Speech and Computer*. 2004.

Karrer, T., Krämer, J.P., Diehl, J., et al. (2011) “Stacksplorer: Call graph navigation helps increasing code maintenance efficiency.” In *UIST’11 - Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. UIST ’11 New York, NY, USA, 2011. ACM. pp. 217–224. doi:10.1145/2047196.2047225.

Kennedy, P.R., Bakay, R.A.E., Moore, M.M., et al. (2000) Direct control of a computer from the human central nervous system. *IEEE Transactions on Rehabilitation Engineering*, 8 (2): 198–202. doi:10.1109/86.847815.

Khamis, M., Alt, F., Hassib, M., et al. (2016) “Gazetouchpass: Multimodal authentication using gaze and touch on mobile devices.” In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 2016. pp. 2156–2164.

Kim, Y.S., Dontcheva, M., Adar, E., et al. (2019) “Vocal shortcuts for creative experts.” In *Conference on Human Factors in Computing Systems - Proceedings*. 2019. ACM. doi:10.1145/3290605.3300562.

Ko, A.J., Myers, B.A., Coblenz, M.J., et al. (2006) An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering*, 32 (12): 971–987. doi:10.1109/TSE.2006.116.

Koester, H.H. (2001) User Performance With Speech Recognition: A Literature Review. *Assistive Technology*, 13 (2): 116–130. doi:10.1080/10400435.2001.10132042.

Kopp, S. and Bergmann, K. (2017) Using Cognitive Models. *The Handbook of Multimodal-Multisensor Interfaces, Volume 1: Foundations, User Modeling, and Common Modality Combinations*, p. 239.

Kowalski, J., Jaskulska, A., Skorupska, K., et al. (2019) “Older Adults and Voice Interaction: A Pilot Study with Google Home.” In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI EA ’19 New York, NY, USA, 2 May 2019. Association for Computing Machinery. pp. 1–6. doi:10.1145/3290607.3312973.

Krämer, J.-P., Karrer, T., Kurz, J., et al. (2013) “How tools in IDEs shape developers’ navigation behavior.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’13 New York, NY, USA, 27 April 2013. Association for Computing Machinery. pp. 3073–3082. doi:10.1145/2470654.2466419.

Krämer, J.-P., Kurz, J., Karrer, T., et al. (2012) “Blaze: Supporting Two-phased Call Graph Navigation in Source Code.” In *CHI ’12 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’12 New York, NY, USA, 2012. ACM. pp. 2195–2200. doi:10.1145/2212776.2223775.

Krausz, G., Scherer, R., Korisek, G., et al. (2003) Critical Decision-Speed and Information Transfer in the “Graz Brain–Computer Interface.” *Applied Psychophysiology and Biofeedback*, 28 (3): 233–240. doi:10.1023/A:1024637331493.

Krishna, G., Tran, C., Yu, J., et al. (2019) “Speech Recognition with No Speech or with Noisy Speech.” In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2019. pp. 1090–1094. doi:10.1109/ICASSP.2019.8683453.

Kristensson, P.O. and Vertanen, K. (2012) “The potential of dwell-free eye-typing for fast assistive gaze communication.” In *Proceedings of the Symposium on Eye Tracking Research and Applications - ETRA ’12*. New York, New York, USA, 2012. ACM Press. pp. 241–241. doi:10.1145/2168556.2168605.

Kumar, C., Hedeshy, R., MacKenzie, I.S., et al. (2020) “TAGSwipe: Touch Assisted Gaze Swipe for Text Entry.” In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20 New York, NY, USA, 21 April 2020. Association for Computing Machinery. pp. 1–12. doi:10.1145/3313831.3376317.

Kumar, C., Menges, R., Müller, D., et al. (2017) “Chromium based framework to include gaze interaction in web browser.” In *26th International World Wide Web Conference 2017, WWW 2017 Companion*. WWW ’17 Companion Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee. pp. 219–223. doi:10.1145/3041021.3054730.

Kumar, M., Paepcke, A. and Winograd, T. (2007) “Eyepoint: practical pointing and selection using gaze and keyboard.” In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2007. pp. 421–430.

Kurauchi, A., Feng, W., Joshi, A., et al. (2016) “EyeSwipe: Dwell-free Text Entry Using Gaze Paths.” In *Proceedings of the 2016 CHI Conference on Human Factors in Computing*

Systems. San Jose California USA, 7 May 2016. ACM. pp. 1952–1956. doi:10.1145/2858036.2858335.

Kytö, M., Ens, B., Piumsomboon, T., et al. (2018) "Pinpointing: Precise Head- and Eye-Based Target Selection for Augmented Reality." In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18 New York, NY, USA, 19 April 2018. Association for Computing Machinery. pp. 1–14. doi:10.1145/3173574.3173655.

Labour Statistics (2021) *Employed persons by disability status, occupation, and sex, 2021 annual averages - 2021 A01 Results*. Available at: https://www.bls.gov/news.release/disabl.t03.htm (Accessed: 9 October 2022).

Lai, W.C., Erickson, B.J., Mlynarek, R.A., et al. (2018) Chronic lateral epicondylitis: challenges and solutions. *Open Access Journal of Sports Medicine*, 9: 243–251. doi:10.2147/OAJSM.S160974.

Lancioni, G.E., O'Reilly, M.F., Singh, N.N., et al. (2008) Microswitch-based programs for persons with multiple disabilities: An overview of some recent developments. *Perceptual and Motor Skills*, 106 (2): 355–370.

Lankford, C. (2000) "Effective eye-gaze input into Windows." In *Proceedings of the 2000 symposium on Eye tracking research & applications*. ETRA '00 New York, NY, USA, 8 November 2000. Association for Computing Machinery. pp. 23–27. doi:10.1145/355017.355021.

LaToza, T.D., Venolia, G. and DeLine, R. (2006) "Maintaining Mental Models: A Study of Developer Work Habits." In *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06 New York, NY, USA, 2006. ACM. pp. 492–501. doi:10.1145/1134285.1134355.

Lau, J., Zimmerman, B. and Schaub, F. (2018) Alexa, Are You Listening? Privacy Perceptions, Concerns and Privacy-seeking Behaviors with Smart Speakers. *Proceedings of the ACM on Human-Computer Interaction*, 2 (CSCW): 102:1-102:31. doi:10.1145/3274371.

Lea, C., Huang, Z., Jain, D., et al. (2022) "Nonverbal Sound Detection for Disordered Speech." In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2022. pp. 7397–7401. doi:10.1109/ICASSP43922.2022.9747227.

Lee, T.-H. (2005) Ergonomic comparison of operating a built-in touch-pad pointing device and a trackball mouse on posture and muscle activity. *Perceptual and motor skills*, 101 (3): 730–736.

Leigh, P.N. and Ray-Chaudhuri, K. (1994) Motor neuron disease. *Journal of Neurology, Neurosurgery, and Psychiatry*, 57 (8): 886–896.

Lenoir, H., Mares, O. and Carlier, Y. (2019) Management of lateral epicondylitis. *Orthopaedics & Traumatology: Surgery & Research*, 105 (8, Supplement): S241–S246. doi:10.1016/j.otsr.2019.09.004.

