Programming by Voice:
A Hands-Free Approach for Motorically Challenged Children

Proposal
June 2013

Author: Amber (Krug) Wagner
Advisor: Dr. Jeff Gray
Committee Member: Dr. Marcus Brown
Committee Member: Dr. Sandra Nichols
Committee Member: Dr. Randy Smith
Committee Member: Dr. Eugene Syriani

Department of Computer Science
College of Engineering
University of Alabama
Abstract

Computer Science (CS) educators frequently develop new methodologies, languages, and programming environments to teach novice programmers the fundamental concepts of CS. A recent trend has focused on new environments that reduce the initial challenges associated with the heavy syntax focus of traditional textual environments. There are numerous Initial Programming Environments (IPEs) available that have been created for student use that in some cases have fostered self-discovery and inquiry-based exploration. In this proposal, three IPEs are discussed: Scratch (Scratch 2012), Lego Mindstorms LabVIEW (Lego Mindstorms 2012), and App Inventor (MIT App Inventor 2012). These three IPEs were selected due to the high number of students and teachers currently utilizing them, as well as my personal experience with these environments through past teaching activities. The creative and graphical nature of these three IPEs have been shown to increase student and teacher interest in CS through adoption in classrooms worldwide at the K-12 levels, as well as interest in introductory university courses.

Although the graphical nature of these IPEs can be helpful for learning concepts in CS, a small group of students is being left out from learning experiences and engagement in CS (approximately 5% of students). Graphical environments often require the use of both a mouse and keyboard, which motorically challenged users often are unable to operate. Based on research performed and presented in this proposal, a Vocal User Interface (VUI) is a viable solution that offers a “Programming by Voice” (PBV) capability (i.e., a capability to describe a program without the use of a keyboard or mouse). There are two primary disadvantages with VUIs as a technology to address the limitations of motorically challenged users: 1) vocal strain can emerge for technical solutions that require a deep amount of vocal interactions, and 2) the process of integrating voice controls into a legacy application (e.g., an IPE) is a very time consuming process. There are existing vocal tools (e.g., the generic Vocal Joystick) that could be integrated into various applications; however, due to the needs of IPEs and the duration of IPE usage, the Vocal Joystick is not feasible due to the potential vocal strain, which is why a more command-driven approach may offer benefits for the PBV concept. The command-driven approach leads to a time-consuming process in terms of adapting legacy applications, particularly, if multiple applications (such as the three IPEs previously mentioned) require specialized VUIs. Each environment has its own layout and its own commands; therefore, each application requires a different VUI. In order to create a more generic solution, Model-Driven Engineering (MDE) and Domain-Specific Languages (DSLs) can be applied to create a semi-automated process allowing a level of abstraction that captures the specific needs of each IPE. From the specification of each IPE, a customized VUI can be generated that integrates with the legacy application in a non-invasive manner.

This proposal presents background information on IPEs focusing on Scratch, LabVIEW, and App Inventor. Accessibility issues are discussed as well as previous work surrounding the concept of PBV. Six challenges discovered during the mapping process are presented in addition to the solutions used to solve those challenges. A small pilot study was conducted to evaluate the viability of the VUI, Myna, with positive results: the difference in time between the mouse/keyboard and the VUI is insignificant, and overall, the participants were satisfied using Myna. Future evaluations and proposed work regarding the application of MDE and DSLs to support generalization are also introduced.
## Table of Contents

Abstract .................................................................................................................................................. 2

1. Introduction ........................................................................................................................................ 4
   1.1 Motivation ....................................................................................................................................... 4
   1.2 Impact on Motorically Challenged Users ....................................................................................... 5
   1.3 Research Goals ............................................................................................................................... 6

2. Related Work ...................................................................................................................................... 7
   2.1 Learning Environments .................................................................................................................. 7
   2.2 Accessibility Issues ........................................................................................................................ 12
   2.3 Programming by Voice ................................................................................................................. 13
   2.4 Voice Limitations ........................................................................................................................... 15

3. Current State ...................................................................................................................................... 16
   3.1 Myna ............................................................................................................................................. 16
   3.2 Challenges and Solutions .............................................................................................................. 18
   3.3 Pilot Study ..................................................................................................................................... 22

4. Proposed Work ................................................................................................................................... 30
   4.1 Expansion of Myna ........................................................................................................................ 30
   4.2 Evaluation ..................................................................................................................................... 31
   4.3 Planned Technique ......................................................................................................................... 32

5. Projected Schedule ............................................................................................................................. 35

6. Conclusion ......................................................................................................................................... 36

Acknowledgements ............................................................................................................................... 37

References .............................................................................................................................................. 38

Appendix A - Pilot Study Survey Summary
Appendix B - Pilot Study Observation Summary
Appendix C - Myna Observation Instrument
Appendix D – Myna Satisfaction Survey
1. Introduction

Currently, there are several nationwide initiatives (e.g., CS10K, CS4HS, code.org) working to increase Computer Science education. A particular strategy is to begin teaching Initial Programming Environments (IPEs) (e.g., Scratch (Scratch 2012), Lego Mindstorms LabVIEW (Lego Mindstorms 2012), MIT App Inventor 2012) in grades 6-12. Unfortunately, these IPEs exploit Graphical User Interfaces (GUIs) requiring a mouse and keyboard, which does not allow for an inclusive classroom for any student with a motor impairment. The primary focus of this research is to provide a method for motorically challenged children (i.e., children with limited limb mobility) to utilize IPEs in the hope of providing these children with an inclusive classroom experience in addition to skills from which they might benefit in future employment opportunities. Section 1.1 presents the motivation for this research, and Section 1.2 elaborates on this motivation by focusing on issues for motorically challenged users. Section 1.3 introduces the research goals for this proposal.

1.1 Motivation

According to the College Board, 3,698,401 students participated in the College Board’s Advanced Placement (AP) program in 2012; however, only 26,103 students participated in AP Computer Science (College Board 2013). This low participation rate suggests that either students do not understand or are unaware of the opportunities in Computer Science, or schools are not offering Computer Science courses. Figure 1 references the trajectory of student participation in the AP Computer Science program over the past decade. While there was a significant decrease in AP CS participation from 2002 to 2009, participation is slowly increasing.

The Computer Science Teachers Association (CSTA) conducted two surveys: one in 2005 and an identical one in 2007 regarding High School Computer Science. In the analysis of the two surveys, Gal-Ezer and Stephenson (2009) attribute the decrease in AP CS to three primary factors:

1. Rapidly changing technology;
2. Lack of student interest; and
3. Lack of staff support.

The College Board discontinued the AP CS AB exam in 2009 due to low participation.
Although teachers cannot control the speed at which technology changes or the amount of staff support available, teachers can provide a pedagogically sound yet interesting curriculum to motivate students rather than discourage them. The College Board is currently working on a new AP exam (CS Principles), which will be offered to students as an introduction to Computer Science course. This course will most likely be taught using an IPE such as Scratch or App Inventor because the current generation of students will not be impressed with the traditional “Hello World” program (Kelleher & Pausch 2005, Wolber 2011) and pedagogical instruction techniques used with previous generations of students. The current generation of students has been playing video games with advanced graphics since they learned to walk, and the notion of internet accessibility is often viewed as a common utility. To increase the interest and attention of students, exciting new programs and technologies are being developed (e.g. Scratch, App Inventor, Lego Mindstorms LabVIEW). Additionally, student engagement is often more successful when the context is driven by current topics that are of direct interest to students. For example, most teenagers are frequent users of smartphones, which provide a unique context for engagement. The adoption rate of mobile computing among students age 13-17 in the US continues to grow and has been estimated at 58% for smartphone adoption (Leggatt 2012). Based on the most current US census results published (U.S. Census Bureau 2010), this suggests that there are roughly 12.8M middle and high school students with smartphones. Educators can take advantage of these devices as a springboard for motivating topics involving Computer Science (College Board 2012, Fenwick et al. 2011, Loveland 2011). Mobile application development is not the only way to reach students, but based on the statistics, it is clearly an excellent starting point.

1.2 Impact on Motorically Challenged Users

Based on the statistics for AP CS participation presented in the previous section, methodologies and tools that introduce new students to CS and programming must be improved. Papert (1980) suggests that programming languages should satisfy three conditions: low-floor (simple to start using), high-ceiling (ability to increase difficulty and complexity over time), and wide-walls (ability to create various types of projects). The current approach instructors are using to meet these suggested requirements is to utilize an environment with a graphical interface that eliminates the concern of syntax (e.g., Scratch, Alice, Lego Mindstorms LabVIEW, and App Inventor) because their focus is on education and pedagogical concerns, rather than the traditional focus of a development environment for a programmer. Based on observations and empirical studies (Malan & Leitner 2007), the use of IPEs offers numerous benefits, such as: removal of the syntactic details of a language, the fostering of creativity, thus increasing engagement and interest, and the ability to start problem solving immediately.

