An Exploratory Study of Blind Software Developers

Sean Mealin and Emerson Murphy-Hill
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
{spmealin,emerson}@csc.ncsu.edu

***Abstract*—As a research community, we currently know very little about the challenges faced by blind software developers. Without knowing what those challenges are, the community cannot effectively address these challenges. In this paper, we describe the first exploratory empirical study, where we conducted eight interviews with blind software developers to identify aspects of software development that are a challenge. Our results suggest that visually impaired software developers face challenges, for instance, when using screen readers to look up information when writing code. We discuss a variety of implications, including that blind software developers need additional support in discovering relevant software development tools.**

I. Introduction

In the United States and around the world, there is a severe shortage of software developers to design, build, and maintain the software that is increasingly critical to our day to day lives [4]. At the same time, current software developers are a largely homogeneous group [1], and diversifying this group could yield significant benefits to society.

People who are blind or visually impaired may be good candidates to fill this shortage and diversify the software development workforce. Since software is not an inherently visible artifact, it would seem that a lack of sight would not put blind software developers at a disadvantage [9]. Blind people's use of screen readers, programs that verbalize text on the screen, suggests that the textual source code used to implement most software should be accessible to blind developers. In fact, blind developers' compensatory abilities, such as enhanced serial memory [5], may give them significant advantages over sighted developers.

At the same time, even a cursory look at modern software development makes it apparent that blind software developers may also face significant challenges. For instance, use of visual languages and notations, such as the Unified Modeling Language, do not appear to be readily accessible to blind developers. As another example, the graphical user interfaces used in integrated development environments (IDEs) may also be a challenge for blind software developers.

Some research has been done in the past to create assistive tools to aid blind people in software development. One is SODBeans, an implementation of the NetBeans IDE created specifically for blind developers [8]. Another is WAD, a debugger for Visual Studio that enables blind software developers to understand dynamic program behavior [7]. The APL programming language was designed specifically for blind software developers [6]. TeDUB allows blind software developers to use diagrams [2].

Despite the interest in creating tools for blind software developers, to our knowledge, there are no empirical studies that systematically characterize existing blind software developers. We contribute the first study that does so. In understanding the challenges and opportunities faced by current software developers who are blind, we envision researchers, educators, and toolsmiths enabling blind people to build the software of the future. This paper is a small step towards that vision.

II. The Study

A. Research questions

To better understand blind software developers, we sought to answer four research questions:

- **RQ1** What tools do blind software developers use?
- **RQ2** What practices do blind software developers use?
- **RQ3** How do blind software developers collaborate with other software developers?
- **RQ4** What attitudes do blind software developers hold about software development?

B. Methodology

To answer our research questions, we conducted qualitative interviews. We chose qualitative interviews as a research method in an attempt to identify as many areas of future research as possible by enabling interviewees to elaborate on specific topics when they felt it was necessary. For each research question, we asked interviewees to answer a set of detailed questions. Depending on each interviewee's response, we elicited more detailed answers. The first author, who is studying software development as a university undergraduate and who also is blind, conducted the interviews via Skype. The interviews ranged from just under one hour to approximately two hours; interviewees controlled the pace of the interview. The full interview script can be found here: http://www4.ncsu.edu/~spmealin/DevStudy/Interview_Script.docx.

We recruited interviewees who are legally blind and actively engaged in software development. We recruited interviewees by sending out a solicitation on the Program-L mailing list, which serves blind software developers. Some recipients also forwarded our solicitation to other mailing lists, as well as their friends.

978-1-4673-0853-3/12/$31.00

TABLE I
DEMOGRAPHICS OF PARTICIPANTS.

ID	Exp.	Degree of Blindness	Programming Languages
P1	21	Basic shape & color ident.	C#, Jaws Script, Python
P2	12	No sight	Assembler, C, Groovy, Java
P3	40	No sight	C++, Fortran 77
P4	10	No sight	Python
P5	25	Basic shape & color ident.	Assembler, C, C++, Python
P6	20	Basic shape & color ident., and large text	Action Script, C, Java, Javascript, Perl
P7	13	Basic color ident.	C#, Javascript, PHP, Ruby
P8	31	Basic shape & color ident.	C#, T-SQL, Visual Fox Pro

C. Demographics

We attracted eight people to participate; demographics are shown in Table I. Each participant is identified with an anonymous **ID**. The **Exp.** column indicates how many years of experience each interviewee had in programming. The **Degree of Blindness** indicates how much usable sight participants reported. **Programming Languages** indicates the major programming languages used by each participant. All interviewees are male. Seven were engaged in developing commercial software, whereas one developed free software. All participants were located in the continental United States, each in a different state. P4 was an undergraduate student while developing software as part of his job. We therefore considered all interviewees to be software developers, with varying levels of experience.