Lewien, R. (2021) “GazeHelp: Exploring Practical Gaze-assisted Interactions for Graphic Design Tools.” In *ACM Symposium on Eye Tracking Research and Applications*. Virtual Event Germany, 25 May 2021. ACM. pp. 1–4. doi:10.1145/3450341.3458764.

Li, Q., Sun, L. and Duan, J. (2005) “Web page viewing behavior of users: an eye-tracking study.” In *Proceedings of ICSSSM ’05. 2005 International Conference on Services Systems and Services Management, 2005.* June 2005. pp. 244-249 Vol. 1. doi:10.1109/ICSSSM.2005.1499470.

Lin, Y., Wu, C., Hou, T., et al. (2016) Tracking Students’ Cognitive Processes During Program Debugging—An Eye-Movement Approach. *IEEE Transactions on Education*, 59 (3): 175–186. doi:10.1109/TE.2015.2487341.

Lincoln, A.E., Vernick, J.S., Ogaitis, S., et al. (2000) Interventions for the primary prevention of work-related carpal tunnel syndrome. *American Journal of Preventive Medicine*, 18 (4, Supplement 1): 37–50. doi:10.1016/S0749-3797(00)00140-9.

Lindsay, S. (2011) Discrimination and other barriers to employment for teens and young adults with disabilities. *Disability and Rehabilitation*, 33 (15–16): 1340–1350. doi:10.3109/09638288.2010.531372.

Liu, Y., Zhang, C., Lee, C., et al. (2015) “GazeTry: Swipe text typing using gaze.” In *OzCHI 2015: Being Human - Conference Proceedings*. New York, New York, USA, December 2015. Association for Computing Machinery, Inc. pp. 192–196. doi:10.1145/2838739.2838804.

London RSI (2023) *Central London RSI Support Group - Home | Facebook*. Available at: https://www.facebook.com/CentralLondonRsiSupportGroup/.

Luger, E. and Sellen, A. (2016) ““Like Having a Really Bad PA”: The Gulf between User Expectation and Experience of Conversational Agents.” In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI ’16 New York, NY, USA, 7 May 2016. Association for Computing Machinery. pp. 5286–5297. doi:10.1145/2858036.2858288.

Lutteroth, C., Penkar, M. and Weber, G. (2015) “Gaze vs. Mouse: A Fast and Accurate Gaze-Only Click Alternative.” In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST ’15 New York, NY, USA, 5 November 2015. Association for Computing Machinery. pp. 385–394. doi:10.1145/2807442.2807461.

Macdonald, S.J. and Clayton, J. (2013) Back to the future, disability and the digital divide. *Disability & Society*, 28 (5): 702–718. doi:10.1080/09687599.2012.732538.

Machado, M.-L., Guincestre, J.-Y., Aimé, M., et al. (2013) Head Pilot: A new webcam-based Head Tracking System tested in permanently disabled patients. *IRBM*, 34 (2): 124–130. doi:10.1016/j.irbm.2013.02.001.

Mackenzie, L., Bhuta, P., Rusten, K., et al. (2016) Communications Technology and Motor Neuron Disease: An Australian Survey of People With Motor Neuron Disease. *JMIR Rehabilitation and Assistive Technologies*, 3 (1): e4017. doi:10.2196/rehab.4017.

Madaan, H. and Gupta, S. (2021) "AI Improving the Lives of Physically Disabled." In Abraham, A., Ohsawa, Y., Gandhi, N., et al. (eds.). *Proceedings of the 12th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2020)*. Advances in Intelligent Systems and Computing Cham, 2021. Springer International Publishing. pp. 103–112. doi:10.1007/978-3-030-73689-7_11.

Majaranta, P., Ahola, U.-K. and Špakov, O. (2009) *Fast Gaze Typing with an Adjustable Dwell Time*. Available at: http://delivery.acm.org/10.1145/1520000/1518758/p357-majaranta.pdf?ip=82.15.1.187&id=1518758&acc=ACTIVE SERVICE&key=BF07A2EE685417C5.B9DB6AC12FAF3618.7F82F319CD1C5BCF.4D4702B0C3E38B35&__acm__=1567517412_7f15fce318648ed0e23ae76d265ce7d6.

Majaranta, P. and Bulling, A. (2014) "Eye Tracking and Eye-Based Human–Computer Interaction." In Fairclough, S.H. and Gilleade, K. (eds.) *Advances in Physiological Computing*. Human–Computer Interaction Series. London: Springer London. pp. 39–65. doi:10.1007/978-1-4471-6392-3_3.

Majaranta, P. and Räihä, K.-J. (2002) "Twenty years of eye typing." In *Proceedings of the symposium on Eye tracking research & applications - ETRA '02*. 2002. doi:10.1145/507072.507076.

Majaranta, P. and Räihä, K.-J. (2007) 9. Text Entry by Gaze: Utilizing Eye-Tracking. *Text Entry Systems: Mobility, Accessibility, Universality*.

Majaranta, P., Räihä, K.-J., Hyrskykari, A., et al. (2019) "Eye Movements and Human-Computer Interaction." In Klein, C. and Ettinger, U. (eds.) *Eye Movement Research: An Introduction to its Scientific Foundations and Applications*. Studies in Neuroscience, Psychology and Behavioral Economics. Cham: Springer International Publishing. pp. 971–1015. doi:10.1007/978-3-030-20085-5_23.

Malik, M., Malik, M.K., Mehmood, K., et al. (2021) Automatic speech recognition: a survey. *Multimedia Tools and Applications*, 80 (6): 9411–9457. doi:10.1007/s11042-020-10073-7.

Maloku, R.S. and Pllana, B.Xh. (2016) HyperCode: Voice aided programming. *IFAC-PapersOnLine*, 49 (29): 263–268. doi:10.1016/J.IFACOL.2016.11.073.

Mani, K. (2018) Ergonomics education for office computer workers: an evidence-based strategy. *Anatomy, Posture, Prevalence, Pain, Treatment and Interventions of Musculoskeletal Disorders*.

Mankoff, J., Fait, H. and Juang, R. (2005) Evaluating accessibility by simulating the experiences of users with vision or motor impairments. *IBM Systems Journal*, 44 (3): 505–517. doi:10.1147/sj.443.0505.

Manogna, S., Vaishnavi, S. and Geethanjali, B. (2010) "Head Movement Based Assist System for Physically Challenged." In *2010 4th International Conference on Bioinformatics and Biomedical Engineering*. June 2010. pp. 1–4. doi:10.1109/ICBBE.2010.5517790.

Marketplace (2023). Available at: https://marketplace.visualstudio.com/search?sortBy=Installs&category=All categories&target=VSCode (Accessed: 7 December 2022).

Marklin, R.W. and Simoneau, G.G. (2001) Effect of Setup Configurations of Split Computer Keyboards on Wrist Angle. *Physical Therapy*, 81 (4): 1038–1048. doi:10.1093/ptj/81.4.1038.

Marshall, B. (2022) *7 Resources for People With Disabilities to Break Into Software Engineering Careers*. Available at: https://www.containiq.com/post/resource-disabilities-software-engineering (Accessed: 9 October 2022).

