Understanding Screen Readers' Plugins

Farhani Momotaz
Pennsylvania State University
University Park, Pennsylvania, USA
fbm5122@psu.edu

Md Touhidul Islam
Pennsylvania State University
University Park, Pennsylvania, USA
mqi5127@psu.edu

Md Ehtesham-Ul-Haque
Pennsylvania State University
University Park, Pennsylvania, USA
mfe5232@psu.edu

Syed Masum Billah
Pennsylvania State University
University Park, Pennsylvania, USA
sbillah@psu.edu

ABSTRACT

Screen reader plugins are small pieces of code that blind users can download and install to enhance the capabilities of their screen readers. In this paper, we aim to understand the user experience of screen readers' plugins, as well as their developers, distribution model, and maintenance. To this end, we conducted a study with 14 blind screen reader users. Our study revealed that screen reader users rely on plugins for various reasons, e.g., to improve the usability of both screen readers and application software, to make partially accessible applications accessible, and to enable custom shortcuts and commands. Furthermore, installing plugins is easy; uninstalling them is unlikely; and finding them online is ad hoc, challenging, and poses security threats. In addition, developing screen reader plugins is technically demanding; only a handful of people develop plugins, and they are well-recognized in the community. Finally, there is no central repository for plugins for most screen readers, and most plugins do not receive updates from their developers and become obsolete. The lack of financial incentives plays in the slow growth of the plugin ecosystem. Based on our findings, we recommend creating a central repository for all plugins, engaging third-party developers, and raising general awareness about the benefits and dangers of plugins. We believe our findings will inspire researchers to embrace the plugin-based distribution model as an effective way to combat application-level accessibility issues.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in accessibility**; *Accessibility systems and tools.*

KEYWORDS

plugins, add-ons, scripts, extensions, screen readers, accessibility, blind, user study.

ASSETS '21, October 18–22, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8306-6/21/10…$15.00
https://doi.org/10.1145/3441852.3471205

ACM Reference Format:
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 (ASSETS '21), October 18–22, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3441852.3471205

1 INTRODUCTION

People who are blind usually interact with computers via screen readers, such as NVDA [2], JAWS [1], and VoiceOver [8]. Screen readers are special-purpose assistive technologies that narrate on-screen visual content serially and support numerous keyboard shortcuts to interact with graphical applications.

Screen readers rely on Operating Systems' (OSes') builtin accessibility support [7, 33] to narrate the textual description of an application's graphical user interface (GUI). Often, this accessibility support falls short for many applications for several reasons—all of which are traced back to how an application is developed. For example, an application becomes partially accessible to screen readers if the developers provided insufficient textual descriptions of individual UI [26], or developed the application using a GUI framework that is not fully compatible with the OS's native accessibility support [15], or encoded information spatially which requires mouse-only interactions [11]. In these cases, blind users who need screen readers face undue challenges in interacting with the application.

To overcome these challenges, at least to some extent, screen readers support **plugins**—small pieces of code that could extend screen readers' builtin functionalities or could repair application-specific accessibility issues. Plugins are also referred to as *add-ons*, *add-ins*, *modules*, *extensions*, or *custom scripts*. Usually, domain experts with technical skills develop plugins, which they publish online so that others can download and benefit from them. Certain screen reader vendors (e.g., NVDA) also distribute popular plugins as part of their original package. As such, it is likely that many screen readers users have used screen reader plugins knowingly or unknowingly. Despite being widely used, to the best of our knowledge, there is no study on understanding the user experience, usability, life-cycle, and design implication of these screen reader plugins.

In this work, we aim to fill that void. We are motivated by a surprising dearth of studies on this area to identify key gaps (if any), make recommendations for best practices, and outline new directions for future accessibility research. More specifically, we asked the following research questions:

- **RQ1**: *Why do blind users use screen reader plugins?*
- **RQ2**: *How screen reader plugins are developed, maintained, and distributed, from the standpoint of the users?*

To this end, we conducted a study with 14 blind participants. The study revealed that plugins are used for a variety of reasons, such as accessing partially inaccessible software, modifying screen readers' features, and receiving additional audio feedback on keystrokes. Furthermore, only a handful of people develop plugins, and they are well-recognized in the community (e.g., `Doug` from NY, `David` from https://blindhelp.net). Although most screen reader users prefer to write their own plugin, the need for technical skills and the complexity of development toolkits are prohibitive. In addition, plugins are easy to install but difficult to find and update. Once installed, users hardly uninstall them. Most screen readers (except for NVDA) do not have a central repository for plugins. As a result, users face difficulty finding "good" quality plugins — they often download plugins from untrusted sources (e.g., unknown websites, Google Drives), and sometimes download malware instead, compromising their digital security. The study also revealed that the current state of maintaining screen readers' plugins is "no maintenance at all" — once deployed, most plugins do not receive regular updates from their developers and become obsolete.

Our findings implicate that plugins can be distributed to screen reader users directly from the developers. Thus, a plugin-based delivery mechanism could inspire accessibility researchers to deploy their research prototype as a plugin and seek early feedback from the blind community. Additionally, educating screen reader users about the potential benefits and dangers of plugins is important because plugins can be instrumental in combating accessibility and usability issues in existing screen readers and application software.

2 BACKGROUND AND RELATED WORK

The plugin-based architecture is commonly used in commercial software, such as web browsers, multimedia players, rich text editors [40]. Each *plugin* is a small piece of code that extends the host application's functionality. These plugins are often neither baked into the source code of the host application nor require access to the source code of that application [17]. Instead, these are linked via well-defined interfaces and extension mechanisms so that the host application can recognize and activate a specific feature or functionality when needed [40].

Thus, a plugin-based architecture allows a complex application to function as a smaller but standalone product, while meeting users' specific needs by loading the third-party plugin (s). This architecture also allows developers to quickly release software upgrades or patches as plugins [17].

