On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science

Andreas Stefik
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL 62026-1656
stefika@gmail.com

Christopher Hundhausen
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164-2752
hundhaus@wsu.edu

Derrick Smith
Department of Education
University of Alabama in Huntsville
Huntsville, AL 35899
derrick.smith@uah.edu

ABSTRACT

The blind and visually impaired community is significantly underrepresented in computer science. Students who wish to enter the discipline must overcome significant technological and educational barriers to succeed. In an attempt to help this population, we are engaged in a three-year research project to build an educational infrastructure for blind and visually impaired middle and high school students. Our primary research goal is to begin forging a multi-sensory educational infrastructure for the blind across the United States. We present here two preliminary results from this research: 1) a new auditory programming environment called Sodbeans, a programming language called Hop, and a multi-sensory (sound and touch) curriculum, and 2) an empirical study of our first summer workshop with the blind students. Results show that students reported a significant increase in programming self-efficacy after participating in our camp.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]; K.4.2 [**Social Issues**]: Assistive Technologies for persons with disabilities

General Terms

Design, Human Factors, Experimentation

Keywords

Accessibility, Visual Impairments, Auditory Debugging, Assistive Technology

1. INTRODUCTION

Blind and visually impaired students have few pathways for entering the computing profession. Successful students and professionals in this community are predominately self taught and have often overcome significant technical and practical barriers. While much work has been dedicated to helping the blind use various computer technologies, more research is needed on finding ways to make it easier for blind users to obtain high-paying and meaningful careers. While computing is potentially appealing as a career option due to its rich sound and tactile facilities, no standardized educational infrastructure exists to help these students succeed. Indeed, with 61% of working adults (aged 16 to 64) with vision loss out of the work force [1], and with households that include a blind member having a significantly higher rate of poverty [16], creating more opportunities for this group of individuals is sorely needed.

Our research attempts to address a broad research challenge: to build a multi-sensory educational infrastructure for the blind and visually impaired across the United States, and to broaden participation from this community in the computing discipline. This is particularly challenging for at least two reasons. First, modern programming environments prove to be quite inaccessibile to the the blind and visually impaired. For example, when the screen reader JAWS® 11 is coupled with Visual Studio® 2010, no sound is generated when the user switches between tabs; a graphical window appears, but JAWS®does not speak. Further, the debugger in Visual Studio®outputs only the key pressed while stepping into or over (e.g., "F11, F11, F11, F11."). To address this problem, we have collaborated with developers at Oracle to develop Sodbeans, a computer programming environment built into NetBeans 6.9. Sodbeans includes a custom designed screen reader, a talking debugger (e.g., "a to 5"), and a custom programming language called Hop. Elements of our tools have been iteratively refined in formal empirical studies for nearly five years [15]. Second, existing programming curricula were not written with the blind and visually impaired in mind, relying heavily upon visual representations to teach key concepts. In contrast, we have opted to build a curriculum tailored to this population's unique needs. As is commonly done in schools for the blind and visually impaired, we have developed learning activities that make extensive use of tactile manipulative objects to teach computing concepts.

This paper presents two primary contributions toward the broad challenge of building an educational infrastructure for the blind and visually impaired. First, we introduce a new auditory programming environment called Sodbeans, a programming language called Hop, and multi-sensory (sound

SIGCSE'11, March 9–12, 2011, Dallas, Texas, USA.

and touch) curriculum that will be used at five schools for the blind over the next three years. Second, we present the results of an empirical study of our first summer programming camp, which introduced computer programming to blind high school students over a three-day period. Results from this preliminary study show that students reported a significant increase in programming self-efficacy through participating in the camp. Our research project constitutes the first large-scale attempt to develop an educational pipeline specifically designed for blind and visually impaired users to enter the computing discipline.

The remainder of this paper is organized as follows. We begin with related work in Section 2 and discuss our educational infrastructure in more detail in Section 3. After presenting an empirical study of our first workshop in Section 4, we present our summary and future work.

