Inclusive Multimodal Voice Interaction for Code Navigation

Bharat Paudyal

DMT Lab, Birmingham City University bharat.paudyal@bcu.ac.uk

Chris Creed

DMT Lab, Birmingham City University chris.creed@bcu.ac.uk

Ian Williams

DMT Lab, Birmingham City University ian.williams@bcu.ac.uk

Maite Frutos-Pascual

DMT Lab, Birmingham City University maite.frutos@bcu.ac.uk

Navigation of source code typically requires extensive use of a traditional mouse and keyboard which can present significant barriers for developers with physical impairments. We present research exploring how commonly used code navigation approaches (e.g. locating references to user-defined identifiers, jumping to function definitions, conducting a search for specific syntax, etc.) can be optimized for multimodal voice interaction. An exploratory study was initially conducted with five developers who have physical impairments to elicit insights around their experiences in navigating code within existing voice-controlled development environments. Findings from this study informed the design of a code editor integrating different navigation features tailored for multimodal speech input. A user evaluation with 14 developers with physical impairments was conducted with results demonstrating that all participants were able to successfully complete a series of standard navigation tasks. Participants also highlighted that the code navigation techniques were intuitive to use and provided a high-level of usability.

CCS CONCEPTS •Human-centered computing~Accessibility~Accessibility systems and tools

Additional Keywords and Phrases: Assistive technology, Programming tools, Speech recognition, accessibility

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 [1, 25, 36]. Research has also highlighted that developers spend a significant portion of their time on navigating code to support comprehension and maintenance [14, 22]. Moreover, studies have demonstrated that effective code navigation is crucial to support developers in building a cognitive model of a codebase that can facilitate understanding of code flow and the location of different code blocks [1, 23]. 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), canvas-based navigation (e.g. visualising all methods in a 2D plane) [8, 13, 16], and a structural navigation approach (e.g. displaying all usages of a method when clicked [36] or jumping to a function definition [34]). These navigation approaches are commonly integrated within mainstream development environments (e.g. Visual Studio Code [39]) where syntax is typically navigated via a mouse and keyboard. However, this can present significant barriers for people with physical impairments who may experience challenges in using traditional input devices to support coding activities. The use of voice control in conjunction with other input modalities (e.g. mechanical switches, eye gaze, etc.) presents an alternative interaction method for enabling disabled developers to write, navigate, and manipulate code [6, 29, 32], although there has been a lack of work around this area to date. Initial research has examined simple inline navigation approaches via speech (e.g. moving the cursor to a specific position on a line [29, 32]) and page scrolling through utilising a grid visualisation [42], 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 [17, 22, 33]. 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, we explored 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. We initially conducted interviews with five developers who have physical impairments to understand further 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 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, as well as demonstrating a high-level of usability.

2 RELATED WORK

2.1 Code Navigation

Code navigation has received significant attention from researchers in terms of both understanding current methods and how new approaches can potentially enhance developer workflows [9, 13, 16, 19, 24, 36]. Common approaches include utilizing a local search function [1, 2, 35] enabling developers to find code based on lexical matching, using a call graph [16, 24] where methods are represented as nodes that developers can utilize to navigate across different methods [16, 19], as well as features such as Flower [36] that enable developers to jump directly into the caller of any method (when selected by a developer). Many modern IDEs also provide a "*find all references*" feature that displays all calls associated with a particular method or function [2, 26], as well as a "*go to definition*" tool that allows developers to find the definition of a specified identifier [19, 34]. Studies have demonstrated that these approaches can present limitations as developers may have to switch between file views and then resize or reflow different file panes, thus increasing development time and increasing cognitive load in understanding the flow of code [9, 13, 16, 22]. Researchers have explored alternative approaches to address these issues such as canvas based code editors where all project code (i.e. multiple files) are collated in a single viewable 2D canvas [9, 13, 21]. Code Bubbles [9] allows developers to display code fragments in resizable bubbles which can be arranged around a 2D canvas. Patchworks [16] and Coderibbon [21] adopt a similar approach and were considered as a promising tools in overcoming the development time, however these methods are yet to be deployed in mainstream development environments.

