An Empirical Evaluation of a Vocal User Interface for Programming by Voice

Amber Wagner and Jeff Gray
University of Alabama
Department of Computer Science
Box 870290, Tuscaloosa, AL 35487
ankrug@bama.ua.edu, gray@cs.ua.edu

ABSTRACT
Although Graphical User Interfaces (GUIs) often improve usability, individuals with physical disabilities may be unable to use a mouse and keyboard to navigate through a GUI-based application. In such situations, a Vocal User Interface (VUI) may be a viable alternative. Existing vocal tools (e.g., Vocal Joystick) can be integrated into software applications; however, integrating an assistive technology into a legacy application may require tedious and manual adaptation. Furthermore, the challenges are deeper for an application whose GUI changes dynamically (e.g., based on the context of the program) and evolves with each new application release. This paper provides a discussion of challenges observed while mapping a GUI to a VUI. The context of our examples and evaluation are taken from Myna, which is our VUI that is mapped to the Scratch programming environment. Initial user studies on the effectiveness of Myna are also presented in the paper.

Keywords
Vocal user interface; accessibility; human-computer interaction, CS education.

INTRODUCTION
GUIs can simplify user interaction by incorporating actions such as "drag and drop" or "click" to perform specified behaviors. The mechanism for this capability is often through the usage of a mouse requiring a minimum level of dexterity to navigate the GUI in addition to manipulating various objects on the screen. This required dexterity can pose challenges for users with physical disabilities. The ACM code of ethics states, "[i]n a fair society, all individuals would have equal opportunity to participate in, or benefit from, the use of computer resources regardless of race, sex, religion, age, disability, national origin or other such similar factors" (ACM, 1992). By not providing alternative means of access, users with disabilities are denied opportunities, such as learning skills that may allow for the exploration of career paths in Computer Science/Information Technology. The recognition that more users should have the opportunity to use GUI-driven applications motivates the underlying theme of the work described in this paper.

As an example case study, Scratch (discussed further in the "Mapping Process Using Myna" section) is an Initial Programming Environment (IPE) using visual blocks as a programming metaphor. Scratch has a vast user base across a large age group and encourages a community of contributors where students can share and learn from each other (Scratch, 2015). Currently, there are nearly 8 million Scratch programs that have been shared among students and teachers worldwide (Scratch, 2015). As an initial study to show the potential for extending an IPE to address accessibility needs, we developed Myna (Wagner et al., 2012), which supports the concept of Programming By Voice (PBV) in Scratch.

The work presented here discusses challenges encountered when mapping a GUI to a VUI and evaluates the effectiveness of the VUI based on Wobbrock et al.'s (2011) definition of ability-based design. The remainder of this paper is organized as follows: Related Work discusses using voice as an input modality in a programming environment, and the subsequent section introduces Myna and discusses the mapping process (Mapping Process Using Myna). The User Testing of Myna section presents both a pilot study and a study with the target audience for this work (i.e., those with motor impairments who desire to learn about computer programming). Final thoughts are then summarized in the Conclusion.

RELATED WORK

GUIs typically require usage of the keyboard and mouse for input, which can be difficult for users with motor impairments. A suggested solution is to provide vocal input, which has proven to be successful in some cases (Begel, 2005; Dai et al., 2004; Désilets et al., 2006; Harada et al., 2009; Shaik et al., 2003). While many tools (e.g., Vocal Joystick) are advantageous and users find them beneficial (Harada et al., 2009), they may not be as helpful when manipulating objects within a GUI, such as in an IPE like Scratch (2015). Such programs require navigation in addition to manipulation of objects on the screen. Beyond providing a means for navigation, we believe that new accessibility tools should be created using ability-based design as described by Wobbrock et al. (2011), which is explained further in the rest of this section.

We believe that the key to creating a successful, universally usable tool is to apply an ability-based design, in which developers strive to take advantage of what abilities the user possesses and make the system adapt to the user, rather than require the user adapt to the system (Wobbrock et al., 2011). This design strategy should be utilized for systems/tools/applications for all users, not just those with disabilities. To ensure that a design is ability-based rather than disability-based, Wobbrock et al. (2011) developed seven principles: Ability, Accountability, Adaptation, Transparency, Performance, Context, and Commodity, which are explained in a later section in a discussion on how our evaluation suggests the degree to which Myna addresses these principles.

Martin et al. (1989) researched two claims regarding the validity of speech as an input modality: 1) Speech is faster than typing; and, 2) Speech increases productivity. Martin's literature search confirmed that using speech increases productivity and some of the research validated that speech is faster than typing, but some uncertainty remained. Martin created an experiment to test each claim by having users navigate a graphical application (called MAGIC). The resulting data confirmed both claims: speech had a 108% time advantage versus typing full-word commands, and speech increased productivity by allowing the user to reduce the amount of glances at the keyboard by providing an "additional user-response modality" (Martin et al., 1989). Jung et al. (2007) and Hauptmann and Rudnicky (1990) performed comparisons of vocal versus keyboard input. Jung et al. (2007) presented a brainstorming experiment and found that utilizing voice in a group setting to collect ideas resulted in a larger quantity of higher quality ideas as compared to typing the ideas. Hauptmann and Rudnicky (1990) had users enter a series of numbers three different ways: 1) Using voice only; 2) Using voice to enter the numbers and a keyboard for error correction; and, 3) Using a keyboard only. Users performed fastest using the second methodology (multimodal) with the first methodology (voice only) less than a tenth of a second slower, and the third methodology (keyboard only) being the slowest at about one second slower. Based on the results presented in the above referenced papers (Hauptmann & Rudnicky, 1990; Jung et al., 2007; Martin et al., 1989), voice is not only viable, but it has the potential to increase the user's productivity.