To address the concern over the challenges of learning the syntax of a programming language, many of the current IPEs have a form of “drag and drop” programming whereby a student connects program constructs together in a manner that forces correct syntax (but may still produce logic errors). The mechanism for this capability is often through the usage of a mouse that requires a level of dexterity to connect the program statements together. However, because this new breed of IPEs requires motor functionality to operate a mouse and keyboard, those students who are motorically challenged are left out from these new learning experiences. 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 2013). By not providing alternative means of access, children with disabilities are being denied learning opportunities that may allow them to explore career paths in computing. Moreover, on June 29, 2010, the Department of Education sent out a letter to colleges and universities across the country requiring that any technology used in a classroom be fully accessible to all students. Although the letter focuses on visually-impaired students, not providing alternative accessibility options to any disabled student violates the Americans with Disabilities Act of 1990 and the Rehabilitation Act of 1973. The driving motivation for the underlying theme of this proposed research is recognition that more children (or adults) should have the opportunity to learn about programming and CS using these new IPEs. Improving the diversity of a user base has several advantages, as noted by Kelleher and Pausch, who wrote that “[i]f the population of people creating software is more closely matched to the population using software, the software designed and released will probably better match [users’] needs” (2005).

A possible solution to widen the gap of accessibility to IPEs is to use a form of a voice-driven interface to assist as an input modality for those with motor impairments. The concept of “Programming By Voice” (PBV) is the long-term focus that drives the topic of this proposed research. Voice-driven applications, such as those presented in (Begel 2005, Dai et al. 2004, Désilets et al. 2006, Harada et al. 2009, Shaik et al. 2003, Wobbrock et al. 2011), primarily cater to specific applications, like the Internet (Gibson 2007, Trewin 2006). These applications are discussed and compared in Section 2.3.

1.3 Research Goals
The overarching research goal, comprised of the four goals described below, is to create a Vocal User Interface (VUI) that will allow motorically challenged children to use graphical IPEs:

**Goal 1 – Creation:** The first goal is to create a VUI prototype for one specific IPE. This goal has been accomplished, and a prototype, Myna, has been created for Scratch; however, this prototype is being improved through user testing. Myna allows users to utilize all of the features available in Scratch via “drag and drop” abilities.

**Goal 2 – Expansion:** According to Wobbrock et al. (2011) (please see Section 2.2 for more detail), it is important to design applications to “fit the abilities” of users rather than compensate for a user’s disabilities; therefore, the vocal commands should be as easy to use as possible. To accomplish this, multilingual capabilities will be added as well as a customizable grammar where users can use a wizard to change commands that might be difficult to say (i.e., a user might have difficulty pronouncing a certain word or phrase, and a customizable grammar would allow the user to adjust the word or phrase to something easier to pronounce).

**Goal 3 – Generalization:** Creating a VUI for one IPE will not resolve the issue motivated in sections 1.1 and 1.2. Additionally, creating a VUI for a legacy application is a time consuming process. The final goal is to create a tool to semi-automate this process, which is described in more detail in Section 4.

**Goal 4 – Evaluation:** The usability of Myna will be evaluated during a United Cerebral Palsy of Birmingham (UCP) summer camp (described further in Section 4.2.2). Additionally, the semi-automated
approach resulting from Goal 3 will be evaluated in comparison to a manual solution (described further in Section 4.2.3).

2. Related Work

Instructors are beginning to utilize “drag and drop” IPEs for novice programmers. These tools are plentiful in number, but I have chosen three for the purposes of this research – Scratch, LabVIEW, and App Inventor. These three environments and their common ancestry are described below along with the accessibility issues introduced by the environments.

2.1 Learning Environments

Since 1963 (Kelleher & Pausch 2005), researchers at universities have been developing IPEs to introduce programming concepts and problem solving with languages aimed at simplifying syntax with groundbreaking simplicity introduced by Papert in 1967 with Logo (Kelleher & Pausch 2005, Papert 1980). Some environments (e.g., BASIC, Turing, and GNOME) were designed for college-level beginning programmers while others (e.g., Play, LogoBlocks, and Alice) were built for a younger target audience, such as elementary school children. The environments (e.g., Karel, GRAIL, and LegoSheets) reviewed by Kelleher and Pausch (2005) “tried to make programming accessible in three main ways, namely, by simplifying the mechanics of programming, by providing support for learners and by providing students with motivation to learn to program”.

The following three learning environments are explained in more detail in the following subsections: Scratch (Scratch 2012), Lego Mindstorms LabVIEW (Lego Mindstorms 2012), and Android App Inventor (MIT App Inventor 2012).

2.1.1 Scratch

MIT created Bongo in 1997, which allowed users to create their own video games and then share them with friends via the web (Kelleher & Pausch 2005). MIT followed this same community-driven sharing model when creating Scratch, which was introduced in 2004 as an IPE that is simplified enough to teach to third graders, yet complex enough to teach to college freshmen or non-CS majors (Resnick et al. 2009). The University of California, Berkeley has adapted MIT’s version of Scratch to a more advanced version called Build Your Own Blocks (BYOB) (BYOB 2012) (recently renamed SNAP), which Berkeley teaches to freshmen and non-CS majors. The creators of Scratch at MIT intended for Scratch to be a “networked, media-rich programming environment designed to enhance the development of technological fluency at after-school centers in economically-disadvantaged communities” (Maloney et al. 2004). The Scratch community has emerged into a social network for its 454,061 project creators who have shared 3,279,054 projects for all 1,519,271 registered members to view (Scratch 2012, Scratch Statistics 2013). Kelleher and Pausch (2005) observed that social environments such as networked applications provide more motivation for students to learn how to program, and Scratch has provided further evidence to support this observation (Resnick et al. 2009). Scratch is visited daily from 16,798 cities (see Figure 2) (Scratch 2012, Scratch Statistics 2013). As an IPE, Scratch is reaching hundreds of thousands of students and educators. The projects that are shared within the community serve to inspire other members in numerous ways (e.g., students share their culture, political and religious views,
students collaborate to create scripts other students can utilize, and students learn how to critique one another) (Scratch 2012). Scratch is more than just a programming environment; it is a socially-driven creative tool that children can use to express themselves and explore their own creativity.

![Figure 2. The location and amount of Scratch users (Scratch 2012).](image)

Scratch is heavily utilized throughout the world because of its ease of use and general availability (Scratch is freely available). Figure 3 is an image of the Scratch UI. All of the commands are “drag and drop” allowing the programmer to focus on logic and problem solving rather than syntax. Also, the programmer can develop his or her own images and import sounds, thereby fostering a creative environment that makes the underlying difficulty of the program transparent to the programmer.

![Figure 3. MIT’s Scratch interface (Scratch 2012).](image)

2.1.2 Lego Mindstorms Software
The University of Colorado created LegoSheets in 1995 to work with the MIT Programmable Brick (“intelligent, built-in computer-controlled LEGO brick that lets a MINDSTORMS robot come alive and perform different operations” (Ranganathan et al. 2008)). LegoSheets was primarily graphical and allowed users to progress slowly from controlling basic motor functionality of a robot to adding conditional statements in a program. In 1996, the MIT Media Lab developed LogoBlocks, which was “a graphical programming language designed for the Programmable Brick” (Kelleher & Pausch 2005). With LogoBlocks, users could “drag and drop” code blocks to manipulate the brick and also allowed the user
to learn about the creation and use of parameterized procedures to support generic reusable functionality.

The successor of the programmable brick is the commercially available Lego Mindstorms LabVIEW environment developed by the MIT Media Lab in 1998. Since then, the programming environment has improved along with the hardware (e.g., the initial RCX Mindstorms robot used a very fragile infrared process for downloading programs, but the newer NXT platform supports Bluetooth and USB connections). The latest Mindstorms hardware is the NXT 2.0. A special programming template has been developed for National Instruments’ LabVIEW (National Instruments), which provides a graphical programming environment that is used by many K-12 schools, including many in the state of Alabama (e.g., Alabama School of Fine Arts, Northridge High School, Tuscaloosa Magnet, Rock Quarry Middle School). The software comes with 46 tutorials to assist educators and/or students in learning how to program and command the NXT robot. The tutorials demonstrate how to build the robot (this is helpful if specific sensors are needed) and how to build the blocks for the code (Lego Mindstorms Software 2012). Lego has created a social media forum for users called NXTLOG 2.0 where users can post their projects and share ideas (Lego Mindstorms 2012), similar to Scratch. Figure 4 is a screenshot of the Mindstorms LabVIEW interface (Lego Mindstorms Software 2012), which is similar to MIT’s original design. There is a block palette on the left, the program editor in the center, and the tutorials are on the right.