Martin, G.L. (1989) The utility of speech input in user-computer interfaces. *International Journal of Man-Machine Studies*, 30 (4): 355–375. doi:10.1016/S0020-7373(89)80023-9.

Mavrou, K. and Hoogerwerf, E.-J. (2016) Towards full digital inclusion: the ENTELIS manifesto against the digital divide. *Journal of Assistive Technologies*, 10 (3): 171–174. doi:10.1108/JAT-03-2016-0010.

Mazumder, S., Liu, B., Wang, S., et al. (2019) *Building an Application Independent Natural Language Interface*. Available at: http://arxiv.org/abs/1910.14084.

McCrocklin, S., Humaidan, A. and Edalatishams, E. (2019) "ASR dictation program accuracy: Have current programs improved." In *Proceedings of the 10th pronunciation in second language learning and teaching conference*. 2019. Iowa State University Ames, IA. pp. 191–200.

McLoone, H., Jacobson, M., Hegg, C., et al. (2010) User-centered design and evaluation of a next generation fixed-split ergonomic keyboard. *Work (Reading, Mass.)*, 37: 445–56. doi:10.3233/WOR-2010-1109.

Mease, P., Strand, V. and Gladman, D. (2018) Functional impairment measurement in psoriatic arthritis: Importance and challenges. *Seminars in Arthritis and Rheumatism*, 48 (3): 436–448. doi:10.1016/j.semarthrit.2018.05.010.

Meena, K., Kumar, M. and Jangra, M. (2020) “Controlling Mouse Motions Using Eye Tracking Using Computer Vision.” In *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. May 2020. pp. 1001–1005. doi:10.1109/ICICCS48265.2020.9121137.

Meena, Y.K., Cecotti, H., Wong-Lin, K., et al. (2017) “A multimodal interface to resolve the Midas-Touch problem in gaze controlled wheelchair.” In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*. September 2017. Institute of Electrical and Electronics Engineers Inc. pp. 905–908. doi:10.1109/EMBC.2017.8036971.

Meng, J., Zhang, J. and Zhao, H. (2012) “Overview of the Speech Recognition Technology.” In *2012 Fourth International Conference on Computational and Information Sciences*. August 2012. pp. 199–202. doi:10.1109/ICCIS.2012.202.

Microsoft (2023) *Speech to Text API | Microsoft Azure*. Available at: https://azure.microsoft.com/en-gb/services/cognitive-services/speech-to-text/.

Microsoft Teams (2023) *Microsoft Teams | Group Chat, Team Chat & Collaboration*. Available at: https://www.microsoft.com/en-GB/microsoft-teams/group-chat-software.

Mihara, Y., Shibayama, E. and Takahashi, S. (2005) “The migratory cursor: accurate speech-based cursor movement by moving multiple ghost cursors using non-verbal vocalizations.” In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*. Assets ’05 New York, NY, USA, 9 October 2005. Association for Computing Machinery. pp. 76–83. doi:10.1145/1090785.1090801.

Miller, R.G., Mitchell, J.D. and Moore, D.H. (2012) Riluzole for amyotrophic lateral sclerosis (ALS)/motor neuron disease (MND). *The Cochrane Database of Systematic Reviews*, (3): CD001447. doi:10.1002/14651858.CD001447.pub3.

Minelli, R., Mocci, A. and Lanza, M. (2015) “I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time.” In *IEEE International Conference on Program Comprehension*. August 2015. IEEE Computer Society. pp. 25–35. doi:10.1109/ICPC.2015.12.

Miniotas, D., Špakov, O., Tugoy, I., et al. (2006) “Speech-augmented eye gaze interaction with small closely spaced targets.” In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*. ETRA ’06 New York, NY, USA, 2006. ACM. pp. 67–67. doi:10.1145/1117309.1117345.

Modak, S., Vikmani, S., Shah, S., et al. (2016) “Voice driven dynamic generation of webpages.” In *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*. August 2016. IEEE. pp. 1–4. doi:10.1109/ICCUBEA.2016.7860153.

Møllenbach, E., Lillholm, M., Gail, A., et al. (2010) “Single gaze gestures.” In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications - ETRA ’10*. Austin, Texas, 2010. ACM Press. p. 177. doi:10.1145/1743666.1743710.

Monaco Editor (2023) *Monaco Editor*. Available at: https://microsoft.github.io/monaco-editor/ (Accessed: 26 March 2022).

Mondal, S., Das, P.P. and Bhattacharjee Rudra, T. (2022) Measuring code comprehension effort using code reading pattern. *Sādhanā*, 47 (3): 117. doi:10.1007/s12046-022-01876-5.

Morimoto, C.H. and Mimica, M.R.M. (2005) Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98 (1): 4–24. doi:10.1016/j.cviu.2004.07.010.

Morrison, K. and McKenna, S.J. (2002) Automatic visual recognition of gestures made by motor-impaired computer users. *Technology and Disability*, 14 (4): 197–203. doi:10.3233/TAD-2002-14408.

Mott, M.E., Vatavu, R.-D., Kane, S.K., et al. (2016) “Smart Touch: Improving Touch Accuracy for People with Motor Impairments with Template Matching.” In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI ’16 New York, NY, USA, 7 May 2016. Association for Computing Machinery. pp. 1934–1946. doi:10.1145/2858036.2858390.

Mott, M.E., Williams, S., Wobbrock, J.O., et al. (2017) “Improving dwell-based gaze typing with dynamic, cascading dwell times.” In *Conference on Human Factors in Computing Systems - Proceedings*. New York, New York, USA, 2017. ACM Press. pp. 2558–2570. doi:10.1145/3025453.3025517.

Mozilla Speech (2023) *DeepSpeech 0.6: Mozilla’s Speech-to-Text Engine Gets Fast, Lean, and Ubiquitous – Mozilla Hacks - the Web developer blog*. Available at: https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine (Accessed: 11 April 2021).

Müller, J.A., Wendt, D., Kollmeier, B., et al. (2016) Comparing Eye Tracking with Electrooculography for Measuring Individual Sentence Comprehension Duration. *PLoS ONE*, 11 (10): e0164627. doi:10.1371/journal.pone.0164627.

Müller, P., Buschek, D., Huang, M.X., et al. (2019) “Reducing calibration drift in mobile eye trackers by exploiting mobile phone usage.” In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. ETRA ’19 New York, NY, USA, 25 June 2019. Association for Computing Machinery. pp. 1–9. doi:10.1145/3314111.3319918.

Nacke, L.E., Stellmach, S., Sasse, D., et al. (2010) *Gameplay experience in a gaze interaction game*. Available at: http://arxiv.org/abs/1004.0259 (Accessed: 13 August 2022).

Naftali, M. and Findlater, L. (2014) "Accessibility in context: understanding the truly mobile experience of smartphone users with motor impairments." In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. ASSETS '14 New York, NY, USA, 20 October 2014. Association for Computing Machinery. pp. 209–216. doi:10.1145/2661334.2661372.

Nagendran, G.A., Singh, H., Raj, R.J.S., et al. (2021) "Input Assistive Keyboards for People with Disabilities: A Survey." In *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*. 2021. IEEE. pp. 829–832.