2.2 Code Navigation via Voice Interaction

Whilst prior studies have highlighted the potential of voice as an interaction approach within a coding environment [10, 18, 27, 40, 41], less work has investigated the potential of speech input to support code navigation. Begel and Graham [5, 6] explored the use of voice control to navigate code utilizing approaches such as "*jump to constructor linked list*" (i.e. to move the cursor to a LinkedList constructor) and "*go to previous page*" (i.e. to navigate to the previous page). Rosenblatt et al [32] utilized vocal navigational commands such as "*go to line*" and "*up X lines*" to move between lines of code in their VocalIDE prototype. To support inline navigation, they used color context editing where individual characters within a line were highlighted with different colors (e.g. red, green, yellow, etc.) which could then be verbalised to navigate to that position (i.e. "green", "blue", etc.). A similar approach is also adopted in Cursorless [43] (a commercial development environment) which is integrated within TalonVoice [42] to provide additional inline navigational features such as moving the cursor to the start/end of a line or after specific characters. Paudyal et al. [29] used voice commands such as "*go to the line x*" and "*left x*" for navigating to different lines and re-positioning the cursor on a specific line, alongside a gaze-based onscreen keyboard where directional keys could also be used for manipulating the cursor position. TalonVoice is a speech based tool that has been widely used by disabled developers to support coding activities (including code navigation), although is yet to be formally evaluated in user studies. Similarly, Serenade [44] provides navigation features (e.g. inline navigation and navigating to methods, etc.) via natural language speech input, but has not been empirically evaluated to date.

While the existing literature can help to inform the development of voice-based code navigation techniques, there remain several areas where little or no work has been conducted. For instance, it remains unclear how common code navigation features (e.g. "*go to definition*, "*find all references*", etc.) can be tailored and adapted for voice interaction. Moreover, there has been a lack of work around evaluating speech interaction approaches that support code navigation in collaboration with developers who have physical impairments. Our work addresses these points through exploring whether voice-based navigation approaches are feasible and the extent to which they may be able to enhance accessibility within coding environments.

3 CODE NAVIGATION – AN EXPLORATORY STUDY

Interviews were initially conducted with five developers who have physical impairments to understand their current code navigation strategies. Participants were aged between 24 to 42 years (M=31.2, SD=6.3) - four participants were diagnosed with Repetitive Strain Injury (RSI) while one had Tendonitis/Carpal Tunnel. All participants were fluent in English and had at least a year of coding experience (M=10.2, SD=8.7) – they also all currently use voice as an interaction approach to perform coding activities. Interviews were conducted on Slack [38] (a team collaboration tool) and focused on exploring challenges participants experience in relation to navigating code, any strategies used to overcome barriers, and future features they felt could support them further. Responses across all participants were collated into three key themes:

Navigation Challenges: All participants highlighted that they use speech-based tools (e.g. Talon [42], Dragon Dictate [45]) for development work and emphasized 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 cursor to line x, *"/ word*" to search for the term "*word*", etc.) - *"…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 difficult to learn (e.g. "*slap*" will move the cursor to end of a line, "*command up*" will move the cursor to line 1, etc.). Whilst tools such as Talon support customization of commands, participants highlighted that this process can be time consuming and requires broader understanding of technical skills (such as configuring a microphone and coding environment, etc.) 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, etc.) 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, etc.) 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 use 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 optimized approaches that facilitate the use of 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 emphasized that any new approaches developed using solely voice as the primary method of interaction should be simple and intuitive "out-of-the-box", as opposed to requiring additional configuration (which can potentially be tedious and time consuming).

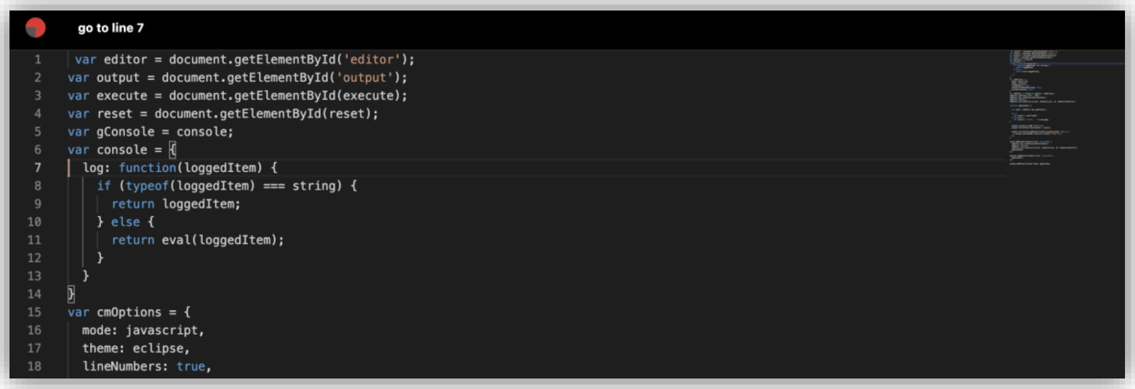
The findings from this initial study provided a solid starting foundation for the project and informed the focus of new multimodal voice code navigation approaches that support a wider range of features (such as navigating to definitions and listing references) that have not been explored in the existing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

