Taming User-Interface Heterogeneity with Uniform Overlays for Blind Users

Utku Uckun
uuckun@cs.stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Rohan Tumkur Suresh
rtumkursures@cs.stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Md Javedul Ferdous
mferd002@odu.edu
Old Dominion University
Norfolk, Virginia, USA

Xiaojun Bi
xiaojun@cs.stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

IV Ramakrishnan
ram@cs.stonybrook.edu
Stony Brook University
Stony Brook, New York, USA

Vikas Ashok
vganjigu@cs.odu.edu
Old Dominion University
Norfolk, Virginia, USA

ABSTRACT

For many blind users, interaction with computer applications using screen reader assistive technology is a frustrating and time-consuming affair, mostly due to the complexity and heterogeneity of applications' user interfaces. An interview study revealed that many applications do not adequately convey their interface structure and controls to blind screen reader users, thereby placing additional burden on these users to acquire this knowledge on their own. This is often an arduous and tedious learning process given the one-dimensional navigation paradigm of screen readers. Moreover, blind users have to repeat this learning process multiple times, i.e., once for each application, since applications differ in their interface designs and implementations. In this paper, we propose a novel *push-based* approach to make non-visual computer interaction *easy*, *efficient*, and *uniform* across different applications. The key idea is to make screen reader interaction 'structure-agnostic', by automatically identifying and extracting all application controls and then instantly 'pushing' these controls on demand to the blind user via a custom overlay dashboard interface. Such a custom overlay facilitates uniform and efficient screen reader navigation across all applications. A user study showed significant improvement in user satisfaction and interaction efficiency with our approach compared to a state-of-the-art screen reader.

CCS CONCEPTS

• **Human-centered computing** → **Accessibility technologies**; **Empirical studies in accessibility**.

KEYWORDS

Desktop Usability, Accessibility, Screen Reader, Visual Impairments, Uniform Interaction

UMAP '22, July 4–7, 2022, Barcelona, Spain

ACM ISBN 978-1-4503-9207-5/22/07…$15.00
https://doi.org/10.1145/3503252.3531317

ACM Reference Format:
Utku Uckun, Rohan Tumkur Suresh, Md Javedul Ferdous, Xiaojun Bi, IV Ramakrishnan, and Vikas Ashok. 2022. Taming User-Interface Heterogeneity with Uniform Overlays for Blind Users . In *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization (UMAP '22), July 4–7, 2022, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3503252.3531317

1 INTRODUCTION

The ubiquitous use of computing applications in professional settings coupled with the advancement of assistive technologies have significantly improved the employment prospects for people with visual disabilities [4, 42]. Many such employment opportunities require expertise in desktop productivity tools such as the Office suite [7]. However, interaction with these applications is frustrating and tedious for blind users, as the graphical user interfaces (GUI) of these applications are primarily intended for sighted interaction [8, 37].

Blind users rely predominantly on screen reader assistive technology to interact with computing applications. A screen reader narrates the contents of an application and also enables a blind user to serially navigate the application content and controls (e.g., *File*, *Edit*, *View*, and other *Ribbon* options) via an assortment of keyboard shortcuts (e.g., *TAB* for next GUI element). This one-dimensional mode of interaction supported by a screen reader does not gel with the two dimensional GUI of most desktop applications which are typically designed to facilitate convenient 'point-and-click' interaction with a computer mouse or touchpad. As a consequence, blind screen reader users have to expend significantly more time and effort compared to their sighted peers to do the same tasks such as accessing application controls [29]. Furthermore, as the underlying implementations of GUIs are likely to vary between applications and operating system platforms, blind users also face the cognitive burden of remembering different navigation strategies for different applications (e.g., VLC media player vs. Windows Media Player) or even the same application on different platforms (e.g., Microsoft Excel on Windows vs. Microsoft Excel on MacOS).

Existing solutions to improve interaction efficiency and experience for blind screen reader users have predominantly focused on web and mobile applications [6, 25, 27, 30], whereas desktop applications have been largely overlooked. To the best of our knowledge, no general-purpose solution currently exists that enables convenient