Using voice as an input modality for programming or GUI navigation is not a new concept. Prior efforts presented some form of a voice-driven application (Begel, 2005; Dai et al., 2004; Désilets et al., 2006; Harada et al., 2009; Shaik et al., 2003). Begel (2005) and Désilets et al. (2006) focused on voice-controlled applications for textual programming, and Dai et al. (2004) and Harada et al. (2009) discussed voice-driven cursor control techniques for general navigation needs. Dai et al. (2004) evaluated a voice-driven math program for the visually-impaired, but experienced the same issues as Begel (2005) and Désilets et al. (2006). The goal for Begel (2005) was a design based on "natural verbalization." The problem that both Begel (2005) and Désilets et al. (2006) experienced was that code and natural language are far different, and adaptations had to be created to provide the required user flexibility.

Shaik et al. (2003) utilized Java's Robot class along with a rule-based grammar to control the mouse and keyboard programmatically. Rather than translate verbal commands into code like Shaik et al. (2003), Dai et al. (2004) and Harada et al. (2009) focused on cursor movement. Dai et al. (2004) used a grid-based solution to allow the user to identify the location on the screen that he/she wishes to click. The specific approach explored by Harada et al. (2009) used ten syllables such that the sound generated by each syllable was mapped to a specific direction and speed (Vocal Joystick). Wobbrock et al. (2011) supported the tool created by Harada et al. (2009) for meeting design requirements of ability-based design. However, our approach for mapping a VUI to the GUI of Scratch uses the Java Robot class similar to Shaik et al. (2003) due to the possible duration users might spend programming, so as not to cause vocal fatigue.

MAPPING PROCESS USING MYNA

Scratch is an IPE used by students from elementary school through college all over the world. The creators of Scratch (Resnick et al., 2009) minimized commands to keep the interface simple; the minimal number of commands makes Scratch an excellent case study for our research in mapping a GUI to a VUI. The VUI we created, Myna, uses the xy-coordinates of a pixel as a grid to locate each of the Scratch commands. The program uses the Java Robot class, which provides programmatic control of the mouse and keyboard. Figure 1 illustrates the process of how vocal commands are implemented by Myna.

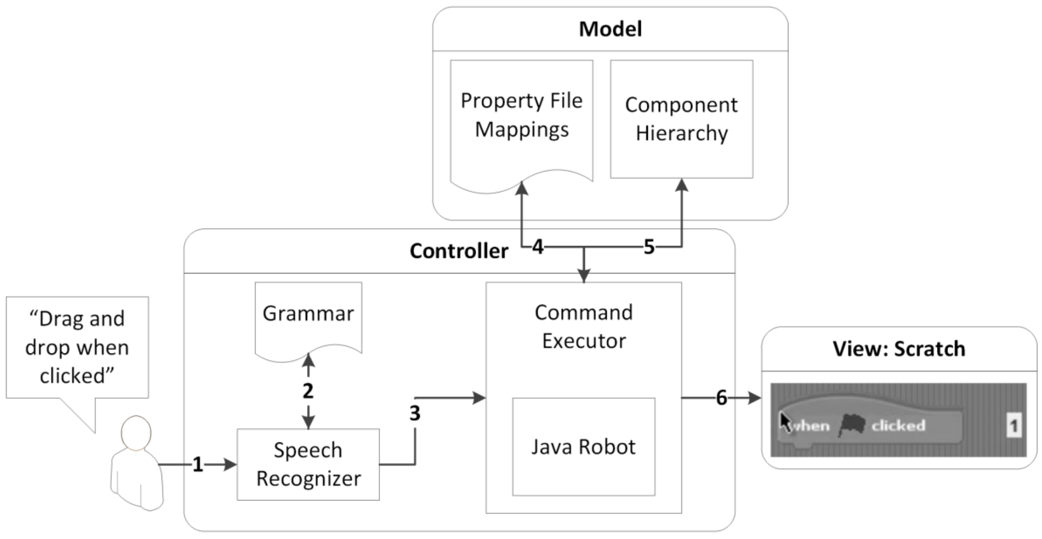


Figure 1: Myna workflow using Model View Controller design.

Myna uses the xy-coordinates of the mapped GUI and processes the vocal commands to perform a specific behavior (e.g., "click," "drag") (Wagner et al., 2011). When commands or code blocks are dropped into the code editor, Myna assigns a numeric label (implemented in the code editor of Figure 2 as transparent frames) to the command for ease of access when performing subsequent tasks (e.g., "delete," "drop before") (Wagner et al., 2011). Myna is a separate background application that runs on top of Scratch – no changes to the underlying Scratch code were needed to create the VUI to support PBV. This design also allows us to explore the application of Myna to other similar programming environments that are block-based, such as Lego Mindstorms robots, Snap!, and the Hour of Code activities hosted by Code.org.

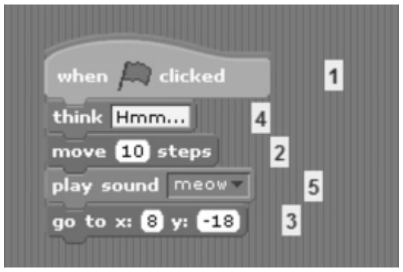


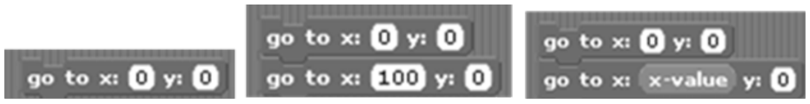
Figure 2. Transparent frames (labels) identifying each block.

While creating this transparent interface, we discovered an assortment of needed features (e.g., pausing speech recognition, horizontal parameter expansion, and deleting blocks) that present new challenges in mapping a GUI to a VUI due to losing the context of the xy-coordinate (i.e., the location was moved because of another action) or because the command was not native to Scratch. We describe these three challenges in detail and their corresponding solutions in the next section.

Mapping a GUI to a VUI: Challenges Encountered and Solutions Considered
The following are three challenges we encountered during the mapping process; our solutions are also presented.

1. *Parameters – horizontal expansion*: Various commands (e.g., "go to x y") in Scratch require multiple parameters (Figure 3a). Depending on the length of the information entered in the first parameter slot, the xy-coordinate of the latter parameter slot(s) will change (Figure 3b).