2. RELATED WORK

While there is a rich legacy of work in auditory display (using sound to convey information), and an extensive literature regarding how to make programming easier for novices, there is relatively little work on programming blind or with audio. In this section, we present related research on the following topics: 1) programming blind, 2) auditory display, and 3) novice and end-user programming. Perhaps the most well known work on programming blind was conducted by Smith et al. [13]. In this work, Smith noticed that blind students tended to have difficulty in courses like data structures, which often rely extensively on visual representations. Moreover, this work provided some pertinent observations on how blind students program. Notably, Smith coined the "Where am I?" problem, the idea that when programming blind, the need to determine one's location (e.g., in an editor, during an execution, general context of use), is crucial.

More recently, Bigham et al. used instant messaging chatbots to inspire blind individuals to enter computer science as part of the National Federation for the Blind's Youth Slam program [3]. Students used off-the-shelf-technologies (The JAWS screen reader and TextPad 4.73), which works well for basic text editing. In contrast, while our tools are compatible with JAWS and other screen readers, we have built our own programming language, compiler, debugger, and supporting auditory technologies, which aurally narrate programming sessions. Our tools are publicly available on sourceforge.net (search for Sodbeans).

Sánchez and Flores [12] created a custom programming language called APL (auditory programming language), which was designed for the blind. APL was tested with the blind population, but supplied a limited set of commands for a blind person to use, making it difficult to scale such tools to a general programming curriculum such as the one we are developing. Since our tools are based on NetBeans, students have a rich set of programming environments and tools they can use in our program (e.g., Java, PHP, Ruby), although our talking debugger is only available for our language: Hop.

The use of auditory technologies in programming is not new. Boardman [4] created LISTEN to explore mappings between source code and sound. Similarly to LISTEN, our tools use a code-to-audio mapping architecture, which is integrated with our compiler, debugger, and virtual machine. However, LISTEN focused on facilitating code-to-audio mappings, not on the design of specific applications, like blind programming environments. Brown and Hershberger [5] augmented algorithm animations in their Zeus system with "algorithm auralizations." Their musical auditory displays mapped higher-pitched tones to larger magnitude data in the algorithms being auralized. Brown and Hershberger claimed that their auditory displays assisted in the comprehension of the algorithms; however, they did not carry out any empirical evaluations to test these claims.

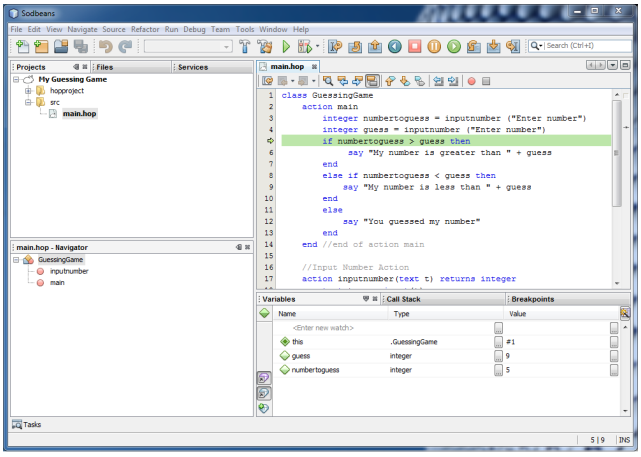

Figure 1: Snapshot of Sodbeans.

Several authors have promoted the specific use of *musical* cues as auditory representations of computer source code. Vickers [17], for example, built a program auralization system called CAITLIN, which used musical auditory cues to represent the execution of computer programs written in Pascal. In experimental studies, Vickers failed to show that participants could find more bugs with musical cues, although he claimed to have found that the effectiveness of the musical cues increased with the cyclomatic complexity of the source code. Subsequent work has shown that musical auditory cues are, in fact, difficult to learn [10].

Finally, there is a rich legacy of research into novice programming environments (see Kelleher and Pausch [7] for an excellent overview). Within this large body of work, some of the themes that resonate with our work are studying programming practices of novices using traditional languages [14], trying to find "natural" ways of programming [9], and the use of multimedia and graphics in programming [6]. Our work is similar in the sense that we have conducted formal studies on our environment as a multimedia novice environment and have completed formal studies on the efficacy of the words chosen in our programming language [15], to try to make the language easier to understand.