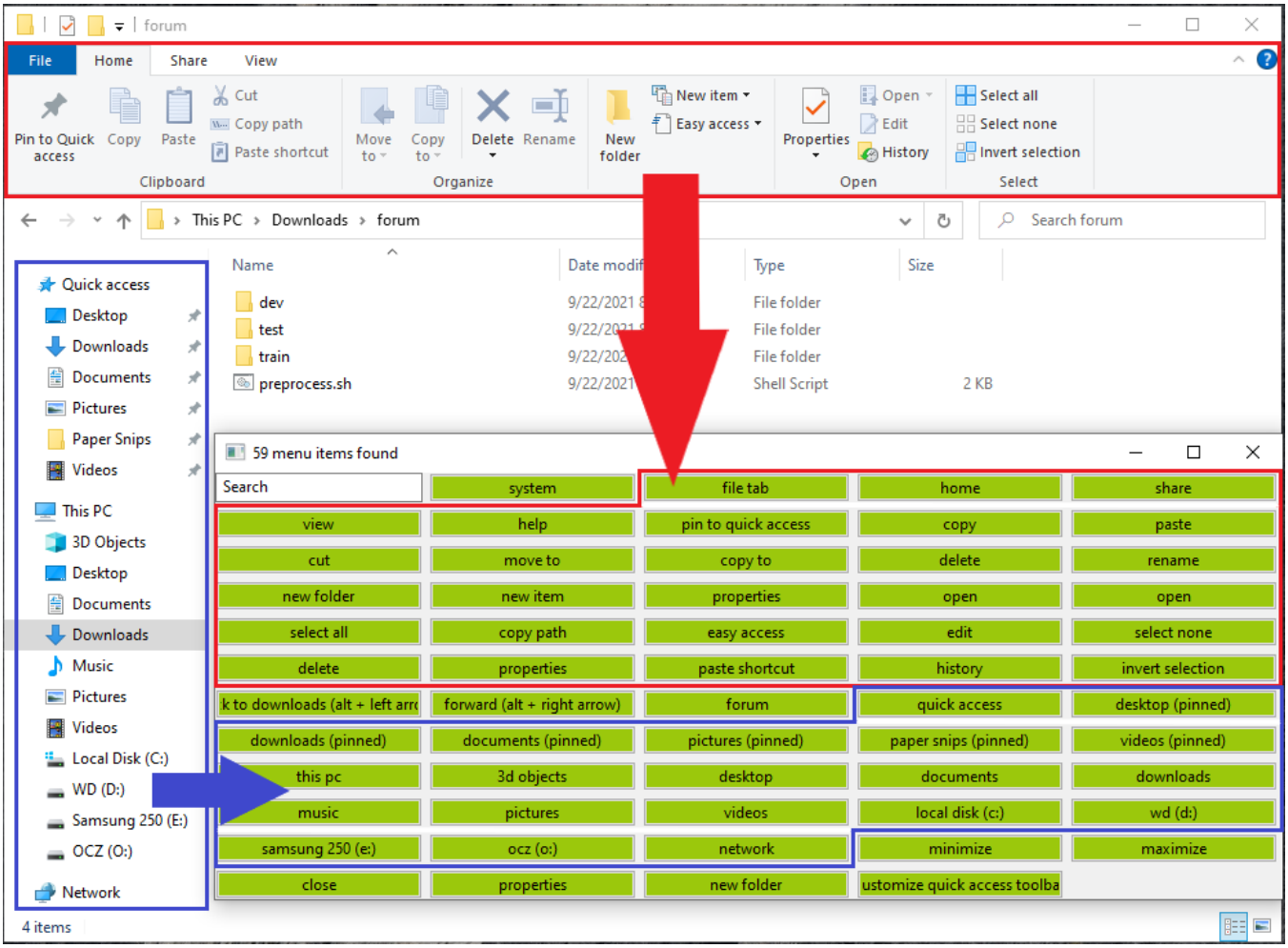


Figure 1: Example use of AccessBolt with File Explorer application. Buttons highlighted with red and dark blue boundaries in the pop-up dashboard correspond to controls highlighted by the same colored rectangles in the application's GUI.

non-visual interaction with the GUIs of a wide range of desktop applications. To better understand the usability issues faced by blind screen reader users when they interact with different desktop applications, we conducted an interview study with 12 blind participants. Analysis of the interview data revealed many interaction challenges of blind users, most notably the need to memorize different sets of keyboard shortcuts and navigation strategies for different desktop applications, a tedious navigation method involving a multitude of keyboard shortcut presses, and long learning curves for unfamiliar desktop applications.

As an initial step towards addressing the above users' concerns, we built a proof-of-concept assistive technology, namely AccessBolt, to facilitate convenient interaction with arbitrary applications for blind screen reader users. The design of AccessBolt was based on the following core principles: (i) Uniformity: AccessBolt automatically extracts all the application controls and then makes them available to the screen reader users via a custom dashboard interface (see Figure 1) that is navigable using the same shortcuts in all applications; (ii) Efficiency: AccessBolt provides a special shortcut for blind users to instantly access the dashboard menu, thereby eliminating the need for tedious manual exploration of application GUI via screen reader shortcuts. AccessBolt also provides a search feature for the dashboard menu controls which further reduces the time needed to access the desired control; and (iii) Usability: The design of AccessBolt dashboard menu is such that blind users can easily navigate its contents via basic shortcuts such as ARROW keys, TAB, and ENTER, thus obviating the need to memorize a wide array of shortcuts to access the application controls.

A pilot study with 6 screen reader users showed that with AccessBolt, the participants needed 40% (avg.) fewer keystrokes to access the desired application controls compared to the status quo condition without AccessBolt. Furthermore, the participants also indicated that they experienced a much lighter interaction workload with AccessBolt – nearly 4 times lower NASA-TLX [20] score compared to status quo. In sum, our contributions in this paper are:

- The findings of a preliminary interview study uncovering the difficulties faced by screen reader users when interacting with desktop application interfaces.
- The design of AccessBolt interface which aims to provide a uniform, efficient, and usable interaction experience for blind screen reader users in all desktop applications.

2 BACKGROUND

2.1 Usability Issues of Screen Readers

A screen reader (e.g., JAWS [16], NVDA [35], and Voiceover [3]) is a special purpose assistive technology that enables people with severe visual impairments to interact with computer applications by listening to their content. The popularity of screen readers among the community of blind users stems from the the fact that they are relatively cheaper and easier to access than hardware based solutions such as braille displays [19, 47, 48]. A screen reader provides a set of special keystrokes that allow a blind user to navigate an application's GUI elements in a one-dimensional paradigm. However, since the modern two-dimensional graphical user interfaces are primarily intended for sighted interaction, screen reader users struggle to efficiently and conveniently interact with applications, as evidenced by the findings of many prior studies [12, 40, 41, 44, 45].

For instance, Kurniawanand and Sutcliffe [26] investigated the mental models utilized by the screen reader users while navigating desktop application interfaces, and noticed that the screen reader users either created a structural mental model to remember the locations of the controls within the interface or memorized the complex shortcuts for directly accessing these controls. As a consequence, when they encountered an application that did not fit their mental model, they failed to properly navigate the application content. Moreover, their mental model was also likely to be challenged in a familiar application, if they had to access an unfamiliar application control. Similarly, Billah et al. [5] observed how differences in shortcut mappings between different screen readers and sometimes even between applications, created confusion and interaction difficulties for blind screen reader users. As there is no universal screen reader that supports uniform interaction across all operating systems and applications, blind users are required to constantly learn new navigational paradigms and strategies, which can be frustrating and time consuming. The challenge of convenient navigation/orientation was also uncovered by Shinohara and Tenenberg in their study [39]. This study revealed how easily an accidental key press could completely disorient a blind user, and also how the user would need to expend considerable mental effort and time to detect and recover from such mistakes.

2.2 Usability Enhancing Techniques for Screen Reader Users

There exist a few prior works that have focused on overcoming the usability challenges and thereby improving the desktop screen reader experience for blind users [13, 15, 33, 50]. Even the screen readers themselves are now offering additional support to offset usability issues caused by the one-dimensional navigational paradigm. For instance, the VoiceOver screen reader has an in-built Rotor [2] feature which allows users to filter and navigate GUI elements based on their types (e.g. navigating only the buttons or links). While this feature allows user to skip over irrelevant GUI elements, content navigation is none-the-less still one-dimensional; the interaction benefits depend on the number of the filtered elements. Moreover, the user should have apriori knowledge of the type (button, link, tab, etc.) of a target control; remembering such application-specific details can induce considerable cognitive load for blind users [5].

To address this issue concerning navigational efficiency, Lee et al. [28] proposed a hardware based solution for Microsoft Word application, where they re-programmed an external dial device to serve as an additional auxiliary input modality for quickly accessing the application's controls. Specifically, with the help of dial interface, screen reader users could easily navigate the ribbon controls (e.g., *Home*, *Insert*, *Design*) of Microsoft Word without getting disoriented and also without needing to remember shortcuts and navigation strategies. Hendy et al. [22] on the other hand developed a Command-Line-Interface tool for Microsoft Word, where the user could simply type the name of the control they wanted to access, instead of doing the same by manually navigating the application's interface. There also exist other similar works that have strived to improve the screen reader usability for word processing applications [10, 34, 38, 43]. However, the scope of all these aforementioned works were limited to a few specific productivity applications, and as such did not generalize to arbitrary applications.

In sum, findings of prior research emphasize the need for an efficient and uniform interface for blind users to conveniently interact with arbitrary desktop applications, and thereby bridge the usability gap between the desktop interaction experiences of blind and sighted users.

3 PRELIMINARY STUDY

We conducted an IRB-approved interview study with 12 screen reader users in order to better understand the usability challenges they face while interacting with desktop applications.

