Understanding the Usages, Lifecycle, and Opportunities of Screen Readers' Plugins

FARHANI MOMOTAZ, MD EHTESHAM-UL-HAQUE, and SYED MASUM BILLAH,
Pennsylvania State University

Screen reader plugins are small pieces of code that blind users can download and install to enhance the capabilities of their screen readers. This article aims to understand why blind users use these plugins, as well as how these plugins are developed, deployed, and maintained. To this end, we conducted an interview study with 14 blind users to gain individual perspectives and analyzed 2,000 online posts scraped from three plugin-related forums to gain the community perspective. Our study revealed that screen reader users rely on plugins for various reasons, such as to improve the usability of screen readers and application software, to make partially accessible applications accessible, and to receive custom auditory feedback. Furthermore, installing plugins is easy; uninstalling them is unlikely; and finding them online is ad hoc, challenging, and sometimes poses security threats. In addition, developing screen reader plugins is technically demanding; only a handful of people develop plugins. Unfortunately, most plugins do not receive updates once distributed and become obsolete. The lack of financial incentives plays in the slow growth of the plugin ecosystem. Further, we outlined the complex, tripartite collaboration among individual blind users, their online communities, and developer communities in creating a plugin. Additionally, we reported several phenomena within and between these communities that are likely to influence a plugin's development. Based on our findings, we recommend creating a community-driven repository for all plugins hosted on a peer-to-peer infrastructure, engaging third-party developers, and raising general awareness about the benefits and dangers of plugins. We believe our findings will inspire HCI researchers to embrace the plugin-based distribution model as an effective way to combat accessibility and usability problems in non-visual interaction and to investigate potential ways to improve the collaboration between blind users and developer communities.

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

Additional Key Words and Phrases: NVDA, screen reader, assistive technology, visual impairments, plugin, extension, scripts

ACM Reference format:
Farhani Momotaz, Md Ehtesham-Ul-Haque, and Syed Masum Billah. 2023. Understanding the Usages, Lifecycle, and Opportunities of Screen Readers' Plugins. *ACM Trans. Access. Comput.* 16, 2, Article 17 (July 2023), 35 pages.
https://doi.org/10.1145/3582697

This work was supported in part by NIH sub-award 87527/2/1159967. The content is solely the responsibility of the authors and does not necessarily represent the views of the National Institutes of Health.
Authors' addresses: F. Momotaz, Md Ehtesham-Ul-Haque, and S. M. Billah, College of Information Sciences and Technology Pennsylvania State University, Westgate Building, University Park, PA 16802; emails: {fbm5122, mfe5232, sbillah}@psu.edu.

1936-7228/2023/07-ART17 $15.00
https://doi.org/10.1145/3582697

1 INTRODUCTION

Screen readers, such as NVDA [3], JAWS [2], and VoiceOver [9], are special-purpose assistive technologies that enable blind users to interact with computers. The screen reader compensates for the blind users' inability to see a **Graphical User Interface (GUI)** by narrating on-screen visual content via audio and implements keyboard shortcuts as an alternative to using a mouse to point and click on graphical elements. These technologies are essential for blind people to access education, employment, and other aspects of life in the digital era.

From a technical standpoint, screen readers depend on the built-in accessibility support of the **Operating System (OS)**, known as *Accessibility APIs*. Examples of such APIs include MSAA [53] and **User Interface (UI)** Automation [55] on Windows, and NSAccessibility [8] on OSX. These APIs extract textual metadata of the GUI elements (e.g., the name of a button, the alt-text of an image) on the screen readers' behalf. Further, they allow screen readers to register various listeners to receive notifications of UI changes, such as when the focused UI has been changed or a drop-down has been expanded/collapsed. Using these APIs, screen readers thus create a *manifest* interface that blind users can perceive and manipulate non-visually through a keyboard [69].