   Solution: The distance from the beginning of the command block to each parameter slot is stored in a property file. As the user enters information vocally, Myna captures the information to measure the length and edits the appropriate property file. For example, if the user submits the number "100" in the first parameter of the block in Figure 3b, Myna takes a constant (the width of one character – 8 pixels), multiplies it by three (the length of the string entered), and calculates the new distance from the latter parameter slot to the beginning of the command block. If a variable is inserted in a parameter slot (Figure 3c), a similar process is used. The variable's name is stored; based on the name, Myna calculates the length of the variable name and adds the edges surrounding the string in the variable block, which is consistent for all variables.



a) Multiple parameters *b) Value alters length* *c) Passing a variable*

Figure 3. Passing parameters.

2. *Delete*: Scratch does not provide a "delete" command. Instead, the user drags the block to be deleted from the code editor to the command palette or right-clicks on a block and chooses "delete." If the desired block is in the middle of a code segment (Figure 4), the user must separate the desired block from the segment, drag the desired block to the command palette, and reconnect the remaining blocks in the code editor.

   Solution: The solution, while more difficult to implement within Myna, is easier on the end-user than the mouse/keyboard method for deleting. The user will state, "delete" followed by the number of the block to be deleted (Figure 5). Using the Java Robot class and information regarding the location of the blocks on the screen, Myna separates the blocks within the code segment, if necessary, and drags the desired block to the command palette, mimicking what the user would do with the mouse. After deleting the block, the number labels are re-generated to avoid leaving an empty label on the screen.

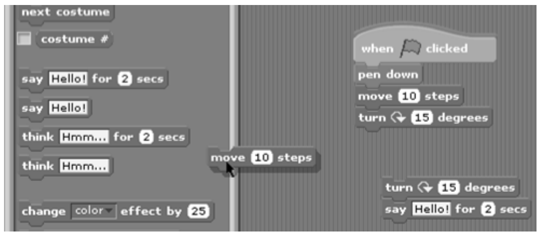
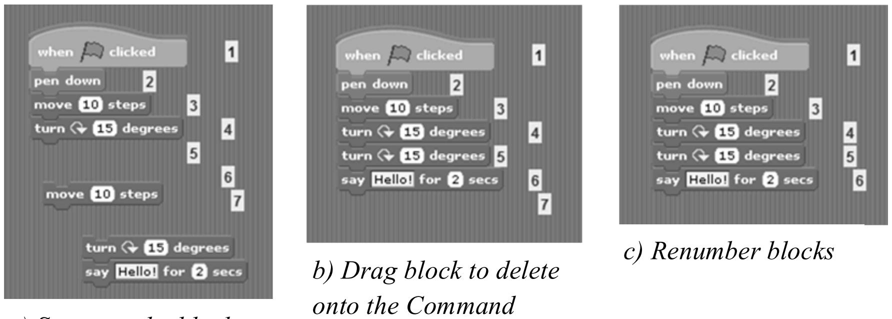


Figure 4. Deleting a middle block in Scratch

3. *Pause*: "Pause" is a non-native command primarily because there is no need for it within Scratch. However, in Myna the speech recognition engine is always in the listening state and interpreting utterances. If someone were to enter the room and begin speaking to the user, the speech recognition engine would receive the conversation and begin trying to perform actions based on the utterances collected; therefore, the user requires the ability to "pause" the application and "resume" it as needed.

   Solution: When the user says, "pause," a dialog window will appear informing the user that Myna has been paused. A new thread will start allowing the speech recognition to continue monitoring; however, only the word "resume" is a valid command. After the user says, "resume," Myna will return to its normal state.



a) Separate the blocks

b) Drag block to delete onto the Command Palette, and reconnect remaining blocks

c) Renumber blocks

Figure 5. Deleting a middle block with Myna.

USER TESTING OF MYNA

We conducted two separate studies: the first was a pilot study with five graduate Engineering students, and the second was a target study with clients from United Cerebral Palsy of Birmingham (UCP). Each study is described individually in this section.

Pilot Study Description

Although we want our VUI, Myna, to function as well as the corresponding GUI in Scratch, the goal is not to compare the two to each other, but to provide similar functionality requiring the participants in the pilot study to use both tools. The type of study performed was a one-group post-test only approach. The target population cannot currently utilize Scratch because of the dexterity required by the Windows Icon Mouse Pointer (WIMP) metaphor; therefore, a control group is not reasonable. The initial pilot study was intended to evaluate the study design and instruments used; however, the pilot study also provided an opportunity to evaluate the current state of Myna.

The following hypotheses were evaluated in the pilot study:

H_0: Myna is not a suitable alternative to the mouse/keyboard.

H_1: The time difference between the mouse/keyboard and Myna is not significant.

H_2: Myna is easy to learn: The number of errors (e.g., saying the incorrect Myna command) occurs less with each use.

H_3: The implemented solutions for Delete, Pause, and Parameters are successful.

Beyond evaluating the above hypotheses, we also needed to determine if Myna meets the ability-based design criteria defined by Wobbrock et al. (2011).

Pilot Study Testing Procedures

The participants evaluated Myna using a laptop and microphone, where each participant was given a script with three programs to write. The programs were designed to be very simple, but they each test various commands such as: “drag and drop,” “drop after,” “pause,” “delete,” “set property at” (parameters), “help,” and menu navigation. The simplicity of the programs allowed the participants to focus on testing the “pause,” “delete,” and “set property at” (parameters) commands.

During testing of the Pilot Study, the participants created each program in Scratch using mouse/keyboard input first. The reasoning behind having the pilot participants start by using the mouse/keyboard method was twofold:

1. To time how long the participants took to complete the program using both mouse/keyboard and voice to determine whether voice took a comparable amount of time (the target audience does not have the mouse/keyboard as an option, Myna should be comparable to the mouse/keyboard but does not need to be better); and

2. To allow the participants to obtain a brief understanding of how Scratch works since the goal is to evaluate Myna’s usability rather than Scratch’s usability.

