A Probabilistic Model and Metrics for Estimating Perceived Accessibility of Desktop Applications in Keystroke-Based Non-Visual Interactions

Md Touhidul Islam
Pennsylvania State University
University Park, PA, United States
mqi5127@psu.edu

Donald E Porter
University of North Carolina
Chapel Hill, NC, United States
porter@cs.unc.edu

Syed Masum Billah
Pennsylvania State University
University Park, PA, United States
sbillah@psu.edu

ABSTRACT

Perceived accessibility of an application is a subjective measure of how well an individual with a particular disability, skills, and goals experiences the application via assistive technology. This paper first presents a study with 11 blind users to report how they perceive the accessibility of desktop applications while interacting via assistive technology such as screen readers and a keyboard. The study identifies the low navigational complexity of the user interface (UI) elements as the primary contributor to higher perceived accessibility of different applications. Informed by this study, we develop a probabilistic model that accounts for the number of user actions needed to navigate between any two arbitrary UI elements within an application. This model contributes to the area of computational interaction for non-visual interaction. Next, we derive three metrics from this model: complexity, coverage, and reachability, which reveal important statistical characteristics of an application indicative of its perceived accessibility. The proposed metrics are appropriate for comparing similar applications and can be fine-tuned for individual users to cater to their skills and goals. Finally, we present five use cases, demonstrating how blind users, application developers, and accessibility practitioners can benefit from our model and metrics.

CCS CONCEPTS

• **Human-centered computing** → **Accessibility design and evaluation methods**.

KEYWORDS

Perceived accessibility, usability; blind users, screen readers, keyboards; computational interaction, probabilistic model; desktops.

ACM Reference Format:
Md Touhidul Islam, Donald E Porter, and Syed Masum Billah. 2023. A Probabilistic Model and Metrics for Estimating Perceived Accessibility of Desktop Applications in Keystroke-Based Non-Visual Interactions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), April 23–28, 2023, Hamburg, Germany.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3544548.3581400

CHI '23, April 23–28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04.
https://doi.org/10.1145/3544548.3581400

1 INTRODUCTION

Software developers make choices at design time that affect the user experience of their software for individuals with disabilities. For instance, individuals with blindness—who must use assistive technologies (ATs), such as screen readers, to interact with computers—are disproportionately impacted by how developers organize visual content. Interleaving ads with text in web content, for example, can lead to confusion and cognitive load, because visual delimiters, such as whitespace, do not translate well to audio by the ATs [11, 71]. Specifically, visual proximity in a graphical user interface (UI) does not necessarily translate to a simple navigation path via the keyboard and audio; rather, users face an additional cognitive load to learn and recall a path through a different, logical representation of the UI. As a result, decisions such as how elements are logically grouped or the selection and placement of keyboard shortcuts are critical to the experience of blind users.

To measure the user experience of the software, usability and accessibility are often used in HCI literature. *Usability* measures to what extent the software offers effective, efficient, and satisfying user experience; and *accessibility* indicates to what extent the software can be used by people with various disabilities via assistive technologies [77]. Usability does not entail accessibility [47]. For example, a piece of software (e.g., remote desktop clients) can be usable to a particular user group (e.g., sighted users) but can be inaccessible to another (e.g., blind users) [5]. Conversely, a particular user group (e.g., blind users) can access the software (e.g., Microsoft Word) via ATs, but their experience can be poor.

Researchers have attempted to integrate both qualities, coining new terms, such as universal usability, where the underlying theme is to enable access to technology plus success for all users [62, 63]. We subscribe to this notion that accessibility is an integral part of usability, and thus, usability principles can be used to measure accessibility. We recognize that universal usability has remained a theoretical concept, mainly because the access needs of individuals with different disabilities are understudied [22], and most developers are unaware of these needs [16]. As such, we believe it is important to understand how individuals with a particular disability *perceive* the usability and accessibility of different software. To that end, we use a term, *perceived accessibility*, to capture how well an individual with a disability experiences software while interacting via their preferred assistive technology.

This work focuses on vision disability, specifically individuals with blindness, and their perceived accessibility of *desktop applications* with screen readers and a keyboard. It is important because desktop applications are still the primary ways people use computers in education and employment.

We first conducted a study with 11 expert blind participants to identify what matters the most in their interaction. The study reveals that perceived accessibility is subjective; it depends on individuals' skills and goals in using the software. Additionally, it is intertwined with usability and other broad concepts, such as independence; ease, i.e., the ease of understanding, learning, and describing an application; the reliability and deterministic behavior of keyboard shortcuts; and the ability to transfer the knowledge of knowing one application to another.

Our study also revealed that a common desire among blind screen reader users is to be able to find a specific functionality within *three* to *four* keystrokes (Section 3.4.8). For example, our analysis shows that without the use of a shortcut in Microsoft Word, the average task takes 17 keystrokes using a screen reader versus one to two clicks for a sighted user. The requirements for Microsoft PowerPoint and Excel are almost 23 keystrokes without using shortcuts. As a result of such a cumbersome user experience, our participants mentioned that they often simply forego functionality and limit themselves to a small fraction of the features of an application. Continuing the example of Microsoft Word, most participants use only 10–15% of the functionalities.

Through this study, we also collected coarse data on participants' perceived accessibility of commonly used desktop applications and usage frequency of different UI elements (e.g., menu, ribbons, and buttons) within these applications. Based on the findings and collected data, we propose a ***probabilistic model*** and derive *three metrics*, namely, ***complexity***, ***coverage***, and ***reachability***, relevant to the usability of keyboard-based interaction of an application for blind users. A key to our model is that the number of user actions to navigate between any two UI elements has a major contribution to the perceived accessibility of screen reader users. Our proposed metrics reveal important statistical characteristics of an application's structure (i.e., UI hierarchy, comparable to an HTML DOM tree), which is the graphical user interface equivalent of the application for screen reader users.

Our model contributes to the area of computational interaction for non-visual navigation. We depart from prior work in cognitive modeling (e.g., KLM models [18]) in a way that prior models estimate task completion time as a proxy for the complexity of a task, whereas we estimate the overall complexity of an application in keystrokes, abstracting individual tasks or task-specific parameters (e.g., source, target, objective). As a result, our metrics can be measured without end-user involvement and earlier in the design cycle. Furthermore, developers can plug these metrics into an optimizer to receive recommendations for optimal application structure and optimal UI elements to place keyboard shortcuts. We demonstrate **five** usages (Section 5), including one that suggests the metrics can align with users' perceived accessibility of different applications.

Three properties of our probabilistic model and metrics are:

- **Automatable Measurement:** The number of user actions to navigate between UI elements can be measured solely from the logical representation of the UI in software and a simple model of how screen readers traverse this representation. This can also be measured by end-users without developer involvement or vice versa.
- **Comparable:** Users can compare the perceived accessibility of similar applications of (e.g., text editors, multimedia players) by these metrics. For instance, the complexity metric for Notepad and Microsoft Word is 1.63 and 3.51 keystrokes, respectively. This indicates that Notepad is less complex than Microsoft Word and is perceived as more accessible by users. Additionally, blind users can estimate the efficiency they would gain by learning new shortcuts for an application.
- **Extensible Model:** Our baseline results show that a simple model can be quite informative about the statistical characteristics of an application, but one can incrementally refine these models for more accurate results. For instance, our simple, unweighted model assumes users will navigate from any element to any other with equal probability. We show how to refine this model based on simple survey-like questions (e.g., "How often do you use the `'Format'` menu?") and translate their responses to a *discrete staircase probability distribution* to better capture user behavior. Similarly, by factoring in the role of keyboard shortcuts, one can incrementally improve the fidelity of the results.

The contributions of this paper are as follows:

(1) Identifying key aspects in blind users' perceived accessibility and connecting those to specific, measurable software design choices (Section 3).
(2) Probabilistic interaction models (Section 4.3 and Section 4.4) and three derived metrics (Section 4.5) that can be measured without user or developer involvement, giving the developer and end-user more rapid feedback on the usability of the design and giving the user better insights into the accessibility of a software package.
(3) A technique to model user behavior from survey-like questions and coarse data (Section 4.4).
(4) Five use cases demonstrating how these metrics can be used (Section 5), including an analysis of 11 applications commonly used by blind users (Section 5.1).

2 BACKGROUND AND RELATED WORK

This section presents prior work on perceptions of usability and accessibility, inaccessibility, and user interaction modeling.

2.1 Accessibility, Usability, and Inclusion

Accessibility, usability, and inclusion are closely related concepts in creating products (e.g., software, applications, and web pages) that work for everyone. Traditionally, usability is measured by how much time an average user takes to complete certain tasks (i.e., task completion times), as well as by collecting subjective feedback (e.g., SUS scores [43] and NASA-TLX scores [44]). Developers are motivated to increase the usability of their applications to gain a competitive advantage. Ensuring good usability requires substantial user testing, which is often laborious and expensive. Accessibility, in contrast, is commonly reported by how many accessibility guidelines an application violates (i.e., conformance test). Developers are often obligated to check such violations in order to comply with regulations, such as the U.S. Rehabilitation Act of 1973 and Americans with Disabilities Act [67]. The most widely used set of accessibility guidelines is Web Content Accessibility Guidelines

(WCAG 2.1) [66], which has pushed the use of usability-related criteria, such as effectiveness, efficiency, and satisfaction, in web accessibility [29, 56, 86]. This push is known as usable accessibility [35] or second wave inclusion [56]. It strongly corroborates the theoretical concept of universal usability that accessibility is an integral part of usability and the importance of making a product usable to all users, regardless of their abilities, backgrounds, skills, motivations, personalities, cultures, or ages [62, 63].

Embracing accessibility for a particular user group can improve the overall usability of a product [55, 75, 84]. However, the key challenge is to understand the needs of users with different disabilities [3, 22, 46, 68, 69]. In this regard, web accessibility is a notable exception because a large body of work exists to understand the needs of different users in the web [31, 79]. We complement prior work by understanding the perception of accessibility of *desktop applications* for blind users.

Although developers must adhere to the best practices of user interface design and accessibility guidelines, the effects of these practices and guidelines are not the same. For example, users perceive a website as more accessible if it adheres to the best practices of user interface design [87]. In contrast, only adhering to accessibility guidelines, such as WCAG, has a minor impact on a website's perceived usability [76, 78]. Therefore, instead of pushing a guidelines-based design, prior work recommends combining user-centered and participatory design approaches with conformance testing to achieve usability and accessibility in a product [25]. We follow this recommendation.

2.2 Inaccessibility and Automated Testing

Besides conformance tests, much of the literature on accessibility measurement and testing has focused, rightly so, on detecting inaccessibility, i.e., identifying UI components with insufficient or incorrect accessibility metadata. Common inaccessibility issues include UI elements (e.g., buttons or images) that are not labeled with alternate text or are labeled incorrectly; UI elements that are not discoverable (e.g., by setting an ARIA label incorrectly); or by simply omitting metadata (e.g., remote desktop systems, where the screen reader sees a blank void [5]). Numerous tools (e.g., [34, 50, 58]) exist that implement rule-driven conformance testing to find inaccessible UI elements within an application automatically. Examples of rule-driven tools include Expresso [33] and Robolectic [58] for Android, KIF [42] and EarlGrey [34] for iOS, and Inspect [51] and AccChecker [50] for Windows.

These tools have several limitations [48, 57, 74]. First, the rule-driven tests cannot capture all accessibility issues [48] and are usually incomplete [48]. For instance, Vigo et al. [74] show that, on average, the current popular web accessibility testing frameworks have around 50% coverage, 14-38% completeness, and 66-71% correctness. Second, some accessibility guidelines are open to interpretation and thus challenging to translate into rules [12, 13]. This can be further challenged by a lack of easily quantifiable metrics. Although several metrics exist for the web, these are mostly based on the number of barriers or passes found through an accessibility evaluation [48]. Yan and Ramachandran [83] proposed two additional metrics for inaccessibility: inaccessible element rate to estimate the percentage of UI elements that are inaccessible; and accessibility issue rate to calculate the percentage of the actual number of accessibility issues relative to the maximum number of accessibility issues. Unfortunately, none of these metrics are a good match for evaluating user experience above a base level of functionality. In contrast, our proposed metrics can estimate the perceived accessibility of desktop applications as a proxy for user experience, which can be used during automated testing.

