# VocalIDE: An IDE for Programming via Speech Recognition

Lucas Rosenblatt
Brown University, CMU HCI
6324 Douglas St.
Pittsburgh, PA 15217
lucas_rosenblatt@brown.edu

## ABSTRACT

We believe that developing tools using a publicly available speech recognition API can provide a basis for keyboard free programming. We set out to develop a specialized voice-based IDE for this purpose. Early research revealed the strengths and limitations of current vocal text editors and environments. We report on a Wizard of Oz (WOz) experiment we conducted asking participants to edit code using vocal instructions given to a "human computer." Following the study, we developed a prototype, VocalIDE, which implements specialized editing mechanisms to better facilitate vocal programming.

## CCS Concepts

**• HCI → Interaction devices → Sound-based input / output •HCI → Interactive systems and tools → User interface toolkits • HCI → Accessibility → Accessibility technologies**

## Keywords

speech recognition; IDE; upper-limb impairment;

## 1. INTRODUCTION

While 6.7% of Americans have upper limb physical impairments [1], less than 4% of developers report having any physical disability whatsoever [2]. This suggests that those with upper body disabilities are under-represented in the developer community. One potential reason for this disparity is that coding remains inaccessible to those who struggle with or cannot use standard keyboards. Current coding practice implicitly requires the use of a keyboard. This requirement prevents a person who cannot type on a keyboard from coding. While soft keyboards or specialized trackballs might allow a user to code, these systems can be inefficient, difficult to learn, or expensive [5]. They are also primarily for general computer use rather than specialized to the domain of computer programming

IDEs have yet to be fully supported by easy-to-use speech-to-text software. Most programming languages contain varied commands that must be associated with speech patterns. Systems have been built by individuals for personal use, and for those willing to dedicate a significant amount of time to learning a new "language" (see Tavis Rudd's system [7] and Ben Meyer's VoiceCode [6]).

A system of voice programming is an expert task; it will always require some user education/instruction in order for interactions to produce results (sample commands, understanding of editing mechanisms, etc.). In both Rudd and Meyer's solutions, a new user must learn a whole host of non-natural vocal commands to be able to effectively use the software. Yet asking a user to memorize a new language discourages adoption, and this instructional period should be minimized. Conventional user experience wisdom suggests that by reducing use "friction" (in this case educational overhead), users will have a higher system adoption rate.



ASSETS'17, Oct. 29–Nov. 1, 2017, Batlimore, MD, USA.



DOI: http://dx.doi.org/10.1145/3132525.3134824

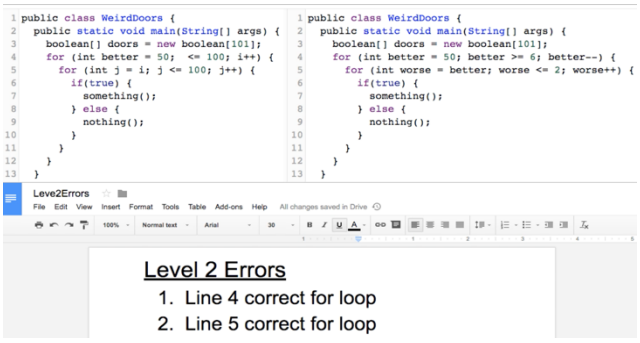


**Figure 1. Participants were asked to correct the code on the left until it matched the code on the right using vocal instructions.**

## 2. PRELIMINARY STUDY

An initial Wizard of Oz study (WOz) was conducted, seeking to gain an understanding of user instincts when giving vocal commands to a computer. What commands are programmers inclined to give a computer (using their voices) in order to write and edit code? What are the largest points of friction, and the most difficult tasks to complete? The study was conducted with 10 participants, none of whom had motor impairments, but who each had some coding experience. Future studies will be conducted on those who have upper limb impairments. The average participant age was 20 years ± 1.15, the study was conducted on 5 men and 5 women, and the average years of programming experience was 2.7 ± .95. All had taken a college programming course, and more than half had taken a high school programming course.

### 2.1 Method

Participants were asked to direct a "human computer" (a researcher) to correct code that was provided to them. The participant's screen mirrored the display of the "human computer's" screen, who operated the text editor according to a

| P | LD | Common words w/ % | Time |
|---|---|---|---|
| 1 | 16.33 | "right" 39 %, "backspace" 9.4 % | 7:26 |
| 2 | 26.13 | "jump-to" 10.0 %, "space" 6.7 % | 9:54 |
| 3 | 24.20 | "the" 10.1 %, "after" 5.9 % | 10:46 |
| 4 | 20.30 | "and" 9.5 %, "then" 9.3 % | 10:55 |
| 5 | 21.91 | "the" 9.1 %, "after" 6.6 % | 10:36 |
| 6 | 35.81 | "to" 6.4 %, "line" 6.4 % | 9:00 |
| 7 | 25.36 | "to" 7.4 %, "backspace" 7.0 % | 10:26 |
| 8 | 28.47 | "type" 6.6 %, "to" 6.1 % | 13:57 |
| 9 | 33.90 | "the" 13.2 %, "line" 7.1 % | 10:42 |
| 10 | 26.17 | "type" 13.4 %, "to" 6.9 % | 10:02 |

**Table 1. Notice that participants shared a common vocabulary, but that all of the Lexical Densities (LD) suggest inefficiency, falling below 45+ (normal speech).**

very literal interpretation of participant's speech. During each of six levels, the participant was presented two blocks of code. The right of their screen displayed the correct and complete code, and the left displayed a similar code snippet that contained some errors (Figure 2). Participants were instructed to speak to the "human computer" to transform the incorrect code into the correct code. Errors in the code were listed to control for variation in debugging time and strategy.