Next, the pilot participants recreated each program using Myna (voice only). Each participant’s experience using both input modalities was observed and documented. Finally, each participant completed an experience survey regarding how he/she felt about using Myna.

Data Collection

For the Pilot Study, two data collection methodologies were utilized: observation and survey. No data was captured electronically. During the observation, errors were recorded: an error could be the fault of the user (e.g., the user says the incorrect command) or the fault of Myna (e.g., speech recognition error or xy-mapping issue). The amount of time to complete each task was documented to determine if completing the task using the VUI took longer than when using the GUI. Furthermore, for any questions on the survey that appeared to contradict observed behavior (e.g., the participant appeared frustrated but commented otherwise on the survey), the participant was asked for an explanation, which was also recorded.

The survey asked more specific questions regarding the user's experience during the experiment. The questions asked the participants about their opinion of using specific features of Myna (such as the solutions presented in the Mapping section); these included: if the commands were relevant and performed the expected actions, if help or error messages were useful, the user's overall impression of Myna, and basic demographic information.

Pilot Study Results
The results from the pilot observation are presented in Table 1. Survey results are summarized throughout the Discussion section.

Observation	Program 1	Program 2	Program 3
Type A - Participant stated incorrect vocal command (Average count)	0.8	1	0.2
Type B - Participant stated incorrect Scratch command (Average count)	0.6	0	0.4
Type C - Speech recognition was inaccurate (Average count)	4.8	2.4	1.8
Type D - Myna placed the block in the incorrect location (Average count)	1.2	0.8	0.8
Time to complete with Scratch (Average seconds)	114.2	83.2	88.6
Time to complete with Myna (Average seconds)	98.6	113.6	112.8

Table 1. Summary of observation data.

As can be seen in Table 1, there are four types of errors that can occur. Errors of Type A and D are directly related to Myna's performance; errors of Type B and C, while of interest, are not directly related to Myna's performance. To evaluate the ease of learning Myna, we focus on Type A errors, which occur when a user forgets one of the Myna commands (e.g., "drag and drop," "set property at") and causes an undesired action in Scratch. Type B errors occur when the user misreads or misunderstands one of the Scratch blocks (e.g., user says "think for seconds" instead of "think"). This erroneous command was not what the program script stated; thus, it is a type of logic error rather than an error reflecting poorly on Myna's functionality. Type C errors are due to speech recognition issues and are not a function of Myna's performance. Although speech recognition errors give the appearance of being a Myna issue, our work is more focused on the research of the viability of a VUI, not the evaluation of speech recognition. To avoid these errors, additional training must occur between the user and the initial configuration of the speech recognition engine. Type D errors are Myna functionality issues, and need to be corrected before further user testing.

Pilot Study Data Analysis
In order to show that Myna is a suitable alternative to mouse/keyboard interaction, we must determine the degree to which the null hypotheses can be rejected. To do so, the remaining hypotheses must be supported by the observations and survey data collected. In order to support H_1, a paired t-test was performed comparing the time to complete the programs with mouse/keyboard and the time to complete the programs with Myna (see Figure 6). The result of the t-test is 0.323347 with an alpha of 0.05, which supports H_1.

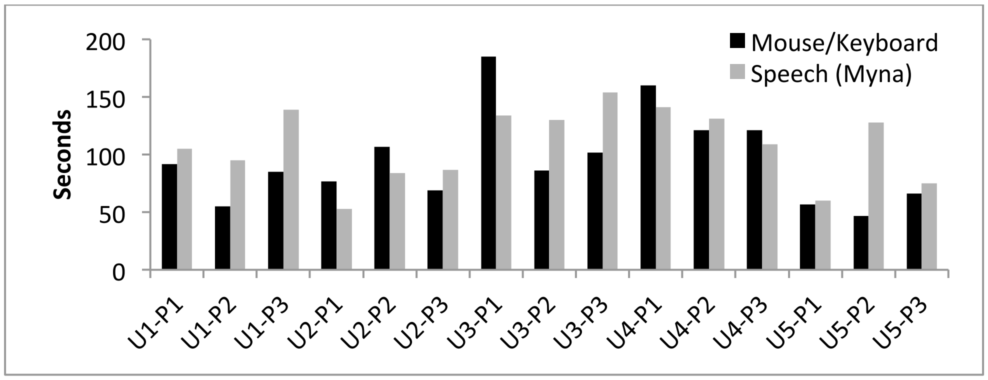


Figure 6. Time to complete using mouse/keyboard vs. Myna (U1-P1 = User 1 – Program 1).

For H_2, both quantitative and qualitative data was analyzed. First, the total number of errors due to a participant stating the incorrect Myna command (Type A error) were summed. Then, the qualitative data provided by the participants was analyzed, and the median of the participants' responses was calculated. On all three questions, the median was four ("Somewhat easy"). Based on the number of errors decreasing dramatically from programs one and two to program three, in combination with the qualitative data provided by the participants, H_2 is supported: Myna is easy to learn.

To determine if expected functionality has an impact on perceived ease of learning, a chi-squared test was performed between two survey questions:

1. I feel the vocal commands in Myna perform the action that I expect ("Always," "Most of the time," "Half of the time," "Some of the time," or "Never"); and
2. I perceive the degree to which Myna is easy to learn as ("Very difficult," "Somewhat difficult," "Neither difficult nor easy," "Somewhat easy," or "Very easy").

The result was 0.513, which is greater than alpha = 0.05. We may not have a large enough sample to adequately perform this test; therefore, this requires further investigation in future studies.

The percentage of the number of errors relating to each feature was calculated to evaluate H_3. Of the 14 total Myna errors (Type D), one was due to Parameters; none were caused by Delete or Pause. Thus, 7% of the errors were caused by the special features mentioned in the Mapping section. User feedback was positive for all but Parameters (the median was calculated and resulted in scores of 4s and 5s for all categories except for Parameters, which received a score of 3); therefore, H_3 can neither be supported nor rejected.