III. RESULTS

A. RQ1: Tools

Accessibility Tools. Interviewees used several accessible technologies to access the software that they need to use for their development tasks. All interviewees with the exception of P7 primarily use the commercial screen-reader Jaws for Windows, and P1, P4 and P8 secondarily use the open-source screen reader Non Visual Desktop Access. Only P7 uses Non Visual Desktop Access as a primary screen reader. When looking at code, P2, P3, P7 and P8 reported using a refreshable Braille display, which is a device that converts a single line of text on screen to a single line of Braille that can then be felt with the fingers. P7 uses a Braille display because it makes it easier to do parentheses matching. Interviewee P2 said he used the Braille display when looking at hexadecimal output; he reported the reason was because screen readers typically try to pronounce adjacent letters as a single word. For instance, screen readers make it difficult to distinguish between the hexadecimal numbers 'EF' and 'EFF' because they sound identical when spoken aloud.

Editors and IDEs. When writing code, interviewees most commonly reported using text editors, while a few use integrated development environments (IDEs). In terms of editors, P1, P2, P4 and P7 reported primarily using Notepad or another text editor; for example, P7 reported using Notepad++, P4 uses Notepad, and interviewees P1 and P2 use EdSharp, a feature-rich editor specifically created for the blind. In terms of IDEs, P6 uses Eclipse, both P3 and P7 use Visual Studio, P8 uses Visual FoxPro, and P5 uses the IDE Code Warrior. Some interviewees reported using different editors or IDEs, depending on their development needs. For example, P2 primarily uses a text editor, but has used Eclipse in the past when he has worked with Java, and P7 spends his time both in Notepad++ and Visual Studio. Interviewees who said they tended to use alternatives to mainstream IDEs cited lack of accessibility in those environments as the motivation; however, the fact that five interviewees use IDEs for some tasks suggests that at least specific portions of IDEs are usable. These results motivate existing research into making IDEs more accessible [8].

Other Tools. Apart from editors and IDEs, interviewees reported sparse usage of other programming tools. For instance, we asked all participants about static analysis tool usage, but only P5 responded positively; apparently most interviewees were simply not aware of them. We also asked the last six interviewees (P1, P3, P4, P6, P7 and P8) about their use of debuggers. P3 and P6 used basic debugger functionality such as breakpoints and stepping through code, while all six interviewees used "printf debugging." Interviewees mentioned that accessibility of debugging tools could be a significant barrier to use. P7 gave the example of FireBug, a javascript debugger, as an example of an inaccessible tool; he used javascript alert boxes as his preferred "printf" debugging tool. These results motivate existing research into accessible debugging tools [7].

B. RQ2: Practices

Getting an Overview. We were curious about how blind developers get a high-level overview of their code, because a typical mechanism employed by sighted developers, code beacons [10], would seemingly be unavailable to blind developers. This is because screen readers work in a highly linear fashion, forcing blind users to read through an entire document. For instance, depending on the language, finding the start of blocks of code may not be as easy without reading through all of the lines in the file.

To get an overview of code, all interviewees except P8 indicated that they rely heavily on API documentation. By reading the documentation, they are able to get an overview of the available methods and the structure of the code without delving into implementation details. Interviewees that work with C-like languages reported that reading header files is similarly helpful. P1, P2, P7 and P8 said that using the *find* tool in their editor is another way of getting structural information without having to read every line. However, they said that it is not as helpful as good documentation because finding keywords to jump to can be difficult. For example, jumping to the `public` keyword in a Java source file will help find all of the methods that are public, but it will not show methods that are `private` or declared with package-level access.

Information Seeking. We were interested in how blind developers look up information when writing code. For instance, when a sighted developer needs to look up an identifier, they have several options such as scrolling the editor to view the identifier. When it is found, they can start typing, because the location of the cursor has not changed. Users of screen readers are not able to do the same thing; it is not possible to look

through the code without moving the cursor as well. In order to find something such as the spelling of a variable's identifier, it is necessary to move the cursor to where the variable is declared so the screen reader will verbalize the information. Once the information has been obtained, a screen reader user must then move the cursor back to where they were editing.

To help alleviate this problem, all interviewees except P1 indicated that they regularly have a temporary text buffer open to write notes, such as method or variable names. By frequently inserting identifiers into this buffer, interviewees reported being able to quickly reference those identifiers without losing their place in the code that they were editing.