Popular applications usually have their online repositories containing hundreds of plugins. For example, the repositories for Firefox and Microsoft Office 365 contain over 450, 000 [6] and 2, 800 [32] plugins respectively. Compared to popular applications, the number of screen reader plugins is relatively small. For example, the official plugin repository [34] of NVDA contains less than 100 plugins, whereas there exists no such repository for JAWS screen reader.

In the following subsections, we first situate our work in the broader context of accessibility research. We then describe the usage of application-specific plugins to enhance accessibility, followed by browser-based plugins to increase web accessibility, and its similarity to developing plugins in screen readers.

2.1 Accessibility Issues with Screen Readers

User interfaces (UIs) are typically designed with the assumption that the users have no perceptual and cognitive impairments and use a typical set of input and output devices [24]. Thus, any mismatch between users' effective abilities and the underlying assumptions seriously hampers the effectiveness of user interface design. Often, this diversity of needs is either ignored; or addressed via a manual redesign of the application UI; or via external assistive technologies.

Although the manual redesign approach is arguably the best [24], it is neither feasible nor scalable—users' abilities and preferences may vary, which UI designers may not anticipate [10].

Fortunately, these challenges are well-addressed in the assistive technology-based approach. Assistive technologies like screen readers are designed to adapt application UIs and create a *manifest* interface that users with vision impairment can perceive and manipulate [35]. For example, blind users cannot use computer mouses to point and click on a UI because a mouse only provides a *visual* feedback (e.g., the cursor) [14] which blind users cannot perceive. Therefore, screen readers manifest 2D graphical content, suitable for mouse-based point and click, to a 1D list, where users can select individual UIs without using a mouse but using a keyboard's arrow keys.

However, this manifestation comes at the expense of user experience. Prior research reported that when the output modality of a UI-rich application is manifested from its original form to another modality, i.e., consuming an application aurally instead of visually, the adaptation introduces undesired side effects, e.g., the app may become partially accessible [26].

It is also reported that the quality of many screen readers plateaued in the mid 2000s [28]. Since then, these technologies have neither improved fundamentally nor kept pace with the rapid technological changes of the post-PC era, where users own multiple devices (e.g., desktops, laptops, and smartphones). For instance, screen readers are not portable, difficult to learn, inefficient, cumbersome to use, and do not provide a uniform interaction experience to date [15]. Further, switching from one screen reader to another is disruptive because one has to learn a whole slew of new interaction procedures [16]. As such, it is unlikely that screen readers alone could address all accessibility-related issues. That's why plugin-based extensible architecture has promise and potential in screen reader design.

2.2 Application-Specific Plugins to Enhance Accessibility

Many applications support some degree of extensibility by the end-users through scripting, plugins, or extensions [21]. For example, Mac OS supports *AppleScript*, a scripting interface to control applications and OS features. Such scripts can automate repetitive tasks, combine features from multiple apps, and create complex workflows to increase system-wide accessibility.

Independently of screen readers, some plugin-based frameworks are designed to make OS-level configuration easy and accessible. Heron et al. [27] proposed such a framework, ACCESS, that maps

user inputs to user needs. User inputs such as mouse clicks and pointer movement can be mapped to user needs, such as increasing the delay between two clicks in double-click events, increasing the pointer size, or highlighting the pointer location.

Researchers also proposed application-specific plugins to tackle a single accessibility issue in isolation. For example, WebAnywhere [12] and HearSay [18] are browser plugins that enable screen reading of webpages on public computers that do not have a screen reader installed. Ferres et al. [22] develop a plugin for MS Excel to generate accessible graphs in spreadsheets. The generated graphs contain static textual descriptions of the original graphs' content and visual attributes as HTML tags. The users can query and navigate the content of the graphs easily. Baker et al. [9] develop a plugin for Eclipse IDE to assist blind programmers in navigating Java source code. It constructs a hierarchical tree of the nesting structure of a program's source code.

While these frameworks and plugins require low-level integration with the OS and the host application, we focus on screen reader plugins that run within the scope of a screen reader.

2.3 Browser-Based Plugins to Enhance Web Accessibility

Most web browsers offer plugins and extensions, allowing one to directly modify the `HTML`, `CSS`, or `DOM` of a webpage prior to rendering it on the browser. Because of this extensibility, a plethora of browser-based plugins are designed to enhance web accessibility [13, 23, 25, 30, 31, 36–39]. These plugins usually share three key pieces: (i) an end-user first reports an accessibility issue; (ii) an expert user repairs the accessibility issue by writing a plugin or custom script using a research tool; and (iii) finally, that plugin is shared with others so that future users could benefit from the repair. Example of some repairs include providing alt-texts for images [37–39], reordering content navigation [36], performing website-specific customizations [13], and representing web content for aural glancing [23]. However, unlike screen readers' plugins, most of these scripts and plugins are research prototypes, either discontinued or have achieved little impact beyond the lab, and are not widely used by the visually impaired community.

2.4 How Screen Readers Support Plugin Development

Developing screen reader plugins is analogous to that of web browsers' plugins. A web browser usually exposes the HTML DOM of a webpage to its plugin as a JavaScript object. Similarly, a screen reader constructs a DOM-like structure of an app using the operating system's native Accessibility API [7, 33] and exposes it to a screen reader plugin. The syntax and semantic of this exposure are screen reader-dependent, as specified by the plugin developer guidelines (e.g., for NVDA [4], for JAWS [3]). A screen reader plugin can utilize many functions of the host screen reader, e.g., obtaining the reference of the focused UI, walking up/down in the DOM-like structure to inject metadata to a reference representing a UI. It can also bind new keyboard shortcuts to trigger an action (e.g., `CAPS LOCK + Alt + T` to announce `''Hello, World!''`), as well as bind a new event (e.g., `valueChanged`) to a UI element.

3 USER STUDY

To understand our research questions, we conducted a user study (IRB-approved) with 14 blind participants. Participants' demographics and study details are described below.