Errors per person	11
Errors per native speaker	4
Errors per non-native speaker	21.5

Table 2. Average number of speech recognition errors per person.

Errors of type C, speech recognition errors, have the highest occurrence. Two of the five Pilot Study participants were non-native English speakers, and their accent might have resulted in more errors (see Table 2 for the average number of errors per non-native/native speaker). To validate this idea, we performed a non-equivalent variance, one-tailed t-test with a result of 0.061, which is greater than alpha = 0.05, but it is not statistically significant.

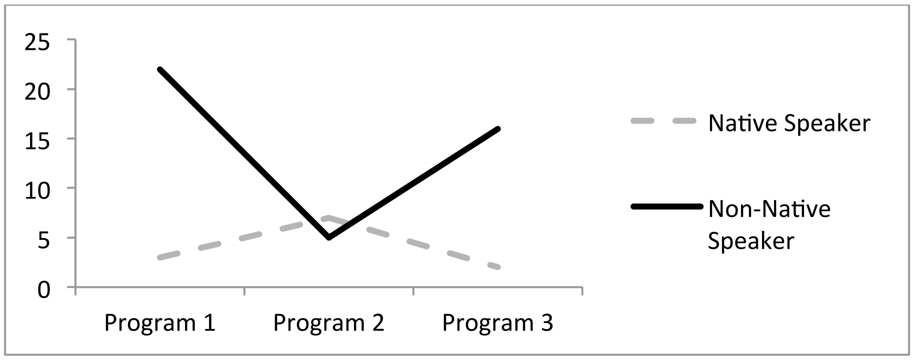


Figure 7. Sum of speech recognition errors per program by speaker.

Program 2 seemed to have the fewest speech recognition errors for non-native English speakers, but the most speech recognition errors for native English speakers (see Figure 7). This highlights that word pronunciation differs for each user and suggests a need for a custom grammar wizard, which would allow users to customize the grammar to better match his/her needs. One native English speaker experienced zero speech recognition errors throughout all three test programs, which illustrates the accuracy of the speech recognition is based on articulation and pronunciation.

UCP Study Description

The UCP Study was conducted during a six-week summer camp for young adults (ages 14-23) sponsored by UCP. Participants worked one-on-one for approximately 30 minutes once a week. We spent the first week of the camp getting to know the participants in order to make the participants feel more comfortable working with us. The type of study performed was a one-group post-test only approach.

Initially, seven attendees of the camp elected initially to participate in the study; however, only two attendees were able and willing to participate (reasons for this are described in the Threats to Validity subsection).

UCP Study Testing Procedures

Participants were introduced to Scratch during the second week of camp and given a brief tutorial on how Scratch worked by creating a simple program (the program drew a square on the screen, and then we worked with the students to edit the code to draw additional shapes). Session one (during week three of camp) began the vocal testing portion of the study, which was performed on the same laptop as in the Pilot Study with a directional microphone. At first, the participants were given an exact script of the vocal commands to execute in addition to a list of all available vocal commands. The exact script was provided during the target study because some of the participants did not have the cognitive ability to fully understand how to solve the program; however, the participant who completed four sessions was able to program without the script by the fourth session. This participant spent the final session building his own program. Participants executed the commands while we made observations. At the end of the study, the participants completed a brief satisfaction survey regarding their experience using Myna. Only one participant completed all four testing sessions; a second participant completed two testing sessions (as the other two sessions were spent working on the meter of his speech); and two other participants did not have the vocal ability to complete a full session.

UCP Study Results

Because all participants in this study had a lack of fine motor skills, we only evaluated their use of Myna (as opposed to both the mouse/keyboard and Myna). The participants were not timed in order to ensure each participant was comfortable and did not feel pressured. Therefore, the data collected during observation was minimal and focused primarily on the same errors documented during the Pilot Study. The errors observed for participant one are listed in Table 3. The errors observed for participant two are listed in Table 4.

Observation	**Session 1**	**Session 2**	**Session 3**	**Session 4**
Type A - Participant stated incorrect vocal command	0	1	0	0
Type B - Participant stated incorrect Scratch command	0	0	0	0
Type C - Speech recognition was inaccurate	1	0	0	0
Type D - Myna placed the block in the incorrect location	0	2	0	0

Table 3. Error counts for Participant One.

Observation	**Session 1 – Invalid**	**Session 2**
Type A - Participant stated incorrect vocal command	0	0
Type B - Participant stated incorrect Scratch command	0	0
Type C - Speech recognition was inaccurate	5 – stopped session	2
Type D - Myna placed the block in the incorrect location	0	2

Table 4. Error counts for Participant Two.

UCP Study Data Analysis
Due to low levels of participation, there is very little data from the UCP Study to analyze. In Tables 3 and 4, we see few errors (one) of Type A and B across all four sessions. This was primarily due to the fact that we wrote out the vocal commands necessary to create the desired program. The participants only had to read the scripts when using Myna; however, Participant One was able to use Myna to create his own program by Session Four without Type A and Type B errors. Participant Two needed assistance with understanding the script because his reading level was not advanced enough to read the scripts.

Type C errors also decreased for both users from session to session, which represents Myna's goal. The Type D errors that occurred were related to incorrect vocal command and speech recognition errors.

User Testing Data Discussion
For the Pilot Study, in the first program, Myna averaged 15.6 seconds less time than the mouse/keyboard; however, in programs two and three, Myna averaged 30.4 and 24.2 seconds longer than the mouse/keyboard for a total average increase of 13 seconds. In the second program, the increase in time might be due to the extra step of using "pause" (there is no pause function in Scratch; it is not necessary when using the mouse/keyboard). Some participants, however, took less time to complete the program in Myna. Based on the small increase in time on average and decrease in some circumstances combined with the t-test value of 0.32 (which is greater than the alpha), Myna may not increase productivity, but the increase in time is not significant. Again, our goal is not to replace the mouse/keyboard input modality, but to instead provide an alternative option for those who cannot use the mouse/keyboard.