![Figure 4. Lego Mindstorms’ LabVIEW interface (Lego Mindstorms Software 2012).](image)

2.1.3 Android App Inventor

Google’s Android App Inventor (MIT App Inventor 2012, Wolber et al. 2011), now maintained by MIT’s Center for Mobile Learning, enables educators to take advantage of the interest that high school students have in smartphones. App Inventor is a visual programming language that allows users to write apps using a block-oriented “drag and drop” interface to create an app’s user interface and to specify its behavior and functionality (the latter is based off of Scratch’s visual block structure). An emulator is available so that apps can be executed on a local desktop. App Inventor also integrates with Android smartphones and tablets, which enables the user-made applications to be tested on a physical device.
Similar to Scratch, app developers can package their apps for a phone and then share that app with friends on a community-based gallery; thus, creating a networked social environment and motivating the students further (Kelleher & Pausch 2005).

The popularity of mobile devices has inspired much interest as a context for teaching CS principles (Mahmoud 2011). In fact, a new learning model has emerged, LOCAL (Location and Context Aware Learning) (Barbosa et al. 2008), which combines mobile devices and wireless networks to create a new learning context. David Wolber integrated App Inventor into a general education course over three semesters at the University of San Francisco (USF) and found that “teaching App Inventor has been [his] most satisfying teaching experience in seventeen years” (Wolber 2011). Before teaching App Inventor, he used Lego Robots (Lego Mindstorms 2012) and Media Computation (Guzdial & Ericson 2009), similar to the approach followed at the University of Alabama for summer camps. Using App Inventor at USF taught students not only how to program, but also offered a context for the potential impact that apps can have on the community (Wolber 2011). Although the students attending the general education course at USF are not CS majors, the students successfully learned how to solve problems, and more importantly, the students felt successful at the conclusion of the course. Because students were able to start building applications immediately with App Inventor, they were motivated to learn “how to solve hard logic problems and specify interactive behavior with a static (though visible) language” (Wolber 2011). As described in (Fenwick et al. 2011), Fenwick and Kurtz from Appalachian State University and Hollingsworth from Elon University taught their senior classes using the Android SDK (Eclipse 2012) and App Inventor. The Elon University course started as a lecture-based course with hands-on activities followed by a project-based approach. Appalachian State University’s course was project-based from the start. Both universities found that student engagement was high and students once again exhibited an “entrepreneurial and independent spirit” (Fenwick et al. 2011).

To assist in teaching the importance of human-computer interaction, Loveland used App Inventor as a platform to motivate students in a new course (Chang et al. 2010). At the conclusion of the semester,
one student commented on his appreciation for using new technologies in the course. This is why many educators (e.g., as evidenced by positive adoption stories detailed in (Barbosa et al. 2008, Fenwick et al. 2011, Goadrich & Rogers 2011, Loveland 2011, Mahmoud 2011, Wolber 2011)) are taking advantage of mobile computing as a teaching context. Students are able to learn about programming using tools that are very personal and relevant to their daily lives. Another successful educational use of AI is the creation of data collection apps for the betterment of a community (Abelson, H. 2011); particularly, Hal Abelson mentions a medical app, which collects data in Nicaragua and is used as a mobile laboratory (Abelson, H. 2011).

2.1.4 Observations from Common and Distinguishing Characteristics among Modern IPEs

Scratch, Lego Mindstorms LabVIEW, and App Inventor have a common ancestor: LogoBlocks (Begel 1996). LabVIEW is a direct descendent of LogoBlocks (Kelleher & Pausch 2005), Lego blocks were the inspiration for Scratch (Resnick et al. 2009), and App Inventor followed the trend set by LogoBlocks in 1996. Scratch was intended for eight to 16 year old students (Resnick et al. 2009); however, Malan and Leitner (2007) found success using it in introductory CS courses at the college-level. Similarly, Lego Mindstorms LabVIEW was designed for K-12 classrooms, yet Ranganathan, Schultz, and Mardani (2008) found success using it in a freshman Electrical Engineering course. App Inventor was built for a broader audience and has had success in college classrooms (Uludag et al. 2011, Wolber 2011) and is slowly being integrated into high school classrooms (Gray et al. 2012, Roy 2012). Unlike Scratch and LabVIEW, there is no evidence that App Inventor is appropriate for students younger than high school; however, it is still appropriate for this research as it provides a higher ceiling (Papert 1980) for the older students.

While Scratch, LabVIEW, and App Inventor have many commonalities (e.g., similar age range, graphical blocks that snap together in a program editor, and a command palette), there are differences among them. Both LabVIEW and App Inventor have physical components that allow students to observe their programs run on a device other than the computer. Because of the associated physical devices, App Inventor and LabVIEW have a duality in that the user must design a Graphical User Interface (GUI) for the screen of the phone in addition to programming the behavior of the phone in App Inventor, and the user must ensure the robot is built appropriately for the LabVIEW program. While Scratch is intended to run on the computer only and is not associated with a physical device, it does have a similar duality as users must design the stage (or background) of their world prior to building the program.

An obvious commonality among all of these graphical programming environments is the dependence on the WIMP metaphor, which “provides ease of use but assumes dexterity of human hands to use a mouse and keyboard” (Wagner et al. 2012). Clearly, this dependence results in a failure to achieve universal usability (Wobbrock et al. 2011) and “[address] the needs of all users” (Shneiderman &

---

2 In our own state of Alabama, Gina McCarley from Lawrence County attended a University of Alabama summer teacher workshop and then encouraged her students to design an app for tracking wild hogs, which was the national winner of Samsung’s “Solver for Tomorrow” technology contest worth $125,000 (Samsung).
Plaisant 2010). Based on the ACM code of ethics (ACM 2013), it is important that these accessibility issues be addressed.

2.2 Accessibility Issues

Despite the growing popularity of IPEs as a context for introducing CS to young students, there are several challenges that remain related to extending the impact of such environments to a broader audience. Because these IPEs utilize the WIMP metaphor, motorically challenged users are unable to take advantage of them.

Understanding how users with motor impairments currently use a computer and what issues they have with a standard computer is imperative to creating a better solution to “fit the abilities” (Wobbrock et al. 2011) of these users. Harada et al. (Harada et al. 2009) performed a study to evaluate the effectiveness of a voice-controlled cursor, in which five of the participants suffered from various degrees of motor impairment. Participant number one in the Harada study was 52 years old and was afflicted with multiple sclerosis for seven years. Although he could type, he used Dragon Naturally Speaking instead because he easily tires from the arthritic pain he feels in his hands. Participant number three (age 20) suffered from muscular dystrophy since birth. Muscular dystrophy causes “progressive weakness and degeneration of the skeletal muscles that control movement” (NIH 2012); thus, this participant has a small range of mobility and must be in a wheelchair. Because her mobility is limited to the point that she cannot turn her hands so that her palms are face down, she has taught herself how to type with the backs of her hands. She tried using voice recognition in the past, but she was displeased with the performance. Participant number four (age 30) was diagnosed with cerebral palsy at birth. Due to her condition, she has violent spasms frequently and weakened muscles. In addition to cerebral palsy, she suffers from fibromyalgia, a “chronic condition” (Harada et al. 2009) causing the nerves to misfire constantly. Her input device of choice is a touchpad on her laptop as a keyboard and traditional mouse are tiring for her. She is also on a ventilator, which concerned the researchers since the study was regarding voice-control (although the ventilator makes some noise during operation, it did not impact the application).

Changing input devices from a traditional device to a new modality like voice is a very difficult process; both the user and the application must learn or train. The user needs to learn the grammar, and the application needs to learn the user’s accent/articulation (Begel 2005, Dai et al. 2004, Désilets et al. 2006, Harada et al. 2009, Shaik et al. 2003, Wobbrock et al. 2011). Because the traditional input devices for a computer cause extreme fatigue for individuals suffering from these conditions and they truly are not designed with their specific needs in mind, these individuals are willing to devote the extra time it takes to learn and train the applications with better accessibility features. Harada et al. (Harada et al. 2009) point out the positive attitude these users have towards taking the time to learn new tools, “a number of people with disabilities...are willing to invest more time in learning a system, especially if there are no other alternatives due to situational, monetary, or availability constraints.” Data from experiments performed by Dai et al. (2004) confirms Harada et al.’s (Harada et al. 2009) claim. Dai et al. (2004) presented a voice-controlled navigation system using grids, and the participants in the experiment found the grid-based solution to be fast, accurate, and comfortable. Conversely, Wobbrock et al. (2011) argue that the “burden of change” should fall on the system rather than the user, and designers should
focus on ability-based design resulting in adaptive systems (discussed further in Section 4). Similar to Wobbrock et al. (2007), Gibson (2007) complains that “assistive technology is not always able to interpret the user interaction model” implying that the system is not adapting to the user appropriately. Until individuals with impairments become the application developers, “[a]ccommodating diverse human perceptual, cognitive, and motor abilities [will be] a challenge to every designer” (Shneiderman & Plaisant 2010).