Interestingly, P3, P4, P6, P7 and P8 also reported using such buffers for out-of-context editing. These interviewees said that copying a block of code to a buffer and then editing it before copying it back was preferable to editing it in-place. They did this because using a separate window for the snippet made it easier to jump to the beginning or end of the code block; if they were to try this in the original source file, they would just jump to the beginning or end of the file and completely lose context. In this way, out-of-context editing serves as a mechanism to predictably constrain cursor movement.

C. RQ3: Collaboration

Teams. P2, P5, P6, P7 and P8 reported on working in teams. When working with teammates, P7 reported working on separate computers and verbally communicating via line numbers in source code that the team shared using a version control system. He reported several positive experiences working in a team, such as the ability to bounce ideas off of coworkers, to distribute knowledge about the software, and to have sighted coworkers be able to inspect the visual elements of the software he writes, such as the user interface to his web application. P8 reported interacting with other developers at meetings, where he usually asks for any material discussed at the meeting to be sent to him ahead of time. However, he reported infrequent interaction with his teammates. One difficulty he faces when interacting with teammates on the same computer is that when sighted developers watch him work with a screen reader, it embarrasses him that he cannot accomplish some tasks as fast as they can.

Diagrams. We asked interviewees to relate their experiences with communicating UML and other visual notations with their teammates. P7 noted that whenever coworkers use UML diagrams, he asks them to verbalize the content of those diagrams, such as method names and class names. If he needs to keep track of this information, he reported that he writes it down in a textual format, a practice he was satisfied with. P5 said that he did not have problems with diagrams because the same information was usually present elsewhere, such as in the documentation. P8 said that he was occasionally handed a UML diagram to implement, but found them inaccessible. P3 and P8 encountered UML diagrams at work, but avoided using them, while P1, P2 and p4 did not encounter visual diagrams at all. P6 found that diagrams that illustrated complicated systems try to communicate so much information to a viewer that some ambiguity due to interpretation is always present. He forced developers on the team that he led to explain diagrams in "simple English," which he thought is a lot more explicit and reduced errors due to interpretation.

D. RQ4: Attitudes

Career. Interviewees reported ending up as software developers through a variety of career paths, though most said that their blindness played some role in their career choice. P2 and P4 said that they became interested in development after using a note taker designed specifically for the blind that happened to come with an interpreter for the basic language. Interviewee P1 said that he started to write software because he could not find accessible software that would help him accomplish his tasks. P7 thought that software development would be a relatively accessible field that he could turn into a career. P3 moved to software development after he found that his vision was not enough for him to do tasks that he felt was crucial to the research process, such as sharing data with colleagues on a blackboard. P8 started as an architect, but was encouraged by friends to look into software development; he regarded this as fortunate, because losing the remainder of his vision would have made tasks such as drawing or computer aided design difficult. Overall, interviewees reported finding their work motivating and rewarding for reasons such as pride in creating something new and overcoming their disability.

Challenges. We also asked interviewees about their attitudes towards their disability, specifically whether it put them at a disadvantage with respect to software development. A recurring theme was difficulty with development tasks with a visual component, such as program architecture and user interface layouts. P3 and P4 said that finding mathematical formulas in an accessible form is another disadvantage, because most formulas on the web are posted as images that cannot be used with a screen reader. Interviewee P5 complained about the inaccessibility of visual languages such as LabVIEW. P8 said that he felt that he was at a disadvantage because his sighted teammates are able to navigate and use software with a mouse much faster than he is able to using a keyboard. When asked specifically about user interface design tasks, such as the use of Cascading Style Sheets, nearly all of the interviewees said that they felt that they are not able to do them as efficiently or confidently as sighted developers.

Opportunities. When we asked the interviewees about their advantages versus sighted colleagues, interviewees P3, P5, P6, P7 and P8 were able to give a software development subject that they feel like they excel in, P1 and P4 did not know, and P2 felt that he does not have any comparative advantages. P7 and P8 felt that they do well with tasks that are not easy to visualize, such as algorithm design. They said that they are able to create a detailed mental model of the code that helps them understand it. P6 said that he has an advantage because he can read extremely quickly with his screen reader; this is useful, for instance, for reading through documentation. He also feels that he excels in program architecture, an aspect of development that other interviewees try to avoid.

IV. Discussion

Multi-line Braille Displays. We found that some interviewees use a Braille display quite heavily while writing code. Braille displays come in multiple sizes, but most are limited to displaying a single line of text at once. While this is perfectly acceptable for tasks such as reading documents, where it is common to move forward sequentially, working with software may be a task where multi-line displays are beneficial. For instance, since one interviewee used a single-line Braille display to match parentheses on a single line of code, it seems reasonable that a multi-line display would help blind developers match brackets across multiple lines of code. Multi-line Braille displays could also help blind developers navigate code structures more quickly than they can with a screen reader by helping them "feel" the shape of code.