3.1 Participants

We recruited 14 blind participants (10 males, 4 females) through mailing lists, university mailing lists, and public posts on Facebook. Our inclusion criteria included visually impaired adults who use screen readers on their computers, are familiar with screen readers' plugin, and fluent in English. The participants varied in age from 24 to 52 ($M = 37$, $SD = 7$). Most of them were expert screen reader users and used two screen readers. They came from diverse professional backgrounds: 2 were business entrepreneurs, 2 worked in information technology, 3 worked for the government, 1 worked in a non-government organization, 1 worked in student disability service in a university, 2 in textile and apparel industry, 1 in sales, 1 was a physical therapist, and 2 were students. Participants were geographically distributed as follows: 40% North America and 60% South Asia. Table 1 presents their demographics.

3.2 Interview Method

The interviews were semi-structured, performed remotely via video/audio conferencing tools such as Zoom and phone calls. Two researchers administered the study: one conversed with the participants, while the other took notes. Following the completion of the first five interviews, the researchers analyzed the transcripts using an iterative coding process with initial coding and identified concepts [19], categorized them, framed new questions for subsequent interviews, and updated the concept list.

3.3 Interview Protocol and Data Analysis

After verbal consent, we began by asking participants to introduce themselves, their history of visual impairment, their expertise in screen readers (self-disclosed), and their use of screen reader plugins. We noted that their self-disclosed expertise closely matched our observation during the interview.

We believe semi-structured interviews provided us the maximum leverage to explore an under-researched topic like screen reader plugins systematically. This semi-structured format also allowed participants to talk freely about their experience using screen reader plugins.

The initial interview questions were similar to our original research questions (`RQs`). For example, why do they use screen reader plugins? What do they know about plugin developers? Who maintains plugins? What are the common places for finding plugins?

We also adapted questions under each `RQ` to match individual's expertise. A sampler of questions under `RQ1` were as follows: (i) Do you use any application that needs plugins? (ii) Why do some applications need more plugins than others? (iii) What types of features plugins usually provide? (iv) How often do you use a plugin? and (v) How many plugins do you use on average? Sample questions under `RQ2` included: (i) Where do you find a plugin or script? (ii) How do you install a plugin? (iii) How do you vet a plugin? (iv) Have you developed a plugin (why or why not)? (v) Do you want to develop plugins by yourself (why or why not)? (vi) How often

Table 1: Participant demographics. LP stands for Light Perception.

ID	Age/ Sex	Expertise	LP?	History of Blindness	Profession	Screen Readers
P1	39/M	Expert	No	Advantageous	Business Entrepreneur	NVDA, JAWS
P2	35/M	Expert	Yes	Advantageous	IT Instructor	NVDA, JAWS
P3	31/F	Expert	Yes	Advantageous	Information technology	JAWS, NVDA, VoiceOver
P4	40/M	Expert	Yes	Congential	Educator and Business Entrepreneur	System Access, NVDA
P5	44/M	Expert	No	Advantageous	Government employee	NVDA, JAWS
P6	43/M	Expert	No	Advantageous	Government employee	NVDA, JAWS
P7	28/M	Expert	No	Advantageous	Physical therapist	NVDA, JAWS
P8	24/M	Expert	No	Congential	Student	JAWS, NVDA
P9	40/M	Beginner	Yes	Congential	NGO worker	JAWS, NVDA
P10	38/F	Beginner	Yes	Advantageous	Sales representative	NVDA
P11	35/M	Expert	No	Advantageous	Student, Chessmaster	JAWS, NVDA
P12	38/F	Beginner	No	Advantageous	Garment worker	JAWS
P13	52/M	Intermediate	No	Congential	Government employee	NVDA, JAWS
P14	40/F	Expert	No	Congential	Student Disability Service	JAWS, NVDA, VoiceOver

do you update, install, or uninstall a plugin? (vii) Is there any issue with updating a plugin?

Towards the end, we educated some participants about existing plugins that could address some of the issues they raised during the interview. All interviews were audio-recorded and transcribed for analysis. Each session lasted for an hour, and participants were compensated with a $30 (USD) Amazon or other e-gift card. Each interview culminated with participants making suggestions and recommendations.

4 FINDINGS

In this section, we describe our findings emerged from the study. Note that some participants refer screen reader plugins as add-ons or scripts.

4.1 Why do Blind Users Use Screen Reader Plugins?

The participants mentioned several reasons why they use plugins. We categorize these reasons into 4 themes: blind users need plugins to make inaccessible or partially accessible software accessible, to modify screen readers' existing features, to receive audio feedback on keyboard shortcuts or commands, and to add new shortcuts to screen readers. Overall, plugins make software applications more usable. We elaborate on each of these themes below.

4.1.1 Plugins can Make Applications More Accessible. The participants mentioned that they search for a plugin when they realize certain feature of an application is not accessible. For example, P1, who uses NVDA screen reader, said: *'When I use a software and I find it inaccessible through NVDA, I start searching for the plugins"*. Several participants complained that developers often focus on sighted users and do not fully invest in accessibility. As such, blind users experience difficulty using most software, including the popular ones. For example, P3, a music enthusiast, who uses JAWS screen reader, described how plugins help her using music editing software:

> *For anything musical I do – working with programs like GarageBand, Audacity, Pro Tools, complete control – all those music kinds of programs for playing music, creating music, are not the most accessible programs to start with. And there are people in the blind music world who can also create scripts along the way so that they become a little bit more usable.*

Inaccessibility is more prevalent in web applications, e.g., "things are not always labeled properly", and screen readers "do not know what to make of them". So, screen readers often say "button, button, button", or sometimes stay silent. A plugin "usually comes up with some keystrokes" in these regards, and provides appropriate metadata to screen readers.

4.1.2 Plugins can Modify Screen Readers' Existing Feature. Sometimes, a software is accessible but provides a sub-optimal user experience with a screen reader. Plugins can improve the usability of such software by providing more control to the users. For example, P8, who uses Zoom teleconferencing software for work, explained why Zoom's user experience is poor, even though it is fully accessible. NVDA is "too chatty" on Zoom – it announces each event that may occur during a Zoom meeting, such as someone enters or leaves the meeting, someone mutes or unmutes themselves. Further, every time a chat message appears, which may be a casual conversation between two attendees, NVDA reads that too. P8 complained that with so many distracting voice outputs from NVDA, he could not concentrate on the actual meeting. It becomes more annoying when he shares his audio, and everyone else is also disturbed by the excessive voice notifications. So, he uses a plugin in NVDA that lets him configure Zoom's notification.