Many existing IPEs (e.g., Scratch, LabVIEW, and App Inventor) were not built with a VUI in mind; therefore, adding a VUI to these tools is complicated. Despite the need for “out of the box” solutions to lower barriers for those with impairments (Wobbrock et al. 2011), an “out of the box” voice recognition system (e.g., Dragon Naturally Speaking) will not integrate with these legacy applications without a bridge to connect the voice recognition system to the legacy application. This bridge needs to connect vocal commands to those the user sees on the screen. Moreover, the voice recognition system must be able to accommodate the behavior of the application, which is a far more difficult process. There is a significant difference in adding a “click” or “drag and drop” action than an action to “drag and drop” one block within another.

2.3 Programming by Voice

Programming by voice (PBV) is not a radical concept. Prior efforts, exemplified by (Begel 2005, Dai et al. 2004, Désilets et al. 2006, Harada et al. 2009, Shaik et al. 2003), present some form of a voice-driven application. Begel (2005) and Désilets et al. (2006) discussed voice-controlled applications for textual programming, whereas Dai et al. (2004) and Harada et al. (2009) focused on voice-driven cursor control techniques. 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 is that code and natural language are far different, and algorithms had to be created to provide the required flexibility a user would need. Begel (2005) provided an illustrative example that highlights this challenge within the context of verbalizing a common loop structure:

\[
\text{for (int i = 0; i < 10; i++)} \\
\]

If a programmer were to speak this statement aloud, it might be spoken as:

“for int i equals zero i less than ten i plus plus”

As a first effort, this appears to be a reasonable way to state the loop structure, and it is quite probable that every programmer says this in his/her head while typing a “for” loop. Unfortunately, this statement has two problems: 1. the brackets and punctuation are not dictated, and 2. some of these terms are not usual for a typical natural language processor. Instead of the above verbalization, Begel (2005) explains that the voice recognition engine may understand the words spoken as:

“4 int eye equals 0 aye less then ten i plus plus”
Although Begel solves some of the issues, particularly the punctuation, there is still some responsibility placed on the programmer to learn the structure and terminology. Désilets et al. (2006) discovered the same issue and solved it in a similar manner. There is a proper grammar the user must learn; however, this same grammar translates to any programming language. Désilets et al. (2006) also interjected code navigation and error correction abilities. In the SpeechClipse tool, Shaik et al. (Shaik et al. 2003) utilized a rule-based grammar in coordination with Java’s Robot class to translate the commands recognized by the speech recognition engine into activities performed by the keyboard. In their approach, as the user speaks, the mouse or keyboard performs actions based on the commands (i.e., the mouse and keyboard are programmatically controlled by the Java Robot class, which is driven by parsed words spoken by the user).

Rather than understand code, 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 where on the screen he/she wishes to click. The grid begins as the entire screen is divided into nine sections. The user selects in which of those nine segments he/she wishes to click by verbalizing the number. That portion of the screen is then partitioned into nine sections, and again the user verbalizes the section in which he/she wishes to click. This process repeats three times until the specific item of interest is identified. Dai et al. (2004) conducted a small study that compared whether a grid-based system works best with nine cursors or with one cursor; the difference being that with nine cursors, the user makes three verbal clicks, and with the one cursor solution, the user makes four verbal clicks. The nine cursor solution was determined the fastest method.

The specific approach explored by Harada et al. (Harada et al. 2009) used a very different method for cursor control. Rather than use words such as “move right, move right, stop,” they identified ten syllable sounds that are mapped to direction and speed (Figure 6). The syllables are selected based on the International Phonetic Alphabet (IPA). Of the ten syllable sounds, eight are vowel sounds, and they “were chosen because they represent the eight most distinct sounds in the vowel map periphery that are present in as many of the world’s major languages as possible” (Harada et al. 2009). A disadvantage of using the syllable mapping is that it is not instinctive and some syllables are difficult to pronounce at first; therefore, it requires some training. The actual tool, called the Vocal Joystick, was developed with the goal of user-friendliness. It provides multiple methods of feedback for the user, which is particularly helpful in the training sessions. If a certain phonetic sound is difficult for a user to pronounce, the system can be modified for that user. Additionally, it is built in such a way that applications can be
developed around it such as VoiceDraw, which is supported by Wobbrock et al. (2011) as meeting three of their seven guidelines: adaptation, performance, and commodity. The most important aspect of the Vocal Joystick, and thus VoiceDraw, is the ability to adapt to the user rather than the user having to adapt to the system.

Martin (1989) researched two claims regarding the validity of speech as an input device: 1) Speech is faster than typing; and 2) Speech increases productivity. Martin’s literature search proved claim number two and some of the research validated claim number one, but with some uncertainty. Martin created an experiment to test each claim by having users navigate a graphical application (MAGIC). The resulting data proved 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 and providing an “additional user-response modality” (Martin 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 document collected ideas resulted in a larger quantity of higher quality ideas as compared to typing the ideas collected in a group setting. 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 (Martin 1989, Jung et al. 2007, Hauptmann and Rudnicky 1990), voice is not only viable, but it has the potential to allow the user to be more efficient.

2.4 Voice Limitations
Several papers in the area of PBV (Begel 2005, Dai et al. 2004, Désilets et al. 2006, Harada et al. 2009, Shaik et al. 2003) demonstrate that voice can be a reasonable alternative to the mouse and keyboard. However, an undesirable consequence is the potential for vocal strain. Haxer et al. (2001) reported a case study of a doctoral candidate/lecturer who began experiencing pain due to tendonitis and decided to utilize a voice recognition program instead of her keyboard. She used voice recognition approximately four to six hours daily. Due to the heavy usage, she began to experience vocal strain and vocal fatigue. She then went to the Multidisciplinary Voice Clinic of the University of Michigan Vocal Health Center for evaluation. After performing several tests, it was determined that her vocal quality and sound were below average. She then started speech therapy where she was taught proper vocal techniques such as warming-up her vocal chords before speaking for long periods of time and hydrating periodically during vocal use. After learning how to properly exercise her vocal chords, she began using voice recognition again for four to six hours daily, and she no longer experienced vocal strain or fatigue. Moreover, she was re-evaluated, and her vocal quality and sound had improved to normal levels. De Korte and Van Lingen (2006) performed a study to determine if voice recognition in the workplace could improve posture and productivity and how user friendly this technology was perceived among participants in the study. While voice recognition did improve posture, most participants found they were more productive, and most participants found the voice recognition to be user friendly; five of the 15 participants complained about sore throats as a result of the frequent speaking.
Another limitation of voice as an input is speech recognition errors. Oviatt (2000) mentions that errors primarily occur when a user alters his or her voice and articulates words differently than when the user trained the speech recognition system. Oviatt (2000) continues to discuss that users’ articulation alters with changes in emotional states, the surrounding noise, and the tasks the user is performing at the time. She proposes using multimodal input to reduce these errors and to increase the efficiency of correcting speech recognition errors. A multimodal approach is not appropriate for the work presented in this proposal; however, by understanding the primary reason for errors occurring, users can work to speak consistently, thereby reducing the error rate.

3. Current State
Currently, research goal one (Section 1.3) is almost complete. After conducting a user study and making final changes to the application, goal one will be complete. More detail regarding the current state of Myna is presented below in Sections 3.1 (general introduction to Myna) and 3.2 (mapping challenges discovered), and the pilot study is described in detail in Section 3.3.

3.1 Myna
The PBV research summarized in Section 2 indicates that voice is a viable modality for providing accessibility for users with motor impairments. A tool affectionately named Myna, which is a species of birds that are well-known for their imitative skills, was created by previous students of Dr. Gray at the University of Alabama Birmingham (Rudraraju et.al 2011). Myna has been extended in this proposal as a Java program that runs parallel to Scratch and utilizes Sphinx (CMU Sphinx 2013) as a voice recognition system to allow for further portability. When Myna is started and Scratch is opened, the user can begin creating a program within Scratch solely through voice. The grammar for the user is relatively simple as the verbal commands match the commands on the screen with the exception of select action commands (e.g., “drop after”). A prototype was created and populated with a small vocabulary to perform basic functionality (Rudraraju et.al, 2011). The grammar was then extended to include all commands native to the program. Myna development continues at UA with deep extensions to the grammar and capabilities related to navigation. Clients of UCP will soon be working with us to evaluate the application and determine how feasible it is for someone with Cerebral Palsy to work with a voice-driven IPE. Stevens and Edwards (1996) present an approach for designing a solid evaluation for assistive technology, which will be consulted during our evaluation design process. The future interactions with UCP require a fully functional application rather than a prototype.