4 SYSTEM DESIGN

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

4.1 Prototype Design

The prototype was built-upon an open-source JavaScript based editor (Monaco Editor [46]) that provides a range of standard coding features. The system architecture consists of three components: (1) a speech recognition interface to convert speech into text (using the Web Speech API [47]), (2) a command interpreter to parse a user's intent and command (using wit.ai [48] - an open source natural language platform) 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 studies [29, 32], 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 VSCode 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[49], Brackets[7], Visual Studio Code[39], etc.) and utilized a similar theme to the default one used in VS Code (Figure 1). The header area within the interface contains a microphone icon by default and provides feedback in relation to the speech commands that have been issued by the user. 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 recognized 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 recognizer failed to understand the command. The area below the header included the main editor where syntax can be written, navigated, and edited. A list of example commands is also available 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 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.

4.2 Code Navigation via Multimodal Voice Input

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*" , etc.), positioning the cursor position within a line of code (e.g. *"left x", "right x"*, etc.), 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 [16, 34, 36], 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 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 1.

Table 1: List of features along with example vocal commands used for navigation. ■ denotes that the cursor has to be placed around the identifier before verbalizing the command.

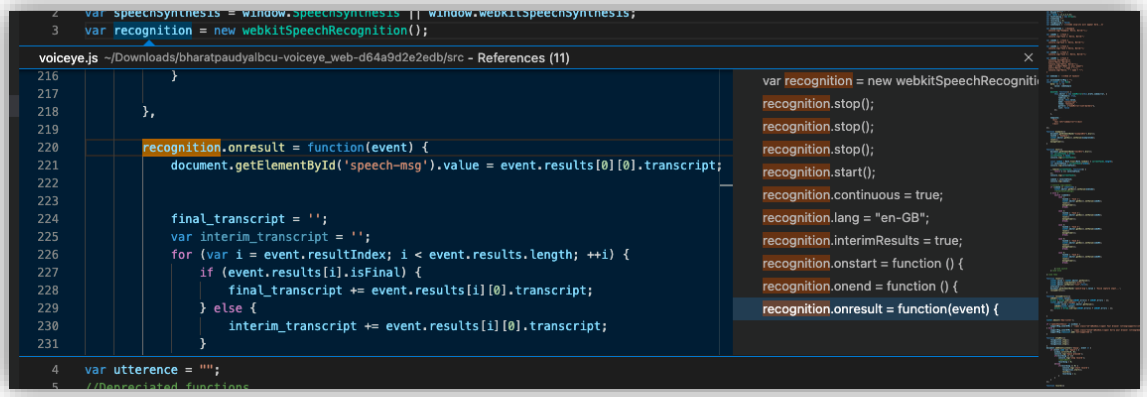
Features	*Usage*	*Example utterances*
Go to Definition	To navigate to the definition of Identifier (function/class/variable)	'define', 'go to definition', 'definition'
Find all references	To display all the possibilities of a selected Identifier	'reference', 'get reference', 'refer'
Search	To find/search a word using a caret	'find getarea', 'search function hello'
Select Next Item	To move forward to next item	'next', 'next 12', 'next 5th '
Select Previous Item	To go back to previous item	'previous', 'move previous', 'previous 2nd'
Left/Right/Up/Down	To navigate to desire position across the code caret. Defaulted to one position if 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 current line	'end of line', 'end line', 'line end'

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, etc.). Figure 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, etc.). They can then issue a command to activate the "*Find all References*" feature (e.g. "*reference*", "*find reference*", etc.) and a list of references are displayed on the right-side of the interface (Figure 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 verbalize 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();
```

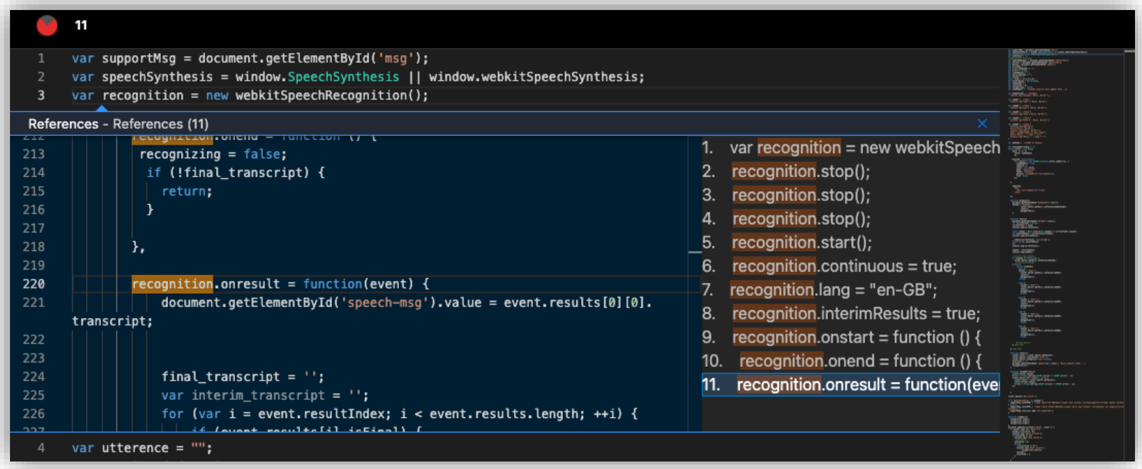
voiceye.js ~/Downloads/bharatpaudyalbcu-voiceye_web-d64a9d2e2edb/src - References (11)