4.1.3 Plugins can Provide Audio Feedback on Keystrokes and Shortcuts. The participants use plugins to receive simple but necessary feedback and audio cues from screen readers. P1 presented two examples: NVDA does not provide audio feedback on popular clipboard command, `Crtl+C`, but with a plugin it provides useful feedback such as *"copied"* or *"nothing was copied"*. Similarly, NVDA is quiet on VLC Media player when a media is fast-forwarded, but with a plugin it announces the number of seconds the media was fast-forwarded. According to P1, these plugins are very simple scripts, but without them, he feels clueless.

4.1.4 Plugins can Add Useful Shortcuts to Screen Readers. The participants reported that plugins could add a set of useful and "make-sense" shortcuts to screen readers. P11, who is a professional chess player, explained how he added a new shortcut (e.g., `Crtl + 2`) to Win Board, a chess software, to report the position of a piece backward (e.g., Knight at 2E instead of Knight at E2).

In summary, most participants stated that plugins can solve accessibility problems and improve the usability of applications—half (7) of the participants wished to have more plugins.

The other half who did not wish for more, 3 of them were beginners, 1 was intermediate, and 3 were experts (P4, P7, P14). The beginner and intermediate participants mentioned that they usually interacted with a few websites and software for work, which were accessible through their existing plugins. P7, an expert working in healthcare, mentioned not using computers extensively and was content with his collection of plugins. However, the circumstances for P4 and P14 were different. They both were experts, but their primary screen readers were `System Access` and `VoiceOver`, respectively, for which the support for plugins is limited. As such, their expectation for having more plugins was low. Furthermore, they were financially well-off to employ sighted assistants who could provide technical support.

4.2 How Plugins are Developed?

4.2.1 Only a Handful of People Develop Plugins. The participants reported that only a handful of people develop plugins and they are well-respected and well-reputed in the blind community. The community recognizes them by their first name. For example, P3 mentioned a highly reputed blind plugin developer, named `Doug`, *"...he is known, he has been active for a very long time, 20-30 years almost. He is very established and very good at what he does."*. P1 mentioned another reputed developer named `David`, who publishes his plugins in https://blindhelp.net, a website popular among blind users in South Asia. Similarly, P8 mentioned reaching out to `Joseph`, who leads a team of plugin developers, to create a "Table Navigator" plugin for NVDA. P4 also named 3 sighted plugin developers who can be trusted but need to be paid to write a custom plugin.

Besides these reputed developers, the participants mentioned that they need to use plugins from unknown developers, from their personal websites, blogs, and GitHub repositories. Some of these plugins are not up to the quality that they get from the reputed developers.

4.2.2 Challenges of Developing Plugins. All participants were enthusiastic about the idea of writing their own plugin. They considered this idea "empowering" to blind users. P3 shared why she wants to write her plugin:

> *I would love to. There are so many things I come across every day that are frustrating for no reasons because if I were sighted, I would just click click click, and good to go. It is very frustrating and can be very demoralizing that I need all of this help and sighted assistance when it's no fault of my own. It's not that I don't have the cognitive ability, it's simply that I can not see and the person who created it didn't think of someone like me using it. So, yes I would definitely write scripts by myself if I could for sure.*

However, 2 participants who attempted to write a plugin before, informed the challenges in doing so. Only P11 succeeded in writing a JAWS script. P1, who failed to write an NVDA add-on in Python, attributed his failure to the inaccessibility of programming IDEs and Compilers. For programming languages like Python, accessibility does not entail reading out texts (i.e., code) but the scopes, indentations, missing variables, and other contextual information, P1 elaborated. Neither the Editor nor the Compiler provides adequate audio feedback to blind users to write their code, he added. In fact, P1 failed to write a plugin before, with another programming language, C++, because of the same issue – Compilers not being accessible.

Besides accessibility issues of IDEs and Compilers, the participants who did not have a prior programming background expressed that it would be daunting for them to learn how to code for the sake of writing their own scripts. They were also concerned about the steep learning curve and the return-on-investment of their time. P11 mentioned another challenge in developing plugins as navigating the lengthy plugin development guidelines from NVDA [4] and JAWS [3]. Considering the amount of time needed to develop programming skills and navigate the guidelines, P4 commented: *"I probably would not. Even if I really had to, I probably would not look at it"*. P14 thinks it is unfair for blind users to learn programming to use a partially-accessible software that most people use without putting in an extra effort.

4.2.3 Lack of Financial Incentive. 5 participants believe that not having financial incentives plays a vital role in the lack of enthusiasm to develop new plugins. P11 believes that relying on third parties to develop plugins out of kindness is unlikely to come true. P2 was interested in developing plugins only if it benefits his career. In his words: *I would love to start writing my own scripts if it helps me make myself more marketable.*

4.3 How Plugins are Distributed and Deployed?

4.3.1 No Centralized Plugin Repository. All participants emphasized that there is no central repository for plugins. Only NVDA has a community add-ons page, where developers can upload plugins, which the participants recognized as "a good sign because it makes things easier". However, they considered the number of add-ons (i.e., plugins) in NVDA community page is inadequate — many plugins they use routinely are not available there, for which they must rely on individual developers' websites, blogs, or GitHub pages. The participants mentioned that they were not aware of any central repository for other screen readers like `JAWS` or `System`

Access. Thus, finding a script for those screen readers is even more challenging, and they described the ordeal as ad-hoc, unstructured, and predominantly "Google Search"-driven. In fact, not having a central repository affects the users and developers alike—the users cannot group and compare similar plugins; the plugin-developers cannot upload their work to a more "recognized" place.