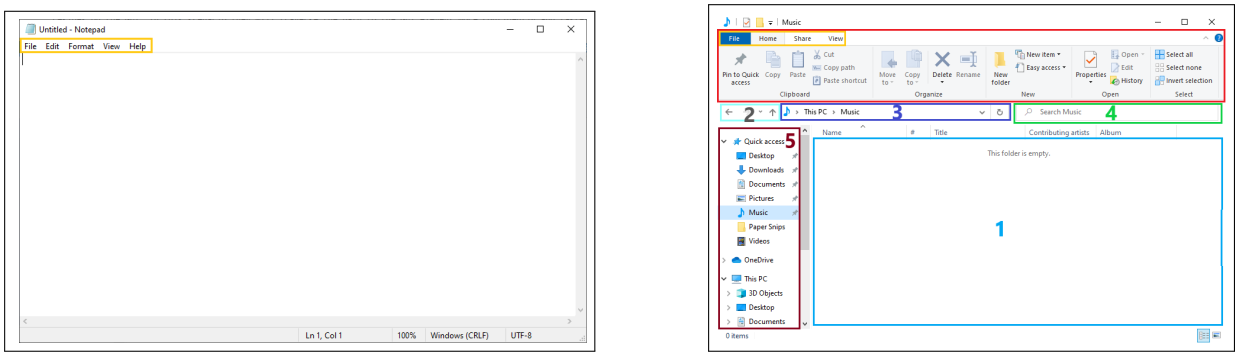
3.1 Participants

We recruited 12 (P1 to P12) screen reader users for the study. The average age was 48.2 (median=48.5, SD=10.5, Range=29 – 64) and the gender representation was equal (6 female, 6 male). Recruitment was done using an email list and snowball sampling. Inclusion criteria were: (i) exclusive dependence on a screen reader with no functional vision, and (ii) proficiency with a screen reader software. Except two participants (P1 and P2), all others owned a computer[1]. Participants P1 and P2 stated that they accessed computers in libraries and at their friend's homes. Almost all (11) participants had experience with the Windows operating system platform. When asked to self-assess their screen reader proficiency, P1 claimed to be a beginner, P6, P10, P11 claimed to be experts, and the rest of the participants claimed to have intermediate proficiency.

3.2 Interview format

All the interviews were conducted remotely via phone. The interviews were semi-structured, with questions about the following two topics:

[1]Participant details can be found here: https://github.com/UMAP2022-Supplementary-Files/AccessBolt



(a) (b)

Figure 2: Accessing controls in desktop applications. Rectangles highlight the ribbons that contain the controls of the desktop applications. (a): Ribbon for the Notepad. Yellow rectangle highlights the menu ribbon of the Notepad application which is accessible via "Alt" key. Controls in this ribbon are navigable via arrow keys. (b): Ribbons of File Explorer. Ribbon labelled as 1 is the content area. Red ribbon (top) is accessible via "Alt" key but the ribbons tagged with numbers 2-5 do not have a key shortcut. They are accessible via TAB.

- *Desktop usability*: What desktop applications do you use most frequently? How frequently do you come across unusable desktop applications? What are the causes for usability issues in desktop applications? What do you do to overcome these challenges?
- *Shortcut usage and GUI navigation habits*: Do you memorize shortcuts for specific applications? How frequently do you use shortcuts in desktop applications? How frequently do you use navigational shortcuts (ARROW keys and TAB) in desktop applications? Do you memorize screen reader specific shortcuts?

Each interview lasted about 10 minutes. The interviews were recorded, with the permission of the participant, and transcribed for analysis. Open coding technique [46] was used for analyzing the collected interview data; authors iteratively went over the user responses and identified key insights and themes that reoccurred in the data. Participation in the study was voluntary and no monetary compensation was given to the participants.

3.3 Findings

Heterogeneous keyboard shortcuts. Six participants (P1, P3, P5, P6, P10, P12) mentioned poor interface navigability as the source of many usability problems across multiple applications. In this regard, P5 and P6 specifically highlighted the lack of a uniform access paradigm to applications' controls, and additionally stated that many controls in applications could only be accessed via a combination of different kinds of shortcuts. P10 and P12 also mentioned that some controls could not be reached using the common navigational shortcuts, and instead required use of less familiar and hard-to-remember shortcuts. The participants also indicated that the usability problems were further exacerbated whenever they interacted with new applications due to the interaction burden associated with learning new shortcut mappings.

In order to further examine this issue, we used a screen reader to navigate interfaces of different desktop applications. We observed that simple applications such as Notepad used a single menu ribbon that was accessible via single keystroke and navigable with basic shortcuts (see Figure 2a). However, complex applications such as File Explorer and Microsoft Word used multiple ribbons to organize their controls, and these ribbons in turn had different access methods (i.e., shortcut mappings) for navigating the controls within them (see Figure 2b). As screen reader users often do not have (or remember) this application-specific shortcut knowledge, they end up manually exploring the interface using different combinations of ALT, TAB and ARROW keys to reach their target control.

Repeating actions and hoping for different outcome. Four participants (P4, P5, P6 and P11) stated that they often were unable to reach their target control in the first navigational attempt, and therefore they repeated their actions and tried different shortcut strategies with the hope of succeeding in one of the attempts. These participants however asserted that the strategy of repeating actions, even if successful, was annoying and frustrating. Eight participants (P1, P2, P4, P5, P9, P10, P11 and P12) stated that if even repetition and alternative exploration strategies did not help them in locating target controls, they sought help from either sighted friends or online forums.

Using general navigational shortcuts more than application specific shortcuts. All 12 participants claimed that they mostly relied on basic screen reader shortcuts that are common across multiple applications. Specifically, all of them mentioned that they frequently used ARROW and TAB keys for navigation, and 7 of them indicated that they also sometimes relied on basic 'skip' shortcuts (e.g., skipping between paragraphs, headings, and links). Only five participants (P4, P5, P6, P7 and P12) stated that they had memorized slightly more advanced shortcuts such as open/close, minimize, maximize, application title and menu bar. As for application-specific shortcuts (e.g., *Insert Table* in Excel *Align Center* in Word), 9 participants said that they less frequently used such shortcuts, except for the standard "Cut", "Copy", and "Paste" shortcuts. Only the remaining 3 participants (P6, P7 and P12) mentioned that they knew some

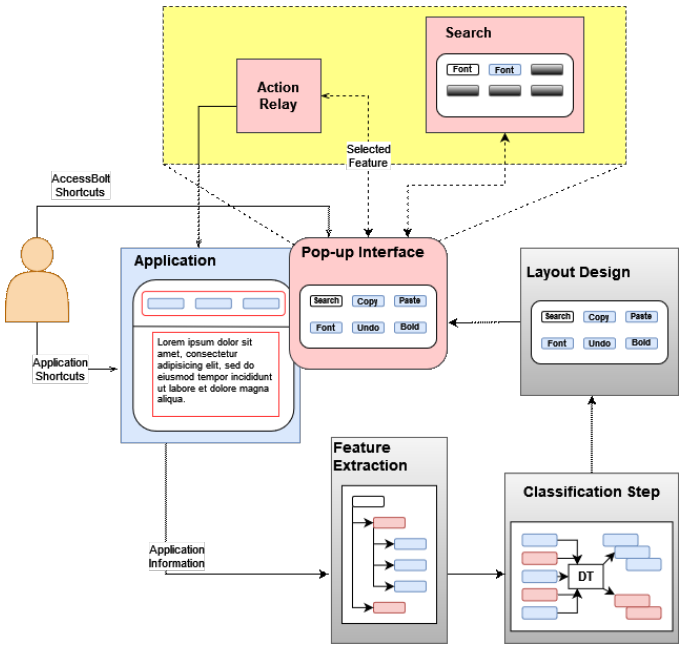


Figure 3: Architectural workflow of AccessBolt.

application-specific advanced shortcuts such as "Find", "Replace", "Indent Left/Right/Center", New document, "AutoSum"(Excel), and "Row/Column header names"(Excel). However, all participants agreed that using the application specific shortcuts to access controls, although efficient, was not feasible on most occasions due to the cognitive burden of remembering a plethora of complex shortcuts.