```
            }

        },

        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 2: Find all reference utilized in VSCode.



11

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

References - References (11)

```
         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) {
```

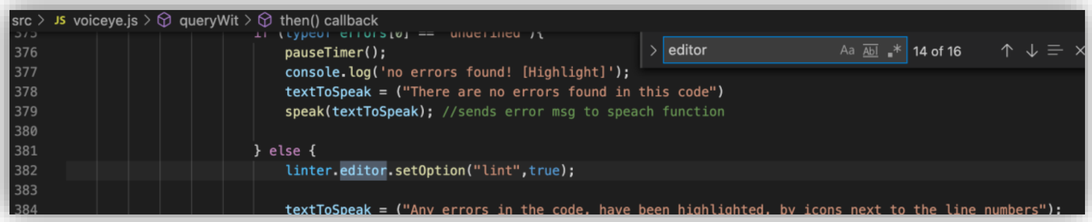
1. var recognition = new webkitSpeech
2. recognition.stop();
3. recognition.stop();
4. recognition.stop();
5. recognition.start();
6. recognition.continuous = true;
7. recognition.lang = "en-GB";
8. recognition.interimResults = true;
9. recognition.onstart = function () {
10. recognition.onend = function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

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

Figure 3: Find all references in our prototype

Find / Search

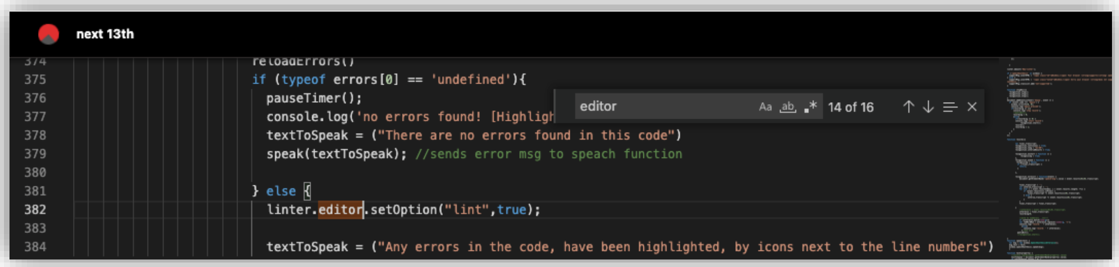
A "*Find*" feature is also a common component of mainstream IDEs and enables users to search for code through providing a search term (via the keyboard). For example, in Visual Studio Code developers can initiate the tool either through a menu item or via keyboard shortcut (CTRL/CMD + F). Users can then enter their search terms (e.g. "*editor*") into a text field and press the enter key to navigate to 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 onto the next result or use the mouse to directly select the arrows within the "*Find*" area located towards the top of the interface (Figure 4).



```
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 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

We adapted this feature for speech interaction through enabling the user to issue an associated verbal command (e.g. "*find x*", "*search x*", etc) to search for the desired term. The search would then be performed and a similar "*Find*" box is 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 the instance they would like to select. Figure 5 shows the implementation of the "*Find/Search*" feature in our 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 has resulted in navigating to the 14th search result.



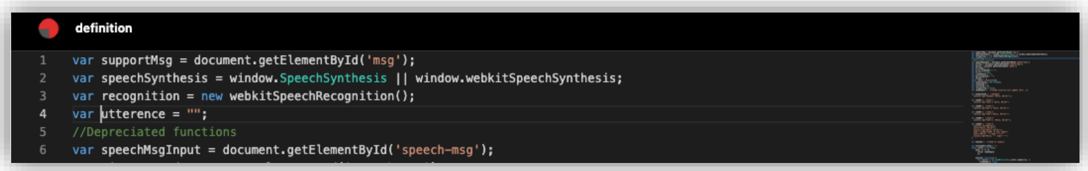
```
reloadErrors()
if (typeof errors[0] == 'undefined'){
  pauseTimer();
  console.log('no errors found! [Highligh
  textToSpeak = ("There are no errors found in this code")
  speak(textToSpeak); //sends error msg to speach function

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

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

Figure 5: Implementation of 'Find' feature in the prototype.