The number of errors occurring on average is minimal with the exception of speech recognition errors. Speech recognition often requires some training for both the speech recognition system and the user. The largest number of errors occurred for participants who are non-native English speakers. If the extreme outlier of participant U4 for program one is removed, the average number of speech recognition errors for program one becomes 0.75, and if we do the same for Program three, the average number of errors becomes 1.75, both of which are far more reasonable. As Table 2 illustrates, the average number of errors per person is substantially higher than the average number of errors per native speaker, and substantially lower than the average number of errors per non-native speaker.

Regarding the UCP Study, the speech recognition errors decreased over time, as can be seen in Tables 3 and 4. The results for participant two improved dramatically as the initial session ended early because the participant's meter of speech was not conducive for the speech recognition engine; however, after brief vocal training, participant two was able to change his meter and successfully build programs using Myna.

We found it interesting that user performance improved over time. Type A errors and Type D errors (both indicative of Myna performance) decreased over time in both studies. As previously mentioned, Type C errors do not meet this goal for the Pilot Study; however, a decrease in errors from Program 1 to Program 3 is shown in the data, but the trend is inconsistent. Type B errors occurred infrequently, but they are important to note. With more training, these errors should disappear. Type B errors did not occur during the UCP Study because the participants were given an exact script of the commands to say, and the participant who eventually stopped using a script had more training, which demonstrates that the target audience may require more initial training of programming concepts, which is not unexpected.

Overall, the Pilot Study participants felt Myna was "somewhat easy" to learn. The participants from both studies felt "satisfied" after using Myna and were not frustrated during the study. Additionally, Pilot Study participants felt that the vocal commands were "somewhat predictable" and that Myna performed the expected actions "most of the time." Because H_3 is inconclusive, the null hypothesis cannot yet be rejected; however, H_1 and H_2 are supported.

Threats to Validity

The primary threat to validity is the small participant group: five participants in the Pilot Study and two participants in the UCP Study (total of seven) is a small sample size. Initially, three other UCP camp attendees showed interest in participating, but opted not to participate when prompted throughout the summer camp. Two of the three preferred to spend time with friends over "playing on the computer." This particular group of students does not have the opportunity to interact with each other very frequently; therefore, their main goal for the summer camp was to spend time with friends. The third attendee preferred to participate in a different activity by himself, and therefore, he chose not to participate in the study. Two additional attendees lacked the vocal articulation necessary to participate. This project requires training and observation over time, which means that it is a longer study and more difficult to recruit a large number of participants, especially for the target population. There are positives to this type of study (e.g., richer data, participant becomes comfortable with the observer and provides honest feedback) and negatives (e.g., lower number of participants).

Internal Validity

By asking each Pilot Study participant to create all of the programs in Scratch using the mouse/keyboard and then again in Myna using voice, there could be a threat to the validity of the time recorded when using Myna. Since the participant would know how to complete the program, this knowledge could reduce the time to recreate it using Myna. However, the aspect being timed was not how long the user took to successfully complete the logic of the program but, rather, how long it took the user to move the commands from the command palette to the command editor. Moreover, the user had to learn Scratch commands when using the mouse/keyboard and Myna commands when using Myna, which produced a learning factor in both scenarios.

Also, Pilot Study participants were given a two minute introduction to Myna and no training in order to determine how easy Myna is to learn. However, this approach may have had a negative impact on the data as participants may have performed better and felt more satisfied with Myna after a longer training session.

Ability-Based Design Evaluation
To determine if Myna meets the design requirements set forth by Wobbrock et al. (2011), we analyzed the survey data based on each category of Wobbrock's definition.

1. Ability is defined as focusing on what users can do, which Myna does by utilizing speech rather than dexterous movement. To understand if users felt Myna met their abilities, we analyzed the Learning Myna category of the Pilot Study survey by looking at the median responses. The participants had a median response of 4/5 for each question, suggesting they felt "able" to use Myna. Additionally, the UCP Study participants both stated they would use Myna again on the satisfaction survey, and one participant stated, "[Myna] got somewhat easier the more I used it."
2. Accountability states that the system should change, not the user. The solutions described in the Mapping section are an example of how we have adapted Myna to meet user needs. The median response varied between 3/5 and 5/5. 3/5 is lower than what we desire. The Parameters issue scored the 3/5, which matches with the number of errors observed (7% of errors, 1/14, were caused by Parameter issues). While the functionality performed as expected, participants appeared to struggle with how to use the vocal command properly. Simplifying the command from "set property one at" to "edit one at" will potentially improve this functionality.
3. Adaptation states that interfaces should be user-adaptable; thus, Myna requires a customizable command wizard, which would allow users to change vocal commands to better meet his/her vocal needs.
4. Transparency is defined as giving users awareness of what the application is doing. We reviewed questions regarding how users were provided information throughout the testing process. Only the category of error messages received below a 4. The existing error messages were not called for during testing (e.g., there were no errors starting the microphone); however, the users did experience errors, because of speech recognition issues, for which there were no messages. Further development is necessary to provide more error message feedback to the user. Also, one participant commented that it would be helpful if Myna displayed the command it interpreted from the user's speech to better understand what Myna is doing, which is currently being implemented.
5. Performance should monitor user actions and then predict behaviors necessary to meet those actions in the future. While Myna does not meet this goal entirely, Myna should have predictable commands allowing the user to feel he/she can predict the action associated with each command. Both questions related to this topic received a 4/5; therefore, users found the commands to be predictable.
6. Context states that the system should sense the context and anticipate its effects. This ability is not yet built into Myna, but there are future plans to include it via a dynamic grammar. When the user chooses a particular menu, only the commands on the selected menu will be available to the user.
7. Commodity is defined as being low-cost. Commodity is one of Wobbrock et al.'s (2011) most important requirements because individuals with motor impairments already have high costs

associated with the adaptive technologies used to assist with their disability. Myna is a free tool and, other than a computer, requires only a microphone.