The interview study findings clearly illuminate the need for an efficient and uniformly navigable interface for screen reader users to interact with desktop applications. This interface should also minimize the cognitive and learning overhead with respect to shortcut memorization and moreover should be scalable, i.e., assist users in a wide range of desktop applications across different operating system platforms. By casting these interface requirements as design principles, we developed AccessBolt as described next.

4 ACCESSBOLT DESIGN

Figure 3 presents an architectural schematic of AccessBolt. When a user presses the special shortcut "CTRL + K" to instantly access the AccessBolt interface, AccessBolt first automatically detects and extracts all the controls in the application using an accessibility API. Then, the command controls (e.g., font dropbox in Word, New Slide button in PowerPoint etc.) are separated from non-command controls (e.g., structural controls such as panes and ribbons) and main content area controls (e.g., cells in Excel Table, files/folder in File Explorer) using custom built machine learning-based classification models. The identified command controls are subsequently passed to the layout design module which in turn dynamically creates a pop-up interface for displaying these commands to the user. Navigation within the popup interface is uniform, i.e., the user only needs to press the basic TAB shortcut to navigate the list of commands. The interface also provides a search functionality to assist the user in quickly locating the desired command. When the user selects a command in our interface, the Action Relay module triggers a selection or click event on the actual corresponding command in the application GUI. Also, the pop-up interface gets automatically closed and the screen reader focus returns to the application GUI (specifically the GUI element that was in focus when the user pressed the AccessBolt shortcut). This way the user can resume without having to readjust the screen reader focus via multiple key presses. The AccessBolt popup interface can also be manually closed at any instant using "CTRL + K" shortcut.

4.1 Extracting Application Controls

The accessibility API of the operating systems in general has access to the underlying UI trees of applications so as to ensure their accessibility for screen reader users. For instance, on the Windows operating system platform, the Microsoft UI Automation framework [32] serves as the accessibility API that gives screen readers access to application GUI elements. Specifically, this framework provides access to the UI Automation tree (also referred to as *control tree* from this point onwards for convenience) of an application that contains its structural information along with detailed information such as name, description, control type, and default action for each of the application's controls (buttons, lists, ribbons, etc.) [21]. When the user seeks to opens AccessBolt's popup interface, AccessBolt detects the application that is currently in use, and extracts all of its controls by leveraging the UI Automation framework.

4.2 Classifying Application Controls

After extracting the controls, AccessBolt classifies them as belonging to one of the two groups – command or non-command. Structural controls such as ribbons and panes do not offer any direct functionality to the user as their sole purpose is to group and visually arrange other controls for sighted users, therefore they are placed under the non-command category. Functional controls are the controls that can be triggered by the user (e.g. clicked, toggled, expanded, selected) and they execute operations on behalf of the user. For example, ribbon controls or commands such as *Bold*, *Left Indent*, *Fonts* in a word processor; *Increase volume*, *Play/Pause* buttons in a music player, are all functional controls. However, not all functional controls are command controls, some are content controls such as cells in Excel or files/folders in File Explorer. Classifying structural controls as non-command controls is relatively easy, but distinguishing between content controls and command controls is challenging due to their shared similarities. Therefore, we built custom machine learning models to aid us in this binary classification (command controls vs. everything else).

Classification Models. We trained three machine learning models for this classification task – decision tree, support vector machine (SVM), and neural network. We leveraged the scikit-learn software library [14] to train the machine learning model. The decision tree model used the Gini impurity metric and the SVM model used the Radial Basis Function kernel. The neural network model comprised two hidden layers (36 nodes in each layer) with rectified linear unit function. The learning rate was set to 0.001 and the maximum number of iterations was 200. In addition to these three

machine learning models, we also built an ontology consisting of manually designed classification rules.

We handcrafted features[2] to train the machine learning models. This included a categorical *control type* feature that was vectorized using the term frequency-inverse document frequency (tf-idf) method for training. Features capturing patterns (e.g., selection, toggle) were binary, and the features relevant to a control's relative screen location (e.g., horizontal and vertical position) were continuous. Note that we also tested the models both with and without the screen location-based features to assess their influence on the models' performances.

The manually constructed ontology-based classification model on the other hand had only 3 simple rules. The first rule checked if the type of an application control belonged to one of the predefined types in a custom predetermined list. This handcrafted list included types such as buttons, checkboxes, etc., that are commonly used in applications to represent command controls. The second rule checked whether an application control was a leaf node in the UI Automation control tree; our manual inspections revealed that structural controls such as panels and ribbons were almost always the inner nodes of the UI control tree. The last rule used *default action* attribute value of the application control to check if it was interactive via a keyboard or a mouse, as command features are always interactive e.g. clickable, toggleable, selectable, etc.

Dataset. To evaluate our models, we built a dataset by selecting 15 desktop applications and manually annotating all their controls as either command or non-command. Specifically, we chose a diverse set of applications to ensure generalizeability of the classification models – Calendar, Excel, File Explorer, Google Chrome, MS To do, Mozilla Firefox, Notepad, Notepad++, Outlook, PowerPoint, Skype, Weather, Word, WordPad and Zoom. In total, there were 1014 command (25%) and 3095 non-command (75%) controls in our dataset summing up to a total of 4109 controls.[3]

Evaluation. We used a folding method, where in each fold an application was left out for testing, and the models were trained on the data collected from the remaining 14 applications. The precision and recall metrics were then averaged across all applications, and the results are presented in Table 1.

Analysis. As seen in Table 1, the ontology model had the highest average recall but lowest average precision across all applications. A deeper analysis revealed that this was due to the model not leveraging the screen location information of the controls; it labelled controls even outside of the typical menu region in applications as command elements. The SVM model on the other hand performed much better than the ontology model, even when excluding the location information. However, it performed poorly on applications with relatively fewer command controls such as MS To Do, Calendar, Weather, Google Chrome and Firefox. Inclusion of location-related features slightly improved the SVM model performance for these specific applications, but the overall performance dropped significantly. The performance of the neural network model was similar to the SVM model, however its overall performance improved with the addition of location-related features. The decision tree model without the location features performed very poorly on applications such as Excel and Calendar where it labelled cells of the table and days on the calendar respectively as command controls. However, once the location features were included, the performance of the decision tree model drastically increased, and it yielded the best F1 score. We therefore selected the decision tree model with location features included as the classifier model in AccessBolt.