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, etc.). 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 relevant definition of the specific identifier chosen by the user and the navigation is completed (Figure 6).

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.



```
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 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.

5 EVALUATION

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

5.1 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 Repetitive Strain Injury (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 2).

5.2 Apparatus

All study sessions were conducted remotely via Zoom[50] and Microsoft Teams[51]. 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, etc.) to trigger the speech recognizer.

5.3 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 of 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., in terms of selecting buttons for starting and completing tasks), as well as for initiating the speech recognizer. However, they were still required to use the speech commands available within the prototype for completing the navigation tasks (via their own microphone). Four participants opted to use an eye tracker, while the remainder confirmed they would use a keyboard as 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. Participants then completed training tasks (e.g., moving to different lines/positions, finding references and moving to a specified position, etc.) 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 categorized based on real case debugging scenarios utilized in previous studies [3, 29, 32].

Table 2: 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

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/Capral 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

Similar to Shakil et al [34], 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 steps to uncover for each code snippet. The code snippets were taken from an online (open source) code repository [11] with common JavaScript syntax errors integrated (i.e. undefined methods, incorrect method parameters, typographic errors, etc.) [15, 28]. This approach was adopted to present realistic navigation scenarios where manipulation of the onscreen cursor was required over both shorter and longer distances within different code snippets. Participants were initially presented with one of these snippets and then required to navigate to each of the errors embedded within the file. To eliminate code comprehension time impacting on task completion, participants were provided with the exact steps required for navigating to errors (thus utilizing the same approach used in previous related studies [33, 34]). 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 use of the different navigation features available 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 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 also required to complete them (i.e., "*go to line", "left", "right", "up", "down*" and "*end of the line*").

Once participants had familiarized 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 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 [4] 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.

5.4 Measures

Task Completion: Task completion times were measured in milliseconds from when participants started each task (i.e., after they selected to "Start Task" button) until the task had been completed.

Usability: Perceptions of usability were measured using SUS [4] which was administered after all navigation tasks had been completed.

Speech Recognition Accuracy: Errors were categorized into two themes: <u>*Speech Recognition*</u> – where the recognizer was inaccurate (e.g. "go to line 160" misrecognized as "Go to wine 160") and <u>*Unrecognized Commands*</u> – where the recognizer misinterpreted the intention behind the user voice commands or the command was not recognized (e.g. "write down").

5.5 Results

Task completion time and Usability: All participants were able to complete all the tasks and utilize the navigation features within the prototype. The average completion time for all the tasks was 11.9 minutes (SD=3.5 min). 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). The navigation features also obtained an average SUS score of 83.4 (SD=14.3) which can be labeled as ‘Excellent’ [4].

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 unrecognized commands. Some of the common speech misrecognition commands included “write” instead of “*right*”, “*done*” instead of “*down*”, and “*to*” instead of “*two*”, etc. Whilst common unrecognized commands included “*write position*”, “*last 20”,* and “*west 6th*”. Despite misrecognition in a small number of cases (*N*=30), the model was still able to identify the correct intention of the command issued by the user. For instance, in multiple cases, “*right*” was misrecognized as “*write*”, although the model would accurately identify the intent as “*right*” and would perform the appropriate action.

Subjective Feedback: A thematical analysis was conducted on qualitative data obtained during the open-ended questionnaire and observations during the evaluation sessions.