There are three types of navigation in Myna (Rudraraju et.al 2011, Wagner et al. 2012):

1. **Drag and Drop Navigation:** This mimics the idea of clicking on an object, dragging it to another location, and dropping it. The user will say, “drag and drop” followed by the command block they wish to add to the program, and the block will be placed after the last block in the program.
2. **Continuous Navigation:** The user drives the cursor by stating, “move right” and “keep moving,” which contradicts the research in (Harada et al. 2009, Wobbrock et al. 2011); therefore, this might be an area for improvement.
3. **Transparent Frame Navigation**: The transparent frames allow small numbers to be placed next to commands within the program (see Figure 7). The user will state “drag” followed by the command on the desired block, and upon determining where the user wishes to place the block, he/she will state one of three macro commands (“drop before,” “drop in,” or “drop after”) and the number from the label.

The primary design goal for Myna was to avoid invasive modification of the Scratch source code (e.g., the actual source code of Scratch itself). Myna is a separate entity that runs parallel to Scratch, as a type of monitor that sits on top of Scratch to process the voice-driven commands. This allows Myna to be flexible in that it can be applied to other IPEs such as LabVIEW or App Inventor. This generalized capability will be discussed further in Section 4.3.

![Image of prototype and use of transparent frames.](image)

As the user creates a program in Scratch using Myna, the mouse moves without the need for any physical activity from the user. This is due to the Java Robot class executing the mouse and keyboard events translated by the speech recognition engine. SpeechClipse (Shaik et al. 2003) was also developed using this approach. Myna allows the user to have full control over the interface with just his/her voice. The following is an example of how Myna maps the voice commands to actions (refer to Figure 8) (Rudraraju et.al 2011, Wagner et.al 2012, Wagner 2013):

1. **User gives a voice command.**
2. **The input command is identified by the speech recognizer and checked against the grammar file.**
3. **If the command is present in the grammar file, an appropriate action is invoked in the Command Executor.**
4. **The Command Executor obtains the current mappings of the component.**
5. **If necessary, the Command Executor requests the Model to change its current state.**
6. **The Command Executor calls into the Java Robot [class] to perform the corresponding mouse/keyboard action.**
3.2 Challenges and Solutions

The following are six of the issues we encountered during the mapping process; our proposed solutions are also presented.

1. **Parameters – horizontal expansion**: Various commands (e.g., “go to x y”) in Scratch require multiple parameters (Figure 9a). 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 4b)

   **Solution**: The program knows the distance from the beginning of the command block to each parameter slot. As the user enters information vocally, the program must capture the information to measure the length. For example, if the user submits the number “100” in the first parameter of the block in Figure 9b, the program takes a constant (the length of one character), 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 9c), a similar process is used. The variable’s name is stored as the user enters the variable name one letter at a time via voice. Based on the name, Myna calculates the length of the variable name by summing a constant number of pixels per character entered and adds the edges surrounding the string in the variable block, which is consistent for all variables.

2. **Delete**: Scratch does not contain a “delete” command. Instead, the user drags the block to be deleted from the code editor to the command palette. If the desired block is in the middle of a code segment (Figure 10), 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 proposed solution, while more difficult to implement for the programmer, is easier on the end-user. The user will state, “Delete” followed by the number of the block to be deleted (Figure 11). Using the Java Robot class and information regarding the location of the blocks on the screen, Myna will separate the blocks within the code segment if needed and drag 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 11. Deleting with Myna

3. **Scrolling**: There are two issues with scroll bars within Scratch: 1. Four different scroll bars exist (Figure 12): editor vertical, editor horizontal, palette vertical, and palette horizontal; and 2. How does Myna know when the end of a scroll bar is reached? If a user were to say, “scroll down,” Myna needs to know which scroll bar to which the user is referring. Moreover, where is the end of the scroll bar? For example, the user can specify which scroll bar and navigate down, but if the user mistakenly states, “scroll down,” and the scroll bar has already reached its end, what should happen? Because Myna has been developed without modifying the Scratch code, Myna cannot read from Scratch that the end of the scroll bar has been reached.

Solution: The first issue is simple. The user should specify which scroll bar to use by stating the area followed by specifying either vertical or horizontal and then the direction (e.g., “editor horizontal right”). The distance scrolled is more challenging. When clicking on the arrow of a scroll bar, the
distance scrolled is consistent. When the user commands “editor vertical down,” the Java Robot will allow for the mouse to click on the appropriate scroll arrow, and the xy-coordinate of all blocks in the corresponding area (e.g., code editor) will be updated to match the amount scrolled. In order to track the length of a scroll bar, Myna must know the resolution of the screen. The length (or width, if discussing a horizontal scroll bar) of the scroll bar increases a constant amount as needed. For example, as the user adds code to the editor, Myna knows the drop location of the block. If the drop location exceeds the resolution of the screen, the scroll bar will be increased. This increase is constant. By tracking this length, if the user reaches the end of a scroll bar and tries to go beyond the end, Myna can inform the user that the end has been reached.

![Scratch with scroll bars](image)

**Figure 12.** Scratch can display up to four scroll bars simultaneously.

4. **Dynamic GUI changes:** Scratch allows users to create custom variables and lists. This menu begins with two buttons (Figure 13a), but when the user adds a variable or a list, the variable or list is added to the menu along with associated commands (Figure 13b). The user can add as many variables or lists as desired; however, with each additional variable or list, the xy-location of the items listed below it will shift (Figure 8c). Also, the user is responsible for naming new variables and lists, which must be stored for future use.

**Solution:** Regarding the naming issue, when the user creates a new variable or list, Myna will create a property file (this is how all other command information is stored) for this item and give it an xy-coordinate location. Myna must track the number of custom variables and lists created in order to calculate the xy-coordinate of the next new item (Figure 13c). The commands are equidistant in the command palette; therefore, this is a simple calculation. The difficult part is ensuring all information
is stored appropriately, and as new items are added, the necessary property files are updated with the new location.

5. **Resolution**: Because the VUI depends heavily on the xy-coordinates of the pixels on the screen, the resolution is critical to the functionality of Myna. A temporary solution is to use 1024x768 since it is the most widely used resolution (StatCounter 2012). All xy-coordinates are based on this resolution, and users are asked to change their resolution to 1024x768. Wobbrock et al. (2011) explain that systems should adapt to the user rather than the user adapting to the system; therefore, this is an unacceptable solution.

   **Solution**: Myna will first make a call to the OS to determine the current resolution; this call must be OS independent (e.g., Windows, Mac, Linux). All of the commands will still be mapped to the 1024x768 resolution, but Myna will calculate the xy-coordinate in the user’s resolution relative to the xy-coordinate in the 1024x768 resolution. This new value will be stored in the property file for the corresponding command.

6. **Pause**: This is a non-native command primarily because there is no need for it within Scratch. 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; 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. See Figure 14 for an image of the “pause” dialog window.

![Pause dialog window.](image)

All of the issues described in this section will be implemented in Myna by July 2013. The VUI was evaluated in a pilot study (see Section 3.3) and reviewed by three UCP clients. The three clients at UCP found Myna to be very interesting and appropriate for their lives. One gentleman in particular was unable to speak but used a laptop to state the Myna commands, and the digitized speech was successful. He really liked the idea of being able to use Myna in the future. Next, a user study will be performed during the Summer of 2013 (see Section 4.2.2). The usability of the resulting VUI will be determined and any necessary changes will be made from a formative assessment. The primary goal during this testing will be for the VUI to be transparent to the user (i.e., the VUI should be clear, concise, and intuitive to the user).

3.3 Pilot Study

Although we want the VUI to function as well as the GUI, the goal is not to compare the two to each other, but to provide similar functionality requiring the participants in this study to use both tools. The type of study performed was a one-group post-test only approach (see Figure 15). 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 study presented here is a pilot study intended to evaluate the study design and instruments used, but also to evaluate the current state of Myna. Five CS graduate students (native and non-native English speakers, all male) participated in this IRB approved pilot study.

![Study design.](image)

The following hypotheses will be evaluated:

$H_0$: 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 (i.e., 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 want to determine if Myna meets the ability-based design criteria defined by Wobbrock et al. [12].

3.3.1 Testing Procedures
The participants evaluated Myna using a laptop and microphone, where each participant was given a script with three programs to write. First, the participants created each program in Scratch using mouse/keyboard input. The reasoning behind having the participants use the mouse/keyboard method first 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 more or less time; and
2. To allow the participants to get a brief understanding of how Scratch works since the goal is to evaluate Myna’s usability rather than Scratch’s usability.

Next, the 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 felt about using Myna.

3.3.2 Data Collection
Two data collection methodologies were utilized: observation and questionnaire. 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 questionnaire that appeared to contradict observed behavior (e.g., the participant appeared frustrated but commented otherwise on the questionnaire), the participant was asked for explanation, which was also recorded.

The questionnaire asked more specific questions regarding the user’s experience during the experiment. The questions asked the user how he felt using specific features of Myna (e.g., the solutions presented in Section 3.2), 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.

3.3.3 Results
The results from the observation are summarized in Table 1. Survey results are summarized throughout the discussion (please see Appendix A).

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. These are the occurrences 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 syntax 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, they are not. To avoid these errors, more training must occur. Type D errors are Myna functionality issues, and need to be corrected before further user testing.