[2]Due to space constraints, detailed list of features along with their descriptions can be found here: https://github.com/UMAP2022-Supplementary-Files/AccessBolt

[3]The dataset can be found here: https://github.com/UMAP2022-Supplementary-Files/AccessBolt

4.3 AccessBolt Popup Interface

Once the command controls are identified in the application GUI, AccessBolt creates *proxy* commands for them in its popup interface as shown in Figure 1. As seen in this figure, all applications commands regardless of their containing ribbons, are grouped together into one list in the popup, thereby eliminating the need for memorizing different shortcuts and strategies for accessing different commands within and across applications. This design choice was also motivated by prior work [24] and our own observations, which indicated that blind users were generally familiar with the command names (e.g., "Find") and not their container names(e.g., "Editing"). The linear arrangement of commands also facilitates simple uniform navigation with basic TAB or ARROW shortcuts.

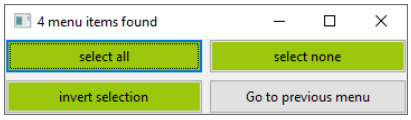


Figure 4: Search feature illustration using the keyword "Select" for the interface in Figure 1.

4.3.1 ***Search Feature.*** To further reduce the time need to access a target command, we included a search feature at the beginning of the popup interface. Specifically, while navigating the commands in the popup, the user can press the "S" shortcut anytime to instantly focus on the search bar, and then type the name of the command. In this regard, we used Levenshtein distance based matching [31] to provide more flexibility with the search queries; only the commands with the highest matching scores are shown in the popup thereby reducing the search space for the user. For example, Figure 4 depicts how the AccessBolt popup interface gets updated with fewer relevant commands in the list, when the user enters a query "Select" in the search bar. Without search functionality, the user would need 20 key presses to reach the "Select all" button in the original list, whereas with the search functionality, the same can be achieved with fewer key presses by typing *select* and then pressing the ENTER key.

4.3.2 ***Action Relay.*** Once the user selects a command from the popup interface, AccessBolt leverages Microsoft UI Automation API to automatically trigger the default action (click, expand, toggle, etc.) on the actual corresponding command control in the application GUI. AccessBolt then automatically closes the popup interface and returns the screen reader focus to where it was prior to accessing the popup interface, thereby ensuring seamless interface transition and context preservation for the user.

	Averaged Scores (Standard Deviations)		
Models	Precision	Recall	F1
Ontology	0.37 (0.28)	1.0 (0.0)	0.54 (0.24)
SVM	0.84 (0.32)	0.83 (0.39)	0.83 (0.34)
SVM with Location features	0.68 (0.21)	0.70 (0.16)	0.69 (0.18)
Neural Network	0.83 (033)	0.83 (0.38)	0.83 (0.34)
Neural Network with Location features	0.80 (0.18)	0.91 (0.19)	0.85 (0.15)
Decision Tree	0.39 (0.34)	0.82 (0.38)	0.53 (0.34)
Decision Tree with Location features	0.88 (0.25)	0.89 (0.21)	**0.88 (0.22)**

Table 1: Precision, Recall and F1 scores of the classification models.

5 EVALUATION

5.1 Pilot Study

Participants. We conducted an IRB-approved pilot study with 6 screen reader users (see Table 2) to evaluate AccessBolt. The average age of the participants was 47.5 (median=47, min=37, max=58) and the gender representation was equal (3 male, 3 female). Recruitment of the participants was done via word of mouth and snowball sampling. The inclusion criteria were: (i) exclusive dependency on screen readers for interacting with computers; (ii) at least basic proficiency with one screen reader (JAWS or NVDA); and (iii) experience with the Windows operating system platform. To ensure external validity, there was also no overlap between the participant pools of the two studies.

Design. In the study, the users had to complete four typical desktop-application tasks using a screen reader. These tasks along with the comprising sub-tasks are listed in Table 3. Since the goal of the study was to evaluate ease and speed with which a user could access application controls with and without AccessBolt, we did not want the user's familiarity with the application to be a confounding factor. Therefore, each task was divided into a sequence of clearly specified sub-tasks such that all participants had equal knowledge about the task. In a within-subject experimental setup, the participants had to complete these tasks under the following two conditions, with two tasks per condition:

- **Screen Reader**: Using just the screen reader shortcuts (status-quo baseline)
- **AccessBolt:** Using the popup interface generated by AccessBolt (proposed approach)

As noticeable in Table 3, we considered both a Microsoft application and a third-party application for our tasks in order to have a more general representation of desktop applications. Specifically, we chose the Windows File Explorer and Zoom Cloud Meetings [49] applications, as both of these applications provide important service to desktop users. The File Explorer tasks requires 3 and 4 mouse clicks for a sighted person while the Zoom tasks requires 3 and 5 mouse clicks. The assignment of tasks to conditions and the ordering of the conditions and tasks were counterbalanced using the Latin square method [9].

Procedure. The experimenter first allowed the participants to refresh their memory regarding the standard screen reader shortcuts and also get accustomed to the AccessBolt popup interface. For this practice session, we selected the Wordpad application, where they were asked to type, navigate between "Home" and "View" menu tabs, and find "Bold" and "Align text to center" controls in the Wordpad ribbons. After practice, the participants were asked to complete the main study tasks in the predetermined counterbalanced order. To complete a task, the participants had to complete all of its subtasks as listed in Table 3. We allocated a maximum of 10 minutes for each task. For each participant, the keystrokes and task completion times were recorded. After completing each task, the participants were asked to answer an SEQ (Single Ease Question)[4], on a scale of 1(hard) to 7(easy). The participants were also administered the NASA-TLX [20] questionnaire to estimate their perceived workload after each study condition. Each study lasted for nearly 2 hours, and the participants were given monetary compensation for their time and contribution.

5.1.1 *Results.*

Task Completion. With 6 participants and 4 tasks per participant, there were in total 24 tasks (12 per condition). Under the AccessBolt condition, all participants were able to complete all of the tasks within the time limit; however, they were only able to complete 5 tasks (cumulative) under the baseline Screen Reader condition (see Table 4). Also, as shown in Table 4, in the Screen Reader condition, the participants could not finish 3 tasks (cumulative) within the stipulated time limit, and further they even gave up in 4 tasks (highlighted in red) due to frustration.