4.3.2 Challenges in Finding the "Right" Plugin. Not having a central repository makes it difficult for blind users to find the "right" plugin. Unfortunately, this difficultly is further compounded by the following factors: (i) the name of a plugin can be changed in an update without keeping a change-log; (ii) a plugin for NVDA does not work for JAWS (and vice versa); (iii) most plugins are not backward compatible, i.e., the same version of a plugin may not work for a different version of the same screen reader; and (iv) misleading or no documentation, which is common for third-party plugins in NVDA.

P8, who identified himself as an expert, pointed out that tech-savvy users are mostly interested in trying out different plugins from third-party websites, but they, too, could struggle to find the "right" plugin at times. For example, he shared his experience of finding a plugin named Eloquence – he originally downloaded it from the community page of NVDA, but it stopped working after an update. He tried to make this plugin work for a long time in vain. Later, he discovered another version of this plugin from a different source, an IBM website, which was functional.

4.3.3 Ad-Hoc Approaches to Find a Plugin. P8 mentioned a Telegram[1] community, where fellow blind users share Google Drive links. Usually, these links point to zip files containing a bundle of plugins. P1 and P3 also mentioned that they regularly check the websites of popular plugin developers. P8 pointed out a number of GitHub repositories where third-party developers create and store plugins. P3 mentioned that she is fortunate to have a developer friend who usually directs her to a website or make a plugin for her, if needed.

Not all participants were as fortunate as P3. Most participants had difficulty finding a plugin they need – they usually had no way of requesting a developer to write a plugin. 4 non-expert participants expressed their frustration as not knowing where to look at to find a plugin they need. Their frustrations were so dire that 3 of them asked the interviewers during this study whether they (interviewers) could help them find certain plugins.

2 participants reached out to screen reader's vendors for plugin support. However, their experience was mostly negative. For example, when P2 reached to Freedom Scientific, the vendor of JAWS screen reader, requesting a plugin, they declined.

> *You call Freedom Scientific and tell them what you want. If they need to create a script, they will tell you how much it's going to cost you. Sometimes, they will just tell you that they don't do scripting for these tasks.*

P4 also had a similar experience with the vendor of System Access. He said, *"They did not give any explanation. I am not happy about it. I might have to find another screen reader".*

[1] https://telegram.org/

4.3.4 Installing Plugins is Easy. All participants stated that installing a plugin is surprisingly simple – plugin files usually have a known extension (e.g., .exe or .nvda-addon) on which they need to press Enter. P3 articulated this process as follows:

> *I go to the website, make sure this is the one I'm looking for or a similar one. There should be an executable file. I will download it, save it, go to the folder where I saved it, and press Enter on it. Usually, that does the trick, like bringing up the wizard or self-install and say get ready. Next time I run the program, these scripts are already in place. It is a straightforward process.*

Installing a plugin is so easy and convenient that participants preferred plugins for solving their accessibility issues over other techniques.

4.3.5 Uninstalling Plugins is Unlikely. The participants mentioned that the Plugin Manager in their screen readers supports disabling or uninstalling a plugin, but they expressed less to no interest in doing so. For instance, P1 never uninstalled any of his plugins, even though some of those were unnecessary. When asked, P1 replied that uninstalling a plugin is tricky—one needs to remember the name of the plugin to uninstall it. But like most participants, he often forgets the association of a plugin's name to its function after some time and fears uninstalling a wrong one. On a separate note, he informed that NVDA introduced a plugin that can check the compatibility of installed plugins, and notifies if a plugin becomes unsupported in the current version of the screen reader. However, it does not remove those (unsupported) plugins automatically.

We note that unnecessary plugins can listen to certain keyboard shortcuts or overlap with a newly installed plugin, causing undesirable accessibility issues. In sum, once installed, it is unlikely that participants uninstall a plugin.

4.3.6 Insufficient Plugins for Non-native English Speakers. 4 participants, who needed Text-to-Speech support for dual languages, had difficulty finding plugins that work for non-Latin scripts. They experienced a similar problem with plugins for recognizing optical characters (OCR) in non-Latin scripts. P6 stressed the lack of plugins to support non-Latin languages in general. He once reached out to Google's regional development head regarding OCR issues but received no response.

4.4 How Plugins are Maintained?

One of the recurrent themes throughout our interviews was the maintenance, updates, and compatibility of plugins. All participants complained that the plugins are not well-maintained. Once a plugin is distributed, it is hardly maintained or updated; and users have nowhere to report bugs. Without maintenance, most plugins become unusable after some time.

P3 pointed out a deeper issue with plugin development—most plugins are developed on ad-hoc basis, and the developers have no incentive to maintain their plugins after deployment. As a result, many plugins do not receive updates to work with the newer versions of a screen reader. Sometimes, a plugin receives updates, but breaks some functionalities and causes compatibility issues.

P1 and P8 reported that when NVDA migrated from Python 2 to Python 3 (v.19.3), most plugins in their systems became stale. But

P1 was yet to give up his plugin collection, thus reverted back to NVDA v.19.2, despite knowing the benefits of the newer version. He recounted that moment as follows:

> *The plugins I use, are not updated with the versions of NVDA. That is the reason I don't update NVDA. ... I will lose what I already have ... The thing is, these plugins support NVDA up to version 19.2. Basically, these plugins are written by third-party developers. If they do not update their plugins, no one else does.*

The participants also mentioned that sometimes a plugin no longer work when the operating system or application software gets updated. P6 provided an example about a *Railway* service app that broke with JAWS after getting updated.

4.4.1 Quality Control of Plugins. The participants complained about "so many" poor quality plugins, i.e., plugins that do not work properly, and have missing shortcuts or unassigned shortcuts. P3 and P8 mentioned installing several plugins before finding the one that worked as described. P4 identified a memory leak problem with a plugin in System Access when using Microsoft Excel.

Most participants were frustrated by the fact that there is not much effort to ensure the quality of plugins. Only NVDA checks the quality of plugins before posting them on their community page. However, this effort is not sufficient, because users install plugins from other sources that do not ensure quality. Moreover, there is no such community page for other popular screen readers like JAWS or System Access. P1 emphasized that all plugins must go through a review process. But he was not specific about the reviewer(s) when asked.