3. EDUCATIONAL INFRASTRUCTURE

We are working toward a *broad research challenge*: to *create* a multi-sensory educational infrastructure for the blind and visually impaired and to implement it widely, so as to *broaden* the participation of this community in the computing discipline. Our preliminary approach toward this challenge has three primary components: 1) Technology creation, 2) curriculum development, and 3) infrastructure.

3.1 Technology Creation

We are developing a programming environment for the blind called Sodbeans, shown in Figure 1. Recognizing the

Figure 2: Manipulatives for teaching programming concepts. (a) representing a variable as a box; (b) representing a numeric value as a die; (c) representing Boolean value as a light switch; (d) representing a string value as a string of beads.

difficulty of describing auditory user interfaces in written prose, a blind user interacts with our environment similarly to how they would using a screen reader, with the exception that we "take over" the screen reader, on occasion, to supplement or replace the information that it provides. For example, if a user installs Sodbeans on Windows, and that user has JAWS®installed, we ask JAWS®to speak additional words and phrases designed for our language (e.g., information about the execution of a program in the debugger or text editor). If, however, a blind user does not own a screen reader, Sodbeans has an optional, and free, drop-in screen reader replacement that starts up automatically.

In contrast to modern screen readers, this "take over" process works through what we call a *push* accessibility model. For example, when JAWS®attempts to read the screen, it *pulls* information from an accessibility API (if one exists), for that environment and attempts to aurally present information to the user. Unfortunately, such APIs are often poorly maintained by corporations, as they are not particularly flashy or profitable. In our approach, we integrate deeply into the listener architecture in NetBeans, allowing us to *push* appropriate information to a screen reader, without reliance upon such APIs. This gives us significant freedom in how we approach accessibility internally, and has made it much easier to build tools like talking debuggers.

Sodbeans includes a custom virtual machine, compiler, and debugger for the Hop programming language. To give an example of how this works, suppose the Hop auditory debugger executes the line of code `integer a = a + 1`. In this case, the debugger would say "a to 5" (or another value). Similarly, if the user executes an `if` statement, like `if a < b then end` (the `then` and `end` statements serve as a block in Hop), the debugger would say either "if true" or "if false." The auditory cues we are using, and the syntax and semantics in Hop, have been carefully selected and chosen in formal empirical studies. For example, we use the word `repeat` over `for` and `while`, or `cycle` (see e.g., Sánchez and Flores [12]), because a recent study showed that the word `repeat` represents the concept of iteration significantly better than these words [15]. The full documentation for our language and environment is available in our online wiki at `https://sourceforge.net/apps/trac/sodbeans/`.

Our previous work on the design of auditory cues includes complex statistical analysis and formal studies [15]. We offer here a condensed overview of the results of our empirical observations related to how we design our auditory cues. Generally, we think that *good* auditory cues follow a pattern similar to the following: 1) they are short, 2) they are "browsable," and 3) the most important information comes first. First, auditory users rarely want to listen to lengthy cues, and we would encourage cue designers to remove extraneous words. Second, users interacting with our auditory debugger have a tendency to press the debugger keys (e.g., step over, step into) rapidly, which means that users often hear only the first few milliseconds of an auditory cue before moving on to the next. As such, an auditory cue like "variable a set to 5" is inferior to "a to 5," as a user browsing with the former could hear "va, va, va, va, va (the first part of the word variable)," which has little meaning for browsing. We *suspect* such cues make it more difficult for users to answer Smith et al. [13]'s classic question: "Where am I?"

Finally, sometimes an auditory cue must accommodate complex information, an example of which is in the use of "meta-auditory cues." A meta-auditory cue supplements another cue. For example, NetBeans 6.9 uses visual display information called *editor hints*, which try to help a user determine information about their code (e.g., a yellow lightbulb on the left gutter of an editor, a red underline for a compiler error). This kind of information can potentially be embedded into auditory cues as well, but we have observed that such information can get in the way if presented first. For example, a cue like "integer a = 5x; compiler error" appears to be more usable in practice than "compiler error, integer a = 5x;" as the former is more browsable and gives the most pertinent information immediately, allowing a user to easily skip over the meta-auditory cue.