Our observations showed that the participants spent a lot of time and key presses navigating the application GUI, because the right navigational instructions were not communicated to them clearly by the applications in real time. For example, when searching for the "Target" folder in File Explorer task 1, most participants managed to navigate to the correct ribbon using the TAB shortcut, however, they failed to realize that they were supposed to use the ARROW keys to navigate within this ribbon. This need for changing the shortcut from TAB to ARROW keys for navigation was not communicated to them by either the application or the screen reader, and consequently the participants were never able to reach their goal; instead they simply cycled over the remaining elements leading to frustration. In the Zoom application, the TAB shortcut was needed to navigate between the ribbons, but the ARROW keys were required to navigate within the top ribbon. The participants therefore struggled to find the "Meetings" tab, and they kept navigating in

[4] https://measuringu.com/seq10/

ID	Gender & Age	Condition	Screen reader	Proficiency	Computer Usage	Most Used Application
P1	Female / 48	Blind	JAWS	Beginner	Once a week	Notepad
P2	Female / 37	Blind	JAWS, VoiceOver	Intermediate	Twice a week	Outlook
P3	Male / 55	Light Perception	NVDA, VoiceOver	Intermediate	Once a week	Word, Notepad
P4	Male / 46	Blind	JAWS	Intermediate	Once a month	Windows Media Player
P5	Male / 41	Light Perception	JAWS, VoiceOver	Expert	Everyday	Word, Notepad, Skype
P6	Female / 58	Blind	JAWS	Expert	Everyday	Excel, Calculator, Word

Table 2: Pilot study participant demographics. Screen reader proficiency is self-assessed

Application	Task Number	Subtasks
File Explorer	1	Navigate to "Target" Folder -> Select "Target.txt" file -> Go to "View" Tab -> Invoke "Hide Selected Item" button
	2	Select all items in the folder -> Go to "Share" Tab -> Invoke the "Zip" button
Zoom	1	Go to "Meetings" Tab -> Select the "Target" meeting -> Invoke the "Edit" button -> Check the "Host Video Off" checkbox -> Invoke the "Save" button.
	2	Go to "Chat" Tab -> Select the "Target" chat -> Type and send message

Table 3: Pilot study tasks along with sub-tasks.

circles over the application elements. As a consequence, two participants P1 and P4 run out of time whereas the participant P6 simply gave up (see Table 4). On the other hand, none of the participants navigated the AccessBolt dashboard interface more than once. We observed that once the participants got accustomed to the search bar, they used it frequently which reduced the task completion times further, while also improving the application usability.

Number of Key Presses. Keystroke statistics for 5 participants are shown in Table 5 (note that key press data for the participant P1 was lost due to technical issues). In Table 5, we also show the number of basic navigational key presses (TAB and ARROW keys) used by the participants while doing the tasks.

The participants performed 178.35 key presses on average in the Screen Reader condition, whereas they only needed an average of 72.18 keystrokes in the AccessBolt condition, a result that is in accordance with the prior observations regarding task completion times. The high number of key presses in the Screen Reader condition was mainly due to repeated cyclic navigation of interface elements due to lack of proper navigational instructions. In the AccessBolt condition however, none of the participants had to navigate the controls in the popup interface more than once, which significantly decreased the number of keystrokes for doing the tasks. The use of search bar lowered the number of key presses by reducing the command search space. As shown in Table 5, the average ratio of navigational keystrokes (TAB and ARROW keys) to total number of keystrokes was 0.87 in the baseline condition, whereas it was just 0.22 with AccessBolt. This indicates that the participants preferred to use the search bar for quickly accessing the controls rather than navigating the interface element by element.

SEQ and NASA-TLX Scores. The average SEQ score was 2.08 (min=1 , max=4, median=1.5) for the baseline condition and 5.66 (min=3, max=7, median=6) for the AccessBolt condition (higher is better) suggesting that participants had an easier time completing the tasks using AccessBolt. The average NASA-TLX score for the baseline and the AccessBolt conditions were 11.08 and 3.33 respectively (lower is better). A detailed examination of the individual components of the TLX scores revealed that the average scores for the performance, effort and frustration sub-scales were very high for the baseline condition (14.5, 13.3 and 13 respectively out of 20), which indicates that the participants felt they were exerting more effort for completing the tasks, while feeling that they have not performed well and getting increasingly frustrated in the process. Meanwhile the scores for the same categories in the AccessBolt condition were much lower – 5.3, 2.5 and 1.8 respectively, which confirms that the participants had a more pleasant experience under the AccessBolt condition.

Subjective Feedback. Subjective feedback of the participants was collected at the end of the study via exit interviews. In the interviews, all participants explicitly stated that they preferred the AccessBolt condition over the baseline screen reader condition. Few of them (P2, P3) mentioned that they liked how quickly they could find a desired GUI element using the proposed AccessBolt interface. Quoting participant P2 – "I just open [interface] and searched. It's quick. Very basic, and useful." All participants also liked that the arrangement of having all functional application controls in a single popup interface. For instance, P4 stated that – "I liked it because all the options were in place. There was a search method in place. If you don't do the old method you can search it." Lastly, the participants felt that AccessBolt was easy to learn and easy to use. To quote P1, "It was fewer keystrokes, less to remember, controls come easy.", and P5 said "Simple and user friendly." Two participants suggested that the AccessBolt interface should also be capable of reminding them of their relative location in the interface. In this regard, P5 said "Screen reader should read where you exactly are.", while P6 suggested that the interface should "Remind which element is currently selected". Using sound prompts to communicate the status of the popup interface and also the location of the user within the

ID	Baseline				AccessBolt			
	File Explorer		Zoom		File Explorer		Zoom	
	1	2	1	2	1	2	1	2
P1	>10:00 (3)	N/A	>10:00 (1)	N/A	N/A	2:00 (7)	N/A	1:24 (7)
P2	N/A	2:31 (2)	N/A	6:33 (1)	6:10 (5)	N/A	2:39 (6)	N/A
P3	7:12 (3)	N/A	N/A	8:14 (3)	N/A	3:05 (4)	2:55 (3)	N/A
P4	N/A	5:08 (4)	>10:00 (1)	N/A	3:56 (7)	N/A	N/A	1:52 (7)
P5	N/A	4:01 (4)	N/A	3:53 (1)	4:21 (7)	N/A	4:16 (5)	N/A
P6	7:38 (1)	N/A	3:49 (1)	N/A	N/A	2:05 (4)	N/A	1:17 (6)

Table 4: Task completion times and SEQ scores for each participant. Tasks with greater than 10 minute completion times and those that were abandoned before the time limit (indicated by red) are considered as failures. N/A indicated the participants did not do the task in the specified condition.

Key Type	Baseline				AccessBolt			
	File Explorer		Zoom		File Explorer		Zoom	
	1	2	1	2	1	2	1	2
Navigational	154.5 (142.1)	98.3(33.7)	177.5(102.5)	189.3(35.7)	37(32,8)	17.5(20.5)	10.3(12.8)	0 (0)
Total	188.5 (126.6)	115.3(31.8)	196(108.9)	213.6(41)	118.6(38.4)	47.5(24.7)	91.6(25.8)	31(7.1)
Average	Navigational: 154.9 / Total:178.35				Navigational: 16.2 / Total: 72.18			

Table 5: Average number of key presses with standard deviation for subtasks in baseline and AccessBolt conditions. Navigational keystrokes only consist of TAB and ARROW keys. Last row shows the average number of key presses per conditions.

interface was a common request by a majority (P1, P3, P5, P6) of the participants.