Necas, M. (1996) Musculoskeletal Symptomnatology and Repetitive Strain Injuries in Diagnostic Medical Sonographers: A Pilot Study in Washington and Oregon. *Journal of Diagnostic Medical Sonography*, 12 (6): 266–273. doi:10.1177/875647939601200604.

Nguyen, N. and Nadi, S. (2022) "An Empirical Evaluation of GitHub Copilot's Code Suggestions." In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. May 2022. pp. 1–5. doi:10.1145/3524842.3528470.

Nivala, M., Hauser, F., Mottok, J., et al. (2016) "Developing visual expertise in software engineering: An eye tracking study." In *2016 IEEE Global Engineering Education Conference (EDUCON)*. 2016. pp. 613–620. doi:10.1109/EDUCON.2016.7474614.

Norris, G. and Wilson, E. (1997) "The eye mouse, an eye communication device." In *Proceedings of the IEEE 23rd Northeast Bioengineering Conference*. 1997. IEEE. pp. 66–67.

Nowogdorzki, A. (2018) WRITING CODE OUT LOUD Programmers turn to voice-command tools to give their hands a rest. *Nature*, 559 (5 July 2018): 141–142.

Nowrin, S., OrdóñEz, P. and Vertanen, K. (2022) "Exploring Motor-impaired Programmers' Use of Speech Recognition." In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '22 New York, NY, USA, 22 October 2022. Association for Computing Machinery. pp. 1–4. doi:10.1145/3517428.3550392.

Ntoa, S., Margetis, G., Antona, M., et al. (2014) *Scanning-Based Interaction Techniques for Motor Impaired Users*. doi:10.4018/978-1-4666-4438-0.ch003.

Obaido, G., Ade-Ibijola, A. and Vadapalli, H. (2020) "TalkSQL: A Tool for the Synthesis of SQL Queries from Verbal Specifications." In *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. November 2020. pp. 1–10. doi:10.1109/IMITEC50163.2020.9334088.

Ocariza, F., Bajaj, K., Pattabiraman, K., et al. (2013) “An empirical study of client-side JavaScript bugs.” In *International Symposium on Empirical Software Engineering and Measurement*. 2013. pp. 55–64. doi:10.1109/ESEM.2013.18.

Oh, J., Vidal-Jordana, A. and Montalban, X. (2018) Multiple sclerosis: clinical aspects. *Current Opinion in Neurology*, 31 (6): 752. doi:10.1097/WCO.0000000000000622.

Ohno, T., Mukawa, N. and Kawato, S. (2003) “Just blink your eyes: a head-free gaze tracking system.” In *CHI ’03 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’03 New York, NY, USA, 5 April 2003. Association for Computing Machinery. pp. 950–957. doi:10.1145/765891.766088.

Okafor, O. and Ludi, S. (2022) “Helping Students with Motor Impairments Program via Voice-Enabled Block-Based Programming.” In Antona, M. and Stephanidis, C. (eds.). *Universal Access in Human-Computer Interaction. User and Context Diversity*. Lecture Notes in Computer Science Cham, 2022. Springer International Publishing. pp. 62–77. doi:10.1007/978-3-031-05039-8_5.

Oliver, M. and Barnes, C. (2012) Back to the future: the World Report on Disability. *Disability & Society*, 27 (4): 575–579. doi:10.1080/09687599.2012.686781.

Ossmann, R., Thaller, D., Nussbaum, G., et al. (2012) AsTeRICS, a Flexible Assistive Technology Construction Set. *Procedia Computer Science*, 14: 1–9. doi:10.1016/j.procs.2012.10.001.

Oviatt, S. (1997) Mulitmodal Interactive Maps: Designing for Human Performance. *Human-Computer Interaction*, 12 (1): 93–129. doi:10.1207/s15327051hci1201&2_4.

Oviatt, S. (2017) “Theoretical foundations of multimodal interfaces and systems.” In *The Handbook of Multimodal-Multisensor Interfaces: Foundations, User Modeling, and Common Modality Combinations - Volume 1*. Association for Computing Machinery and Morgan & Claypool. pp. 19–50. Available at: https://doi.org/10.1145/3015783.3015786 (Downloaded: 7 January 2024).

Oviatt, S., Cohen, P., Wu, L., et al. (2000) Designing the User Interface for Multimodal Speech and Pen-Based Gesture Applications: State-of-the-Art Systems and Future Research Directions. *Human–Computer Interaction*, 15 (4): 263–322. doi:10.1207/S15327051HCI1504_1.

Oviatt, S. and Cohen, P.R. (2022) *The Paradigm Shift to Multimodality in Contemporary Computer Interfaces*. Springer Nature. (Google-Books-ID: ZYlyEAAAQBAJ).

Oviatt, S., Coulston, R., Tomko, S., et al. (2003) “Toward a theory of organized multimodal integration patterns during human-computer interaction.” In *Proceedings of the 5th international conference on Multimodal interfaces*. ICMI ’03 New York, NY, USA, 5 November 2003. Association for Computing Machinery. pp. 44–51. doi:10.1145/958432.958443.

Oviatt, S. and VanGent, R. (1996) “Error resolution during multimodal human-computer interaction.” In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP ’96*. October 1996. pp. 204–207 vol.1. doi:10.1109/ICSLP.1996.607077.

Päivi, M. (2011) *Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies: Advances in Assistive Technologies*. IGI Global. (Google-Books-ID: HuWeBQAAQBAJ).

Papoutsaki, A., Laskey, J. and Huang, J. (2017) “Searchgazer: Webcam eye tracking for remote studies of web search.” In *Proceedings of the 2017 conference on conference human information interaction and retrieval*. 2017. pp. 17–26.

Park, K.S. and Lee, K.T. (1996) Eye-controlled human/computer interface using the line-of-sight and the intentional blink. *Computers & Industrial Engineering*, 30 (3): 463–473. doi:10.1016/0360-8352(96)00018-6.

Park, T.H., Dorn, B. and Forte, A. (2015) An Analysis of HTML and CSS Syntax Errors in a Web Development Course. *ACM Transactions on Computing Education*, 15 (1): 1–21. doi:10.1145/2700514.

Patel, R. and Patel, M. (2014) *Hands free JAVA (Through Speech Recognition)*. Available at: www.ijcsit.com.

Pearl, C. (2016) *Designing Voice User Interfaces: Principles of Conversational Experiences*. O’Reilly Media, Inc. (Google-Books-ID: MmnEDQAAQBAJ).

Peixoto, N., Nik, H.G. and Charkhkar, H. (2013) Voice controlled wheelchairs: Fine control by humming. *Computer Methods and Programs in Biomedicine*, 112 (1): 156–165. doi:10.1016/j.cmpb.2013.06.009.

Penguin (2021) Penguin Ambidextrous Vertical. *Ergomax*. Available at: https://ergomax.ca/en/Products/penguin-ambidextrous-vertical-2/ (Accessed: 15 November 2022).

Penkar, A. (2014) *Hypertext Navigation with an Eye Gaze Tracker*. PhD Thesis, ResearchSpace@ Auckland.