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

Table 1. Summary of observation data.

3.3.4 Data Analysis
We want to show that Myna is a suitable alternative to mouse/keyboard interaction; therefore, we must reject the null hypotheses. 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 16). The result of the t-test is 0.323347 with an alpha of 0.05; therefore, we cannot reject the null hypotheses for $H_1$, which means that the time difference is statistically insignificant.

For $H_2$, both quantitative and qualitative data was analyzed. First, the total number of errors due to the participant stating the incorrect Myna command (Type A error) were summed and charted (see Figure 17). Then, the qualitative data provided by the participants was analyzed. 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; and
2. I perceive the degree to which Myna is easy to learn as ....

The result was 0.513, which is substantially higher than alpha = 0.05. We may not have a large enough sample to adequately perform this test (having only five participants), but it will be investigated further 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 Section 3.2. User feedback was positive for all but Parameters (the median was calculated and resulted in 4s and 5s except for Parameters, which received a 3); therefore, $H_3$ can neither be supported nor rejected.

Errors of type C, speech recognition errors, have the highest occurrence. Two of the five 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 nonequivalent variance, one-tailed t-test with a result of 0.061, which is greater than alpha = 0.05. While the result of the t-test is not statistically significant, the result is not insignificant.

<table>
<thead>
<tr>
<th>Errors per person</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors per native speaker</td>
<td>4</td>
</tr>
<tr>
<td>Errors per non-native speaker</td>
<td>21.5</td>
</tr>
</tbody>
</table>

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

Program 2 seemed to have the fewest speech recognition errors for non-native English speakers, but the most for speech recognition errors for native English speakers (see Figure 18). This highlights that word pronunciation differs from person to person and further demonstrates the needs for a custom grammar wizard, which would allow users to customize the grammar to better match his/her needs.

![Figure 18. Sum of speech recognition errors per program by speaker type.](image)

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 et al.’s (2011) definition.

1. Ability is defined as focusing on what users can do. Myna does this by utilizing speech rather than dexterous movement. To understand if users felt it met their abilities, we analyzed the Learning Myna category of the survey by looking at the median responses (see Figure 14 - Ability). The participants had a median response of 4/5 for each question implying they felt “able” to use Myna.

2. Accountability states that the system should change, not the user. The solutions described in Section 3.2 are an example of how we have adapted the system to meet user needs. The median response (see Figure 19 - Accountability) varied between 3/5 and 5/5. 3/5 is lower than what we want to see. The Parameters scored the 3/5, which matches with the number of errors observed (7% of errors, one, were caused by Parameter issues). The ability to address issues related to parameters needs more work before this design goal can be met.

3. Adaptation states that interfaces should be user-adaptable. At this time, Myna does not have the wizard we intend to build; therefore, there is no data to evaluate.
4. Transparency is defined as giving users awareness of the application. We reviewed questions regarding how users were provided information (see Figure 19 - Transparency). Only the category of error messages received below a 4 based on user feedback. The errors for which there are existing error messages built into the system were not encountered during testing (e.g., there were no errors starting the microphone); however, the users did experience errors for which there were no messages (e.g., speech recognition errors). Further development is necessary to provide more feedback to the user (e.g., a dialog window displaying the utterance Myna understood from 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.

5. The performance category requires that the system monitors user interaction and then predicts user needs in the future. Although 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. Figure 19 illustrates the median responses to questions on the topic of performance. Both of these questions received a 4/5; therefore, the commands are 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.

7. Commodity is defined as being low-cost. This is one of Wobbrock et al.’s (2011) biggest requirements because individuals with impairments already have high costs associated with the impairment. Myna is a free tool and other than a computer, requires only a microphone (the cost of which depends on the user’s preference).

![Figure 19. Graphs based on survey responses.](image-url)
Based on the above, 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 Adaptation (e.g., grammar wizard mentioned previously) and Context.

3.3.5 Discussion
In the first program, Myna averaged 15.6 seconds less time than the mouse/keyboard; however, in programs 2 and 3, 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 took less time to complete the program in Myna (please see Appendix B for full observations and Figure 16 for a graphical representation of the time). Based on the small increase in time on average and the 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.

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 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 3 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. The longitudinal study to be performed during the Summer of 2013 should reflect different results due to increased training.

We found it important that user performance improve over time. Figure 8 illustrates that Type A errors and Type D errors (both indicative of Myna performance) decrease over time. As previously mentioned, Type C errors do not meet this goal; however, the chart does show that there was a decrease from Program 1 to Program 3, but it is an inconsistent trend. Type B errors occurred infrequently, but they are important to note. With more training, these errors should disappear.

Overall, the participants felt Myna was “somewhat easy” to learn. The participants also felt “satisfied” after using Myna and were not frustrated during the study. Additionally, 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.

3.3.6 Threats to Validity
The primary threat to validity is the small participant group; five participants is a very small sample size. It should be noted that this is a preliminary study with the purpose of learning if the study design was adequate for the current state of Myna. During the Summer of 2013, a second study will be performed with a small group of motor impaired children. The 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. 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) (Cozby 2008).

**Internal Validity**

By asking each participant to create all of the programs in Scratch using the mouse/keyboard and then in Myna using voice, there could be a threat to the validity of the time in Myna. Since the participant would know how to complete the program, this 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 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 causing a learning factor in both scenarios.

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

**External Validity**

Initial testing was conducted with graduate students instead of the target population. Although members of the target population (three UCP clients) aided in the design of Myna, testing was conducted with graduate students, specifically Engineering graduate students, in order to get technical feedback while testing usability. Adjustments will be made to Myna based on participant feedback, and a longitudinal study with participants from the target population will be conducted during the Summer of 2013.

3.3.7 Lessons Learned

There were two goals for conducting this pilot study. First, we wanted to evaluate the usability of Myna, particularly the effectiveness of our solutions to the proposed challenges discussed in Section 3.2. Second, we wanted to evaluate the study procedure itself. The target audience is motorically impaired children, and the study should be as seamless as possible.

We learned of a few changes that need to be made to Myna in order to improve the functionality; however, the three more interesting lessons learned apply to the latter goal. The following lessons are in regards to conducting further testing prior to the formal study.

1. Users were first asked to complete three programs using a mouse/keyboard. User testing was performed on a laptop with a touchpad for a mouse rather than a traditional mouse. We noticed several students were unfamiliar with the touchpad and in the future, user testing should be performed with a traditional mouse.
2. In attempting to demonstrate how easy Myna is to learn, the participants received a two minute introduction. Although the participants performed well and experienced few vocal command errors, participants commented that a short, experimental session prior to writing the first program would be helpful. This would give the participants a few minutes to see how the commands work before being timed. Adding this experimental session may introduce a new
threat to validity because participants will have had more time with Myna; however, it will eliminate the current threat where participants have not had enough time with Myna.

3. In reviewing the observational data, it would be helpful to have access to the programs the participants created as well as the console output. In future testing, we will save each program and the console output to allow for more thorough analysis.

4. Proposed Work

Although there are potential advantages for the “Programming By Voice” idea to help children learn more about Computer Science, it is costly to customize each separate environment individually. The effort to provide PBV for Scratch is likely not possible to be reused for a similar purpose on some other environment, such as App Inventor or Mindstorms LabVIEW. This leads to Goal 3 (Section 1.3), Generalization.

It is not realistic to create a new environment for every IPE that exists, or will emerge in the future. Furthermore, every time the arrangement of GUI elements changes in a new version of the IPE\(^3\), the hardcoded dependencies often require much effort to re-tool the voice commands. Section 4.1 presents Goal 2, the expansion of Myna, and Section 4.2 describes evaluating Myna. Section 4.3 describes two concepts that could work together to semi-automate the voice integration process and potentially reduce the cost of creating voice-controlled solutions for existing IPEs.

4.1 Expansion of Myna

Myna will be the model from which future IPEs will be integrated with voice controls. The following are the required tasks that need to be completed in order to have a fully functional PBV framework that can be instantiated for specific IPEs.

Grammar Customization: Users may utilize a specific function more frequently and desire to change the command for that function. To achieve this functionality, a wizard will be created to allow users to customize the grammar. In addition to creating shortcuts, this will allow users to change words that they find challenging to articulate, thus, improving the adaptability of Myna (Wobbrock et al. 2011).

Multi-lingual Support for Code Dictation: Almost all computer programming languages have reserved words that are derived from the English vocabulary (e.g., “if” “for” “begin”). Correspondingly, current PBV tools assume that the user will be speaking English to drive the recognition engine. Yet, many users (especially young children, who are the target group of this research) may feel more comfortable speaking in their native tongue such that the development environment translates their native expression into the corresponding code. Under the proposed future work, wizards for adapting the grammar to match a specific spoken language will be provided. This is not proposed as a major contribution, but such adaptability can make the resulting tools more amenable to those with disabilities who are not native English speakers. United Cerebral Palsy will help to identify a participant who is a non-native speaker (e.g., a child who is fluent in Spanish) in order to evaluate the effectiveness of multi-lingual support in IPEs.