3.2 Computer Programming Curriculum

We are developing a custom computer programming curriculum specifically tailored for the blind and visually-impaired. In order to provide an empirical foundation for this curriculum, we conducted a field study at the Washington State School for the Blind in the summer of 2010. While there, we interviewed two instructors and observed several classes. Our observations and interviews yielded three key and unexpected requirements for a programming curriculum to be used at a school for the blind: 1) introduce concepts, whenever possible, through the use of manipulative objects that students can touch, 2) favor hands-on activities and projects over lecturing, which is virtually non-existent, and 3) since students work at varied paces and most teaching is done one-on-one, provide a rich array of projects and activities that students can work on at their own pace.

In order to address (1), we have created a suite of manipulative objects to help students learn programming concepts. For example, Figure 2 presents examples of manipulatives we have prototyped for teaching variables and variable values. In order to address (2) and (3), our curriculum stresses

hands-on activities, programming projects, and discussions over lectures. A typical class might begin with a hands-on activity involving manipulatives, in order to help students initially explore the concept being taught. Following that, students might open up Sodbeans, load some starter code, and modify it to meet a set of specifications. As students do this, one or more teachers circulates, helping students as needed. Students who work at a faster pace could move on to more advanced projects when they were ready. As students finished projects, they would be given the opportunity to "play" them to the class for feedback and discussion. At our summer 2010 programming camp (see Section 4), we piloted this style of curriculum, with encouraging results.

3.3 Infrastructure

Blind and visually impaired students at schools for the blind generally have few avenues for learning programming. While schools for the blind have expertise in working with blind children, expertise in computer programming is rare. Of our five partner schools for the blind in Washington, Texas, Indiana, Tennessee, and Massachusetts, none reports having a state-approved curriculum for blind students to learn to program, and most teach a curriculum more akin to basic computer skills (e.g., screen reader or keyboarding skills). A state-approved curriculum for middle and high school blind students to learn to program is sorely needed.

We are working with partners to build up an infrastructure for blind children to learn computing. This includes obtaining government approval in our partner states, training teachers from each school on how to program (blind), and creating a support network for all stakeholders involved. Our support network includes the use of 1) our community driven Sodbeans documentation, 2) mailing lists for the project for teachers, students, and developers, 3) a Sodbeans web portal with information on the project and how to get in touch with our team, and 4) a concerted effort to foster a community of like-minded individuals interested in improving the educational infrastructure for the blind. We encourage any group interested in this population to freely use our materials and tools or to work with us to establish an infrastructure for blind students in your local area.

4. EMPIRICAL STUDY

We conducted an empirical study as part of our first annual summer camp, which had the goal of introducing blind students to computer programming. Our empirical study associated with this camp had two research goals: RG1) to obtain insight into student interests and motivations in coming to and participating in a programming workshop, RG2) to determine whether the students felt confident about their ability to engage in computer programming (programming self-efficacy) and to determine whether our workshop might help improve students' programming self-efficacy. While the results in this section should be considered preliminary, we hope the data will provide some insight into how blind children view and consider topics like computer programming.

4.1 Participants

We recruited twelve legally blind students for our first workshop from Washington and Oregon. Students ranged from 13 to 18 years old (M=15.75). Of these students, six reported total blindness, while six reported low visual acuity. Of those with residual vision, the student with the highest visual acuity was rated at 20/300, while the lowest was approximately 20/2200-2400. Students who attended the conference were required to have previously taken at least pre-algebra, although in practice all but one had taken algebra I or higher. The student with the highest previous math experience had taken trigonometry. All students had taken courses in computing, but two students had taken courses in web site design and one had taken a course in computer programming, self reporting some experience in both python and C.