2.3 Cognitive Models in Accessibility

Prior work explored the use of cognitive models [10, 60, 70, 71] as a means for designers to understand the accessibility of user interfaces. For example, Biswas and Robinson proposed a model to simulate how people with low vision and motor impairments interact with graphical user interface [9], which they used to predict task completion times of individuals with a specific impairment [10].

GOMS [19] is arguably the most well-known model to estimate the completion time of a specific task [70], as well as to analyze the complexity of UI elements [38, 39]. Tonn-Eichstädt [70] developed a modified GOMS model [17] for blind users to perform a comparative evaluation of webpage designs. The users performed simple but unfamiliar tasks with a screen reader, which formed the basis of this model. Takagi et al. [64] developed a tool to predict the completion time of expert screen reader users to navigate to an element on a webpage. The completion time includes a summation of the assumed time to speak all page elements that can be traversed en route to the target element. This accounts for individual expertise and navigation strategies (e.g., skipping links and headings in webpages) but not cognitive decision times or other user operations, such as verification of screen reader output and choices between alternative navigation strategies. Therefore, this is not a cognitive model and does not reflect a user's approach to a well-practiced task.

GOMS also paves the way for using other models such as KLM (keystroke-Level Model) [18], which estimates the task execution time for a specific scenario in a given design [41]. Trewin et al. [71] investigated how to model keystrokes of skilled screen reader users using KLM [17]. They concluded that such models could help designers create user interfaces that are well-tuned for screen reader users, without the need for modeling expertise, but also indicated the difficulty in learning KLM. Other researchers also point out cognitive models' steep learning curve, being tedious to use, and error-prone [37].

Both GOMS and KLM require a high-level task to be broken up until the point where each sub-task can be accomplished using one or more predefined operators (either physical or mental). For keyboard-based navigation, the physical operator is a keystroke. The required time for each keystroke is then estimated, along with any delays for the user to think (i.e., mental operator). Different atomic-level tasks and their estimated execution times are available in Kieras et al.'s work [41].

In sum, predicting task completion time is a key focus of prior work. It is also an important usability metric for sighted users who mostly use pointing devices (e.g., mice) to acquire targets and keyboards for entering text (and executing basic shortcuts), but it is not a reliable metric for blind screen reader users who exclusively use keyboards. This is because the task completion time for blind

users depends on the several factors [1]: individuals' expertise with screen readers, their familiarity with the app, navigation strategies supported by the screen readers (e.g., flat or hierarchical [4]), and individuals' the use of shortcuts (e.g., fast versus application-provided [4]).

Compared to GOMS, our model is not suited for estimating task completion times since there is no notion of tasks in our modeling. Instead, we report application-level metrics based on the expected number of keystrokes required to transition between any two UI elements within an app with a probability (either from a uniform or a staircase distribution). We describe the relationship between ours and GOMS modeling in Section 6.2 with an illustration.

2.4 Computational Interaction Models

Computational interaction is a recent development in HCI that is concerned with abstracting the interactions via the use of algorithms and theories (e.g., control theory [54]), data, and mathematical models (e.g., optimization) [54]. Among these, control theory [36] is of special interest to HCI researchers as it helps to model interaction as a continuous event—which is arguably the true nature of interactions [27]. Information Theory-based classic model, Fitts' law [28], is found to be a special use case of control theory. Prior research in computational interaction has mostly focused on optimizing the experience for sighted users, who can observe and interact with the visual cues (e.g., moving target, different font faces, text styles, spacing). However, little to no effort has been given to understanding and optimizing the interactions of blind users. Our probability model can serve as a computational interaction model for non-visual navigation with a keyboard.

3 FORMATIVE STUDY: UNDERSTANDING THE PERCEPTION OF ACCESSIBILITY

To investigate blind screen reader users' general perception of accessibility, as well as variations in their perception across different applications, we carried out an IRB-approved user study. Specifically, we conducted semi-structured interviews, following the recommendations for qualitative interview design [72].

3.1 Research Questions

We structured our questionnaire around 4 predefined research questions (RQ1 to RQ4) and followed up with secondary questions adapted to the participants' expertise, profession, and conversation flow. We describe these questions below:

RQ1: *What are the important qualities of a high-quality, accessible app?* We wanted to know what the participants understood by the term *accessibility* or what qualities of an application contribute to or detract from accessibility. Sample follow-up questions included: (i) What is accessibility to you? (ii) What do you mean when you say that an application is accessible? (iii) What is your view on accessibility? and (iv) What are the characteristics of a highly accessible app?

RQ2: *For commonly used desktop apps, how do you perceive the accessibility of each app?* We asked participants to rate around $10 - 15$ applications on a scale of 1 to 5, 1 being the least and 5 being the most accessible. We also asked them to rationalize their ratings with the use of scenarios and anecdotes. We started with a list of the most commonly used apps in the blind community (e.g., Microsoft Office apps, Calculator, Notepad)—as informed by our prior experience of working with similar participants. This list of applications was updated initially but quickly became stable as the list covered the most frequently used applications of the participants.

RQ3: *What percentage of total functionality on a desktop app do you think that you typically use?* We were purposefully vague in this question because we aimed *to understand the participants' perception of app usage*. Since all participants were experts (self-reported), their perception of app usage could reveal important insight. For better understanding, we specifically asked them about 3 to 4 applications of their choosing; for example: What fraction of all the features in MS Word do you use? The participants were asked for a rough estimate, as this would be difficult to quantify precisely without recording usage data.

RQ4: *What is an acceptable number of keystrokes (in a chain) for you to perform a task?* This question aimed to understand the typical effort (in the number of keystrokes) the participants are comfortable with while issuing a screen reader command. Note that this question was not meant for any specific task (e.g., applying boldface to a text region, increasing text size) but to understand the users' general effort in issuing a command or, inversely, the ease of issuing a command, at a time. Moreover, all of our participants were familiar with application shortcuts for faster navigation. We also asked about their process of learning shortcuts.

3.2 Participants and Recruitment Criteria

We recruited 11 blind participants (8 male, 3 female; details in Table 1) through mailing lists[1]. Some participants had light perception; they all used screen readers full-time. Their ages varied from $33 - 68$ with an average of 45.73. The participants work in varying industries, including Healthcare, Entrepreneurship, IT, and Assistive Technology Training Centers.

Our inclusion criteria included legally blind adults who are experts in screen reading technologies, e.g., power users or assistive technology trainers. We imposed this constraint because research has shown that usability models constructed with expert user behavior can well-approximate outcomes across a range of (visual) tasks and devices for general users [71]. Therefore, we screened participants who considered themselves experts (i.e., expertise was self-reported). The 11 participants in our study had extensive experience using assistive technologies with various applications, and had accumulated knowledge over time about what works and what does not work effectively.

3.3 Interview Method

The interviews were semi-structured and conducted remotely via teleconference (Zoom or telephone). Two researchers administered the study: one conversed with the participant while the other took notes. Each session was audio-video recorded after consent for post-analysis and follow-up.

After verbal consent, we began by asking participants to introduce themselves and tell us a bit about their history of blindness.

[1] freelists.org/list/program-l and nvda.groups.io/g/nvda

ID	Age/ Sex	Profession	Light Perception	Screen Reader Used	Expertise
P1	36/M	Healthcare	Yes	JAWS, NVDA, VoiceOver	Expert
P2	39/M	Entrepreneur	No	NVDA	Expert
P3	35/M	Graduate Student	No	JAWS, NVDA	Expert
P4	45/M	Entrepreneur	Yes	NVDA, System Access	Expert
P5	48/F	University Disability Center	No	JAWS, NVDA, VoiceOver	Expert
P6	40/M	IT Instructor	Yes	JAWS, NVDA	Expert
P7	33/F	Assistive Technology Trainer	Yes	JAWS, NVDA, VoiceOver	Expert
P8	38/F	Assistive Technology Trainer	Yes	JAWS, NVDA, VoiceOver	Expert
P9	67/M	Engineer	No	JAWS, NVDA, VoiceOver	Expert
P10	68/M	Assistive Technology Instructor	No	JAWS, NVDA	Expert
P11	54/M	Software Developer (Python)	No	JAWS, NVDA	Expert

Table 1: Participant demographics in the formative study

Next, we proceeded with our research questions, described earlier in Section 3.1.

Each session lasted an hour, and participants were compensated with a $30 (USD) Amazon or other form of e-gift card. Each interview culminated with participants providing suggestions and recommendations.

3.3.1 Collection of Application Usage Data. After asking our primary research questions (RQ1 to RQ4), we collected data about the usage of different UI elements (e.g., menus, ribbons, buttons) of applications. More specifically, we asked participants to rate their likelihood of using a given UI element (e.g., a menu item) on a scale of 0 to 4, where 4 is *'very frequently'*, 3 is *'frequently'*, 2 is *'sometimes'*, 1 is *'rarely'*, and 0 is *'never'* (e.g., 'never used it' or 'not applicable').

We collected menu-specific information (e.g., menus, ribbons, sub-menus) instead of task-specific information (e.g., button to make selected text boldfaced, increase the font size). This is because users are more likely to be familiar with the name of the menu hierarchy (e.g., File, View, Format) than an individual item within a menu.

We started with major menu groups of visually adjacent functionality (e.g., a top-level ribbon tab in Microsoft Office), and, for groups that are used, we move down and ask the same question about sub-groups or individual elements. This simple, interview-based approach allowed us to collect coarse usage data without the disruption of installing a keylogger or other intrusive software on their personal devices. We used this data in our modeling (described later in Section 4.4).

3.3.2 Data Analysis. Following the completion of the first three interviews, the researchers analyzed the transcripts using an iterative coding process with initial coding and identified concepts [15], categorized them, framed new questions for subsequent interviews, and updated the concept list.

3.4 Findings

We present the findings of our study in the following discussion.

3.4.1 Important Qualities of Accessible Applications. Participant P8, who provides Assistive Technology training to other people stated three important characteristics of accessible applications: *basic, understandable, and explainable.* In her words:

> *"An application should be basic, understandable, and easy to describe to others. By basic, I mean a home screen that is not cluttered with unnecessary buttons and stuff. If you understand something, you can learn it easily and if it is easy to describe, you can pass on what you know easily..."*

Participants P9 and P10 shared the same view as P8. Participants P4, P6, and P11 all stated that they use a screen reader to test the accessibility of an application.

According to P6:

> *"I usually rely on JAWS or NVDA to decide it for me. If JAWS starts saying "blank, blank, blank..." I know I have an inaccessible app in my hand."*

Ease of navigation among UI items was another aspect that came repeatedly during the interview. All of our participants P3, P4, P6, P8-P11 mentioned that they prefer Notepad over Microsoft Word for text editing purposes only because navigating in the former application with fewer options feels much easier to them. The calculator was another application that received multiple mentions as an app with easier navigation.

3.4.2 Accessibility and Usability. When asked about the relationship between accessibility and usability, three participants (P3, P5, and P7) mentioned that accessibility does not always guarantee the usability of an application. Apropos of that, P3 offered a different angle.

> *"To me, if the performance difference between a visual and a non-visual user is negligible in an application, I would consider it accessible... Usability is person-specific. A perfectly usable interface for someone may not be comfortable for others."*

3.4.3 Accessibility and Independence. Another term that came up often during our interviews was *independence*. Participants P3, P6, P7, P8, P9, P10, and P11 mentioned independence as a concept closely related to accessibility. In their opinions, key criteria for evaluation of accessibility are whether users feel comfortable using

and learning the software on their own, as well as teaching other people themselves. In the words of P7:

> *"...if someone is able to approach, learn, and use technology with comfort, that is what accessibility is to me. ... accessibility will lead to independence. "*

3.4.4 Accessibility and Reliability. When asked about what makes an application accessible, P7 stated that *reliability*, or consistent behavior, is essential. Software often has context-dependent behavior, such as a keyboard shortcut only working properly when the user is performing a relevant task. Worse, application context cues may be only visual; in other cases, the application may have simple bugs or omissions that a sighted user can work around with a mouse click. For a blind user, however, these shortcuts also serve a critical role in navigation, and inconsistent or unreliable behavior disrupts navigation. To P7, the reliability of a shortcut means that the shortcut always behaves the same way, regardless of the application state. For instance, P7 mentioned that the desktop version of Outlook has a shortcut that is reliable, but in the web version, the behavior depends on the current focus. P7 also echoed common frustrations related to inaccessibility, such as missing and wrong labels, and unlabeled pictures, which in turn undermine independence. According to P7:

> *"Accessibility also means reliability to me. The labels, actions, and results of those actions should be reliable. If I am pressing a shortcut key and nothing is happening, there is nowhere else to go."*

3.4.5 Learning a New Application and shortcuts. Commensurate with the earlier finding regarding independence, we find that independent exploration is key to how participants learn new software. Most of our participants (P3, P4, P5, P7, P9, P11) prefer to start with keyboard commands, a help menu, or documentation and see whether the application is useful. If they find the software useful after the initial exploration, they are likely to continue learning and exploring. In the words of P3:

> *"If the application is not accessible, I don't bother. If it is, I try it by myself at first. Then, I use the documentation or do a google search to find more features..."*

Some participants also mentioned that they often attempted to transfer the knowledge of knowing one app to a new app when learning. For instance, when asked how they would learn new software, say Microsoft W (a made-up name, we are not affiliated with Microsoft), P7 creatively reacted to this question by describing her strategy:

> *"Well, you already gave me a hint. If it is Microsoft, I can use my awful past knowledge – I'll be automatically teaching myself that it definitely has a file menu, it likely has a [you know] taskbar, [you know] the ribbons at the top, probably. It supports Alt and arrow commands."*

All of our participants were familiar with application shortcuts and the significant improvements in user experience that shortcuts can bring. However, most of our participants were reluctant to invest the effort for a variety of reasons, including the stress and time required for memorization. For example, P2 mentioned that he only memorizes shortcuts for a feature he frequently uses, and P3 avoids using shortcuts altogether.