Based on the analysis of each guideline, Myna meets three of the seven requirements set forth by Wobbrock et al. (2011): Ability, Performance, and Commodity. Future development will be focused on correcting the issues related to Accountability and Transparency in addition to adding the necessary features for Context (dynamic grammar) and Adaptation (customizable command wizard).

FUTURE WORK

Hauptmann and Rudnicky's (1990) work demonstrates that a multimodal approach is faster than speech or mouse/keyboard alone. Wilson (2004) proposes a multimodal programming environment for Dyslexic students, and Ryokai, Lee, and Breitbart (2009) present work on a multimodal programming environment for young children. In more recent research, Zapata (2014) and Slaney et al. (2014) demonstrated success in combining other input modalities with voice. Speech plus touch (Zapata, 2014) provided for improved Translator-Computer Interaction (TCI); however, based on our target audience, touch is not the most viable input modality for our work. Eye gaze and face pose could provide a better second input modality and has been shown to be successful (Slaney et al., 2014). Based on this research, a multimodal approach should be considered. In future revisions of Myna, other input modalities (such as eye tracking, eye gaze, or face pose) will be applied in combination with voice while maintaining the low cost of the tool.

As previously mentioned, creating a VUI for a GUI can be tedious and time consuming. A possible solution is to create a semi-automated tool to shorten the mapping process. This tool should involve gathering the metadata for the graphical components in addition to capturing the dynamic behavior of the application.

A first step in speech-enabling a GUI is to understand all of the widgets that are of interest and must be clickable from vocal commands. There have been several approaches proposed to reverse engineer GUIs, based on techniques such as static analysis of source code (Staiger, 2007), dynamic execution of the application (Memom et al., 2003; Kumar & Sasikumar, 2008; Zettlemoyer & St. Amant, 1999), and reverse engineering of system resource files (ResHacker, 2010). The primary requirement of the screen scraping tool needed is to collect metadata about the different components on the screen, such as the physical coordinates of the component, the dimensions of the component, location of any parameters, the user-defined name of the component, and the type of the component (e.g., is the component static or moveable, is the component a container). Five tools providing some of the required functionality have been reviewed as candidate tools to collect the application metadata: Tesseract (Smith, 2007), VisMap (Zettlemoyer & St. Amant, 1999), Sikuli (Yeh et al., 2009; Chang et al., 2010; Chang 2011), Prefab (Dixon & Fogarty, 2010), and PAX (Chang et al., 2011). Our future work will explore a semi-automated approach based on user input to conserve the effort needed to map the GUI fully.

Aside from the static properties of the IPE screens, the dynamic behavior must also be captured. This represents the valid execution states of the interaction among various screens or modes of an application. For example, the user of a speech-enabled application should only be allowed to vocalize commands that make sense in the current context. The VUI developer must specify all of the interactions among the execution paths of the IPE. A possible solution is the use of a modeling language to capture such interactions. Domain-specific software environments (Gray et al. 2007) are emerging as powerful tools for rapid development of software from higher level visual models. Metamodeling tools have been shown to maximize reuse of domain knowledge by capturing an appropriate level of task abstraction that can be easily and efficiently used to synthesize new applications.

CONCLUSION

This paper has shown that a VUI can be a viable interface for user interaction and is an important input design consideration to ensure that an application focuses on a user's abilities rather than their disabilities (Wobbrock et al., 2011). This work presented the creation of a VUI for a GUI-based application. Interaction required both navigation and screen object manipulation for the purpose of programming by voice. Five Engineering graduate students evaluated Myna and felt "Satisfied" after using it. The difference in time to complete the testing programs is minimal and neither increases nor decreases productivity significantly. The added functionality that vocal input provides outweighs the small increase in time, particularly since motorically challenged children currently cannot use IPEs such as Scratch. Myna is "Easy to learn," but the Parameters feature needs to be strengthened. After the studies, the command for editing a parameter was altered to "edit one at" rather than "set parameter one at" reducing the amount of required vocalizations. Also, users need to spend more time with Myna to decrease speech recognition errors, which was validated in the UCP Study.

The two participants in the UCP Study stated that they felt "Happy" (indicated by choosing a smiley face on the satisfaction survey) while using Myna, and one participant added that he "Had fun!" Despite the inconclusiveness of H_3, Myna is a viable tool as an alternative to the mouse/keyboard. However, to improve user satisfaction, error rates, and to meet Wobbrock et al.'s (2011) design requirements, Myna must be improved further. Although the results illustrate that the vocal commands for Parameters should be simplified, the solutions presented for the three challenges described in the Mapping section were successful, and according to the participants thus far, Myna is easy to use, easy to learn, and fun. However, due to the low participation in the initial two user studies, more user-testing must be conducted to further validate results.

Moreover, changes will be made to Myna to allow for multiple input modalities to increase user-friendliness. Beyond Myna, a semi-automated tool will be developed to improve the VUI creation process. The goals of this semi-automated tool will be to decrease the amount of time it takes to map the xy-coordinates of the graphical components within the desired application, in addition to capturing the dynamic behavior (e.g., "drag and drop", "delete") of the application.

The Myna project page (http://myna.cs.ua.edu) contains videos of Myna in addition to a downloadable version of Myna.

ACKNOWLEDGMENTS

This project is supported by NSF grant IIS-1117940, with previous support from a Google Research Award.

REFERENCES

Association for Computing Machinery (1992). *ACM code of ethics and professional conduct.* Retrieved August 12, 2012 from ACM: http://www.acm.org/constitution/code.html.

Begel, A. (2005, February). *Programming by voice: A domain-specific application of speech recognition*. Paper presented at the AVIOS Speech Technology Symposium – SpeechTek West.