To help determine the experience level each student had with various technologies, we asked participants a series of questions regarding technological usage and experience. Students self-reported on a 1-7 Likert scale high experience with screen readers (M=5.58, SD=1.68), low experience with screen magnifiers (M = 2.833, SD=2.17), moderate experience with embossers (M = 3.875, SD = 2.02), very high experience with personal digital assistants like BrailleNotes designed for the blind (M = 6.33, SD = .99), moderate/high experience with talking books (M = 4.91, SD = 2.5), and high experience with the Perkins Brailler (M = 5.67, SD = 2.27). Students also rated themselves in the use of the Internet (M = 5.375, SD = 1.4) word processing applications (M = 5.33, SD = .89), spreadsheets (M = 2.75 ,SD = 1.49), presentations (M = 3.58, SD = 1.78), online games (M = 3.125, SD= 2.05), web design (M = 2, SD = 1.13), and computer programming (M = 1.875, SD = 1.13).

4.2 Materials and Tasks

We examined whether students with blindness or visual impairments (N =12) ratings on a measure of self-efficacy by Askar and Davenport [2] would change after exposure to a three-day camp in which students used Sodbeans. In addition, we used a scale by Pintrich et al. [11] to measure students' perceptions of 1) Task Value, 2) Self-Efficacy, 3) Critical Thinking, 4) Peer Learning, 5) Task Goal Orientation, 6) Performance—Approach, 7) Performance—Avoidance, 8) Connectedness, and 9) Learning. Students completed all of these 1- 7 point Likert scales at the beginning and conclusion of the workshop. For the Askar and Davenport scale, which was originally designed to test programming self-efficacy in Java, we adapted it for use in our language, Hop. The adaption was trivial, in that we literally replaced the word "Java" with the word "Hop" throughout. All surveys have been validated previously using standard factor analysis procedures (see e.g., Kline [8] for an introduction).

4.3 Procedure

Students arrived at the Washington State School for the blind on July 13th, 2010 in the afternoon. On arrival, students were first welcomed and parents or students signed appropriate informed consent forms. Once forms were signed, a member of our research group guided a student to a room where they were interviewed and given our surveys aurally. On the last day of the workshop, we repeated the procedure, interviewing and surveying the students a second time.

During our workshop, students participated in a series of activities. On day one, our teachers taught programming in two 1.5 hour sessions. Students in these sessions learned some of the syntax of the Hop programming language, how to use the auditory debugger, and how to write code that connects to their screen reader (e.g., a complete program in Hop to connect to a screen reader would be

`say` "`Hello, World!`"). After our two programming sessions were complete, and students had been given lunch, they were brought back to listen to one of several guest speakers. Over the course of the workshop, we brought in 1) an auditory GPS and tactile maps researcher, 2) mechanical braille researchers, and 3) two professional blind programmers to talk to the students about obtaining a career in the computing industry. In sum, we attempted to *engage* students in programming activities, to *inspire* them to consider programming, and to *encourage* them that computer programming is possible as a blind or visually impaired person.

4.4 Results

To begin our analysis, we first used an omnibus repeated measures analysis of variance (ANOVA) with Trial (i.e., pre-test scale scores and post-test scale scores) and Scale (Task Value, Pintrich Self-Efficacy, Critical Thinking, Peer Learning, Task Goal Orientation, Performance —Approach, Performance —Avoidance, Connectedness, and Learning) as the two within-subjects factors. This test tells us whether there existed statistically significant differences in our data set as a whole. This analysis revealed statistically signifiant differences in the main effects for Trial, $F(1, 11) = 4.85, p = .05, \eta_p^2 = .306$, and Scale, $F(8, 88) = 6.66, p = .001, \eta_p^2 = .377$. These main effects were qualified by a statistically significant $Trial \times Scale$ interaction, $F(8, 88) = 3.24, p = .003, \eta_p^2 = .306$. Thus, some of our metrics showed a significant change in the beginning of our workshop compared to the end, but that others did not.

In order to explore differences with respect to specific attitudinal metrics, we performed follow-up repeated-measures ANOVAs. The means for each metric appear in Table 1, divided as a function of Trial (i.e., Pre-Workshop, Post-Workshop). Note that participants' ratings showed a non-significant (unreliable) decrease across time for all scales except Connectedness, which showed an equally unreliable increase. However, the Performance —Approach and Performance —Avoidance scales showed a significant decrease. We ran a separate repeated measures ANOVA for the the Askar and Davenport self-efficacy scale, which showed a significant positive difference with a large effect size, $F(1, 11) = 35.56, p = .001, \eta_p^2 = .764$. Finally, our results continue to hold even if a Bonferroni correction is applied, with the exception of Performance—Approach (it needed to reach $p < .005$ using this correction). In English, for those of our results that showed significant differences, they can be considered reliable even by very conservative estimates.