6 DISCUSSION

The results of our pilot study demonstrated the potential of AccessBolt in improving the usability of desktop interaction for blind screen reader users. However, the study also illuminated key limitations as discussed next.

Fixing the accessibility issues. A limitation of AccessBolt is that it does not fix any existing accessibility problems; it just focuses on making the accessible elements more usable. Applications that do not support any of the accessibility frameworks tend to have unlabelled controls in their interfaces. For example, an application can have a button with a "mute icon" but not provide any label to the accessibility framework. A sighted person can read or recognize the icon of the button on their screen, but a screen reader user will only hear the word *button* when the cursor focus moves to this control. For such scenarios, a machine learning model such as [11] can be trained and integrated into AccessBolt, so as to automatically generate labels for controls with missing labels.

Supporting other accessibility frameworks. The current AccessBolt prototype is limited to Windows applications that support the Microsoft UI Automation accessibility framework, however there exist other operating system-specific [1, 17], and also platform independent accessibility frameworks[18, 36] that are yet to be incorporated into AccessBolt. In order to support the applications that use the other OS frameworks, we have to extend the AccessBolt workflow pipeline by making changes to the Controls Extraction module, and retrain the classification models if necessary.

Expanded scope of search functionality. In most applications, all ribbons are not present on the screen at the same time. For example, in the File Explorer application, controls under the "Home" tab disappear when the user switches to the "View" tab. The present AccessBolt prototype is unable to capture the controls under these *hidden* tabs because they are not rendered on the screen and therefore not present in the UI control tree. As future work, we plan to develop a pre-processing *click monkey* application, similar to [23], that automatically traverses all these *hidden* tabs and dynamic options, and then saves this information, so that AccessBolt always has information regarding all the controls in the application. This way when a user searches for a hidden control in the dashboard interface, AccessBolt can first make it visible and then automatically trigger it if selected by the user.

Adding alternative modes of interaction. Adding additional modes of interaction to AccessBolt interface can provide more options to the user and thereby improve the overall experience. For example, opening the popup interface followed by navigating, searching, and invoking the controls within it can all be carried out using speech commands instead of using the key presses and shortcuts. This can further reduce the need for typing when searching and thereby reduce the overall effort and time needed for triggering a control. Speech interfaces can be especially beneficial for novice screen reader users who have limited screen reader skills and knowledge.

Contextual ordering of commands. In the current version of AccessBolt, commands in the popup interface are listed in the same order as they appear in the application's internal UI tree. This can sometimes lead to tedious linear navigation of the list, especially if the user is unable to find the desired command via search (e.g.,

when the user is unaware of the exact name of the command). In such scenarios, dynamic reordering of commands based on their likelihood of being accessed next by the user given the context, can significantly improve the efficiency of command access in AccessBolt's popup interface. For example, if the user highlights a document portion in a text editor application, AccessBolt can prioritize commands related to text formatting and place them at the top of the list in its popup interface.

Large-scale evaluation. An obvious limitation of our work was the small sample size of participants in the evaluation study[5]. The lack of statistically significant observations can be attributed to the small sample size. In the near future, we plan to conduct a large scale study with more participants.

7 CONCLUSION

In this paper, we explored the usability issues blind screen reader users typically face when interacting with the graphical user interfaces of desktop applications. We conducted an interview study which revealed interaction issues such as tedious navigation involving multiple shortcut presses in application interfaces, and the need for memorizing different shortcuts for different applications. To mitigate these issues, we proposed AccessBolt – a system wide service application that can automatically access the interface details of the target application, detect the command controls in it, and then present these controls to the user instantly on demand via a pop-up interface that is easily and uniformly navigable with simple keyboard shortcuts. A pilot study with blind participants reaffirmed our observations about usability issues of desktop applications, and demonstrated the potential of AccessBolt in mitigating the negative impact of these usability issues. The study also provided promising directions for future work.

ACKNOWLEDGMENTS

This work was supported by NSF Awards: 1805076, 1815514, 1936027, 2113485, 2125147 and NIH Awards: R01EY030085, R01HD097188.

REFERENCES

[1] Apple. 2021. Accessibility API. https://developer.apple.com/accessibility/macos/.
[2] Apple. 2021. Rotor function for VoiceOver. https://support.apple.com/guide/voiceover/with-the-voiceover-rotor-mchlp2719/mac
[3] Apple. 2021. Vision Accessibility - Mac - Apple. https://www.apple.com/accessibility/mac/vision/
[4] Edward C Bell and Natalia M Mino. 2015. Employment outcomes for blind and visually impaired adults. (2015).
[5] Syed Masum Billah, Vikas Ashok, Donald E Porter, and IV Ramakrishnan. 2017. Ubiquitous accessibility for people with visual impairments: Are we there yet?. In *Proceedings of the 2017 chi conference on human factors in computing systems*. 5862–5868.
[6] Yevgen Borodin, Jeffrey P Bigham, Glenn Dausch, and IV Ramakrishnan. 2010. More than meets the eye: a survey of screen-reader browsing strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. ACM, 13.
[7] Diego Bovenzi, Gerardo Canfora, and Anna Rita Fasolino. 2003. Enabling legacy system accessibility by web heterogeneous clients. In *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.* IEEE, 73–81.
[8] Lawrence H Boyd, Wesley L Boyd, and Gregg C Vanderheiden. 1990. The graphical user interface: Crisis, danger, and opportunity. *Journal of Visual Impairment & Blindness* 84, 10 (1990), 496–502.
[9] James V Bradley. 1958. Complete counterbalancing of immediate sequential effects in a Latin square design. *J. Amer. Statist. Assoc.* 53, 282 (1958), 525–528.

[5]Due to the COVID-19 pandemic, participant enrollment was much lower than usual.