4.4.2 Rating System for Plugins. The participants wished to have a rating system for plugins to foster trust. P4 described a hypothetical system, where users could upvote a useful plugin. According to him, users should create a poll about a plugin before using it. Other users, who used that plugin, could vote for it. A higher rating (e.g., "50 people found this plugin helpful") would indicate a good-quality plugin.

4.5 Usability Issues of Plugins

4.5.1 Poor Documentation. The participants indicated that most plugins do not have proper documentation. P8, who received several plugins inside a zip folder, reported getting no descriptions or documentation of those plugins. He also mentioned that many plugins do not assign any hotkeys (i.e., shortcuts) by default, and the user must assign hotkeys to use them for the first time. Unfortunately, developers often do not provide enough documentation on how to assign a hotkey. Consequently, these plugins become useless.

Some participants tried plugins with pre-assigned shortcuts. However, without a description, they failed to execute the shortcuts, thereby rendering those plugins equally useless. Surprisingly, some participants tried plugins with pre-assigned shortcuts, plus a description of shortcuts, yet, they found those plugins useless because shortcuts do not function the way they were described.

In general, we found it hard for blind users to configure a plugin. For example, P1 tried to configure `Dual Voice` plugin in NVDA but failed, even though he followed the documentation. P8 expressed difficulty in configuring `Table Navigator` plugin in NVDA— the documentation instructed to set an action key, which he failed to find. He described the problem as follows:

> *So, the problem is to find out what are the hotkeys and how to assign them. Even if I assign something, I need to know if it is going to collide with another or not. So there is no direct connection between the users and contributors. Maybe someone is uploading it to GitHub, working on it, and then it becomes available elsewhere. But there is no plan for how to use it, how to assign new hotkeys, whether it is contradictory or not.*

4.5.2 Introducing New Shortcuts. We note that blind users need to memorize numerous keyboard shortcuts to efficiently interact with a screen reader. Unfortunately, using a new plugin implies adding a new set of shortcuts in their shortcut repertoire. Sometimes, these new shortcuts could conflict with existing shortcuts. Furthermore, these shortcuts might be unconventional, making them difficult to memorize.

The issue of multiple plugins having the same shortcut came up often in our study. To resolve such conflicts, the participants reported uninstalling the newly installed plugin. Occasionally, they uninstalled a less useful one from their existing plugins to keep the new plugin.

There is no guideline on how developers should assign shortcuts in a plugin. In this regard, P3 disappointingly said, *"Nothing is universal, there is no established code. Every program, every website, every app used to have their own idea of keyboard shortcuts."* Also, there is no easy way to manage screen readers' built-in shortcuts along with new shortcuts offered by the installed plugins. The participants identified these "new" shortcuts as the pain-point of using plugins.

4.5.3 Accessibility being Ignored during App Development. 4 participants stated that the application developers are not always concerned about making their apps accessible. P6 was frustrated and said, *"People who build software, they do not build software for people like us"*. P12 believes that the minority of the users with disabilities could be a contributing factor. P11 urges on developing strong advocacy for people with disabilities. In his own words:

> *We need strong leadership to promote and maintain our ideas and causes. Consider a central website that would list all available screen reader plugins and a tool that could guide people to build new plugins. A third necessary thing would be advocacy. Think about a scenario where making a script will not make the software accessible, such as using graphics in place of texts. The task of this advocacy group would be to let developers know that their software is not accessible because of their fault.*

4.6 Security Issues with Plugins

Because of the unstructured distribution of plugins, it is possible to package malware as plugins. In fact, all participants were concerned about mixing plugins with malware because most plugins are distributed without any security check.

To vet a plugin, P3 stated that she considers plugins with very poor documentation, including typos, grammatical errors, and misleading descriptions, as potential malware. P8 indicated that he looked at a developer's history to vet their plugins. He reminded that most of his peers were unaware of the security risk of using plugins contained in a zip folder. He additionally shared his experience of being a victim of ransomware, giving away the full control of his computer to a hacker who provided an executable file in the name of a plugin. With frustration, he said, *"My Facebook, Skype, Gmail, and about 350 GB of data were all gone".*

4.6.1 Preventive Measures. The participants reported the following measures to protect themselves from potential malware in the guise of plugins: (i) relying on trusted friends (P3, P14); (ii) relying on word-of-mouth from the community (P3, P8); (iii) downloading plugins from trusted developers (P1, P3, P8, P14); (iv) looking for typos and inconsistencies in a plugin's description (P3, P14); and (v) checking the history of a plugin developer (P3, P8). P4 believes *"it is a bad idea"* to install plugins from unknown sources without checking for malware.

5 RECOMMENDATIONS

5.1 Raising Awareness about Screen Reader Plugins

Due to our inclusion criterion that a participant should be familiar with screen readers' plugins, we found that most participants in our study were expert screen reader users who are well-aware of plugins. However, participants who were non-experts, did not realize that plugins could be shipped as part of their screen readers, and they were already using some of these plugins without knowing. A few participants reported installing malware instead of a plugin and one also reported getting scammed by someone who promised to help him out with a plugin. As such, we realize that there is a big value in educating blind users about plugins, their benefits, and potential dangers. An equally important step is to make them aware of the security risks of certain plugins, how to flag them, and best practices in online safety.

5.2 Building a Central Plugin Store

We found a pressing need for a central repository to store, distribute, and update plugins. Thus, we recommend building a central repository for all available plugins, categorized by screen readers and their versions. In this store, all plugins should have a rating based on their quality, merits, and usefulness. In addition to a rating system, we also suggest each plugin to have have a clear and informative description, possible security issues, documentation on shortcuts, and a simple tutorial showcasing its functionalities. In this regard, we note that the community add-on page[2] of NVDA is a good starting point. We can draw further inspiration from the *App Store*[3] for iOS apps. Like the apps in *App Store*, the plugins in this store should receive automatic updates without causing compatibility issues.