Tool Use and Discovery. We were somewhat surprised about the general lack of software development tool usage by blind software developers; we posit that this may be because the learning curve for software development tools is different for blind developers than for sighted developers. Sighted developers can expand their tool repertoire by exploring the graphical affordances of their IDEs, such as by clicking on toolbar icons that they notice. Blind developers do not have this luxury, both because screen readers cannot verbalize graphical icons, and because a sighted developer cannot notice tools on the periphery of their work because screen readers focus the developer on the location of her cursor. Alternative mechanisms for helping blind software developers learn about tools that they do not know about may be beneficial, such as command recommender systems [3].

In retrospect, one kind of tool that may have been beneficial to interviewees is code navigation tools. For example, in the Eclipse IDE, there is a command that moves the cursor back to the last source code that was edited; this command may be useful when blind developers use the cursor and screen reader to look up relevant information, but then need to navigate back to the last edited text position. It may be that interviewees were either not aware of this tool, or that it does not completely solve their navigational challenges. Further research is necessary to determine whether such existing navigational tools would be useful for blind developers, or if novel tools are needed; such research could likely take the form of observational studies of blind developers performing program editing tasks.

Education. Education may be another fruitful avenue to increase tool usage among future blind software developers. Our limited experience has been that blind computer science students are excused from learning aspects of software development that are not readily accessible. Instead, a better approach would be to equip students with the tools they need to efficiently complete tasks that are traditionally difficult for blind software developers, such as code navigation tools.

Language Choice. We were surprised that Python is such a popular programming language among interviewees. As some interviewees pointed out, white space is sometimes difficult to use when using a screen reader, yet whitespace is critical to the semantics of Python. When asked why he uses Python as a primary language, P4 said that Python has features that more than make up for any inconvenience that indentation may cause. For example, due to Python being an interpreted language, he is able to dynamically inspect and alter a program's objects in the Python interpreter. He believes that being able to do that is much more useful than using debuggers of compiled languages, such as C. Because the Python interpreter is text based, he said that almost everything is accessible through his screen reader.

V. Conclusion and Future Work

In a series of interviews, we have explored the challenges faced by blind software developers. Some results were expected, such as the inaccessibility of IDEs, but others were not, such as the frequent use of out-of-context editing. We have discussed several implications, but future work is still needed to quantify the extent to which our conclusions generalize and to understand the causes of some of our observations. For example, interviewees hinted about some of the tasks that they may be especially skilled at because of their blindness, but it was apparent to us that interviewees had not spent a lot of time reflecting on what they themselves are good at. A different type of study may further illuminate this particular subject; for instance, observational studies or interviews with blind developers' colleagues may help expose the unique skills that blind people bring to the software development process.

Acknowledgments

The authors would like to thank those who agreed to be interviewed, and our anonymous reviewers for their suggestions.

References

[1] M. Fowler. The developer world has serious issues with diversity. http://java.dzone.com/articles/developer-world-has-serious, Jan. 2012.

[2] A. King, P. Blenkhorn, D. Crombie, S. Dijkstra, G. Evans, and J. Wood. Presenting UML software engineering diagrams to blind people. In *Computers Helping People with Special Needs*, volume 3118 of *Lecture Notes in Computer Science*, pages 626–626. Springer, 2004.

[3] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. CommunityCommands: Command recommendations for software applications. In *Proceedings of UIST '09*, pages 193–202, 2009.

[4] C. Nickisch. For software developers, a bounty of opportunity. News Story, National Public Radio, Sept. 2011.

[5] N. Raz, E. Striem, G. Pundak, T. Orlov, and E. Zohary. Superior serial memory in the blind: a case of cognitive compensatory adjustment. In *Current Biology*, volume 17, pages 1129–33, 2007.

[6] J. Sánchez and F. Aguayo. Blind learners programming through audio. In *Proceedings of CHI '05*, pages 1769–1772, 2005.

[7] A. Stefik, R. Alexander, R. Patterson, and J. Brown. WAD: A feasibility study using the wicked audio debugger. In *Proceedings of ICPC'07*, pages 69–80, June 2007.

[8] A. Stefik, A. Haywood, S. Mansoor, B. Dunda, and D. Garcia. Sodbeans. In *Proceedings of ICPC'09*, pages 293–294, May 2009.

[9] T. D. Sterling, M. Lichstein, F. Scarpino, and D. Stuebing. Professional computer work for the blind. *Commun. ACM*, 7(4):228–230, Apr. 1964.

[10] A. von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28:44–55, 1995.