[10] Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Giulio Mori. 2012. Designing e-learning collaborative tools for blind people. *E-Learning-Long-Distance and Lifelong Perspectives* (2012), 125–144.
[11] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhut, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 322–334.
[12] Michael F Chiang, Roy G Cole, Suhit Gupta, Gail E Kaiser, and Justin B Starren. 2005. Computer and world wide web accessibility by visually disabled patients: Problems and solutions. *Survey of ophthalmology* 50, 4 (2005), 394–405.
[13] Michael Connolly, Christof Lutteroth, and Beryl Plimmer. 2010. Document resizing for visually impaired students. In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*. 128–135.
[14] David Cournapeau. 2021. Scikit-learn software library for Python. https://scikit-learn.org/stable/
[15] Christin Engel, Emma Franziska Müller, and Gerhard Weber. 2019. SVGPlott: an accessible tool to generate highly adaptable, accessible audio-tactile charts for and from blind and visually impaired people. In *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. 186–195.
[16] Freedom Scientific. 2021. JAWS® - Freedom Scientific. https://www.freedomscientific.com/products/software/jaws/
[17] GNOME. 2021. Accessibility Toolkit. https://developer.gnome.org/atk/.
[18] Andres Gonzalez and Loretta Guarino Reid. 2005. Platform-independent accessibility api: Accessible document object model. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. 63–71.
[19] Yoichi Haga, Wataru Makishi, Kentaro Iwami, Kentaro Totsu, Kazuhiro Nakamura, and Masayoshi Esashi. 2005. Dynamic Braille display using SMA coil actuator and magnetic latch. *Sensors and Actuators A: Physical* 119, 2 (2005), 316–322.
[20] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
[21] Rob Haverty. 2005. New Accessibility Model for Microsoft Windows and Cross Platform Development. *SIGACCESS Access. Comput.* 82 (June 2005), 11–17. https://doi.org/10.1145/1077238.1077240
[22] Jeff Hendy, Kellogg S. Booth, and Joanna McGrenere. 2010. Graphically Enhanced Keyboard Accelerators for GUIs. In *Proceedings of Graphics Interface 2010* (Ottawa, Ontario, Canada) *(GI '10)*. Canadian Information Processing Society, CAN, 3–10.
[23] Sven Hertling, Markus Schröder, Christian Jilek, and Andreas Dengel. 2017. Where is that Button Again?!-Towards a Universal GUI Search Engine.. In *ICAART (2)*. 217–227.
[24] Harry Hochheiser and Jonathan Lazar. 2010. Revisiting breadth vs. depth in menu structures for blind users of screen readers. *Interacting with Computers* 22, 5 (2010), 389–398.
[25] Shaun K. Kane, Chandrika Jayant, Jacob O. Wobbrock, and Richard E. Ladner. 2009. Freedom to Roam: A Study of Mobile Device Adoption and Accessibility for People with Visual and Motor Disabilities. In *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility* (Pittsburgh, Pennsylvania, USA) *(Assets '09)*. Association for Computing Machinery, New York, NY, USA, 115–122. https://doi.org/10.1145/1639642.1639663
[26] Sri Hastuti Kurniawan, Alistair G Sutcliffe, and Paul Blenkhorn. 2003. How Blind Users' Mental Models Affect Their Perceived Usability of an Unfamiliar Screen Reader.. In *INTERACT*, Vol. 3. 631–638.
[27] Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. 2007. What frustrates screen reader users on the web: A study of 100 blind users. *International Journal of human-computer interaction* 22, 3 (2007), 247–269.
[28] Hae-Na Lee, Vikas Ashok, and IV Ramakrishnan. 2020. Rotate-and-Press: A Non-visual Alternative to Point-and-Click?. In *International Conference on Human-Computer Interaction*. Springer, 291–305.
[29] H. N. Lee, V. Ashok, and I. V. Ramakrishnan. 2020. Repurposing Visual Input Modalities for Blind Users: A Case Study of Word Processors. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2714–2721. https://doi.org/10.1109/SMC42975.2020.9283015
[30] Barbara Leporini, Maria Claudia Buzzi, and Marina Buzzi. 2012. Interacting with Mobile Devices via VoiceOver: Usability and Accessibility Issues. In *Proceedings of the 24th Australian Computer-Human Interaction Conference* (Melbourne, Australia) *(OzCHI '12)*. Association for Computing Machinery, New York, NY, USA, 339–348. https://doi.org/10.1145/2414536.2414591
[31] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. 707–710.
[32] Microsoft. 2021. Microsoft UI Automation. https://docs.microsoft.com/en-us/windows/win32/winauto/entry-uiauto-win32
[33] Lourdes Morales, Sonia M Arteaga, and Sri Kurniawan. 2013. Design guidelines of a tool to help blind authors independently format their word documents. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 31–36.
[34] Giulio Mori, Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Victor MR Penichet. 2010. Making "Google Docs" user interface more accessible for blind

people. In *International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability*. Springer, 20–29.
[35] NVDA team. 2021. NVDA. https://www.nvaccess.org/.
[36] ORACLE. 2021. JAVA Accessibility API. https://docs.oracle.com/javase/7/docs/technotes/guides/access/jaapi.html.
[37] Leonard H Poll and Berry H Eggen. 1996. Non-visual interaction with GUI objects. In *People and Computers XI*. Springer, 159–168.
[38] John Gerard Schoeberlein and Yuanqiong Wang. 2014. Usability Evaluation of an Accessible Collaborative Writing Prototype for Blind Users. *Journal of Usability Studies* 10, 1 (2014).
[39] Kristen Shinohara and Josh Tenenberg. 2007. Observing Sara: A Case Study of a Blind Person's Interactions with Technology. In *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility* (Tempe, Arizona, USA) *(Assets '07)*. Association for Computing Machinery, New York, NY, USA, 171–178. https://doi.org/10.1145/1296843.1296873
[40] Reeta Singh. 2012. Blind Handicapped Vs. Technology: How do Blind People use Computers? *International Journal of Scientific & Engineering Research* 3, 4 (2012), 1–7.
[41] J. Thatcher. 1994. Screen Reader/2: Access to OS/2 and the Graphical User Interface. In *Proceedings of the First Annual ACM Conference on Assistive Technologies* (Marina Del Rey, California, USA) *(Assets '94)*. Association for Computing Machinery, New York, NY, USA, 39–46. https://doi.org/10.1145/191028.191039
[42] Katie Wang, Laura G Barron, and Michelle R Hebl. 2010. Making those who cannot see look best: Effects of visual resume formatting on ratings of job applicants with blindness. *Rehabilitation psychology* 55, 1 (2010), 68.
[43] Mirza Muhammad Waqar, Muhammad Aslam, and Muhammad Farhan. 2019. An intelligent and interactive interface to support symmetrical collaborative educational writing among visually impaired and sighted users. *Symmetry* 11, 2 (2019), 238.
[44] Tetsuya Watanabe, Shinichi Okada, and Tohru Ifukube. 1998. Development of a GUI screen reader for blind persons. *Systems and Computers in Japan* 29, 13 (1998), 18–27.
[45] Brian Wentz and Jonathan Lazar. 2011. Usability evaluation of email applications by blind users. *Journal of Usability Studies* 6, 2 (2011), 75–89.
[46] David Wicks. 2017. The coding manual for qualitative researchers. *Qualitative research in organizations and management: an international journal* (2017).
[47] Cheng Xu, Ali Israr, Ivan Poupyrev, Olivier Bau, and Chris Harrison. 2011. Tactile display for the visually impaired using TeslaTouch. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. 317–322.
[48] Levent Yobas, Dominique M Durand, Gerard G Skebe, Frederick J Lisy, and Michael A Huff. 2003. A novel integrable microvalve for refreshable braille display system. *Journal of microelectromechanical systems* 12, 3 (2003), 252–263.
[49] Zoom. 2021. Zoom Meetings. https://explore.zoom.us/meetings.
[50] Hong Zou and Jutta Treviranus. 2015. ChartMaster: A Tool for Interacting with Stock Market Charts Using a Screen Reader. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (Lisbon, Portugal) *(ASSETS '15)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/2700648.2809862