A noteworthy exception was P6, who believes memorizing shortcuts for faster navigation is worthwhile. He reported that he knew about 50 shortcuts for MS Word alone, and even created tutorial audio clips to help others memorize shortcuts. Because of his profession (IT Instructor), he collaborated with many sighted and blind computer users. Memorizing many shortcuts allowed him to perform a task faster to be in sync with his sighted collaborators.

3.4.6 Perception of Application-Specific Accessibility. We asked each participant to rate the accessibility of commonly used applications on a scale of 1 to 5, where 1 is the least, and 5 is the most accessible.

The participants consistently rated `Notepad` and `Calculator` as the most accessible applications. `Visual Studio` scored the lowest, and participants shared many accessibility issues with Visual Studio.

For `Microsoft Excel`, almost all the participants agreed that cell-to-cell navigation is simple but tiresome. Nonetheless, they rated Excel well overall because the rich, comprehensive features for data manipulation make Excel a useful application.

A color chart of all the applications with their ratings is shown in Figure 1. A darker tone is used here to indicate larger counts, while lighter tones indicate smaller counts. Not all applications have 11 ratings, as we omitted scores from participants that were not familiar with a given application, or for applications added after that interview.

We also note that one source of divergent scores among our participants fell along whether they could perceive light or color. Some participants with light perception were able to get some visual cues, improving their overall experience relative to users without light or color perception. For applications that do not require visual cues for navigation, such as the calculator or the notepad, there was little disagreement.

3.4.7 Usage of Application Functionality. In order to understand whether all application functionality was equally useful, we asked the participants what percentage of all the features they use in an application like MS Word. Then, we repeated the same process for a few other applications, including Notepad and MS Excel. Responses for Word varied widely, from 10–70%, although most reported 10%, as shown in Figure 2a. For MS Excel, the number was about the same. For Notepad, however, the numbers jumped around from 50–90%, given its very limited set of functionalities.

3.4.8 Acceptable Number of Keystrokes to Perform a Task. Most of our participants report that two or three keystrokes are acceptable for a given task, as shown in Figure 2b. After five keystrokes, the experience becomes cumbersome. In the words of P6:

> *"... ... 4 is also acceptable. 5, I would say, is pushing a bit, and any more than that, I might as well just use the menu. Because, something that can be done with one click, should not take 5 or more keystrokes ... ..."*

Application	Participant Rating				
	1	2	3	4	5
Notepad	0	0	1	0	8
MS Word	0	0	3	4	4
MS Excel	0	1	1	3	6
Calculator	0	0	0	2	5
Acrobat Reader	0	2	5	2	1
Outlook	0	1	1	5	1
Zoom	0	0	3	2	4
VLC Media Player	0	1	0	1	0
Audacity	0	0	1	1	1
MS PowerPoint	0	2	2	4	1
Visual Studio	1	2	1	0	0
Google Meet	0	0	0	1	2
FileZilla	0	0	0	2	0

Figure 1: Accessibility scores of commonly used applications as rated by participants – one being the least and five being the most accessible. The shade of a cell indicates the frequency of response, which is also shown numerically within a cell.

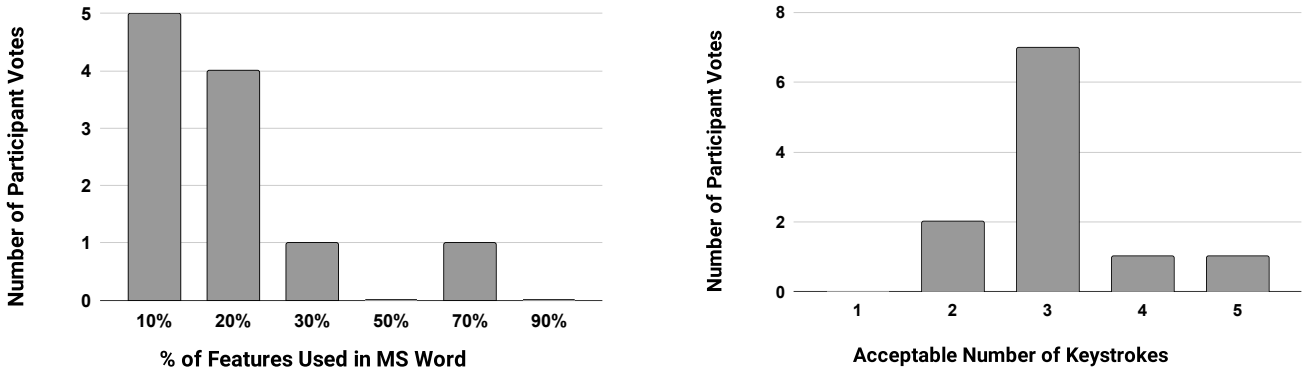


Figure 2: Results from the formative study.

One exception was P3, an advanced MS Word user. For P3, five keystrokes are comfortable, and he was willing to go as high as $10 - 15$ keystrokes if the task at hand was worth it, although this was beyond his comfort level, and he was quick to note that he considered himself an outlier among users with blindness.

3.5 Perception of Accessibility and Usability in Screen Reader-Mediated Interaction

We now discuss our findings. Our study revealed that the participants' perception of accessibility was intertwined with usability, similar to the concept of universal usability [62, 63]. They also associated accessibility with other broad concepts, such as independence, the reliable and deterministic behavior of keyboard shortcuts, and knowledge transfer.

The frequent association of "independence" with accessibility is unsurprising because "independence" is central to blind users [45], and it is considered the primary goal of any assistive device [80]. Some participants extended the notion of "independence" to "ease" and "comfort", such as the ease of understanding an app, the ease of learning, and the ease of describing an app to others. Ease and comfort are tenets of usability, and the fact that they asked for these qualities suggests that AT-mediated interactions lack usability.

Another perception of accessibility is the deterministic application behavior and the reliability of keyboard shortcuts. This is insightful because many user interactions are not deterministic. For example, re-organizing menu items based on usage is not deterministic; it forces blind users to update their mental map each time. Sometimes, navigating between two UI elements is not reversible (e.g., moving from A to B takes 4 left arrows, but moving from B

to A takes more or fewer than 4 right arrows). These are bugs that often remain undetected during system testing.

The ability to transfer the knowledge of knowing one application to another came up often. This concept is broadly connected with the uniformity of interaction, a known usability principle. However, prior work [4] indicates that transferring knowledge is a major challenge for screen reader users. For example, switching from one screen reader to another (e.g., from JAWS to NVDA), from one application to another (e.g., from MS Word to Open Office), or from one platform to another (e.g., from Mac to Windows) is all but difficult. Therefore, although the underlying usability principle is a known art, its application for blind users is noisy and unforced.

Further, most participants were aware of the upper limit of accessibility: when *"the performance difference between a visual and non-visual interaction is negligible in an application"*(P3). This is interesting because most accessibility researchers evaluate the performance of their system or prototype with a set of participants having the same disability (e.g., blindness) [2]. But our findings indicate that researchers should compare their system's performance to those without disabilities to set the bar for future development.

Understanding these nuanced relationships between accessibility and usability in AT-mediated interaction is one of the contributions of this paper.

4 PROBABILISTIC MODELS AND METRICS ESTIMATING PERCEIVED ACCESSIBILITY

The previous section highlights how blind users organically associate usability with accessibility and how their assistive technology (AT)-mediated interaction lacks well-known usability principles. Take the inability to transfer knowledge as an example. For sighted users, transferring knowledge is facilitated by the graphical user interface, which creates a de facto standard for the application interface that gives sighted users ease of learning, ease of use, and ease of transfer of knowledge gained from using one application to another because of a consistent look and feel [73]. Often, this consistency is enforced by the systems (e.g., GUI libraries, frameworks, or Operating Systems).

However, no such enforcer exists for blind users. Their primary mode of interaction is keyboard-based shortcuts that vary from application to application. This is analogous to forcing sighted users to interact with applications only via command line interfaces, where the command set for different applications differs.

Therefore, the key usability challenge in AT-mediated interaction is how users navigate and interact with a graphical user interface—approximated by an AT, using a keyboard. Since visual proximity in a graphical user interface does not necessarily translate to a simple navigation path via the keyboard and audio; decisions such as how elements are logically grouped or the selection and placement of keyboard shortcuts are critical to the experience of blind AT users.

This section first explains how we model assistive technology navigation through a given application UI. Then, we use probabilistic modeling to replicate UI navigation. Finally, we derive three metrics using this modeling; namely, ***Complexity***, ***Coverage***, and ***Reachability***, which can reveal important statistical characteristics of an application relating to its usability for blind AT users.

4.1 Modeling Application Navigation

To build a navigation model for an app, we start by extracting a logical model of the UI using standard accessibility APIs. Accessibility APIs expose a logical representation of an application's user interface as a tree of UI elements, similar to an HTML Document Object Model (DOM) tree; for brevity, we use the term DOM in this paper to describe the OS's logical representation of the UI.

To extract an application's DOM, we developed a tool in `C#`, similar to `Accessibility Insights`[2]. Internally, this tool utilizes `UI Automation` API [52] to register a hook onto the target application and extract the metadata associated with each UI element in that app. Once extracted, our tool exports the constructed DOM as an XML file. We then calculate our metrics on this XML file. The DOM extraction process is mechanical and requires only a few clicks. We plan to make our tool open-sourced and release it in the future.

For a graphical user interface, this DOM is ultimately rendered into a bitmap, whereas for an assistive technology, such as a screen reader, the DOM is traversed and converted to speech. In the case of an AT, we need to posit the notion of *focus* or a *cursor*, which identifies the current element the AT is rendering (into speech, for a screen reader). The AT user must move through the DOM, one element at a time.

As an example, Figure 3 shows a screen capture of Microsoft Word, with a few visual elements under the "Home" ribbon highlighted and labeled with numbers. Figure 4 shows a simplified DOM representation of the labeled elements, where the "Home" ribbon tab (label 1) is a parent of the other elements, which are siblings within the hierarchical representation. Note that a "down arrow" navigation (from 1) always goes to a single element (2), and then the user must navigate laterally to other siblings. From any of the elements 2..5, the user can type the "up arrow" to move back to 1, creating an asymmetric path, indicated by unidirectional arrows in the figure. We note that the organization of elements in the DOM may not match the visual layout.

The user of a screen reader uses a keystroke to navigate from one node in the navigation model to another. In moving from a DOM to a navigation model, we remove some hidden elements from the DOM that would not affect navigation, such as "container" elements that logically group several UI elements, but do not contain any information to present or action to take. Screen readers implement similar heuristics, such as flattening containers that have only other containers as children.

In practice, keyboard shortcuts can also add edges to the navigation model. A keyboard shortcut can jump to a given UI element, reducing the number of keystrokes required to navigate from one element to another. Shortcuts are also unidirectional; if the user is on element A and uses a shortcut to jump to B, the application may not provide a matching shortcut back to A. We add edges to our navigation model for all keyboard shortcuts. In short, our navigation model introduces one directed edge per transition of focus, and a node is a UI element.