Perceptions of “Find All References”: All participants provided positive feedback in relation to the “Find all References” navigation approach with comments emphasizing 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) specifically 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 verbalize 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., verbalizing the exact numeric value e.g., *“5”, “9th”)* and relative numbers (i.e., verbalizing the number with respect to current selected item position, e.g. “*next 6th”, “previous 5*”, etc.) 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 recognized properly) with participants suggesting the possibility to incorporate customizable vocabulary where commands can be tailored depending on user preferences: “…*In Talon I can customise my own thing and has less recognizing bugs, so including custom grammar can improve the accuracy of recognizer 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 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…*”. Participant 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",* etc.) in addition to relative commands (e.g. "*previous 2*" which would select the instance two places before the current 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 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", etc.) ,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 utilize them to complete the tasks. Four participants suggested potential improvements around the accuracy of recognizer through incorporating a local speech recognizer. 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.

6 DISCUSSION AND FUTURE WORK

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) [6, 29, 32, 41]. 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" [2, 26]) 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 paper 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 speech interaction.

The results from a user evaluation with developers who have physical impairments found the code navigation approaches to be 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. [32] 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 our study. Paudyal et al. [29] used similar navigation commands within their Voiceye prototype (i.e. "go to line x") which was also rated positively by developers with physical impairments, although participants highlighted the requirement for commands to support more advanced "Find" features. Moreover, whilst navigation features such as "Find all References" and "Go to Definition" have been explored and evaluated within systems supporting eye gaze interaction [34], no previous work has investigated these types of features in relation to voice coding. Our work also confirms that 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, 11 participants, who use Talon as their primary development environment expressed a desire for the approaches to be integrated into existing mainstream IDEs (such as VSCode and Talon) to support their development workflow. This work therefore addresses key challenges associated with navigation of code via voice interaction through presenting fully functional techniques that can be utilized by developers with physical impairments.

One limitation of the work is the accuracy of speech recognition (which is a known issue within the field [20, 32, 40]), - the system currently utilizes a cloud-based speech recognizer 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 recognizer (stored on the client-side) to help further address these issues. Moreover, the model is currently trained with a limited range of samples and could developed further through integrating the terms collected during the evaluation, as well as through additional samples from a wider and more diverse userbase (e.g. female developers, non-English speakers, etc.) [37].

In terms of future work, it will be important to further evaluate the navigation approaches in longitudinal studies to explore how developers' perceptions and usage evolve over multiple interactions (including comparison with alternative methods – e.g. those found in commercial applications such as Talon [42]). We feel that increased exposure will likely result in more positive perceptions (as users become increasingly familiar with navigating code via voice control), although it will be important to empirically evaluate this in future studies. Scrolling through code via voice control is another important area where there has been a lack work completed to date. Whilst related work has looked at the use of other modalities to support scrolling within interfaces (e.g. via gaze [14, 31, 34]), it is unclear how this might best be facilitated via speech input. The use of additional assistive inputs in conjunction with speech (e.g. gaze, keyboards, switches, etc. [12, 29, 33]) is also an important research area that could provide insights on how additional modalities can complement code navigation (e.g. using gaze to perform selections when users experience challenges in terms of speech recognition accuracy). Furthermore, investigating how other key coding activities such as debugging can be integrated with code navigation techniques is a crucial area where additional research is required. The findings from this paper may also present wider benefits for non-disabled developers through augmenting and enhancing their current development workflow via voice control (in combination with mouse and keyboard input) [17], although further studies are required to empirically investigate and validate whether this is the case.

7 CONCLUSION