[2]https://addons.nvda-project.org/index.en.html
[3]https://www.apple.com/app-store/

5.3 Engaging Third-Party Developers

Since the number of third-party plugin developers is limited, and most of them maintain their own websites, blind users often struggle to find certain plugins online. To alleviate this problem, there should be a central website or may even be a certain portion of the plugin store that would contain all the necessary information associated with the known plugin developers. Like plugins, the developers should have their own rating too. The system we suggest should also contain the contact information of the developers so that a user can reach out to them if needed.

We also found some of our participants are willing to purchase a plugin if it is useful. Thus, a trusted, easy-to-navigate, and accessible transaction system should also be in place so that users could send money to the plugin developers. We also need a standard set of rules and regulations to manage these interactions.

5.4 Listing Third-Party Websites

The participants spoke about third-party websites, forums, GitHub repositories that they found useful when looking for a plugin. Incorporating these third-party websites in a system is important. Thus, we believe that there should be a central website or a portion of the plugin store to contain the links to all these websites, as well as their helpfulness ratings. Some participants also reported the unusually long waiting times to get their queries answered. We believe that introducing financial incentives would encourage more people to help out on these websites and reduce the waiting time for the users.

5.5 Encouraging Individual Plugin Development

Almost all of our participants were interested in developing their own plugins. However, they reported a number of issues that prevent them from trying to do so. The most common issues they reported were the steep learning curve and the extensive plugin developer's guide. Instead of going through 20- to 30-page long documentation, we recommend to have a step-by-step dialog-based system that could guide them to develop their own plugin.

6 DISCUSSION

Our study revealed that plugins are effective in addressing application-specific accessibility issues. However, only a handful of people develop plugins, and they are well-recognized in the blind community. Usually, these developers first release a plugin in their blog sites, then online forums, and GitHub. If enough users download a plugin and provide positive feedback, screen reader vendors digest that plugin and distribute it as part of the main product bundle. Most participants consider that plugins are useful. In some cases, plugins solved a certain problem that they face; in other cases, plugins just made them more productive. Thus, we believe a good quality plugin can have a large-scale impact on the blind community.

Since installing a plugin is easy for blind users, we are intrigued by this mode of delivering patches that can address various software accessibility issues. When blind users face certain accessibility issues with an application, they often need immediate solution rather than waiting for the screen reader developers to fix, presumably in

the next release cycle. Plugins, in contrast, could be a fast, effective alternative in this matter.

Designing a centralized, trustworthy *plugin store* for the blind community poses an open research challenge. We believe that part of this challenge is addressed by research in community-based recommendation system [20], peer-production [5], and App Store review guideline [29].

The hunt for a plugin was identified as a major hurdle. The participants mentioned no winning strategy—they needed to search around different websites, including personal blogs, online forums, and social media. They also mentioned that it is impossible to check the validity of a particular plugin without installing it first. They only rely on the descriptions provided by plugin developers to make a conscious decision on whether they would install it. Sometimes, they rely on word-of-mouth or the reputation of a plugin developer. These strategies and and challenges pose an interesting research problem for security researchers.

Keeping the installed plugins updated is challenging. Many plugins do not receive regular updates from the developers and become obsolete as soon as the screen reader or the application updates. This finding is consistent with the literature [15] and is a common phenomenon with assistive technologies. We believe that while eliminating the need for retrofitting an application via screen reader plugins would be ideal, it is a laudable goal and unlikely to happen in the near future. Hence, accessibility researchers should embrace the plugin-based distribution model and focus on improving this model.

Accessibility researchers who work on making screen readers better can make their work more reachable if they can create plugins based on their prototype. This would serve two purposes: firstly, it would serve as a test run for the work the researchers have done; secondly, it could help blind users in need. If the researchers can make their plugins available to the central repository, the users could easily find these plugins and can rate them accordingly.

Limitation. Most participants in the study were experts due to our inclusion criteria (e.g., familiarity with screen readers' plugin). Without this criterion, we were afraid that we could not have gained a deeper insight about the ecosystem of plugins. As such, our current findings lacked input from those who are not familiar with screen reader plugins. However, we believe that the impact of not having their inputs is small because the plugin literacy among general screen reader users is low. As an example, our study revealed that non-expert participants who were aware of plugins, were unaware of many aspects of plugins (e.g., some plugins are shipped with screen readers).

7 CONCLUSION

In this paper, we aim at understanding the screen readers' plugins. To that end, we conducted a study with 14 blind users. Our study revealed that plugins could enhance the capabilities of screen readers in making applications more accessible and usable. While finding these plugins is challenging, the participants reported the installation to be very easy. In addition, all participants stated that when developed properly, plugins can be valuable parts of screen readers. The study also revealed that the current state of maintaining screen readers' plugins is "no maintenance at all". Furthermore, plugins have issues with availability, compatibility, unstructured distribution, poor documentation, and security risks. Finally, we found a big gap between the plugin developers and the end-users.

To make screen reader plugins more effective, we recommend creating a central repository, a rating system, and a financial model. Furthermore, we envision a system providing step-by-step instructions to blind users that could empower them to create their own plugins. In the future, we will conduct a larger-scale analysis on existing plugins to sketch out a community-based central plugin repository that is secure, accessible, and well-maintained. Making the plugin-based content distribution more streamlined is also left for future research.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their insightful feedback. Research reported in this publication was supported in part by National Eye Institute (NEI) of the National Institutes of Health (NIH) under award number R01EY03008501A1 (subaward number 87527/2/1159967). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