Petrie, H., Carmien, S. and Lewis, A. (2018) “Assistive Technology Abandonment: Research Realities and Potentials.” In Miesenberger, K. and Kouroupetroglou, G. (eds.). *Computers Helping People with Special Needs*. Lecture Notes in Computer Science Cham, 2018. Springer International Publishing. pp. 532–540. doi:10.1007/978-3-319-94274-2_77.

Pfeuffer, K. and Gellersen, H. (2016) “Gaze and touch interaction on tablets.” In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 2016. pp. 301–311.

Pi, J., Koljonen, P.A., Hu, Y., et al. (2020) Dynamic Bayesian Adjustment of Dwell Time for Faster Eye Typing. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28 (10): 2315–2324. doi:10.1109/TNSRE.2020.3016747.

Pieper, M., Morasch, H. and Piéla, G. (2003) Bridging the educational divide. *Universal Access in the Information Society*, 2 (3): 243–254. doi:10.1007/s10209-003-0061-y.

Piorkowski, D.J., Fleming, S.D., Kwan, I., et al. (2013) "The whats and hows of programmers' foraging diets." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13 New York, NY, USA, 27 April 2013. Association for Computing Machinery. pp. 3063–3072. doi:10.1145/2470654.2466418.

Pireddu, A. (2007) *Multimodal Interaction: an integrated speech and gaze approach*. Available at: https://www.researchgate.net/publication/239542563.

Poole, A. and Ball, L. (2004) Eye Tracking in Human-Computer Interaction and Usability Research : Current Status and Future Prospects. *undefined*. Available at: https://www.semanticscholar.org/paper/Eye-Tracking-in-Human-Computer-Interaction-and-%3A-Poole-Ball/92bc546258e9b6560cea225ca9f6745fa636ae6a (Accessed: 16 July 2022).

Pope, D. and Bambra, C. (2005) Has the disability discrimination act closed the employment gap? *Disability and Rehabilitation*, 27 (20): 1261–1266. doi:10.1080/09638280500075626.

Porcheron, M., Fischer, J.E., Reeves, S., et al. (2018) "Voice Interfaces in Everyday Life." In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Montreal QC Canada, 21 April 2018. ACM. pp. 1–12. doi:10.1145/3173574.3174214.

Porta, M. and Ravelli, A. (2009) "WeyeB, an eye-controlled web browser for hands-free navigation." In *Proceedings - 2009 2nd Conference on Human System Interactions, HSI '09*. 2009. pp. 210–215. doi:10.1109/HSI.2009.5090980.

Pradhan, A., Mehta, K. and Findlater, L. (2018) "Accessibility Came by Accident." In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18 New York, NY, USA, 2018. ACM. pp. 1–13. doi:10.1145/3173574.3174033.

Radevski, S., Hata, H. and Matsumoto, K. (2016) "EyeNav." In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction - NordiCHI '16*. NordiCHI '16 New York, NY, USA, 2016. ACM. pp. 1–4. doi:10.1145/2971485.2996724.

Raghavan, P. (2015) Upper Limb Motor Impairment Post Stroke. *Physical medicine and rehabilitation clinics of North America*, 26 (4): 599–610. doi:10.1016/j.pmr.2015.06.008.

Raja, D.S. (2016) Bridging the disability divide through digital technologies. *Background paper for the World Development report*.

Rajanna, V. and Hammond, T. (2018) *A Gaze-Assisted Multimodal Approach to Rich and Accessible Human-Computer Interaction*. doi:10.48550/arXiv.1803.04713.

Rajanna, V., Russel, M., Zhao, J., et al. (2022) PressTapFlick: Exploring a gaze and foot-based multimodal approach to gaze typing. *International Journal of Human-Computer Studies*, 161: 102787.

Ramadan, Z., F. Farah, M. and El Essrawi, L. (2021) From Amazon.com to Amazon.love: How Alexa is redefining companionship and interdependence for people with special needs. *Psychology & Marketing*, 38 (4): 596–609. doi:10.1002/mar.21441.

Rayner, K. (1998) Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124 (3): 372.

Rayner, M., Chatzichrisafis, N., Bouillon, P., et al. (2005) “Japanese speech understanding using grammar specialization.” In *Proceedings of HLT/EMNLP 2005 Interactive Demonstrations*. 2005. pp. 26–27.

Reddy, D.R. (1966) Approach to Computer Speech Recognition by Direct Analysis of the Speech Wave. *The Journal of the Acoustical Society of America*, 40 (5): 1273–1273. doi:10.1121/1.2143468.

Reichle, J. (2011) Evaluating Assistive Technology in the Education of Persons with Severe Disabilities. *Journal of Behavioral Education*, 20 (1): 77–85. doi:10.1007/s10864-011-9121-1.

Rochon, J., Gondan, M. and Kieser, M. (2012) To test or not to test: Preliminary assessment of normality when comparing two independent samples. *BMC Medical Research Methodology*, 12 (1): 81. doi:10.1186/1471-2288-12-81.

Rodrigues, A.S., da Costa, V.K., Cardoso, R.C., et al. (2017) “Evaluation of a Head-Tracking Pointing Device for Users with Motor Disabilities.” In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA ’17 New York, NY, USA, 21 June 2017. Association for Computing Machinery. pp. 156–162. doi:10.1145/3056540.3056552.

Rodriguez-Cartagena, J.K., Claudio-Palacios, A.C., Pacheco-Tallaj, N., et al. (2015) “The Implementation of a Vocabulary and Grammar for an Open-Source Speech-Recognition Programming Platform.” In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. ASSETS ’15 New York, NY, USA, 26 October 2015. Association for Computing Machinery. pp. 447–448. doi:10.1145/2700648.2811346.

Rosen, K. and Yampolsky, S. (2000) Automatic speech recognition and a review of its functioning with dysarthric speech. *Augmentative and Alternative Communication*, 16 (1): 48–60. doi:10.1080/07434610012331278904.

Rosenblatt, L. (2017) “VocalIDE.” In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS ’17*. ASSETS ’17 New York, NY, USA, 2017. ACM. pp. 417–418. doi:10.1145/3132525.3134824.

Rosenblatt, L., Carrington, P., Hara, K., et al. (2018) “Vocal Programming for People with Upper-Body Motor Impairments.” In *W4A*. 2018. pp. 10–10. doi:10.1145/3192714.3192821.

Rozado, D., McNeill, A. and Mazur, D. (2016) VoxVisio–Combining Gaze and Speech for Accessible HCI. *Resna 2016*. Available at: https://www.resna.org/sites/default/files/conference/2016/pdf_versions/cac/rozado.pdf.

Ruan, S., Wobbrock, J.O., Liou, K., et al. (2016) *Speech Is 3x Faster than Typing for English and Mandarin Text Entry on Mobile Devices*.

Rubio, S., Díaz, E., Martín, J., et al. (2004) Evaluation of Subjective Mental Workload: A Comparison of SWAT, NASA-TLX, and Workload Profile Methods. *Applied Psychology*, 53 (1): 61–86. doi:10.1111/j.1464-0597.2004.00161.x.