We introduced a range of common code navigation approaches (used in mainstream IDEs) and tailored them for multimodal voice interaction to support developers with physical impairments (i.e. "*Find All References", "Go to Definition", "Find*"). Results from a user evaluation found that the techniques exhibited a high-level of usability and enabled developers to complete a wide range of common navigation tasks. Our work also highlights important areas where further research is now required to support the development of more inclusive coding environments.

REFERENCES

[1] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 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), Association for Computing Machinery, New York, NY, USA, 91–100. DOI:https://doi.org/10.1145/3132525.3132550

[2] Vinay Augustine, Patrick Francis, Xiao Qu, David Shepherd, Will Snipes, Christoph Braunlich, and Thomas Fritz. 2015. A Field Study on Fostering Structural Navigation with Prodet. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 229–238. DOI:https://doi.org/10.1109/ICSE.2015.151

[3] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 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*, Association for Computing Machinery, New York, New York, USA, 3043–3052. DOI:https://doi.org/10.1145/2702123.2702589

[4] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: adding an adjective rating scale. *Journal of Usability Studies* 4, 3 (2009), 114–123.

[5] Andrew Begel and Susan L. Graham. 2005. Spoken programs. In *Proceedings - 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE, 99–106. DOI:https://doi.org/10.1109/VLHCC.2005.58

[6] Andrew Begel and Susan L Graham. 2006. An assessment of a speech-based programming environment. In *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006*, 116–120. DOI:https://doi.org/10.1109/VLHCC.2006.9

[7] Brackets. 2018. Brackets - A modern, open source code editor that understands web design. Retrieved from http://brackets.io/

[8] Andrew Bragdon, Robert Zeleznik, Steven P Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola. 2010. Code bubbles. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10* (ICSE '10), ACM, New York, NY, USA, 2503–2503. DOI:https://doi.org/10.1145/1753326.1753706

[9] Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola. 2010. Code bubbles: a working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10), Association for Computing Machinery, New York, NY, USA, 2503–2512. DOI:https://doi.org/10.1145/1753326.1753706

[10] Hashmeet Chadha, Satyam Mhatre, Unnati Ganatra, and Sujata Pathak. 2018. HTML Voice. In *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, IEEE, 1–4. DOI:https://doi.org/10.1109/ICCUBEA.2018.8697733

[11] Harendra Chhekur. 2022. Colon IDE. Retrieved May 11, 2022 from https://github.com/Chhekur/colon-ide

[12] Chris Creed, Ian Williams, and Maite Frutos-Pascual. 2019. Multimodal Gaze Interaction for Creative Design. (2019). DOI:https://doi.org/10.1145/3313831.3376196

[13] Robert DeLine and Kael Rowan. 2010. Code canvas. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10* (ICSE '10), ACM, New York, NY, USA, 207–207. DOI:https://doi.org/10.1145/1810295.1810331

[14] Hartmut Glücker, Felix Raab, Florian Echtler, and Christian Wolff. 2014. EyeDE: Gaze-enhanced software development environments. In *Conference on Human Factors in Computing Systems - Proceedings* (CHI EA '14), ACM, New York, NY, USA, 1555–1560. DOI:https://doi.org/10.1145/2559206.2581217

[15] Quinn Hanam, Fernando S.De M. Brito, and Ali Mesbah. 2016. Discovering bug patterns in Javascript. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, 144–156. DOI:https://doi.org/10.1145/2950290.2950308

[16] Austin Z Henley and Scott D Fleming. 2014. The patchworks code editor: toward faster navigation with less code arranging and fewer navigation mistakes. In *Proc. CHI* (CHI '14), ACM, New York, NY, USA, 2511–2520. DOI:https://doi.org/10.1145/2556288.2557073

[17] Pranav Joshi and Doina Bein. 2020. Audible Code, a Voice-Enabled Programming Extension of Visual Studio Code. In *Advances in Intelligent Systems and Computing*, Springer, 335–341. DOI:https://doi.org/10.1007/978-3-030-43020-7_44

[18] Hyunhoon Jung, Seongeun So, Changhoon Oh, Hee Jae Kim, and Jinjoong Kim. 2019. TurtleTalk: An educational programming game for children with voice user interface. In *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, New York, NY, USA, 1–6. DOI:https://doi.org/10.1145/3290607.3312773

[19] Thorsten Karrer, Jan Peter Krämer, Jonathan Diehl, Björn Hartmann, and Borchers Jan. 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), ACM, New York, NY, USA, 217–224. DOI:https://doi.org/10.1145/2047196.2047225

[20] Yea Seul Kim, Mira Dontcheva, Eytan Adar, and Jessica Hullman. 2019. Vocal shortcuts for creative experts. In *Conference on Human Factors in Computing Systems - Proceedings*, ACM. DOI:https://doi.org/10.1145/3290605.3300562

[21] Benjamin P. Klein and Austin Z. Henley. 2021. CodeRibbon: More Efficient Workspace Management and Navigation for Mainstream Development Environments. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 604–608. DOI:https://doi.org/10.1109/ICSME52107.2021.00065

[22] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Trans. Softw. Eng.* 32, 12 (December 2006), 971–987. DOI:https://doi.org/10.1109/TSE.2006.116

[23] Jan-Peter Krämer, Thorsten Karrer, Joachim Kurz, Moritz Wittenhagen, and Jan Borchers. 2013. How tools in IDEs shape developers' navigation behavior. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13* (2013), 3073–3073. DOI:https://doi.org/10.1145/2470654.2466419