REFERENCES

[1] 2018. What's New in JAWS 2018 Screen Reading Software. Retrieved September 19, 2018 from https://www.freedomscientific.com/downloads/JAWS/JAWSWhatsNew
[2] 2020. NV Access. https://www.nvaccess.org/. (Accessed on 09/20/2018).
[3] 2021. JAWS Script Developer Guide. https://support.freedomscientific.com/Content/Documents/Other/ScriptManual/01-0_Introduction.htm
[4] 2021. NVDA Add-on Developer Guide. https://github.com/nvdaaddons/devguide/wiki/NVDA%20Add-on%20Development%20Guide
[5] 2021. Peer Production. https://wiki.p2pfoundation.net/Peer_Production
[6] Addons.mozilla.org (AMO). 2020. Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/. Accessed: 2020-06-29.
[7] Apple. 2011. *NSAccessibility*. https://developer.apple.com/documentation/appkit/accessibility_for_macos/nsaccessibility [Accessed: 2020-06-29].
[8] Apple Inc. 2020. VoiceOver. https://www.apple.com/accessibility/osx/voiceover/.
[9] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2015. StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3043–3052. https://doi.org/10.1145/2702123.2702589
[10] Eric Bergman and Earl Johnson. 1995. Towards accessible human-computer interaction. *Advances in human-computer interaction* 5, 1 (1995), 87–114.
[11] Jeffrey P. Bigham. 2007. Accessmonkey: enabling and sharing end user accessibility improvements. *SIGACCESS Access. Comput.* 89 (2007), 3–6. https://doi.org/10.1145/1328567.1328568
[12] Jeffrey P. Bigham, Wendy Chisholm, and Richard E. Ladner. 2010. WebAnywhere: experiences with a new delivery model for access technology. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. ACM, 1806007, 1–4. https://doi.org/10.1145/1805986.1806007
[13] Jeffrey P Bigham and Richard E Ladner. 2007. Accessmonkey: a collaborative scripting framework for web users and developers. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*. 25–34.
[14] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. Ramakrishnan. 2017. Speed-Dial: A Surrogate Mouse for Non-Visual Web Browsing. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 3132531, 110–119. https://doi.org/10.1145/3132525.3132531
[15] 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
[16] 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
[17] Dorian Birsan. 2005. On plug-ins and extensible architectures. *Queue* 3, 2 (2005), 40–46.

[18] Yevgen Borodin, Faisal Ahmed, Muhammad Asiful Islam, Yury Puzis, Valentyn Melnyk, Song Feng, I. V. Ramakrishnan, and Glenn Dausch. 2010. Hearsay: a new generation context-driven multi-modal assistive web browser. In *International World Wide Web Conference (WWW'10)*. 1233–1236.

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

[20] Debajit Datta, Navamani T M, and Rajvardhan Deshmukh. 2020. Products and Movie Recommendation System for Social Networking Sites. *International Journal of Scientific & Technology Research* 9 (10 2020), 262–270.

[21] James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut: User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 225–234. https://doi.org/10.1145/2047196.2047226

[22] Leo Ferres, Petro Verkhogliad, Gitte Lindgaard, Louis Boucher, Antoine Chretien, and Martin Lachance. 2007. Improving Accessibility to Statistical Graphs: The IGraph-Lite System. In *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility* (Tempe, Arizona, USA) *(Assets '07)*. Association for Computing Machinery, New York, NY, USA, 67–74. https://doi.org/10.1145/1296843.1296857

[23] Prathik Gadde and Davide Bolchini. 2014. From Screen Reading to Aural Glancing: Towards Instant Access to Key Page Sections. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility* (Rochester, New York, USA) *(ASSETS '14)*. Association for Computing Machinery, New York, NY, USA, 67–74. https://doi.org/10.1145/2661334.2661363

[24] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/964442.964461

[25] Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. 2014. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. 204–217.

[26] Kip Harris. 2006. Challenges and solutions for screen reader/I.T. interoperability. *SIGACCESS Access. Comput.* 85 (2006), 10–20. https://doi.org/10.1145/1166118.1166120

[27] Michael Heron, Vicki L. Hanson, and Ian W. Ricketts. 2013. ACCESS: A Technical Framework for Adaptive Accessibility Support. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*. ACM, New York, NY, USA, 33–42. https://doi.org/10.1145/2494603.2480316

[28] Chris Hofstader. [n.d.]. The Death Of Screen Reader Innovation. https://www.chrishofstader.com/the-death-of-screen-reader-innovation/.

[29] Ana Manzano León, Cesar Bernal Bravo, and Antonia Rodríguez Fernández. 2017. Review of Android and iOS tablet apps in Spanish to improve reading and writing skills of children with dyslexia. *Procedia-Social and Behavioral Sciences* 237 (2017), 1383–1389.

[30] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1719–1728.

[31] Greg Little, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser Kandogan. 2007. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. 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, 943–946. https://doi.org/10.1145/1240624.1240767

[32] Microsoft. 2020. Business Apps – Microsoft AppSource. https://appsource.microsoft.com/en-us/marketplace/apps?src=office&product=office. Accessed: 2020-06-29.

[33] Microsoft Inc. 2020. UI Automation Overview. http://msdn.microsoft.com/en-us/library/ms747327.aspx

[34] NVAccess. 2020. nvda/source/appModules at master – nvaccess/nvda – GitHub. https://github.com/nvaccess/nvda/tree/master/source/appModules. Accessed: 2020-06-29.

[35] Robert Reimann, Alan Cooper, David Cronin, and Chris Noessel. 2014. *About Face: The Essentials of Interaction Design, 4th Edition*.

[36] Daisuke Sato, Masatomo Kobayashi, Hironobu Takagi, and Chieko Asakawa. 2009. What's next? a visual editor for correcting reading order. In *IFIP Conference on Human-Computer Interaction*. Springer, 364–377.

[37] Daisuke Sato, Hironobu Takagi, Masatomo Kobayashi, Shinya Kawanaka, and Chieko Asakawa. 2010. Exploratory analysis of collaborative web accessibility improvement. *ACM Transactions on Accessible Computing (TACCESS)* 3, 2 (2010), 1–30.

[38] Hironobu Takagi, Shinya Kawanaka, Masatomo Kobayashi, Takashi Itoh, and Chieko Asakawa. 2008. Social accessibility: achieving accessibility through collaborative metadata authoring. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*. 193–200.

[39] Hironobu Takagi, Shinya Kawanaka, Masatomo Kobayashi, Daisuke Sato, and Chieko Asakawa. 2009. Collaborative web accessibility improvement: challenges and possibilities. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*. 195–202.

[40] Reinhard Wolfinger. 2008. Plug-in architecture and design guidelines for customizable enterprise applications. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. 893–894.