[2]https://accessibilityinsights.io/docs/en/windows/overview/

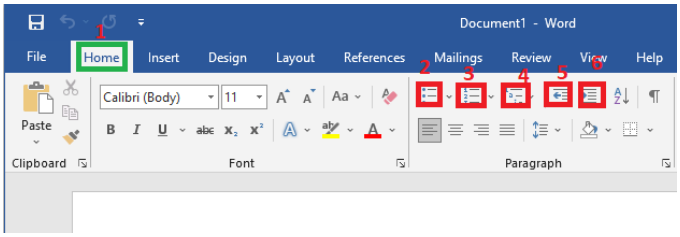


Figure 3: A screenshot of MS Word, with a few clickable elements labeled with numbers (1 to 6).

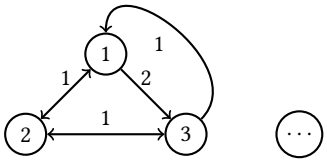


Figure 5: The cost model for nodes 1, 2, 3 from Figure 4 (top-right). Nodes are the same UI elements as in the DOM, but the edges represent the smallest cost to move between a pair of elements. In practice, the graph is fully connected (and simplified for clarity).

4.2 Cost Model

The next step is to build a cost model for each potential transition. *Cost* is simply the shortest path between two nodes in the navigation graph, represented in the number of keystrokes. This includes up/down moves, sibling/level moves, and also shortcut moves if applicable.

For example, let us consider that our source is node 1 and our destination is node 5 in Figure 4. The journey from node 1 to node 5 would have the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$. Here, the move $1 \rightarrow 2$ is a down move and the rest are sibling/level moves. One down move and three sibling moves make the total cost of this journey $1 + 3 = 4$. In the case of keyboard shortcuts, we model a move as the total number of keystrokes needed for that (e.g., 2 for "Alt+H").

Figure 5 shows the cost model with nodes 1, 2, 3 from Figure 4. The model is represented using a fully connected, weighted graph, where each edge's weight is the minimal cost to travel between two nodes (i.e., UI items.)

4.3 Transition Model 1: Uniform Transition Probability

We model UI navigation as a random process in which a user navigates from one UI element to another at discrete time steps. If an application has n interactable UI elements, at each time, the AT's cursor or focus can be on one of these n elements and can transition to any of the other $n-1$ elements. To represent the transition model, we build a fully-connected, weighted, directed graph (G) having n nodes and $n * (n-1)$ edges. The weight of each edge in this model is the probability of that transition taking place, as explained below.

As a simple baseline, we start with a model where each transition is equally likely. In other words, the probability of an edge, e being followed is:

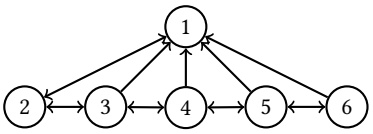


Figure 4: A simplified DOM representation of the labeled UI elements in Figure 3 (left). The Home ribbon tab is a parent of the other elements, which are siblings in the hierarchy.

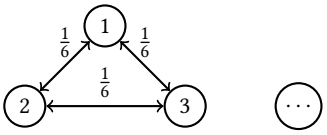


Figure 6: The first transition model for nodes 1, 2, 3 from Figure 4. The edges represent the uniform transition probability (1/6) to move between a pair of elements. For simplicity, we only show 3 nodes with 3 directional edges (i.e., 6 unidirectional edges). In practice, the graph is fully connected containing a total of 6 nodes and 36 unidirectional edges, with an equal transition probability of 1/36 for each edge.

$$P_e = \frac{1}{n * (n-1)} \tag{1}$$

Continuing the Microsoft Word example, Figure 6 presents a transition model for nodes 1, 2, 3 from Figure 4. With $n = 3$, we get $P_e = \frac{1}{3*2} = \frac{1}{6}$. In other words, the probability of each transition is $\frac{1}{6}$, as shown in Figure 6.

4.4 Transition Model 2: Modeling Probabilities from User Interviews

Recall that during our interview, we collected coarse data about the participants' usage of different features in an application (Section 3.3.1). We can use this data to build a user-specific transition model, where each edge (i.e., transition) in the fully-connected graph may have different weights (i.e., probabilities).

During data collection, recall that the participants rated their likelihood of using different UI elements (e.g., menus) in an application on a scale of 0 to 4 (4 is "very frequently", 3 is "frequently", 2 is "sometimes", 1 is "rarely", and 0 is "never"). With this interview data, we can then place every UI element in one of five buckets, corresponding to the average usage frequency reported by participants. We model a five-node "graph of graphs", where the five buckets are each a node in a fully-connected transition graph (Figure 7), but each bucket also contains a transition graph of all the nodes within the bucket (Figure 8). Note that if there are n nodes in the transition graph and 5 buckets in total, and a bucket B_k contains n_k nodes, then $n = \sum_{i=0}^{i=5} n_i$.

For each transition, then, one must consider whether it is an intra-bucket or inter-bucket transition. As a simplification, we treat all items within a bucket (e.g., all the edges in Figure 8) as having equal probability. Since transitioning to a node in the never

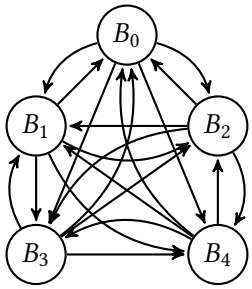


Figure 7: A complete graph-of-graph with 5 preference buckets (B_0 to B_4). Each bucket contains all the nodes with identical personalized probability values (e.g., 0.1 for all nodes in B_1). Only inter-bucket transitions are shown. Intra-bucket transitions are shown in Figure 8.

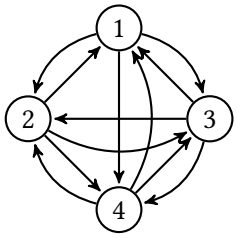


Figure 8: A complete graph showing the intra-bucket transitions for bucket B_i. For an illustration, we assume this bucket contains four nodes: 1, 2, 3, and 4.

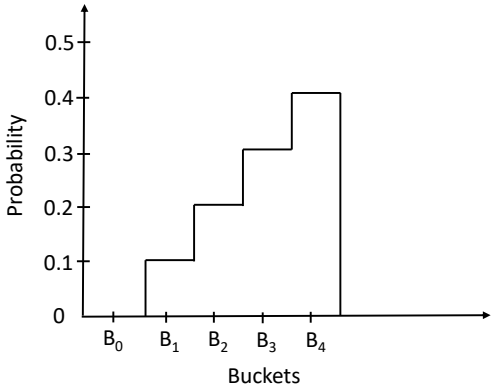


Figure 9: A staircase probability distribution for 5 preference buckets (B_0 to B_4). Nodes or UI items in bucket B_4 are most likely to be visited from a random node.

bucket (B_0) is extremely unlikely (i.e., 'never'), we consider the transition probability to a node in this bucket, $P(B_0)$ is infinitesimally small, i.e., $P(B_0) \approx 0$. Hence, we ignore bucket B_0 from the further calculation.

Next, we model the probabilities for the other four buckets ($B_1 - B_4$) as $P(B_k) = \frac{k}{10}$, modeling it after a discrete staircase probability distribution [23], with hyperparameters $a = 1$, $b = 4$, and $n = 4$. This is shown in Figure 9. The hyperparameter n represents the number of buckets, which is 4 in our formulation (e.g., B_1 to B_4). The values $a = 1$ and $b = 4$ indicate that the likelihood of transitioning to a node in the `highly-frequently` bucket (B_4) is 4 times than that of a node in the `rarely` bucket (B_1). Derivation of $P(B_k)$ is shown in Appendix A. Now, we can calculate the probability of an edge, e (which is from bucket B_k) using the following formula:

$$P_e = \frac{P(B_k)}{n_k * (n_k - 1) + n_k * \sum_{i=1}^{i=4, i \neq k} n_i} \tag{2}$$

Here bucket B_k contains the destination node of the edge e, and the denominator is the total number of edges incoming to nodes in B_k. Here, $n_k * (n_k - 1)$ is for the intra-bucket edges and $n_k * \sum_{i=1}^{i=4, i \neq k} n_i$ for the inter-bucket edges. We show the results of this user-specific usage model in the preliminary feedback section (Section 5.1.4).

We selected the staircase probability distribution because it is a simple and sufficient match for a discrete preferences questionnaire. This approach also leverages the intuition that finer gradations of user probability are unlikely to affect the overall result. This approach to interviewing participants also matches the logical organization of a given piece of software, quickly dismissing swaths of functions that users ignore, without losing outliers.

4.5 The Metrics

We propose three new metrics that we calculate using our cost and transition models. This subsection presents these metrics for estimating blind users' perceived accessibility, along with their significance, mathematical formulae, and statistical meanings.

4.5.1 Complexity. We define **complexity** as the expected number of keystrokes required to navigate from one node to another in an application. We calculate this by the simple formula of mathematical expectation:

$$Complexity = \sum_{e=1}^{e=N} P_e * C_e \tag{3}$$

where $N = n * (n - 1)$ represents the total number of edges or possible transitions, $\{C_1, C_2, C_3, ..., C_N\}$ represents the costs for all N edges (comes from the cost model), and the probabilities $\{P_1, P_2, P_3, ..., P_N\}$ (comes from the transition model) represent the likelihood of a transition.

Complexity captures the difficulty perceived by a user while navigating an application. The higher the expected number of keystrokes for a transition in the application, the more tedious the user experience.

4.5.2 m-keystroke Coverage. We define **m-keystroke coverage** as the percent of possible transitions that can be completed in m or fewer keystrokes. For example, our findings show that users consider three to four keystrokes are an acceptable target, although two to three keystrokes are desirable (see Section 3.4.8). As such, the 3-keystroke coverage would indicate what fraction of the edges in the transition graph have a cost (i.e., weight) of three or less. Higher m-keystroke coverage is better for the end user. We also use the term **coverage** more generically.

4.5.3 x%-reachability. We define **x%-reachability** as the value of m for m-keystroke coverage that would cover x% of the possible transitions in the application. The primary goal of this metric is to understand how many keystrokes the users may need when they want to use a certain percentage of all the features in an application. As reported in §3.4, users only use a very small portion of the available application options, shown in Figure 2(a). Thus, it would make sense to investigate how many keystrokes it would take if we were to cover 10%, 20%, or 50% of all the nodes. The lower the value of x% Reachability, the better the overall navigation complexity for the user. We also use the term **reachability** more generically.

Capturing the Value of Learning Shortcuts. Some of the nodes in an application DOM have shortcuts. When these nodes are the destinations of a transition, the cost is $1 - 3$ depending on the number of keystrokes associated with the shortcut. Shortcuts have a steep learning curve, and generally require the user to memorize them. If the user is not aware of an existing shortcut, they must traverse the application UI in the traditional way using up, down, and sibling moves.

In order to capture the potential gains from learning these shortcuts, we calculate and compare the complexity, coverage, and reachability of an application with and without shortcuts. Of course, the measurements without shortcuts are expected to be worse. For example, it would take *nine* keystrokes to apply boldface to a piece of text in MS Word (Figure 3). With shortcut ($Ctrl + B$), however, it would take just *two* keystrokes.

However, a significant difference between a measure with and without shortcuts indicates that the user can improve their experience by learning these shortcuts. This metric is most relevant when the scores without shortcuts are poor; adding shortcuts to an application with already good metric values will only see diminishing returns.

5 USE CASES OF THE PROPOSED METRICS

In this section, we demonstrate how screen reader users, application developers, and accessibility practitioners can use our metrics.

5.1 Use Case 1: Benchmarking Commonly Used Applications

This section presents the benchmarking results (in terms of the proposed metrics) for 11 applications, followed by validation via a follow-up discussion with study participants. We selected the applications based on the process described in Section 3.1 (RQ2).

5.1.1 Complexity of Commonly Used Application. Complexity measures the expected number of keystrokes to move from one UI element to another in a given application (§4.5.1). Figure 10 presents complexity metrics for 11 different applications. For each application, we report data with and without shortcuts (the difference indicates how much the user experience depends on learning the shortcuts). One can think of the two bars as bounding the range of accessible user experience—ranging from no knowledge of any shortcut to perfect mastery of all shortcuts. The reality is likely in between for most users.