[24] Jan-Peter Krämer, Joachim Kurz, Thorsten Karrer, and Jan Borchers. 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), ACM, New York, NY, USA, 2195–2200. DOI:https://doi.org/10.1145/2212776.2223775

[25] Thomas D Latoza, Gina Venolia, and Robert Deline. 2006. *Maintaining Mental Models: A Study of Developer Work Habits*. Retrieved from http://delivery.acm.org/10.1145/1140000/1134355/p492-latoza.pdf?ip=134.225.30.51&id=1134355&acc=ACTIVE SERVICE&key=3B621857D79A46E2.3B621857D79A46E2.7F82F319CD1C5BCF.4D4702B0C3E38B35&__acm__=1567591177_fde2357a954c9ec7f294e6bc257d8b64

[26] Yun Young Lee, Sam Harwell, Sarfraz Khurshid, and Darko Marinov. 2013. Temporal code completion and navigation. In *2013 35th International Conference on Software Engineering (ICSE)*, 1181–1184. DOI:https://doi.org/10.1109/ICSE.2013.6606673

[27] Rinor S. Maloku and Besart Xh. Pllana. 2016. HyperCode: Voice aided programming. *IFAC-PapersOnLine* 49, 29 (January 2016), 263–268. DOI:https://doi.org/10.1016/J.IFACOL.2016.11.073

[28] Frolin Ocariza, Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2013. An empirical study of client-side JavaScript bugs. In *International Symposium on Empirical Software Engineering and Measurement*, 55–64. DOI:https://doi.org/10.1109/ESEM.2013.18

[29] Bharat Paudyal, Chris Creed, Maite Frutos-Pascual, and Ian Williams. 2020. Voiceye: A multimodal inclusive development environment. In *DIS 2020 - Proceedings of the 2020 ACM Designing Interactive Systems Conference*, 21–33. DOI:https://doi.org/10.1145/3357236.3395553

[30] David J. Piorkowski, Scott D. Fleming, Irwin Kwan, Margaret M. Burnett, Christopher Scaffidi, Rachel K.E. Bellamy, and Joshua Jordahl. 2013. The whats and hows of programmers' foraging diets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13), Association for Computing Machinery, New York, NY, USA, 3063–3072. DOI:https://doi.org/10.1145/2470654.2466418

[31] Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. 2016. EyeNav. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction - NordiCHI '16* (NordiCHI '16), ACM, New York, NY, USA, 1–4. DOI:https://doi.org/10.1145/2971485.2996724

[32] Lucas Rosenblatt, Patrick Carrington, Kotaro Hara, and Jeffrey P Bigham. 2018. Vocal Programming for People with Upper-Body Motor Impairments. In *W4A*, 10–10. DOI:https://doi.org/10.1145/3192714.3192821

[33] Korok Sengupta, Sabin Bhattarai, Sayan Sarcar, I. Scott MacKenzie, and Steffen Staab. 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)*, 1–11. DOI:https://doi.org/10.1145/3313831.3376579

[34] Asma Shakil, Christof Lutteroth, and Gerald Weber. 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), Association for Computing Machinery, New York, NY, USA, 1–12. DOI:https://doi.org/10.1145/3290605.3300306

[35] David Shepherd, Kostadin Damevski, Bartosz Ropski, and Thomas Fritz. 2012. Sando: an extensible local code search framework. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (FSE '12), Association for Computing Machinery, New York, NY, USA, 1–2. DOI:https://doi.org/10.1145/2393596.2393612

[36] J. Smith, C. Brown, and E. Murphy-Hill. 2017. Flower: Navigating program flow in the IDE. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 19–23. DOI:https://doi.org/10.1109/VLHCC.2017.8103445

[37] S. Stumpf, A. Peters, S. Bardzell, M. Burnett, D. Busse, J. Cauchard, and E. Churchill. 2020. Gender-Inclusive HCI Research and Design: A Conceptual Review. *Foundations and Trends in Human–Computer Interaction* 13, 1 (March 2020), 1–69. DOI:https://doi.org/10.1561/1100000056

[38] A. Teckchandani. 2018. Slack: A Unified Communications Platform to Improve Team Collaboration. *AMLE* 17, 2 (June 2018), 226–228. DOI:https://doi.org/10.5465/amle.2018.0061

[39] Tobias Kahlert and Kay Giza. 2016. Visual Studio Code - Code Editing. Redefined. *Mikrosoft* 1, March (2016), 1–26.

[40] Jessica Van Brummelen, Kevin Weng, Phoebe Lin, and Catherine Yeo. 2020. CONVO: What does conversational programming need? In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, Institute of Electrical and Electronics Engineers (IEEE), 1–5. DOI:https://doi.org/10.1109/VL/HCC50065.2020.9127277

[41] Amber Wagner and Jeff Gray. 2015. An Empirical Evaluation of a Vocal User Interface for Programming by Voice. *International Journal of Information Technologies and Systems Approach* 8, 2 (2015), 47–63. DOI:https://doi.org/10.4018/IJITSA.2015070104

[42] 2018. Talon 0.0.7.7 documentation. Retrieved from https://talonvoice.com/docs/index.html#document-index

[43] Cursorless - Visual Studio Marketplace. Retrieved March 26, 2022 from https://marketplace.visualstudio.com/items?itemName=pokey.cursorless

[44] Serenade | Documentation. Retrieved from https://serenade.ai/docs/

[45] Dragon Speech Recognition - Get More Done by Voice | Nuance. Retrieved from https://www.nuance.com/dragon.html

[46] Monaco Editor. Retrieved March 26, 2022 from https://microsoft.github.io/monaco-editor/

[47] Using the Web Speech API - Web APIs | MDN. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API

[48] Wit.ai. Retrieved from https://wit.ai/

[49] Atom. Retrieved from https://atom.io/

[50] Video Conferencing, Web Conferencing, Webinars, Screen Sharing - Zoom. Retrieved from https://zoom.us/

[51] Microsoft Teams | Group Chat, Team Chat & Collaboration. Retrieved from https://www.microsoft.com/en-GB/microsoft-teams/group-chat-software