## 2.2 Level Generation

Levels were created following an analysis of common Java code and errors. The 50 most in-linked algorithm pages from rosettacode.org formed a common Java corpus. We ran further text analysis to create a standard of common Java syntax. Thanks to Jackson, Carver et al. [3] we had an approximation of the 20 most common Java errors. We used this knowledge to create realistic, common errors on each level of the study. This rigorous process ensured that levels were representative of important/common challenges a programmer faces when writing/editing Java code.

## 2.3 Results

The study showed that participants used different commands when vocal programming, although they shared approaches. The most common word was "the", and the most common significant words, barring articles ("the", "to", etc.), were "right," "line," "space," and "after." Significant words were navigational in nature, suggesting that a majority of participants spent their words on referencing locations in the code. For closer text analysis, we use lexical density (LD) (a measure of significant words in a text, or how many words contribute to the overall meaning of a text) [4]. The overall lexical density of the participant corpus was 6.9%, which is far less than average speech (one study suggests that speech interviews tend to have a lexical density of ~45%) [4]. This suggests that users are inefficient with natural speech vocal commands. A number of individuals typed letter by letter, while some participants provided new feature insight that we had not yet conceptualized (for example: combine "search" and "type" into one command "change").

# 3. WORKING PROTOTYPE: VOCALIDE

Drawing from research and the results of our WOz study, we developed VocalIDE using websafe javascript. VocalIDE is a fully capable voice-to-code editor. VocalIDE allows users to program using a myriad of vocal commands. Users can enter new text by word or by letter, enter syntax (+,.,\},) etc.) by common name, navigate using line numbers or minute cursor commands, and select instances of words or phrases anywhere in the document. This constitutes what we deem the "basic" usable functionality of vocal text editor. The results of the WOz study indicate that a user expects for each of these inputs or editing mechanisms to be available.

VocalIDE ignores most articles and filler words, attempting to search a command string for pertinent or significant command information. The WOz study suggested that users will include these words that do not contribute to meaningful commands. Ignoring them provides a smoother transition as the user becomes accustomed to the platform.

The WOz study suggested that users are ineffecient when navigating text using vocal commands. Furthermore, close context editing (i.e. single word level) and wide context changes (i.e. file navigation) are difficult to do with ease and accuracy using vocal commands. To address this, we proposed and implemented Context Color Editing, or CCE. CCE follows the cursor position, and highlights individual syntax and words on the cursor's line, as well as the lines above and below it. The user need only say a color for that color's highlighted text to become the active selection. CCE also highlights file tabs for quickly changing between the current open file. The goal of CCE is to allow users to be more efficient with their commands, replacing cumbersome sentence structures with short, quickly spoken colors. The entire document can be navigated using CCE alone, and we found that the introduction of CCE greatly improved close context editing ability of users by allowing them to be less accurate with their selection commands (as they could quickly reselect with color commands) and faster with their general navigation.


**Figure 2. Active prototype with CCE (a "select doors" command has just been recorded). The main.java and diff.java tabs can be swapped using vocal CCE shortcuts.**

# 4. FUTURE WORK

Another study will soon be conducted with participants who have limited upper body, arm or hand dexterity; following the study, the participants will be interviewed and the system adjusted according to their specific needs. CCE will be expanded into more visual feedback vocal navigation selection tools (for file navigation and running code).

# 5. REFERENCES

[1] Brault, M.W. July 2012. *Americans With Disabilities: 2010.* U.S. Department of Commerce.

[2] IGDA. October 2005. *Game Developer Demographics: Exploration of Diversity.* U.S. Department of Commerce.

[3] Jackson, J, Cobb, M, and Carver, C. 2005. Identifying top Java errors for novice programmers. In *FIE'05. Proceedings 35th Annual Conference.* T4C-T4C.

[4] Johansson, V. 2008. *Lexical diversity and lexical density in speech and writing.* Working Papers 53. 61-79 pages.

[5] Mackenzie, I. S, Zhang, S. X, and Soukoreff, R. W. 1999. Text entry using soft keyboards. *Behaviour & Information Technology.* 18, 4 (1999).

[6] Meyer, B. Accessed May 23, 2017. *Advanced voice-control Platform.* https://voicecode.io/.

[7] Rudd, T. http://pyvideo.org/pycon-us-2013/using-python-to-code-by-voice.html. Using Python to Code by Voice. PyVideo.org, Accessed May 23, 2017