Rudd, T. (2013) *PyVideo.org · Using Python to Code by Voice*. Available at: https://pyvideo.org/pycon-us-2013/using-python-to-code-by-voice.html.

Rudžionis, V., Ratkevičius, K., Rudžionis, A., et al. (2013) “Recognition of Voice Commands Using Hybrid Approach.” In Skersys, T., Butleris, R. and Butkiene, R. (eds.). *Information and Software Technologies*. Communications in Computer and Information Science Berlin, Heidelberg, 2013. Springer. pp. 249–260. doi:10.1007/978-3-642-41947-8_21.

Ruiz, N., Chen, F., Oviatt, S., et al. (2009) Multimodal input. *Multimodal Signal Processing: Theory and applications for human-computer interaction*, pp. 231–255.

Sahadat, M.N., Sebkhi, N., Kong, F., et al. (2018) “Standalone Assistive System to Employ Multiple Remaining Abilities in People with Tetraplegia.” In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. October 2018. pp. 1–4. doi:10.1109/BIOCAS.2018.8584688.

Santella, A., Agrawala, M., DeCarlo, D., et al. (2006) “Gaze-based interaction for semi-automatic photo cropping.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’06 New York, NY, USA, 22 April 2006. Association for Computing Machinery. pp. 771–780. doi:10.1145/1124772.1124886.

Santos, A.L. (2021) “Javardeye: Gaze Input for Cursor Control in a Structured Editor.” In *Companion Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming*. Programming ’21 New York, NY, USA, 22 March 2021. Association for Computing Machinery. pp. 31–35. doi:10.1145/3464432.3464435.

Sarcar, S., Panwar, P. and Chakraborty, T. (2013) “EyeK: an efficient dwell-free eye gaze-based text entry system.” In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction*. APCHI ’13 New York, NY, USA, 24 September 2013. Association for Computing Machinery. pp. 215–220. doi:10.1145/2525194.2525288.

Sarmah, R.J., Ding, Y., Wang, D., et al. (2020) “Geno: A developer tool for authoring multimodal interaction on existing web applications.” In *UIST 2020 - Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA, 2020. ACM. pp. 1169–1181. doi:10.1145/3379337.3415848.

Schiessl, M., Duda, S., Thölke, A., et al. (2003) Eye tracking and its application in usability and media research. *MMI-interaktiv Journal*, 6 (2003): 41–50.

Schmandt, C., Ackerman, M.S. and Hindus, D. (1990) Augmenting a window system with speech input. *Computer*, 23 (8): 50–56. doi:10.1109/2.56871.

Sears, A., Feng, J., Cseitutu, K., et al. (2003) Hands-free, speech-based navigation during dictation: Difficulties, consequences, and solutions. *Human-Computer Interaction*, 18 (3): 229–257. doi:10.1207/S15327051HCI1803_2.

Sears, A., Lin, M. and Karimullah, A.S. (2002) Speech-based cursor control: understanding the effects of target size, cursor speed, and command selection. *Universal Access in the Information Society*, 2 (1): 30–43. doi:10.1007/s10209-002-0034-6.

Seizov, O. and Wildfeuer, J. (2017) *New Studies in Multimodality: Conceptual and Methodological Elaborations*. Bloomsbury Publishing. (Google-Books-ID: J5smDwAAQBAJ).

Sengupta, K., Bhattarai, S., Sarcar, S., et al. (2020) “Leveraging Error Correction in Voice-based Text Entry by Talk-and-Gaze.” In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, (25/04/20 - 30/04/20)*. January 2020. pp. 1–11. doi:10.1145/3313831.3376579.

Sengupta, K., Ke, M., Menges, R., et al. (2018) “Hands-free web browsing.” In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications - ETRA ’18*. ETRA ’18 New York, NY, USA, 2018. ACM. pp. 1–3. doi:10.1145/3204493.3208338.

Serenade (2023) *Serenade | Documentation*. Available at: https://serenade.ai/docs/.

Shaffer, T.R., Wise, J.L., Walters, B.M., et al. (2015) “iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks.” In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015 New York, NY, USA, 2015. ACM. pp. 954–957. doi:10.1145/2786805.2803188.

Shaik, S., Corvin, R., Sudarsan, R., et al. (2003) “SpeechClipse: an Eclipse speech plug-in.” In *Proc. ETX at OOPSLA*. eclipse ’03 New York, NY, USA, 2003. ACM. pp. 84–88. doi:10.1145/965660.965678.

Shakil, A., Lutteroth, C. and Weber, G. (2019) “CodeGazer: Making Code Navigation Easy and Natural With Gaze Input.” In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19 New York, NY, USA, 2 May 2019. Association for Computing Machinery. pp. 1–12. doi:10.1145/3290605.3300306.

Sharafi, Z., Sharif, B., Guéhéneuc, Y.-G., et al. (2020) A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25 (5): 3128–3174. doi:10.1007/s10664-020-09829-4.

Sharafi, Z., Soh, Z. and Guéhéneuc, Y.-G. (2015) A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, 67: 79–107. doi:10.1016/j.infsof.2015.06.008.

Shaw, G. and Coles, T. (2004) Disability, holiday making and the tourism industry in the UK: a preliminary survey. *Tourism Management*, 25 (3): 397–403. doi:10.1016/S0261-5177(03)00139-0.

Sheehan, R. and Hassiotis, A. (2017) Digital mental health and intellectual disabilities: state of the evidence and future directions. *Evidence Based Mental Health*, 20 (4): 107–111. doi:10.1136/eb-2017-102759.

Sibert, L.E., Templeman, J.N. and Jacob, R.J. (2001) *Evaluation and Analysis of Eye Gaze Interaction:* Fort Belvoir, VA: Defense Technical Information Center. doi:10.21236/ADA389984.

Sims, D. (2022) *Two-fifths of adults own and use a voice-activated personal assistant or smart speaker device, finds survey*. Available at: https://www.djsresearch.co.uk/InformationTechnologyMarketResearchInsightsAndFindings/article/Two-fifths-of-adults-own-and-use-a-voice-activated-personal-assistant-or-smart-speaker-device-finds-survey-05114 (Accessed: 13 August 2022).

Sinha, M. and Dasgupta, T. (2021) A web browsing interface for people with severe speech and motor impairment. *Journal of Enabling Technologies*, 15 (3): 189–207. doi:10.1108/JET-07-2020-0029.

Siri (2023) *Siri*. Available at: https://www.apple.com/uk/siri/ (Accessed: 24 March 2023).

Skovsgaard, H., Mateo, J.C., Flach, J.M., et al. (2010) “Small-target selection with gaze alone.” In *Eye Tracking Research and Applications Symposium (ETRA)*. January 2010. pp. 145–148. doi:10.1145/1743666.1743702.

Slack (2022) *Where work happens*. Available at: https://slack.com/intl/en-gb/ (Accessed: 7 May 2022).

Smith, J., Brown, C. and Murphy-Hill, E. (2017) “Flower: Navigating program flow in the IDE.” In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. October 2017. pp. 19–23. doi:10.1109/VLHCC.2017.8103445.