Unfortunately, Accessibility APIs have several notable shortcomings. For instance, they are buggy, idiosyncratic, and can lose some essential notifications [17, 18]. Sometimes, they lead to unbounded memory consumption (i.e., memory leaks) when UI elements change on the screen [27]. Additionally, they are inefficient—can cause a spike in CPU usage as high as 30% [62] and can reduce the battery life in smartphones by as much as 52% [82]. As a result, screen readers frequently freeze, become unresponsive to user inputs, and crash prematurely [59]—all of which cause loss of contextual data and disrupt the user experience.

Besides the preceding shortcomings, accessibility issues can stem from the application developers. Developers often provide unhelpful or generic textual descriptions for different UI elements (e.g., graphics and charts) within an application, making it partially accessible to screen readers [41]. For convenience, they can use a GUI framework to develop an application, but the framework may not be fully compatible with the OS's native accessibility support [17, 25]. In addition, they can choose to encode information spatially that is only interactable with a mouse cursor [13].

To overcome these challenges, at least to some extent, screen readers support *plugins*—small pieces of downloadable code—that can extend screen readers' built-in functionalities, and can repair application-specific accessibility issues retroactively. For example, *Beep Keyboard*[1] is a plugin that notifies NVDA users with a beep sound when certain keyboard events occur (e.g., beep on keystroke when CAPS LOCK is ON). We provide a list of major NVDA plugins and their functions in Table A.1. Plugins are also referred to as *add-ons*, *add-ins*, *modules*, *extensions*, or *custom scripts*. Domain experts with technical skills typically develop plugins, which they distribute online so others can download and benefit from them.

Certain screen reader vendors (e.g., NVDA) distribute popular plugins as part of their original package. Therefore, it is likely that many blind users, although unbeknownst to them, have used screen reader plugins. Although the literature on screen readers' accessibility is extensive and growing, to our knowledge, there is no prior work on understanding how screen readers' plugins contribute to user experience, how they are created and in which contexts, the design implications of using them, and what opportunities these plugin can offer to different stakeholders (e.g., screen reader vendors, blind users, plugin developers, and accessibility researchers). Put differently, we aim to understand screen readers' plugins as first-class citizens. More specifically, we asked the following **Research Questions (RQs)**:

[1]https://addons.nvda-project.org/addons/beepKeyboard.en.html.

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

We investigated the preceding RQs from *two perspectives*: the perspective of individual blind users who consume plugins and the perspective of the communities of blind users and developers who contribute to developing and maintaining these plugins online.

We gained the first perspective by conducting a study with 14 blind participants (Section 3). This study revealed that plugins are used for various 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 rarely 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 Drive) 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. A lack of financial incentives can contribute to the slow growth of the plugin ecosystem.

The second perspective (Section 4) is achieved by analyzing online data related to NVDA's plugins because NVDA has acquired a substantial user base for its quality and zero cost [79]. Additionally, its source code is open [64], which allows us to analyze its plugins at the source-code level. We analyzed NVDA's 160 plugins including 76 third-party [65] and 84 built-in [60] plugins (Section 4.1). In addition, we scraped 41,333 posts from two NVDA user forums and 8,000 issues from NVDA's GitHub repository (Section 4.2). We sampled 1,000 posts and 1,000 GitHub issues from these data sources and analyzed them manually to report our findings.

The analysis of plugin data expanded our initial understanding of why blind users use plugins to what benefits these plugins can offer. For example, we found that some plugins can gracefully kill the screen reader when frozen, so the users do not need to restart their computers. In addition, plugins can improve the accessibility of the clipboard (e.g., reading aloud the copied text without pasting it), as well as rich text (e.g., reading e-books, emoticons, Unicode symbols, and indentation characters in source code like Python). They also enable using multiple audio synthesizers, such as two voices for reading two languages: one with a Latin script and another for non-Latin (e.g., Arabic, Chinese, Hebrew, Persian, and Asian) scripts. In addition, they allow customizing input (e.g., input Braille characters with a keyboard), easing the navigation of complex UI elements (e.g., using Num. keys to access rows and columns in a table), and interfacing with computer vision algorithms (e.g., **Optical Character Recognition (OCR)** and image descriptors). Some plugins capture crash reports in a way that is useful for developers to diagnose the underlying problem.