Among the lowest complexity applications, when factoring in shortcuts, are Calculator, Notepad, VLC Media Player, and MS Word; among the highest are Adobe Acrobat Reader, Audacity, and MS PowerPoint. MS Excel is a special case, in that most of the complexity comes from traversing 250 different cells in our sample input spreadsheet, for which arrow keys are commonly used.

In terms of gains from shortcuts, we find applications that placed shortcuts near difficult-to-reach UI elements fared best, such as MS Word, Notepad, Outlook, and Visual Studio.

5.1.2 3-Keystroke Coverage of Commonly Used Application. 3-keystroke coverage is the percent of total transitions a user can complete in up to 3 keystrokes. We selected 3 as a target based on input from the user study (Figure 2b). The results are presented in Figure 11.

These results indicate that most applications have extremely limited functionality within users' comfort level, without the aid of shortcuts. In most cases, shortcuts make more than half of an application's content reachable within the three-keystroke threshold. For MS Word, the coverage of 82.52% is well above the 10–20% estimate of functionality from our study participants. The same is true for Notepad as well. Both Excel and Acrobat Reader failed to meet this target, but as previously mentioned, Excel is a special case.

5.1.3 20%-Reachability of Commonly Used Application. 20% reachability indicates the number of keystrokes it would take to cover 20% of all possible transitions in an application. We selected 20% as a reasonable bound based on current usage by our sample group as seen in Figure 2a, where the average usage is 21%.

Figure 12 presents reachability metrics for our 11 test applications. Among the 11 applications, 10 of them realize 20% reachability within three keystrokes using shortcuts—a threshold most participants considered acceptable. However, we hasten to note that all of these depend on shortcuts; only the calculator app meets this goal with no shortcuts.

5.1.4 ***Preliminary User Feedback on Metric Values.*** To understand whether the values of our metrics for different applications make sense, we conducted a brief follow-up interview with the same set of participants (demographics in Table 1) remotely over Zoom and phone calls.

In each session, we first reviewed the accessibility ratings (on a scale of 1 to 5) they provided earlier for different applications (Section 3.4.6) and asked whether they wished to adjust any ratings upon further consideration. We then presented our analysis of different applications with our metrics (described in earlier subsections) and explained how to interpret our metrics for an app. We asked participants whether the values of these metrics aligned with their experience.

More specially, we asked two main questions in the follow-up:

- *On average, a task in application **X** takes **k1** and **k2** keystrokes respectively without and with shortcuts. Do these numbers match up with your experience?* (**Complexity**)
- *Without using any shortcut on application **Y**, you would need **k1** keystrokes to perform a task. With shortcuts, the number is **k2**. Does this change your view on learning shortcuts in application **Y**?* [In most cases, **k2** was significantly lower than **k1** as shown in Figure 10] (**Incentives for learning shortcuts**)

Feedback on Complexity Analysis. Most participants agreed that the complexity metric captured each application's usability and accessibility, and tracked their overall experience. For most applications, including Notepad, PowerPoint, and Outlook, all of our participants agreed that the numbers were representative and made sense.

In the case of Word, Participants P1, P3, P4, P5, P6, P7, and P8 agreed with our findings. To enhance usability, P8 recommended limiting the usage of shortcuts to only necessary cases and utilizing helpful mnemonics. Overusing shortcuts can make them difficult to remember, as suggested by P8. P3 said of Microsoft Word's scores:

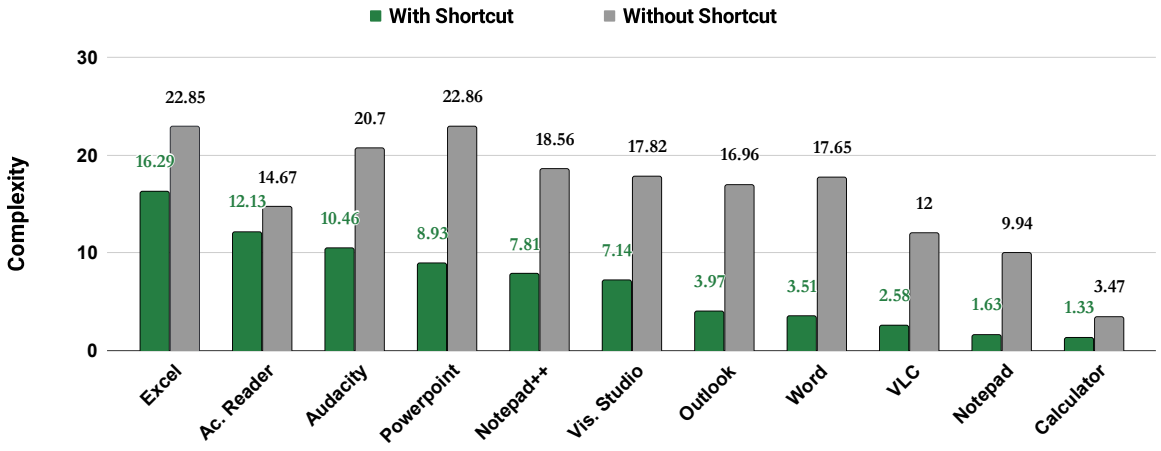


Figure 10: Complexity of Different Applications With and Without Shortcuts (the lower, the better).

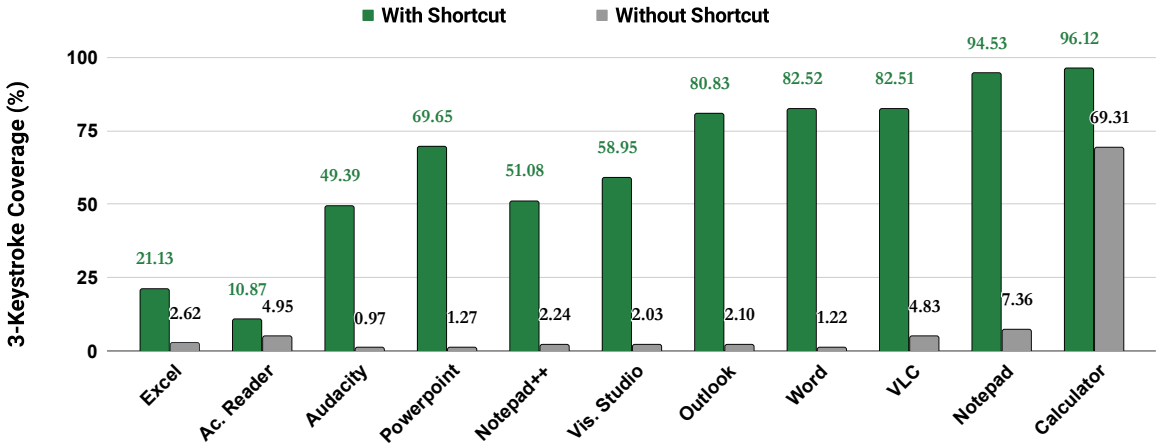


Figure 11: 3-Keystroke Coverage of Different Applications With and Without Shortcuts (the higher, the better).

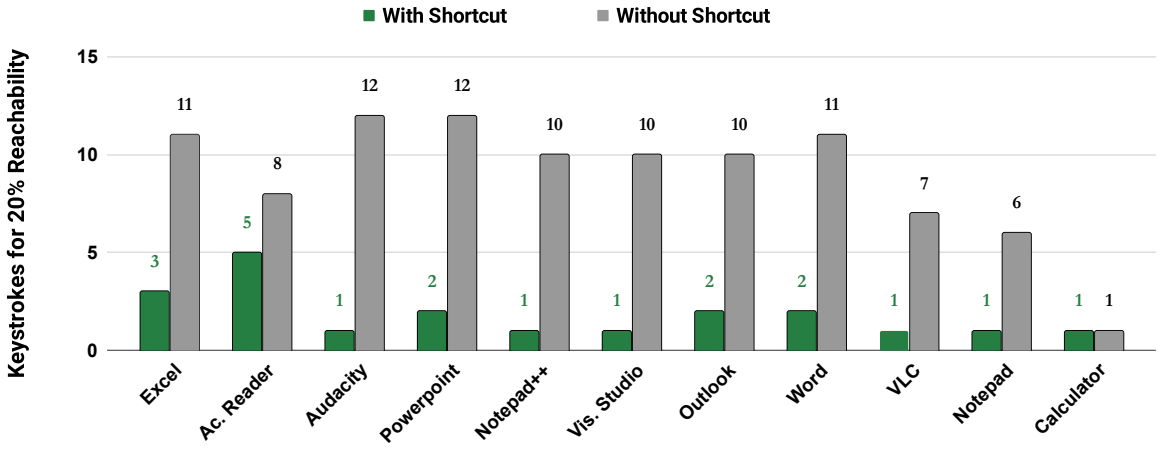


Figure 12: Required Keystrokes for 20%-Reachability of Different Applications With and Without Shortcuts (the lower, the better).

> *"...for Word? Makes absolute sense, both numbers. They have huge objective insights and are very believable."*

Microsoft Excel is an unusual case, and our scores received more nuanced feedback. Our test scenario involved a spreadsheet with 250 cells, which did not have shortcuts for navigation. Participants P1, P3, P6, and P7 felt Excel's complexity scores were representative, whereas P5 and P8 found the complexity score with shortcuts too high. In P7's words:

> *"...22 and 16 keystrokes, yeah; that is accurate, sounds like Excel."*

Feedback on Learning Shortcuts. Next, we revealed complexity data in Figure 10. Given that applications, such as Microsoft Word, have a large difference in complexity with and without shortcuts, we asked whether this information increased their interest in learning more shortcuts. Participants P3-P8 all expressed increased motivation to learn new shortcuts in their commonly used applications. One exception was P2, who still believes that memorizing shortcuts is more trouble than it is worth; he would prefer applications to reduce navigation complexity by personalizing menus based on a given user's usage patterns, similar to what Gazos et al. did in Supple [88].

Modelling Probabilities from User's Preferences. One participant (P4) believed the scores of 3.5 and 17.65 for MS Word were higher than his expectation. To investigate the scenario, we asked him how he uses MS Word and found that he rarely visited UI elements that were *costly to reach* in terms of required keystrokes. Recall from Section 4.4 that we can incorporate user preference-specific usage in our transition model. As such, we completed the questions discussed in Section 4.4 and built a more personalized transition probability model for the participant. As a result, the complexity scores for Word dropped to 1.95 and 16.17 with and without shortcuts for him, respectively. This explains why he was expecting a number lower than 3.5—his usage of MS Word does not involve visiting too many costly-to-reach nodes. This also indicates our model's ability to incorporate individual usage patterns and preferences.

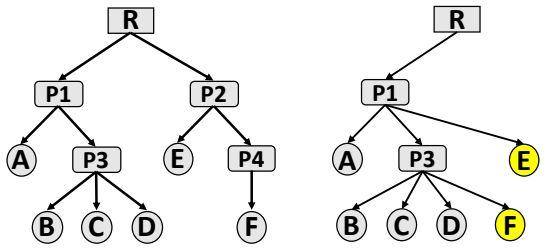
Summary. This follow-up study indicates that the complexity metric accurately quantifies aspects of the AT user experience that could previously only be described qualitatively. The analysis of these metrics, as well as reachability and coverage, can give further insights to the developer about how to improve the usability of an application. The study demonstrates both that these metrics are grounded in feedback from study participants, and how one can incrementally refine the model of user behavior to capture unusual cases and improve the accuracy of the results. Feedback from the follow-up interview suggests that such metrics can benchmark different apps; and, if validated with more users, could be a valuable community resource for blind users.

5.2 Use Case 2: Optimizing UI Hierarchy for Screen Reader-Mediated Interaction

How UI elements are organized in the DOM tree can influence navigational complexity. Developers can use one or more of our metrics (§4.5) as optimization functions to compare different UI layouts [88]. Further constraints, such as the number of newly allowed shortcuts or the number of structural changes, can also be placed. The following examples demonstrate how UI hierarchy optimization can be performed using our metrics.

Goal	Constraints
Minimize Complexity	i) Number of changes, N <= 2
	ii) No new shortcut allowed

Table 2: (Example-1) Goal and constraints for Figure 13



(a) Original Configuration (**complexity = 3.37**)

(b) Optimized Configuration (**complexity = 2.23**)

Figure 13: (Example 1) UI hierarchy optimization obliging the constraints in Table 2. The letter R stands for the root of the hierarchy. P1-P4 stand for the intermediate non-leaf nodes that we ignore in our computation. A yellow shade inside a node indicates that its position in the UI hierarchy has been changed.