Smith, M.J., Karsh, B.-T., Conway, F.T., et al. (1998) Effects of a Split Keyboard Design and Wrist Rest on Performance, Posture, and Comfort. *Human Factors*, 40 (2): 324–336. doi:10.1518/001872098779480451.

Snell, L. (2000) *AN INVESTIGATION INTO PROGRAMMING BY VOICE*. Available at: http://www3.imperial.ac.uk/pls/portallive/docs/1/18619779.PDF.

Song, Y., Wong, R.C.-W., Zhao, X., et al. (2022) "VoiceQuerySystem: A Voice-driven Database Querying System Using Natural Language Questions." In *Proceedings of the 2022 International Conference on Management of Data*. SIGMOD '22 New York, NY, USA, 11 June 2022. Association for Computing Machinery. pp. 2385–2388. doi:10.1145/3514221.3520158.

Soto Munoz, J.G., De Casso Verdugo, A.I., Geraldo Gonzalez, E., et al. (2019a) "Programming by Voice Assistance Tool for Physical Impairment Patients Classified in to Peripheral Neuropathy Centered on Arms or Hands Movement Difficulty." In *Proceedings - 2019 International Conference on Inclusive Technologies and Education, CONTIE 2019*. 1 October 2019. Institute of Electrical and Electronics Engineers Inc. pp. 210–217. doi:10.1109/CONTIE49246.2019.00048.

Soto Munoz, J.G., De Casso Verdugo, A.I., Geraldo Gonzalez, E., et al. (2019b) "Programming by Voice Assistance Tool for Physical Impairment Patients Classified in to Peripheral Neuropathy Centered on Arms or Hands Movement Difficulty." In *Proceedings - 2019 International Conference on Inclusive Technologies and Education, CONTIE 2019*. October 2019. Institute of Electrical and Electronics Engineers Inc. pp. 210–217. doi:10.1109/CONTIE49246.2019.00048.

Špakov, O. and Miniotas, D. (2004) "On-line adjustment of dwell time for target selection by gaze." In *Proceedings of the third Nordic conference on Human-computer interaction*. 2004. pp. 203–206.

Sporka, A.J., Kurniawan, S.H., Mahmud, M., et al. (2006) "Non-speech input and speech recognition for real-time control of computer games." In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*. Assets '06 New York, NY, USA, 23 October 2006. Association for Computing Machinery. pp. 213–220. doi:10.1145/1168987.1169023.

Srinivasan, A., Dontcheva, M., Adar, E., et al. (2019) "Discovering natural language commands in multimodal interfaces." In *International Conference on Intelligent User Interfaces, Proceedings IUI*. New York, NY, USA, 17 March 2019. Association for Computing Machinery. pp. 661–672. doi:10.1145/3301275.3302292.

Stack Overflow (2023) *Stack Overflow*. Available at: https://stackoverflow.com/ (Accessed: 4 January 2023).

StackOverflow (2022) *Stack Overflow Developer Survey 2022*. Available at: https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022 (Accessed: 9 October 2022).

Stahl, C. and Laub, P. (2017) “Maintaining multiple sclerosis patients’ quality of life: a case study on environment control assistance in a smart home.” In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA ’17 New York, NY, USA, 21 June 2017. Association for Computing Machinery. pp. 83–86. doi:10.1145/3056540.3064943.

Standen, P., Brown, D., Roscoe, J., et al. (2014) “Engaging Students with Profound and Multiple Disabilities Using Humanoid Robots.” In Stephanidis, C. and Antona, M. (eds.) *Universal Access in Human-Computer Interaction. Universal Access to Information and Knowledge*. Lecture Notes in Computer Science. Cham: Springer International Publishing. pp. 419–430. doi:10.1007/978-3-319-07440-5_39.

Stumpf, S., Peters, A., Bardzell, S., et al. (2020) Gender-Inclusive HCI Research and Design: A Conceptual Review. *Foundations and Trends in Human–Computer Interaction*, 13 (1): 1–69. doi:10.1561/1100000056.

Suhm, B., Myers, B. and Waibel, A. (2001) Multimodal error correction for speech user interfaces. *ACM Transactions on Computer-Human Interaction*, 8 (1): 60–98. doi:10.1145/371127.371166.

Sweller, J., Chandler, P., Tierney, P., et al. (1990) Cognitive load as a factor in the structuring of technical material. *Journal of Experimental Psychology: General*, 119 (2): 176–192. doi:10.1037/0096-3445.119.2.176.

Tai, K., Blain, S. and Chau, T. (2008) A Review of Emerging Access Technologies for Individuals With Severe Motor Impairments. *Assistive Technology*, 20 (4): 204–221. doi:10.1080/10400435.2008.10131947.

Talon (2023) *Talon 0.0.7.7 documentation*. Available at: https://talonvoice.com/docs/index.html#document-index.

Tiric-Campara, M., Krupic, F., Biscevic, M., et al. (2014) Occupational Overuse Syndrome (Technological Diseases): Carpal Tunnel Syndrome, a Mouse Shoulder, Cervical Pain Syndrome. *Acta Informatica Medica*, 22 (5): 333–340. doi:10.5455/aim.2014.22.333-340.

Tobii (2023) *Tobii Gaming | Eye Tracker 4C for PC Gaming. Buy Now at €169.* Available at: https://gaming.tobii.com/tobii-eye-tracker-4c/.

Tobii 5 (2023) *Tobii Eye Tracker 5 | The Next Generation of Head Tracking and Eye Tracking*. Available at: https://gaming.tobii.com/product/eye-tracker-5/ (Accessed: 26 March 2023).

Trewin, S., Swart, C. and Pettick, D. (2013) “Physical accessibility of touchscreen smartphones.” In *Proceedings of the 15th International ACM SIGACCESS Conference on*

Computers and Accessibility. ASSETS '13 New York, NY, USA, 21 October 2013. Association for Computing Machinery. pp. 1–8. doi:10.1145/2513383.2513446.

Trivedi, A., Pant, N., Shah, P., et al. (2018) Speech to text and text to speech recognition systems-Areview. *IOSR J. Comput. Eng*, 20 (2): 36–43.

Trujillo, J.P. and Holler, J. (2023) Interactionally Embedded Gestalt Principles of Multimodal Human Communication. *Perspectives on Psychological Science*, 18 (5): 1136–1159. doi:10.1177/17456916221141422.

Uludaǧli, M. and Acartürk, C. (2018) User interaction in hands-free gaming: a comparative study of gaze-voice and touchscreen interface control. *Turkish Journal of Electrical Engineering and Computer Sciences*, 26 (4): 1967–1976. doi:10.3906/elk-1710-128.

Valencia, X., Pérez, J.E., Arrue, M., et al. (2017) Adapting the Web for People With Upper Body Motor Impairments Using Touch Screen Tablets. *Interacting with Computers*, 29 (6): 794–812. doi:10.1093/iwc/iwx013.

Vallabhaneni, A., Wang, T. and He, B. (2005) "Brain—Computer Interface." In He, B. (ed.) *Neural Engineering*. Bioelectric Engineering. Boston, MA: Springer US. pp. 85–121. doi:10.1007/0-306-48610-5_3.