\(^{3}\) MIT released Scratch 2.0 in May 2013, which is a browser-based version of Scratch and has an updated interface.
4.2 Evaluation
The proposed research consists of multiple phases, and each phase will require a different user study. Three user studies are proposed and presented below.

4.2.1 Pilot Study
The initial pilot study was completed as discussed in the previous section; however, some features, particularly the challenges described in Section 3.2, are still being evaluated with CS graduate students before beginning the formal user study.

4.2.2 User Study
This is the primary user study to evaluate Myna. This study will occur during Summer 2013, and participants will consist of approximately ten 15-16 year old students taking part in a summer camp with United Cerebral Palsy. These participants will be the target audience (i.e., these participants will have motor impairments) for evaluation of Myna. Unlike the pilot study, these participants will only use Myna. A general introduction to Scratch will be presented, and participants will receive a printout of the Myna commands. Participants will then work one-on-one (a camp counselor or therapist will also be present) with a researcher for 30 minutes once per week for 5-6 weeks. This study will have a post-test design and will attempt to determine if errors lessen over time via participant learning and increased comfort with the vocal commands. I will observe the participants’ interactions and document any observed errors. Additionally, I will ask a few general questions regarding the participants’ experience at the end of each session (Please see Appendices C and D for the instruments to be used).

After completing the study, any necessary changes will be made to Myna before it is finalized and published to the web.

4.2.3 Peer Study
This final user study will be conducted during Spring 2014. The purpose of this study is to evaluate the resulting automation tool (please see Section 4.3). The participants of this study will once again be CS graduate students since the target audience will be users with some software engineering knowledge. Participants will use the VUI generalization tool to create a VUI for a specific IPE. I will observe each participant using the VUI generalization tool one-on-one, and the participants will complete a short experience questionnaire. The data will be analyzed and any necessary changes will be made to the automation tool before finalization and publication.

4.2.3 Sample Execution
One goal of the generalization tool is to have the capability to recreate the VUI when GUIs are updated and modified. As a trial, we will use the generalization tool to recreate a VUI manually for Scratch 2.0 (please see footnote 3). By performing this trial, we will be able to evaluate the VUI creation process empirically by analyzing several aspects (e.g., the time the process takes, the amount of effort in lines of code the user must modify, the accuracy of the end result) and compare these aspects to those required to create the VUI manually.
4.3 Planned Technique

After Myna becomes a more mature application (such as meeting the guidelines set by (Wobbrock et al. 2011)), the next phase will be to create a semi-automatic means by which voice can be integrated into other applications, such as LabVIEW and App Inventor. Figure 20 represents a possible workflow of the semi-automation process; the three steps are outlined below.

**Step 1 - Importing the Static Structure of the GUI:** A first step in speech-enabling a GUI is to understand all of the widgets that are of interest and must be clickable from voice 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 and Sasikumar 2008, Zettlemoyer and 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). The approach that will be explored in the proposed work is a semi-automated approach based on screenshots of the IPE to conserve the effort needed to map the GUI fully.

Five tools providing some of the required functionality have been reviewed.

1. **Tesseract (Smith 2007):** Tesseract, an open-source Optical Character Recognition (OCR) tool developed by HP between 1984 and 1995, was developed to improve the accuracy of HP’s scanners. In 1995, Tesseract proved to be a better OCR engine than commercial engines in the
1995 Annual Test of OCR Accuracy (Smith 2007). Although it has not been updated in over a decade, Tesseract is a useful tool for capturing text; however, its accuracy could be improved significantly by using a Hidden-Markov-Model-based character n-gram model (Smith 2007).

2. VisMap (Zettlemoyer and St. Amant 1999): VisMap allows for programmatic control of an application through the GUI, and it builds a “structured representation of interface objects” by utilizing display input and image processing algorithms (Zettlemoyer and St. Amant 1999).

3. Sikuli (Yeh et al. 2009, Chang et al. 2010, Chang 2011): Sikuli Search allows users to search documentation databases using an image rather than text as the search item, and Sikuli Script allows the user to automate GUI elements on the screen by editing a Python script based on a screenshot of the application (Yet et al. 2009). Sikuli Script allows quality assurance testers to utilize images when writing scripts to test the application; the testers write the appropriate script, and Sikuli Script returns whether or not the automation performed according to the requirements (Chang et al. 2010).

4. Prefab (Dixon and Fogarty 2010): Prefab reverse engineers the pixels of an application to evaluate the types of elements on the screen to allow for the addition of advanced functionality to any application. The primary work presented by Dixon and Fogarty (2010) compares element types found on the screen to that of a prototype library to determine the type of element found. Prefab searches the screen, creates hypotheses about the elements on the screen, and then tests the hypotheses. Dixon and Fogarty (2010) state that Prefab can reverse engineer non-standard interfaces that were not created using a toolkit in addition to those interfaces that were created using toolkits.

5. PAX (Chang et al. 2011): PAX is a framework that combines pixels with accessibility APIs to gather all metadata about an object. This framework improves the accuracy and the quality of the data collected about an object when combined with a pixel-based analysis tool (e.g. Sikuli) (Chang et al. 2011).

As previously mentioned, the metadata required for a semi-automated VUI creation tool are 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). This metadata should be collected without interfering with the underlying code of the application for which a VUI is being created. For example, when creating Myna, the Scratch code was never touched. When creating the new VUI for Scratch 2.0 (Section 4.2.3), the underlying code should again be ignored. This is the primary requirement for this system; therefore, unlike Sikuli, Tesseract, and Prefab, VisMap and PAX are not viable tools as both require access to the application’s API. According to Chang et al. (2011), OCR algorithms are better suited for scanning black text on white backgrounds, and Chang et al. (2011) states, “[u]sing current OCR algorithms on screen text would generate poor results.” Thus, Tesseract is not the best tool since it is an OCR engine; however, in the event that none of these tools demonstrate the required functionalities, the algorithms utilized in Tesseract might be beneficial. Additionally, if a custom tool is developed, the PAX framework will be analyzed to determine which pieces, if any, can be utilized to improve the accuracy of the screen analysis (Chang et al. 2011).
Prefab and Sikuli Script are the best choices of these five (Sikuli Search provides a different functionality entirely as it is only used for searching). Both Prefab and Sikuli Script are based on the previous work of Zettlemoyer and St. Amant (Yeh et al. 2009, Chang et al. 2010, Chang 2011, Dixon and Fogarty 2010); however, the primary improvement is that neither Sikuli Script nor Prefab require access to the application’s API. Prefab differs from Sikuli Script as it finds all occurrences of all widgets on a screen, and it does so significantly faster than Sikuli (Dixon and Fogarty 2010); however, Prefab only finds common widgets. The applications for which VUIs are being proposed in this research do not use typical widgets (e.g., buttons, checkboxes, textboxes) that most application development toolkits (e.g., Visual Studio, NetBeans) use (Dixon and Fogarty 2010). Sikuli Script identifies a component on the screen via a screen shot of that component (provided by the user), and then, that component can be automated through the user-defined script (Yeh et al. 2009, Chang 2011). Based on our previously defined requirements, neither application returns the information required; however Sikuli Script might provide some benefits. Instead of learning the location and dimensions of components, perhaps, the components can be identified by their image.

Although each of these tools provides functionality similar to what is needed for this research, none of them fully meet the defined requirements; therefore, after further research, a custom tool might need to be developed. The previous work discussed in this section may be beneficial in creating that tool.

**Step 2 - Specifying the Dynamic Execution Behavior of the IPE:** 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. There are two possible solutions:

1. The use of a modeling language to capture such interactions. Domain-specific software environments (Gray et al. 2007) are emerging as a powerful tool 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 (Gray et al. 2007). Molina et al. (2012), Mohagheghi et al. (2012), and Mohan and Kulkarni (2009) present various perspectives on the use of MDE in industry. Molina et al. (2012) present the use of a specific tool (CIAT-GUI), while Mohagheghi et al. (2012) and Mohan and Kulkarni (2009) present model-driven approaches to software development. CIAT-GUI is an example of a MDE software tool, which uses a variety of tools to ensure the MDE approach is compatible with the applications with which the developers were already working. The approach used in Mohan and Kulkarni (2009) varies slightly as they utilized UML to build the models and metamodels. This is a different approach than using tools such as EMF, which was used in Molina et al. (2012) and Mohagheghi et al. (2012) to create the models and metamodels.