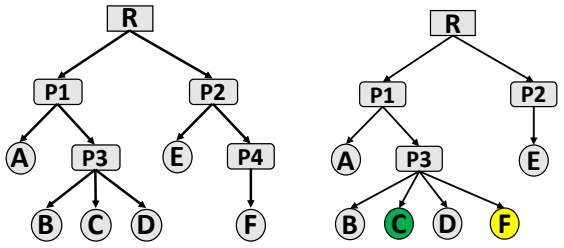
Edge	Original Cost	Optimal Cost	Edge	Original Cost	Optimal Cost
<A,B>	2	2	<D,A>	2	2
<A,C>	3	3	<D,B>	2	2
<A,D>	4	4	<D,C>	1	1
<A,E>	3	2	<D,E>	4	2
<A,F>	4	5	<D,F>	5	1
<B,A>	2	2	<E,A>	3	2
<B,C>	1	1	<E,B>	4	2
<B,D>	2	2	<E,C>	5	3
<B,E>	4	2	<E,D>	6	4
<B,F>	5	3	<E,F>	2	5
<C,A>	2	2	<F,A>	4	2
<C,B>	1	1	<F,B>	5	2
<C,D>	1	1	<F,C>	6	2
<C,E>	4	2	<F,D>	7	1
<C,F>	5	2	<F,E>	2	2

Table 3: (Example-1) Summary of costs for all transitions in Figure 13

Example 1. Here, the original hierarchy is shown in Figure 13a. The goal of the optimization task is to minimize the navigation complexity of the hierarchy with no more than 2 structural changes (e.g., changing the parent of a leaf node) without introducing any new shortcuts. The goal and the imposed constraints are present in Table 2. We consider only the leaf nodes (marked in a circle in Figure 13) as valid sources and destinations for navigation, as mentioned in our model description. As such, the root node R and the intermediate parent nodes P1-P4 are not in our consideration. When the above optimization function (goal) and constraints are put in an LP (Linear Programming) Solver, we get the optimized output as shown in Figure 13b. The cost of all possible valid transitions before and after optimization in Figure 13 is shown in Table 3.

Goal	Constraints
Maximize 3-Keystroke Coverage	i) Number of changes, N <= 1
	ii) 1 new shortcut allowed

Table 4: (Example-2) Goal and constraints for Figure 14



(a) Original Configuration (**3-keystroke coverage = 50%**)

(b) Optimized Configuration (**3-keystroke coverage = 87%**)

Figure 14: (Example 2) UI hierarchy optimization obliging the constraints in Table 4. Node C is marked in green, meaning that it has a direct shortcut available for it. A yellow shade inside a node indicates that its position in the UI hierarchy has been changed.

Example 2. Suppose the original application DOM is similar to before (Figure 14a), but our goal is to maximize the number of transitions we can make within 3 keystrokes, with no more than one structural change and only one new shortcut to any node (Table 4). The output of an LP optimized is shown in Figure 14b. The cost of all possible valid transitions before and after optimization in Figure 14 is shown in Table 5.

Adding a shortcut to node C (marked in green) results in the maximum 3-keystroke coverage. Adding a new shortcut to node C allows it to act as another entry point in the hierarchy. For example, if one needs to go to node D, they can just use the shortcut to get to C and then use a right arrow (→) key to move to D.

Edge	Original Cost	Optimal Cost	Edge	Original Cost	Optimal Cost
<A,B>	2	2	<D,A>	2	2
<A,C>	3	1	<D,B>	2	2
<A,D>	4	2	<D,C>	1	1
<A,E>	3	3	<D,E>	4	4
<A,F>	4	3	<D,F>	5	1
<B,A>	2	2	<E,A>	3	3
<B,C>	1	1	<E,B>	4	2
<B,D>	2	2	<E,C>	5	1
<B,E>	4	4	<E,D>	6	2
<B,F>	5	3	<E,F>	2	3
<C,A>	2	2	<F,A>	4	2
<C,B>	1	1	<F,B>	5	2
<C,D>	1	1	<F,C>	6	1
<C,E>	4	4	<F,D>	7	1
<C,F>	5	2	<F,E>	2	4

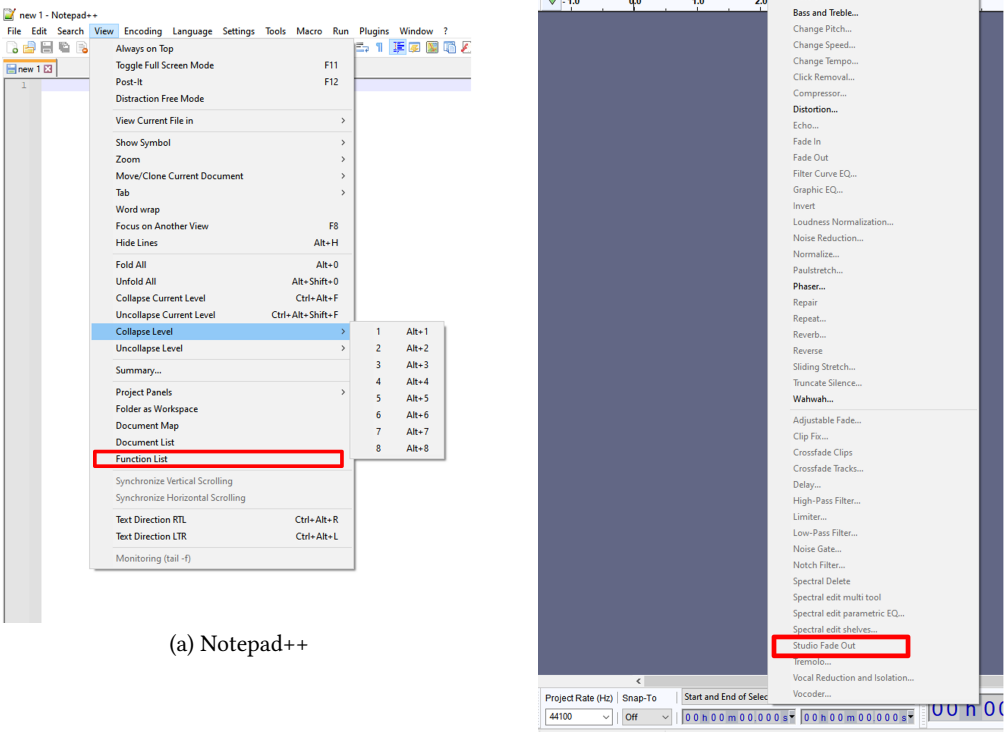
Table 5: (Example 2) Summary of costs for all transitions in Figure 14

Summary. In this possible use case, we showed how the proposed metrics can be used as cost functions when optimizing UI hierarchies for screen reader users. In the first example (Figure 13), the optimizer i) got rid of intermediate non-leaf nodes P2 and P4 and ii) moved the nodes E and F to different parents to minimize the complexity (i.e., cost function) of the hierarchy. In the second example (Figure 14), the optimizer i) got rid of intermediate non-leaf node P4, ii) changed the parent of node F, and iii) assigned a shortcut to node C to maximize the 3-keystroke coverage. The aforementioned changes are made to the DOM structure of the app, which is oblivious to sighted users. As such, sighted users can continue to use the unmodified application hierarchy. The developer could offer the modifications as an accessibility enhancement, making them exclusively available to screen reader users.

5.3 Use Case 3: Finding Hard-to-reach UI Objects and Offering Workarounds

Let us define hard-to-reach UI objects as nodes that require over 20 keystrokes to reach. This is a case when a UI object has too many siblings because the users have to go over these siblings to reach terminal nodes. Two such examples (e.g., Notepad++ and Audacity) are shown in Figure 15 to demonstrate this case. Our method of analyzing applications based on their DOM trees can reveal the existence of such nodes and help the developers make better design choices.

Workaround: Adding Shortcuts For Hard-to-reach Nodes. Recall that any node with a shortcut can be reached using one (or a set) of keystrokes. As such, one idea would be to add shortcuts for all the hard-to-reach nodes. However, from the examples in Figure 15, it is clear that there can be too many such nodes in some cases. For situations like these, developers can create multiple entry points into long list-like menus. For example, if a list-like menu has more than N (developer can pick the value of N) elements, the developer might create a shortcut for the $\frac{N}{2}$-th item in the list. In



(a) Notepad++

(b) Audacity

Figure 15: Applications with too many hard-to-reach UI objects. Example nodes are marked in red.

such a scenario, if the target is closer to the middle of the list, the user can navigate directly to the $\frac{N}{2}$-th item first. Again, dividing the list into two halves is just an example. The developer can pick any number of divisions they deem appropriate. The metrics proposed in this paper would enable the comparisons of different design decisions made by the developer.

We note that designing an effective UI hierarchy is challenging, but existing design frameworks, such as the Information Architecture Principles [14], can provide guidance. For example, keeping a short list of frequently used items at the top level can simplify navigation. For a long list, Hick's law recommends to organize items alphabetically [85].

5.4 Use Case 4: Recommending Guidelines for Application Design/Usage

The hard-to-reach node problem in Section 5.3 can also be addressed by promoting specific app design/usage guidelines, such as ***circular navigation***. Most applications we tested support circular navigation, i.e., pressing the up arrow on the first item in the menu shifts the focus to the last item in the menu. However, none of our participants were familiar with this feature and hence, never used it. Using this feature could result in large decreases in the number of keystrokes required to reach a node at times. For example, consider the *Function List* option in the Notepad++ application in Figure 15a. If traditional linear navigation is used, the average complexity to reach this option is 31. However, with circular navigation, the complexity comes down to 18. In the case of the *Studio Fade Out* option in the Audacity application in Figure 15b, the average complexity is 56 with linear navigation and 16 with circular navigation. For application developers, this metric can facilitate evaluating different UI designs (e.g., circular vs. linear) and their relative difficulties. For users, the benefits of new navigation models require additional training; this metric quantifies the gains in user experience one can expect from such training.

5.5 Use Case 5: Comparing Similar Applications and Estimating the Effect of Learning Shortcuts

Participant P8, who is an Assistive Technology Trainer, believes our metrics could be helpful to end-users as part of a rating system for software accessibility. As reflected in our user study, users currently rely on trial-and-error or advice from others (if available) to decide whether to invest the effort in learning a piece of software.

In the words of P8:

> *"I wish I could tell the people I teach about these numbers. These will help them with the decision*

of choosing, using, and in-depth learning an application."

Figures 10-12 show wide variance in the difficulties of learning and using different applications. Using these, blind screen reader users can choose which application to learn in a given category (for example, text editors). Throughout our interviews, we encountered participants saying that app X is easier to use than app Y. For example, our participants consistently mentioned that they preferred Notepad over Word for basic text editing. This is reflected by our metrics and Figures 10, 11, and 12. The complexity for MS Word is 3.51 and 17.65 with and without shortcuts, respectively. The scores are 1.63 and 9.94 for Notepad. We can also compare the 20% reachability (2 for Word, 1 for Notepad, with lower being better) or 3-keystroke coverage (82.52% for Word, 94.35% for Notepad) for the two applications to come to the same conclusion. Thus, our metrics can help screen reader users, and accessibility trainers choose which applications to learn in a given category.

Figures 10-12 also display the drastic difference in the proposed metric values with or without shortcuts. In the future, these metric values can also be generated for a specific number of shortcuts (e.g., 10, 20, 30), as opposed to 0 (without shortcut) or all (with shortcut) as shown in Figures 10-12. With these additional benchmarks, blind users could estimate their current complexity (say, with 10 shortcuts) and predict how much efficiency they can gain by learning 10 (or, 20) more shortcuts.

6 DISCUSSION

6.1 A Complement to User Testing

HCI community has developed analytical approaches to interface design and evaluation (e.g., GOMS models [18, 19, 40]) that work in tandem with empirical techniques in the iterative design loop and provide satisfactory results when empirical testing is not practical [32, 60, 70]. As software development practices have moved toward the "move fast and break things" mantra, this community needs better "guardrails" against breaking accessibility, which can keep pace with rapid development. Thus, a key benefit of our model is that it can be automated and is task-independent. In contrast, other techniques are task-dependent (e.g., GOMS modeling) or slower and more systematic (e.g., user testing). To be clear, all serve essential, complementary roles.