4.5 Discussion

Our observations suggest three primary results of participation in our workshop: students self reported a 1) significant increase in the Askar-Davenport self-efficacy scale, 2) significant decrease in performance—approach (e.g., I want to do better than other students in my class), and 3) a significant decrease in performance avoidance (e.g., An important reason I do my school work is so that I don't embarrass myself). We generally consider these results to be positive, as they appear to imply that students reported both a gain in self-confidence with programming and a decrease in worrying about their performance in comparison to their peers.

Notice, however, that our two self-efficacy ratings, both of which have been validated in the literature did not achieve the same result. The Pintrich scales shows a small non-significant drop in self-efficacy, while the Askar and Davenport scale shows a positive, significant, result, with an effect size that explains nearly 76.4% of the variance in the sample—an extremely large effect. How should we interpret such results? We speculate that these metrics are measuring different attributes. Consider for example, that the questions in the Pintrich self-efficacy scale are very general (e.g., I'm confident I can understand the basic concepts taught in this course), whereas the Askar self-efficacy scale is more focused on programming skills (e.g., I could write syntactically correct Hop statements), which is of more interest in our case, since we are focused on programming and computer science. We think that future designers of self-efficacy surveys should focus their attention on the external validity of such measures, as this would give the community a better idea of the real-world applicability of each.

5. SUMMARY AND FUTURE WORK

We are engaged in the first large-scale attempt to create a multi-sensory educational infrastructure for blind and visually impaired middle high schools students, starting with five schools for the blind throughout the U.S. We have made two primary contributions in this work: 1) an environment (Sodbeans), which includes auditory debuggers, a programming language (Hop), and a multi-sensory educational curriculum using tactile manipulatives, and 2) an empirical study of our first summer workshop working with the blind and visually impaired students at the Washington State School for the Blind. We are our encouraged that, despite a small sample size and only a few days with our population, we were able to show a significant increase in programming self-efficacy, which, perhaps, could help inspire students from this community to consider learning more about computing. Over the next three years, we are expanding our program to include more schools for the blind, are making regular upgrades and expansions to our software, and are working with local and state governments to approve a multi-sensory curriculum that can be put into practice.

6. ACKNOWLEDGMENTS

We would like to thank the contributors to the Sodbeans project for their help with creating our technology, including Melissa Stefik, Neelima Samsani, Andrew Hauck, Susanna Siebert, Kim Slattery, and Sina Bahram. Sarah Hawkinson and Jason Neufeld contributed to the design of the programming curriculum. We would also like to thank the Washington State School for the Blind, especially Dean Stenehjam and Sherry Haun. This work would also not have been possible without the extraordinary technical expertise of Tim Boudreau (Oracle) and Tom Wheeler (A NetBeans Platform expert). This work was funded by the National Science Foundation under grant no. (CNS-0940521).

7. REFERENCES

[1] American Foundation for the Blind. Interpreting BLS Employment Data. retrieved august 31, 2010, from `http://www.afb.org/Section.asp?SectionID=15&SubTopicID=177`, 2010.

[2] P. Askar and D. Davenport. An investigation of factors related to self efficacy for java programming among engineering students. *The Turkish Journal of Educational Technology*, 8(1):26–32, 2009.