For the purpose of Step 2, a customized modeling language that is based on the semantics of abstract state machines might be designed to allow the VUI developer to specify the dynamic relationships between screenshots and states within the execution of the IPE.
2. In our proposed approach, tool developers will move through the static screens and modes of an IPE and select the GUI components that are of interest in the IPE. For example, the “repeat” block in Scratch could be selected from the list of controls and mapped to an internal representation. Using the screen scraping tool described in Step 1, the developer will click through the screen and identify the necessary metadata for each GUI component (e.g., xy-coordinate, size, name, type). This screen scraping tool should then output the collected information for use in the property files of the VUI and the names added to the grammar. The property files and grammar might be generated using a DSL; however, the information collected via the screen scraping tool might be simple enough to not need a DSL.

Step 3 - Code Generation of Grammars and Programmatic Control Code: After the static structure and dynamic behavior of the IPE have been obtained, the grammar needed by the speech recognition system can be generated as well as the programmatic control of the mouse and keyboard through the Java Robot Class. The proposed research will investigate code generation based on the inputs provided by the VUI developer. A metric for evaluation will be the level of completeness of code that can be generated based on the static and dynamic descriptions (this metric will be evaluated in the sample execution described in Section 4.2.3).

5 Projected Schedule
Section 4 described the proposed work in this project, and the proposed schedule for completing these tasks is below (please see Figure 21 for a Gantt chart for the proposed schedule).

- Spring 2013: Pilot study, Data analysis, and Adjustments based on study
- Summer 2013: Expansion (Goal #2), UCP user study, Screen scraping research, Automation research
- Fall 2013: Data analysis, Adjustments based on study, Create automated tool
- Spring 2014: Complete automated tool, User study, Data analysis, Adjustments based on study
Figure 21. Gantt Chart for proposed task completion.

All necessary work may be completed by the end of Spring 2014 in order to work on final publications and the dissertation during the Summer of 2014.

6 Conclusion


Although there are numerous methodologies and tools to choose from, this proposal focused on Scratch, LabVIEW, and App Inventor as example IPEs. Scratch is useful for students of all ages, but has found a special niche for upper elementary to middle school students, while Berkeley’s BYOB extension of Scratch as well as MIT’s new Scratch release are more appropriate for high school and college students due to their support for parameterized abstraction. LabVIEW is also very appropriate for upper elementary to middle school students. App Inventor can be more complicated and is recommended for high school and college students. These tools, when presented at the appropriate age, can spark a student’s interest in Computer Science (Wolber 2011, Rodger et al. 2009). However, due to the lack of the required dexterity, students with motor impairments are unable to use IPEs such as Scratch, LabVIEW, and App Inventor.

Motorically challenged students cannot use these applications due to the dependence most IPEs have on the WIMP metaphor. Wobbrock et al. (2011) encourages ability-based design and supports voice-driven applications if they are adaptive, perform well, and are cost effective. Research goals were
presented in Section 1.3 to meet these requirements: goal one results in the creation of a tool, Myna, that will allow these users to take advantage of IPEs, user studies will evaluate how well Myna performs, goal two includes an adaptability feature (i.e., the grammar customization wizard), and through adoption of open source tools or tools that are standard on many platforms, Myna will be cost effective. A small user study was already conducted. The results demonstrate that Myna is a viable tool as the users were “satisfied” after using Myna, and the time to complete the programs using Myna was not significantly longer than the time taken to complete the programs via mouse/keyboard.

The concept of a voice-driven IPE is an excellent tool, but can suffer from a static implementation strategy that is focused on hardcoded adaptation for each IPE that is considered. It is time consuming to fully integrate voice into each new IPE. We believe that the realization of the vision illustrated in Figure 20 (goal three) would provide a capability to create voice-controlled alternatives for GUI applications in a way that reduces or eliminates the amount of hardcoded dependencies (i.e., the intelligence of the code generator can use the configuration information of the static and dynamic information to produce what was previously implemented as a fixed-point solution). The planned techniques described in Section 4.3 aim to realize the vision of Figure 20. By utilizing screen scraping and OCR-based tools, we will be able to capture the text and location of components on the screen. Then, DSLs will generate the necessary Java code for the corresponding and required property files and grammar files. These ideas working together in addition to the target user study will drive the focus of the next phase of this doctoral research, leading to a dissertation describing this work.

Acknowledgements

I would like to sincerely thank my advisor, Dr. Jeff Gray, and my committee members: Dr. Marcus Brown, Dr. Sandra Nichols, Dr. Randy Smith, and Dr. Eugene Syriani for their guidance through this process.

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


Chang, T., Yeh, T., and Miller, R. Associating the visual representation of user interfaces with their internal structures and metadata. In *Proceedings of the 24th annual ACM symposium on User Interface Software and Technology*, New York, NY, October 2011, pgs. 245-256.


### Question

#### General Commands
- I feel the vocal commands in Myna are relevant to the action being performed. 4 5 4 4 5
- I feel the commands in Myna perform the actions that I expect. 3 4 4 3 4
- I feel the vocal commands in Myna are. 4 5 5 4 4

#### Specific Features
- The "Delete" command works as expected in Myna. 2 4 5 2 5
- Parameters are easy to edit in Myna. 3 1 3 2 4
- It is easy to pause Myna to stop listening to my voice. 5 4 5 5 5
- It is easy to resume Myna to listen to my voice. 5 5 5 5 5

#### Usability
- I think error messages in Myna are. 1 4 5 2 NA
- I think help messages in Myna are. 5 5 4 4 NA
- I feel tasks can be performed in a straight-forward manner in Myna. 3 4 4 4 4
- I perceive the use of terms throughout Myna to be. 4 5 4 5 5
- I think Myna keeps me informed about what it is doing. 1 3 3 3 4
- I think supplemental reference materials for Myna would be. 4 4 5 4 4

#### Prior Knowledge
- My knowledge of Scratch is. 2 1 1 1 1
- I have used Scratch. 2 1 1 1 1
- My knowledge of Myna is. 2 1 1 1 1
- Prior to this study, I have used Myna. 1 1 1 1 1

#### Learning Myna
- I perceive the degree to which Myna is easy to learn as. 3 4 4 4 5
- I find exploring new features by trial and error in Myna to be. 1 4 4 3 5
- I find remembering names and uses of commands in Myna to be. 3 4 3 4 4

#### Overall Impression
- After using Myna, I am. 3 4 4 3 4
- Using Myna made me. 2 3 4 3 3
- After using Myna, I feel. 3 4 4 4 3

#### Demographic Information
- Gender: Male Male Male Male Male
- Race: White White Asian Asian White
- Native Speaker: Yes Yes No No Yes
### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>101-1</th>
<th>101-2</th>
<th>101-3</th>
<th>102-1</th>
<th>102-2</th>
<th>102-3</th>
<th>103-1</th>
<th>103-2</th>
<th>103-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant error: incorrect vocal command</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Participant error: incorrect Scratch command</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Speech recognition error</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Myna error: xy-location incorrect</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Time to complete with mouse/keyboard</td>
<td>92</td>
<td>55</td>
<td>85</td>
<td>77</td>
<td>107</td>
<td>69</td>
<td>185</td>
<td>86</td>
<td>102</td>
</tr>
<tr>
<td>Time to complete with voice</td>
<td>105</td>
<td>95</td>
<td>139</td>
<td>53</td>
<td>84</td>
<td>87</td>
<td>134</td>
<td>130</td>
<td>154</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>104-1</th>
<th>104-2</th>
<th>104-3</th>
<th>105-1</th>
<th>105-2</th>
<th>105-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant error: incorrect vocal command</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Participant error: incorrect Scratch command</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Speech recognition error</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Myna error: xy-location incorrect</td>
<td>21</td>
<td>2</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time to complete with mouse/keyboard</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Time to complete with voice</td>
<td>160</td>
<td>121</td>
<td>121</td>
<td>57</td>
<td>47</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>141</td>
<td>131</td>
<td>109</td>
<td>60</td>
<td>128</td>
<td>75</td>
</tr>
</tbody>
</table>
Myna Observation Instrument
This instrument will be used to record each observation during user testing.

Participant ID: ________________________________________

Date: _________________________________________________

Observer: _____________________________________________

1. Number of participant errors: _____
   a. Participant stated the incorrect vocal command (“Drag” vs. “Drag and Drop”): _____
   b. Participant stated the incorrect Scratch command (“Go to x” vs. “Go to xy”): _____

2. Number of Myna errors: _____
   a. Voice recognition was inaccurate (Participant said the correct commands, but Myna reacted incorrectly): _____
   b. The xy-location on the screen was incorrect: _____

3. Time to complete experiment using mouse/keyboard: _______

4. Time to complete experiment using voice: _______

5. Note any issues the participant experiences.
Myna Satisfaction Survey

This instrument is to be answered by the participant of the UCP study but completed by the observer.

Participant ID: ________________________________________

Date: _______________________________________________

Observer: ____________________________________________

1. Which face describes how you felt using Myna?

![Emoticons]

2. Did you enjoy creating your own animations with Myna?

   Yes   No

3. Would you use Myna again?

   Yes   No

4. Is there anything else you would like to say about Myna?