6.2 Relationships with GOMS-Based Models

We provide insights into how our approach compares to prior analytical frameworks, especially the GOMS models. The most notable difference is that our approach is application-specific, whereas GOMS models are task-dependent. Put differently, our metrics report applications' *structural properties without assuming a particular task at hand*. With user-specific usage data, our metrics can be fine-tuned. In contrast, GOMS models *assume there is a task at hand* and report its *estimated completion time* by breaking the task into atomic micro tasks whose completion times are known empirically. Therefore, our approach and GOMS models have different focuses, although both are related to usability.

We emphasize this difference with an illustration of GOMS modeling. Suppose the task at hand is to estimate the time to navigate from `'Home'` ribbon pane to the `'Right Align'` button in MS Word (in Figure 3) for a screen reader user. Adopting a reference implementation of the GOMS model for screen reader-mediated interaction [70], we can break the task into program-like steps (shown in Table 6) along with their operators (mental, key press, homing, and system response/wait). The last column of Table 6 describes how a step is abstracted in our approach.

Since our metrics are task-agnostic, our transition models (Section 4.3 and Section 4.4) abstract the notion of a micro-task as the probability of navigating from one UI to another within an app. Introducing probability enables our transition models to account for uncertainty in user interaction, which traditional GOMS-based models do not consider. This would allow us to model errors during navigation in the future. Our cost model (Section 4.2) abstracts the total number of steps (rows) as the number of intermediaries between the source and the target. Finally, although we ignore the mental, homing, and response/wait operators, these are easy to incorporate into our approach like GOMS models.

6.3 Potential Implications

Informing potential users about the accessible usability of different applications is only one usage scenario of our metrics. Other widespread usages may include the following:

- *Assistive Technology (AT) trainers* can use these metrics to explain the value of learning more shortcuts. Although users are not required to memorize shortcuts, they should expect to issue more keystrokes per action.
- *Novice blind users* can mine the model for particularly valuable shortcuts to learn in a given application, letting them prioritize learning effort. After learning a small number of shortcuts (say five to ten), they can observe how much their experience improves and decide whether to learn more.
- *Expert blind users* can use proposed metrics to evaluate their AT mastery. With a lower bound on the average keystrokes per action, they know when further improvement is impossible, or how much more efficient they could become with more effort.
- *Application developers* can use these metrics in an optimizer to get recommendations for optimal UI hierarchy or placement of shortcuts (Section 5.2). They can also integrate these metrics into automated testing frameworks, such as continuous integration (CI/CD) [30], with constraints like the expected number of keystrokes of an application should be fewer than five (i.e., $complexity < 5$). Plugin developers can use these metrics to identify where to insert necessary shortcuts for screen readers to make app navigation easier [8].
- *Employers and policy makers* can use the metrics values for work software to accommodate blind employees in the workplace. For example, suppose the overall complexity of a piece of software is high but fewer than 4 keystrokes (on average) for frequently used features. In that case, blind employees can easily use the software and be productive.

6.4 Challenges in Extracting UI hierarchy for Large Applications

Extracting the internal structure (i.e., DOM) of a piece of software has been a known art in software engineering [24, 26, 49, 81]. Key

#	Steps (GOMS)	Operator (GOMS)	Our Model
1	Initiate the navigation from the `Home`	Mental	Ignored
2	Press the next navigation key	Keystroke	Abstracted by the the cost model
3	Wait for screen reader to narrate the focused item	Wait	Ignored
4	Screen reader narrates the item name	Wait	Ignored
5	Verify if the name is `Right Align` button (**If not, go to 2**)	Mental	The correct intermediate nodes are already abstracted by the cost model
6	Stop the navigation	Mental	Ignored

Table 6: A simplified illustration of GOMS modeling for a task that estimates the time to navigate from `Home` ribbon pane to the `Right Align` button in MS Word (Figure 3) with a screen reader. The steps and operator columns are related to GOMS modeling, and the last column describes how these are abstracted in our approach.

challenges in such extraction are mostly associated with how the software was designed in the first place [24]. For instance, apps that lack proper accessibility metadata are hard to mine using accessibility APIs. Techniques such as "GUI Ripping" [49] reconstruct an application's hierarchy by extensively visiting all the UI options. Our application DOM extraction tool (Section 4.1) uses a similar notion. For very large software, the DOM has to be built incrementally as the extractor moves through hundreds of menu options. Incremental DOM building often brings about inconsistencies and repetitions in detection. The more prominent problem is that large applications often use two or more modal windows and pop-ups. In such cases, our tool expands the DOM as new content is seen on the screen (e.g., menus and pop-ups opened). Then, we calculate metrics separately for modal windows, as screen readers cannot access the parent window until a modal is dismissed. We plan to explore combining metrics for related windows in the future.

6.5 Adaptation for Touchscreen Devices and Other Media

Although we have demonstrated the proposed metrics for desktop applications, these metrics are generic and can be calculated for any graphical UI system that supports the notion of a DOM-like structure for UI elements. For example, smartphone applications running on touch screen devices are accessible to blind users via screen reader-provided touch gestures (e.g., 1-finger swipe left/right in TalkBack [65] to navigate individual UI elements). Similar to desktop screen readers, smartphone screen readers also construct a DOM-like structure for an app [82]. The DOM tree can be built using a similar method to us with accessibility APIs or by using a computer-vision-based UI interpretation approach [49, 81]. Thus, we can easily calculate the proposed metrics for smartphone apps by substituting keyboard shortcuts with touch gestures in the cost model (Section 4.2). In the future, we will explore how to adapt our model to measure the hard-to-type words on a gesture-based touchscreen keyboard [7] and hard-to-navigate printed forms with a wearable device [6].

6.6 Limitations

Our work has several limitations. First, the proposed model and metrics are mathematical constructs that are based on an application's UI hierarchy and the probability of transitioning from one UI to another UI. Therefore, how these metric values correlate to blind users' *true* experience with an application needs further investigation. Second, the interpretability of new metrics like ours is necessary for wide adoption. Although our preliminary user feedback (Section 5.1.4) is encouraging, a large-scale study in the wild is warranted to establish the interpretability of the proposed metrics. This study can be conducted online with participants from diverse backgrounds and skills interacting with different desktop applications using different screen readers. Third, not all participants in our study were familiar with all 11 of the selected applications. As such, the number of reviews for some applications like VLC and Audacity was low (Section 3.4.6), and additional data may raise confidence in these ratings. Finally, our study covers only 11 applications. For this study, our goals were to cover a wide variety of use cases and to select applications with a significant population of knowledgeable blind users. These results can be strengthened with additional applications, although it may be more difficult to get ground truth usability data without experienced blind users. Nevertheless, with additional usability data from users, collecting and comparing our metrics for new applications is straightforward.

7 CONCLUSION

This paper first presents a study with 11 expert blind users to understand their perceived usability and accessibility of desktop applications. Our study reveals that their perceptions of usability and accessibility are nuanced, incorporating various factors beyond the traditional definitions. These factors include independence, ease of learning an application, ability to describe it to others, reliability and deterministic behavior of keyboard shortcuts, and transferability of knowledge to others. Informed by this study, we aimed to capture this notion of perceived accessibility by proposing a probabilistic user interaction model. Based on this model, we compute three statistical properties of desktop applications that indicate the complexity of the keyboard navigation of an application, the primary mode of interaction for screen reader users. Our study participants corroborate our findings. We present five different utilities of our metrics to benefit end users, application developers, and accessibility practitioners.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their insightful feedback. This work was supported in part by NIH subaward 87527/2/1159967.

The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

REFERENCES

[1] Amaia Aizpurua, Myriam Arrue, Simon Harper, and Markel Vigo. 2014. Are users the gold standard for accessibility evaluation?. In *Proceedings of the 11th Web for All Conference*. 1–4.

[2] Cynthia L Bennett and Daniela K Rosner. 2019. The Promise of Empathy: Design, Disability, and Knowing the" Other". In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–13.

[3] Jeffrey P. Bigham, Irene Lin, and Saiph Savage. 2017. The Effects of "Not Knowing What You Don't Know" on Web Accessibility for Blind Web Users. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) *(ASSETS '17)*. Association for Computing Machinery, New York, NY, USA, 101–109. https://doi.org/10.1145/3132525.3132533

[4] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. 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*. ACM, 5862–5868. https://doi.org/10.1145/3025453.3025731

[5] Syed Masum Billah, Donald E. Porter, and I. V. Ramakrishnan. 2016. Sinter: low-bandwidth remote access for the visually-impaired. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2901335, 1–16. https://doi.org/10.1145/2901318.2901335

[6] Shirin Feiz, Syed Masum Billah, Vikas Ashok, Roy Shilkrot, and I. V. Ramakrishnan. 2019. Towards Enabling Blind People to Independently Write on Printed Forms. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 1–12. https://doi.org/10.1145/3290605.3300530

[7] Syed Masum Billah, Yu-Jung Ko, Vikas Ashok, Xiaojun Bi, and I. V. Ramakrishnan. 2019. Accessible Gesture Typing for Non-Visual Text Entry on Smartphones. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 1–10. https://doi.org/10.1145/3290605.3300606

[8] Farhani Momotaz, Md Touhidul Islam, Md Ehtesham-Ul-Haque, and Syed Masum Billah. 2021. Understanding Screen Readers' Plugins. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 1–10. https://doi.org/10.1145/3441852.3471205

[9] Pradipta Biswas and Peter Robinson. 2008. Automatic Evaluation of Assistive Interfaces. In *Proceedings of the 13th International Conference on Intelligent User Interfaces* (Gran Canaria, Spain) *(IUI '08)*. Association for Computing Machinery, New York, NY, USA, 247–256. https://doi.org/10.1145/1378773.1378806

[10] Pradipta Biswas and Peter Robinson. 2010. Evaluating the design of inclusive interfaces by simulation. In *Proceedings of the 15th international conference on intelligent user interfaces*. 277–280.

[11] Yevgen Borodin, Jeffrey P. Bigham, Glenn Dausch, and I. V. 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, 1806005, 1–10. https://doi.org/10.1145/1805986.1806005

[12] Giorgio Brajnik. 2008. Beyond conformance: the role of accessibility evaluation methods. In *International Conference on Web Information Systems Engineering*. Springer, 63–80.

[13] Giorgio Brajnik. 2008. A comparative test of web accessibility evaluation methods. In *Proceedings of the 10th international ACM SIGACCESS Conference on Computers and Accessibility*. 113–120.

[14] Dan Brown. 2010. Eight principles of information architecture. *Bulletin of the American Society for Information Science and Technology* 36, 6 (2010), 30–34.

[15] A. Bryman and R.G. Burgess. 1994. *Analyzing Qualitative Data*. Routledge. https://books.google.com/books?id=KQkotSd9YWkC

[16] Shiya Cao and Eleanor Loiacono. 2019. The state of the awareness of web accessibility guidelines of student website and app developers. In *International Conference on Human-Computer Interaction*. Springer, 32–42.

[17] SK Card, TP Moran, and A Newell. 1983. The Psychology of Human Computer Interaction Lawrence Erlbaum. *Associates, NJ* (1983).

[18] Stuart K Card, Thomas P Moran, and Allen Newell. 1980. The keystroke-level model for user performance time with interactive systems. *Commun. ACM* 23, 7 (1980), 396–410.

[19] Stuart K Card, Thomas P Moran, and Allen Newell. 1983. *The psychology of human-computer interaction*. Crc Press.

[20] Lucas Pedroso Carvalho, Bruno Piovesan Melchiori Peruzza, Flávia Santos, Lucas Pereira Ferreira, and André Pimenta Freire. 2016. Accessible smart cities? Inspecting the accessibility of Brazilian municipalities' mobile applications. In *Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems*. 1–10.

[21] Raphael Clegg-Vinell, Christopher Bailey, and Voula Gkatzidou. 2014. Investigating the appropriateness and relevance of mobile web accessibility guidelines. In *Proceedings of the 11th Web for All Conference*. 1–4.

[22] Mark Colley, Taras Kränzle, and Enrico Rukzio. 2022. Accessibility-Related Publication Distribution in HCI Based on a Meta-Analysis. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–28.

[23] Giliam de Carpentier. 2012. Discrete staircase probability distribution. https://www.decarpentier.nl/staircase-distribution

[24] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. ACM, New York, NY, USA, 767–776. https://doi.org/10.1145/2984511.2984581

[25] Catherine D'Ignazio, Alexis Hope, Becky Michelson, Robyn Churchill, and Ethan Zuckerman. 2016. A Feminist HCI Approach to Designing Postpartum Technologies: " When I first saw a breast pump I was wondering if it was a joke". In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2612–2622.