Chang, T. (2011, October). *Using graphical representation of user interfaces as visual references*. Paper presented at the ACM Symposium on User Interface Software and Technology, New York, NY, USA. doi: 10.1145/2046396.2046411

Chang, T., Yeh, T., & Miller, R. (2011, October). *Associating the visual representation of user interfaces with their internal structures and metadata*. Paper presented at the ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA. doi: 10.1145/2047196.2047228

Chang, T., Yeh, T., & Miller, R. (2010, April). *GUI testing using computer vision*. Paper presented at the ACM SIGCHI International Conference on Human Factors in Computing Systems, Atlanta, GA, USA. doi: 10.1145/1753326.1753555

Dai, L., Goldman, R., Sears, A., & Lozier, J. (2004, October). *Speech-based cursor control: A study of grid-based solutions*. Paper presented at the ACM SIGACCESS conference on Computers and Accessibility, Atlanta, GA, USA. doi: 10.1145/1029014.1028648

Désilets, A., Fox, D., & Norton, S. (2006, April). *VoiceCode: An innovative speech interface for programming-by-voice*. Paper presented at the ACM SIGCHI International Conference on Human Factors in Computing Systems, Montréal, Québec, Canada. doi: 10.1145/1125451.1125502

Dixon, M. & Fogarty, J. (2010, April). *Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure*. Paper presented at the ACM SIGCHI International Conference on Human Factors in Computing Systems, Atlanta, GA, USA. doi: 10.1145/1753326.1753554

Gray, J., Tolvanen, J.P., Kelly, S., Gokhale, A., Neema, S., & Sprinkle, J. (2007). Domain-Specific Modeling. In P. Fishwick (Ed.), *Handbook on Dynamic System Modeling*, Boca Raton: CRC Press.

Harada, S., Wobbrock, J., Malkin, J., Bilmes, J., & Landay, J. (2009, April). Longitudinal study of people learning to use continuous voice-based cursor control. Paper presented at the ACM SIGCHI

International Conference on Human Factors in Computing Systems, Boston, MA, USA. Doi: 10.1145/1518701.1518757

Hauptmann, A. G. & Rudnicky, A. I. (1990). *A comparison of speech and typed input*. Paper presented at the workshop on *Speech and Natural Language*. doi: 10.3115/116580.116652

Jung, J. H., Looney, C. A., & Valacich, J. S. (2007, January). *Fine-tuning the human-computer interface: Verbal versus keyboard input in an idea generation context*. Paper presented at the Hawaii International Conference on System Sciences, Big Island, HI, USA. doi: 10.1109/HICSS.2007.230

Kumar, N. & Sasikumar, M. (2008, February). *Automatic generation of speech interface for GUI tools/applications using accessibility framework*. Paper presented at Techshare India: Breaking the Barriers Conference, India.

Martin, G. (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

ResHacker. (2012). Retrieved from Resource Hacker: http://download.cnet.com/Resource-Hacker/3000-2352_4-10178588.html.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM,* 52(11), 60-67. doi: 0.1145/1592761.1592779

Ryokai, K., Lee, M. J., & Breitbart, J. M. (2009, April). *Multimodal programming environment for kids: A "thought bubble" interface for the Pleo robotic character*. Paper presented at the ACM SIGCHI: Extended Abstracts on Human Factors in Computing Systems, Boston, MA, USA. doi: 10.1145/1520340.1520687

Scratch. (2015). Retrieved January 31, 2015 from Massachusetts Institute of Technology: http://scratch.mit.edu.

Shaik, S., Corvin, R., Sudarsan, R., Javed, F., Ijaz, Q., Roychoudhury, S., Gray, J., & Bryant, B. (2003, October). *SpeechClipse – An Eclipse speech plug-in*. Paper presented at the Eclipse Technology eXchange Workshop (OOPSLA), Anaheim, CA, USA. doi: 10.1145/965660.965678

Slaney, M., Stolcke, A., & Hakkani-Tür, D. (2014, November). *The relation of eye gaze and face pose: Potential impact on speech recognition*. Paper presented at the International Conference on Multimodal Interaction, Istanbul, Turkey. doi: 10.1145/2663204.2663251

Smith, R. (2007, September). *An overview of the tesseract OCR engine*. Paper presented at the International Conference on Document Analysis and Recognition (ICDAR), Washington DC, USA.

Staiger, S. (2007, October). *Reverse engineering of graphical user interfaces using static analyses*. Paper presented at the Working Conference on Reverse Engineering, Vancouver, BC, Canada. doi: 10.1109/WCRE.2007.44

Wagner, A., Rudraraju, R., Datla, S., Banerjee, A., Sudame, M., & Gray, J. (2012, May). *Programming by voice: A hands-free approach for motorically challenged children*. Paper presented at the ACM SIGCHI Human Factors in Computing Systems, Austin, TX, USA. doi: 10.1145/2212776.2223757

Wilson, D. (2004, October). *Multimodal programming for dyslexic students*. Paper presented at the International Conference on Multimodal Interaction, State College, PA, USA. doi: 10.1145/1027933.1028001

Wobbrock, J., Kane, S., Gajos, K., Harada, S., & Froelich, J. (2011). Ability-based design: Concept, principles, and examples. *ACM Transactions on Accessible Computing*, *3*(3). doi: 10.1145/1952383.1952384

Yeh, T., Chang, T., & Miller, R. (2009, October). *Sikuli: Using GUI screenshots for search and automation*. Paper presented at the ACM Symposium on User Interface Software and Technology, Victoria, BC, Canada. doi: 10.1145/1622176.1622213

Zapata, J. (2014, November). *Exploring multimodality for translator-computer interaction.* Paper presented at the International Conference on Multimodal Interaction, Istanbul, Turkey. doi: 10.1145/2663204.2666280

Zettlemoyer, L. & St. Amant, R. (1999, May). *A visual medium for programmatic control of interactive applications*. Paper presented at the ACM SIGCHI International Conference on Human Factors in Computing Systems, Pittsburgh, PA, USA. doi: 10.1145/302979.303039