Van Brummelen, J., Weng, K., Lin, P., et al. (2020b) "CONVO: What does conversational programming need?" In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*. July 2020. Institute of Electrical and Electronics Engineers (IEEE). pp. 1–5. doi:10.1109/VL/HCC50065.2020.9127277.

Van Brummelen, J., Weng, K., Lin, P., et al. (2020c) *Convo: What does conversational programming need? An exploration of machine learning interface design*. Available at: http://arxiv.org/abs/2003.01318.

Van Der Kamp, J. and Sundstedt, V. (2011) "Gaze and voice controlled drawing." In *ACM International Conference Proceeding Series*. 2011. doi:10.1145/1983302.1983311.

Vickers, S. (2011) *Eye-gaze interaction techniques for use in online games and environments for users with severe physical disabilities.*

Visual Studio Code (2023). Available at: https://code.visualstudio.com/ (Accessed: 6 December 2022).

Voicecode (2023) *Advanced Voice-Control, speech to code, program by voice, stop RSI*. Available at: https://www.voicecode.io/ (Accessed: 26 May 2021).

Vornholt, K., Villotti, P., Muschalla, B., et al. (2018) Disability and employment – overview and highlights. *European Journal of Work and Organizational Psychology*, 27 (1): 40–55. doi:10.1080/1359432X.2017.1387536.

Wagner, A. and Gray, J. (2015) An Empirical Evaluation of a Vocal User Interface for Programming by Voice. *International Journal of Information Technologies and Systems Approach*, 8 (2): 47–63. doi:10.4018/IJITSA.2015070104.

Wagner, A., Rudraraju, R., Datla, S., et al. (2012) “Programming by voice.” In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts - CHI EA ’12*. CHI EA ’12 New York, NY, USA, 2012. ACM. pp. 2087–2087. doi:10.1145/2212776.2223757.

Wang, J. (1995) “Integration of eye-gaze, voice and manual response in multimodal user interface.” In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. 1995. IEEE. pp. 3938–3942. doi:10.1109/icsmc.1995.538404.

Wang, J., Zhai, S. and Su, H. (2001) *Chinese input with keyboard and eye-tracking: an anatomical study*. doi:10.1145/365024.365298.

Ware, C. and Mikaelian, H.H. (1986) “An evaluation of an eye tracker as a device for computer input2.” In *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*. 1986. pp. 183–188.

Wav2letter (2023) *Wav2letter*. Available at: https://ai.facebook.com/tools/wav2letter (Accessed: 5 December 2022).

WebSpeech (2023) *Using the Web Speech API - Web APIs | MDN*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API.

Wei, J., Tag, B., Trippas, J.R., et al. (2022) “What Could Possibly Go Wrong When Interacting with Proactive Smart Speakers? A Case Study Using an ESM Application.” In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI ’22 New York, NY, USA, 29 April 2022. Association for Computing Machinery. pp. 1–15. doi:10.1145/3491102.3517432.

Whitehead, M., Clayton, S., Holland, P., et al. (2009) *Helping chronically ill or disabled people into work: what can we learn from international comparative analyses?* Available at: http://phrc.lshtm.ac.uk/project_2005-2011_c206.html (Accessed: 17 August 2022).

Wilcox, M.J., Norman-Murch, T., Oberstein, J.S., et al. (1999) *Assistive Technology: Tips, Tools, and Techniques. A Parent Resource Manual.*

Wilcox, T., Evans, M., Pearce, C., et al. (2008) Gaze and voice based game interaction: the revenge of the killer penguins. *SIGGRAPH Posters*, 81 (10.1145): 1400885–1400972.

Wit (2023) *Wit.ai*.

Wobbrock, J.O., Rubinstein, J., Sawyer, M.W., et al. (2008) “Longitudinal evaluation of discrete consecutive gaze gestures for text entry.” In *Proceedings of the 2008 symposium* *on Eye tracking research & applications*. ETRA ’08 New York, NY, USA, 26 March 2008. Association for Computing Machinery. pp. 11–18. doi:10.1145/1344471.1344475.

Wong, G. (2020) The Role of Assistive Technology in Enhancing Disability Arts. *Review of Disability Studies: An International Journal*, 16 (1): 1–31.

World Health Organization. “How to use the ICF: A practical manual for using the International Classification of Functioning, Disability and Health (ICF).” Exposure draft for comment. Geneva: WHO 13 (2013). (2013).

Yassi, A. (1997) Repetitive strain injuries. *The Lancet*, 349 (9056): 943–947. doi:10.1016/S0140-6736(96)07221-2.

Yiu, E.M. and Kornberg, A.J. (2015) Duchenne muscular dystrophy. *Journal of Paediatrics and Child Health*, 51 (8): 759–764. doi:10.1111/jpc.12868.

Zaheer, A., Malik, A.N., Masood, T., et al. (2021) Effects of phantom exercises on pain, mobility, and quality of life among lower limb amputees; a randomized controlled trial. *BMC Neurology*, 21 (1): 416. doi:10.1186/s12883-021-02441-z.

Zapala, D. and Balaj, B. (2012) *Eye Tracking and Head Tracking–The two approaches in assistive technologies*.

Zargham, N., Pfau, J., Schnackenberg, T., et al. (2022) ““I Didn’t Catch That, But I’ll Try My Best”: Anticipatory Error Handling in a Voice Controlled Game.” In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI ’22 New York, NY, USA, 28 April 2022. Association for Computing Machinery. pp. 1–13. doi:10.1145/3491102.3502115.

Zhai, S., Morimoto, C. and Ihde, S. (1999) “Manual and gaze input cascaded (MAGIC) pointing.” In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. CHI ’99 New York, NY, USA, 1 May 1999. Association for Computing Machinery. pp. 246–253. doi:10.1145/302979.303053.

Zhang, X., Lu, C., Yin, J., et al. (2020) The Study of Two Novel Speech-Based Selection Techniques in Voice-User Interfaces. *IEEE Access*, 8: 217024–217032. doi:10.1109/ACCESS.2020.3041649.

Zhu, S., Ma, Y., Feng, J., et al. (2009) “Speech-Based Navigation: Improving Grid-Based Solutions.” In Gross, T., Gulliksen, J., Kotzé, P., et al. (eds.). *Human-Computer Interaction – INTERACT 2009*. Lecture Notes in Computer Science Berlin, Heidelberg, 2009. Springer. pp. 50–62. doi:10.1007/978-3-642-03655-2_6.

Ziefle, M. (2019) “Is the trackball a serious alternative to the mouse? A comparison of trackball and mouse with regard to cursor movement performance in manipulation tasks.” In *Human-Centered Computing*. CRC Press. pp. 158–162.

Zimmerer, C., Krop, P., Fischbach, M., et al. (2022) "Reducing the Cognitive Load of Playing a Digital Tabletop Game with a Multimodal Interface." In *CHI Conference on Human Factors in Computing Systems*. New Orleans LA USA, 29 April 2022. ACM. pp. 1–13. doi:10.1145/3491102.3502062.

Zoom (2023) *Video Conferencing, Web Conferencing, Webinars, Screen Sharing - Zoom*. Available at: https://zoom.us/.