[26] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. ACM, New York, NY, USA, 1525–1534. https://doi.org/10.1145/1753326.1753554

[27] Gavin Doherty and Mieke Massink. 1999. Continuous interaction and human control. In *Proceedings of the XVIII European annual conference on human decision making and manual control*. 80–96.

[28] Paul M Fitts. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology* 47, 6 (1954), 381.

[29] International Organization for Standardization. 2010. *Ergonomics of Human-system Interaction: Part 210: Human-centred Design for Interactive Systems*. ISO.

[30] Martin Fowler and Matthew Foemmel. 2006. Continuous integration.

[31] SBL Ferreira, RC dos Santos, and DS Silveira. 2007. Panorama of Brazilian web accessibility, Proceedings of the XXXI ANPAD Meeting-EnANPAD, page 17p

[32] Richard Gong and David Kieras. 1994. A validation of the GOMS model methodology in the development of a specialized, commercial software application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 351–357.

[33] Google. 2021. Expresso | Use Espresso to write concise, beautiful, and reliable Android UI tests. https://developer.android.com/training/testing/espresso. Online; accessed 23 January 2023.

[34] Google. 2021. iOS UI Automation Test Framework. https://github.com/google/EarlGrey. Online; accessed 23 January 2023.

[35] Web Accessibility Initiative. 2017. Accessibility, usability, and inclusion: related aspects of a web for all.

[36] Richard J Jagacinski and John M Flach. 2018. *Control theory for humans: Quantitative approaches to modeling performance*. CRC press.

[37] Bonnie E John. 1994. Toward a deeper comparison of methods: A reaction to Nielsen & Phillips and new data. In *Conference Companion on Human Factors in Computing Systems*. 285–286.

[38] Bonnie E John and David E Kieras. 1996. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 4 (1996), 320–351.

[39] Bonnie E John and David E Kieras. 1996. Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 4 (1996), 287–319.

[40] David Kieras. 1997. A guide to GOMS model usability evaluation using NGOMSL. In *Handbook of human-computer interaction*. Elsevier, 733–766.

[41] David Kieras et al. 2001. Using the keystroke-level model to estimate execution times. *University of Michigan* 555 (2001).

[42] KIF. 2021. KIF iOS Integration Testing Framework. https://github.com/kif-framework/KIF. Online; accessed 23 January 2023.

[43] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, (1996), 194, 194.

[44] 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*. Elsevier, 52, 139–183.

[45] Sooyeon Lee, Madison Reddie, and John M Carroll. 2021. Designing for Independence for People with Visual Impairments. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–19.

[46] Barbara Leporini and Fabio Paternò. 2004. Increasing usability when interacting through screen readers. *Universal access in the information society* 3, 1 (2004), 57–70.

[47] Jennifer Mankoff, Gillian R Hayes, and Devva Kasnitz. 2010. Disability studies as a source of critical inquiry for the field of assistive technology. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. 3–10.

[48] Beatriz Martins and Carlos Duarte. 2022. Large-scale study of web accessibility metrics. *Universal Access in the Information Society* (2022), 1–24.

[49] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. 2003. GUI ripping: Reverse engineering of graphical user interfaces for testing. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings*. IEEE, 260–269.

[50] Microsoft. 2021. Accessibility tools - AccChecker (UI Accessibility Checker). https://docs.microsoft.com/en-us/windows/win32/winauto/ui-accessibility-

checker. Online; accessed 23 January 2023.
[51] Microsoft. 2021. Accessibility tools - Inspect. https://learn.microsoft.com/en-us/windows/win32/winauto/inspect-objects. Online; accessed 23 January 2023.
[52] Microsoft Inc. 2020. UI Automation Overview. http://msdn.microsoft.com/en-us/library/ms747327.aspx
[53] Lauren R Milne, Cynthia L Bennett, and Richard E Ladner. 2014. The accessibility of mobile health sensors for blind users. In *International Technology and Persons with Disabilities Conference Scientific/Research Proceedings (CSUN 2014)*. 166–175.
[54] Antti Oulasvirta, Per Ola Kristensson, Xiaojun Bi, and Andrew Howes. 2018. *Computational interaction*. Oxford University Press.
[55] Helen Petrie and Omar Kheir. 2007. The Relationship between Accessibility and Usability of Websites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 397–406. https://doi.org/10.1145/1240624.1240688
[56] Christopher Power, Paul Cairns, and Mark Barlet. 2018. Inclusion in the third wave: access to experience. In *New Directions in Third Wave Human-Computer Interaction: Volume 1-Technologies*. Springer, 163–181.
[57] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, and Mika V Mäntylä. 2012. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test (AST)*. IEEE, 36–42.
[58] Robolectric. 2021. Robolectric is the industry-standard unit testing framework for Android. http://robolectric.org/getting-started/. Online; accessed 23 January 2023.
[59] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O. Wobbrock. 2017. Epidemiology As a Framework for Large-Scale Mobile Application Accessibility Assessment. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) *(ASSETS '17)*. ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/3132525.3132547
[60] Martin Schrepp. 2010. GOMS analysis as a tool to investigate the usability of web units for disabled users. *Universal Access in the Information Society* 9, 1 (2010), 77–86.
[61] Leandro Coelho Serra, Lucas Pedroso Carvalho, Lucas Pereira Ferreira, Jorge Belimar Silva Vaz, and André Pimenta Freire. 2015. Accessibility evaluation of e-government mobile applications in Brazil. *Procedia Computer Science* 67 (2015), 348–357.
[62] Ben Shneiderman. 2000. Universal usability. *Commun. ACM* 43, 5 (2000), 84–91.
[63] Ben Shneiderman. 2002. Promoting universal usability with multi-layer interface design. *ACM SIGCAPH Computers and the Physically Handicapped* 73-74 (2002), 1–8.
[64] Hironobu Takagi, Chieko Asakawa, Kentarou Fukuda, and Junji Maeda. 2003. Accessibility designer: visualizing usability for the blind. *ACM SIGACCESS accessibility and computing* 77-78 (2003), 177–184.
[65] TalkBack. [n. d.]. TalkBack: An Open Source Screenreader For Android. https://support.google.com/accessibility/android/answer/6283677
[66] The World Wide Web Consortium (W3C). 2018. Web Content Accessibility Guidelines (WCAG) 2.1. https://www.w3.org/TR/WCAG21/
[67] Section 508 of the Rehabilitation Act. 2019. https://www.fcc.gov/general/section-508-rehabilitation-act
[68] Mary Frances Theofanos and Janice Redish. 2003. Bridging the gap: between accessibility and usability. *interactions* 10, 6 (2003), 36–51.
[69] Shannon M. Tomlinson. 2016. Perceptions of Accessibility and Usability by Blind or Visually Impaired Persons: A Pilot Study. In *Proceedings of the 79th ASIST Annual Meeting: Creating Knowledge, Enhancing Lives through Information & Technology* (Copenhagen, Denmark) *(ASIST '16)*. American Society for Information Science, USA, Article 120, 4 pages.
[70] Henrik Tonn-Eichstädt. 2006. Measuring website usability for visually impaired people-a modified GOMS analysis. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*. 55–62.
[71] Shari Trewin, Bonnie E. John, John Richards, Cal Swart, Jonathan Brezin, Rachel Bellamy, and John Thomas. 2010. Towards a Tool for Keystroke Level Modeling of Skilled Screen Reading. In *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility* (Orlando, Florida, USA) *(ASSETS '10)*. Association for Computing Machinery, New York, NY, USA, 27–34. https://doi.org/10.1145/1878803.1878811
[72] Daniel W Turner III and Nicole Hagstrom-Schmidt. 2022. Qualitative interview design. *Howdy or Hello? Technical and Professional Communication* (2022).
[73] Andries Van Dam. 1997. Post-WIMP user interfaces. *Commun. ACM* 40, 2 (1997), 63–67.
[74] Markel Vigo, Justin Brown, and Vivienne Conway. 2013. Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*. 1–10.
[75] Beat Vollenwyder, Glena H Iten, Florian Brühlmann, Klaus Opwis, and Elisa D Mekler. 2019. Salient beliefs influencing the intention to consider Web Accessibility. *Computers in Human Behavior* 92 (2019), 352–360.
[76] Beat Vollenwyder, Serge Petralito, Glena H. Iten, Florian Brühlmann, Klaus Opwis, and Elisa D. Mekler. 2023. How compliance with web accessibility standards shapes the experiences of users with and without disabilities. *International Journal of Human-Computer Studies* 170 (2023), 102956. https://doi.org/10.1016/j.ijhcs.2022.102956
[77] W3C Web Accessibility Initiative (WAI). 2016. Accessibility, Usability, and Inclusion. https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/
[78] Cynthia Waddell, Bob Regan, Shawn Lawton Henry, Michael R Burks, Jim Thatcher, Mark D Urban, and Paul Bohman. 2003. *Constructing accessible web sites*. Apress.
[79] Jonathan Lazar, Alfreda Dudley-Sponaugle, and Kisha-Dawn Greenidge. 2004. Improving web accessibility: a study of webmaster perceptions. Computers in human behavior, 20, 2, 269–288 Elsevier
[80] Jacob O Wobbrock, Shaun K Kane, Krzysztof Z Gajos, Susumu Harada, and Jon Froehlich. 2011. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)* 3, 3 (2011), 1–27.
[81] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. UIED: a hybrid tool for GUI element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1655–1659.
[82] Jian Xu, Syed Masum Billah, Roy Shilkrot, and Aruna Balasubramanian. 2019. DarkReader: Bridging the Gap Between Perception and Reality of Power Consumption in Smartphones for Blind Users. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility* (Pittsburgh, PA, USA) *(ASSETS '19)*. Association for Computing Machinery, New York, NY, USA, 96–104. https://doi.org/10.1145/3308561.3353806
[83] Shunguo Yan and P. G. Ramachandran. 2019. The Current Status of Accessibility in Mobile Apps. *ACM Trans. Access. Comput.* 12, 1, Article 3 (Feb. 2019), 31 pages. https://doi.org/10.1145/3300176
[84] Yeliz Yesilada, Giorgio Brajnik, Markel Vigo, and Simon Harper. 2012. Understanding web accessibility and its drivers. In *Proceedings of the international cross-disciplinary conference on web accessibility*. 1–9.
[85] William E Hick. 1952. On the rate of gain of information. *Quarterly Journal of experimental psychology*, Taylor & Francis, 4, 1, 11–26.
[86] Yeliz Yesilada, Giorgio Brajnik, Markel Vigo, and Simon Harper. 2015. Exploring perceptions of web accessibility: a survey approach. *Behaviour & Information Technology* 34, 2 (2015), 119–134.
[87] Amaia Aizpurua, Simon Harper, and Markel Vigo. 2016. Exploring the relationship between web accessibility and user experience *International Journal of Human-Computer Studies*, Elsevier, Vol. 91, 13–23 pages.
[88] Krzysztof Z. Gajos, Daniel Weld, and Jacob Wobbrock. 2010. *Automatically generating personalized user interfaces with SUPPLE*. Vol. 174. 910–950 pages. https://doi.org/10.1016/j.artint.2010.05.005

A APPENDIX

One can calculate $P(B_k)$ using the following equation:

$$P(B_k) = \frac{1}{2 * (a+b)} * (a + (k-1) * \frac{(b-a)}{3}) \tag{4}$$

Proof:

$$\begin{aligned}
\sum_{k=1}^{k=4} P(B_k) &= \frac{1}{2*(a+b)} * [4a + \frac{0*(b-a)}{3} + \\
&\quad \frac{1*(b-a)}{3} + \frac{2*(b-a)}{3} + \\
&\quad \frac{3*(b-a)}{3}] \\
&= \frac{1}{2*(a+b)} * [4a + \frac{(b-a)}{3}(1+2+3)] \\
&= \frac{1}{2*(a+b)} * [4a + \frac{(b-a)}{3}(6)] \\
&= \frac{1}{2*(a+b)} * [4a + 2*(b-a)] \\
&= \frac{1}{2*(a+b)} [4a + 2b - 2a] \\
&= \frac{1}{2*(a+b)} [2a + 2b] \\
&= \frac{1}{2*(a+b)} [2*(a+b)] \\
&= 1
\end{aligned} \tag{5}$$

Equation 5 tells us that the summation of all the transition probabilities is 1. This proves that Equation 4 is correct.