Test	Example Item	pre	post	F	p	η_p^2
Task Value	I think I will be able to use what I learn in this course in other courses.	5.81 (1.08)	5.54 (1.23)	0.83	.38	0.70
Pintrich - Self-Efficacy	I'm confident I can understand the basic concepts taught in this course.	5.54 (.896)	5.42 (1.15)	0.27	0.61	0.102
Critical Thinking	I treat the course material as a starting point and try to develop my own ideas about it.	5.18 (1.05)	4.83 (1.29)	1.25	0.29	0.102
Peer Learning	When studying for this course, I often try to explain the material to a classmate or a friend.	4.56 (1.35)	4.33 (1.41)	0.53	0.48	0.46
Task Goals	I like school work that I'll learn from even if I make a lot of mistakes.	5.68 (.876)	5.57 (1.23)	0.46	0.51	0.40
Performance Approach	I want to do better than other students in my class.	4.50 (1.11)	3.57 (1.44)	11.09	0.007	0.502**
Performance Avoidance	An important reason I do my school work is so that I don't embarrass myself.	4.15 (1.30)	3.06 (1.44)	15.85	0.002	0.590**
Connectedness	I feel connected to others in this course.	4.86 (1.15)	5.08 (1.22)	0.66	0.44	0.056
Learning	I feel that I am encouraged to ask questions.	5.38 (.486)	5.30 (.689)	0.11	0.75	0.010
Askar - Self-Efficacy	I could write syntactically correct Hop statements.	3.178 (1.10)	4.04 (.884)	35.56	< 0.001	0.764 ***

Table 1: Summary table of the statistical results. All but the last metrics are from the Pintrich scale [11], with one example question for each factor. The last row gives an example question from the Askar-Davenport self-efficacy scale [2]. In the columns labeled pre and post, the first number is the mean and the second is the standard deviation. * = p < .05, ** = p < .01, *** = p < .001

[3] J. P. Bigham, M. B. Aller, J. T. Brudvik, J. O. Leung, L. A. Yazzolino, and R. E. Ladner. Inspiring blind high school students to pursue computer science with instant messaging chatbots. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 449–453, New York, NY, USA, 2008. ACM.

[4] D. B. Boardman, G. Greene, V. Khandelwal, and A. P. Mathur. Listen: A tool to investigate the use of sound for the analysis of program behavior. In *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, pages 184–189, Dallas, TX, 1995.

[5] M. H. Brown and J. Hershberger. Colour and sound in algorithm animation. In *Proceedings of the IEEE Workshop on Visual Languages*, pages 10–17, Los Alamitos, CA, 1991. IEEE Computer Society Press.

[6] C. D. Hundhausen, S. F. Farley, and J. L. Brown. Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study. *ACM Trans. Comput.-Hum. Interact.*, 16(3):1–40, 2009.

[7] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, 2005.

[8] P. Kline. *An Easy Guide to Factor Analysis*. Routledge, New York, NY, 2002.

[9] B. A. Myers, A. J. Ko, S. Y. Park, J. Stylos, T. D. LaToza, and J. Beaton. More natural end-user software engineering. In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 30–34, New York, NY, USA, 2008. ACM.

[10] D. K. Palladino and B. N. Walker. Learning rates for auditory menus enhanced with spearcons versus earcons. In *Proceedings of the 13th International Conference on Auditory Display*, pages 274–279, Montréal, Canada, june 2007.

[11] D. Pintrich, D. Smith, T. Garcia, and W. McKeachie. A manual for the use of the motivated strategies for learning questionnaire (technical report no. ncriptal-91-b-004). Technical report, National Center for Research to Improve Postsecondary Teaching and Learning, Ann Arbor, MI, 1991.

[12] J. Sánchez and F. Aguayo. Blind learners programming through audio. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1769–1772, New York, NY, USA, 2005. ACM.

[13] A. C. Smith, J. S. Cook, J. M. Francioni, A. Hossain, M. Anwar, and M. F. Rahman. Nonvisual tool for navigating hierarchical structures. In *The Tenth International ACM SIGACCESS Conference on Computers and Accessibility*, number 77-78, pages 133–139, New York, NY, 2004. ACM Press.

[14] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs: an empirical study. *Communications of the ACM*, 26(11):853–860, 1983.

[15] A. Stefik and E. Gellenbeck. Empirical studies on programming language stimuli. *Software Quality Journal*, pages 1–35, 2010. 10.1007/s11219-010-9106-7.

[16] U.S. Census Bureau. Disability and American Families: 2000. retrieved may 4, 2009, from `http://www.census.gov/prod/2005pubs/censr-23.pdf`, july 2005.

[17] P. Vickers and J. L. Alty. When bugs sing. *Interacting with Computers*, 14(6):793–819, 2002.