Similarly, our analyses of online forums data revealed complex tripartite collaborations among individual users and the communities of blind users and plugin developers. We found that blind users usually create a post on their familiar online forum reporting issues with an application and seeking help with accessibility issues. Sometimes, these posts are accompanied by personal stories or personal commitments (e.g., offering money or can work with developers). Then, forum members can direct them to an existing or an alternative solution (if available). Otherwise, individuals request the developers by creating an *issue* on NVDA's GitHub repository.

We noticed tensions between the users' and the developers' communities in deciding what user requests the developers should address. Nearly half of the user requests (i.e., issues) are declined by the developers for several reasons, such as developers cannot reproduce the problem reported in the issue; the request involves less popular applications or obsolete applications for which an alternative solution exists; the issue needs special skills or sighted assistance to address; and the issue does not affect many users, hence it is less severe. The users often make compelling arguments to support their requests and sometimes frame them as "bugs" to draw the developers' attention. If the developers decide to fix an issue, they assign a priority (p1 to p4, with p1 being the highest); issues with the highest priority are likely to get fixed.

Our findings implicate that building a community-driven plugin repository hosted on a peer-to-peer infrastructure is promising to sustain existing plugins (e.g., containing change logs, documentation, multiple versions, and user ratings), as well as engage third-party developers. Additionally, there is ample opportunity to develop an intelligent, easy-to-use tool that generates plugins (semi-) automatically. This bot-like tool can lower the technical barrier in creating plugins by general blind users and reduce the workload of a handful of plugin developers. Since installing plugins is easy, accessibility researchers can consider deploying their research prototype as a plugin and seek early feedback from the blind community. In addition, educating screen reader users about plugins' potential benefits and dangers and the dynamics of tripartite collaborations is essential. Finally, plugins can be instrumental in combating accessibility and usability issues in existing screen readers and application software.

This article is an expanded version of a conference paper published in ASSETS 2021 [56]. It expanded our findings from the user study by adding the perspectives of online forums of blind users and developers, who collaborate to produce plugins. Further, we reported several phenomena within and between these forums and identified attributes likely to influence a plugin's development.

2 BACKGROUND AND RELATED WORK

The plugin-based architecture is commonly used in commercial software, such as web browsers, multimedia players, and rich text editors [81]. 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 [19]. 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 [81].

Thus, a plugin-based architecture allows a complex application to function as a smaller but stand-alone 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 [19].

Popular applications usually have their online repositories containing hundreds of plugins. For example, the repositories for Firefox and Microsoft Office 365 contain more than 450,000 [7] and 2,800 [54] plugins, respectively. Compared to popular applications, the number of screen reader plugins is relatively small. For example, the official plugin repository [60] of NVDA contains fewer than 200 plugins, whereas no such repository exists for JAWS (screen reader).

In the following sections, 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 their similarity to developing plugins in screen readers. Finally, we describe prior work focusing on collaboration in online communities, knowledge collaboration, and sharing expertise.

2.1 Accessibility Issues with Screen Readers

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 [37]. Thus, any mismatch between users' effective abilities and the underlying assumptions seriously hampers the effectiveness of UI 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 [37], it is neither feasible nor scalable—users' abilities and preferences may vary, which UI designers may not anticipate [11].

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 [69]. For example, blind users cannot use computer mice to point and click on a UI because a mouse only provides a *visual* feedback (e.g., the cursor) [16], 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 a keyboard's arrow keys.

Unfortunately, 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—for example, the app may become partially accessible [41].

2.2 Application-Specific Plugins to Enhance Accessibility

Many applications support some degree of extensibility by the end users through scripting, plugins, or extensions [28]. For example, the Mac OS supports *AppleScript*, a scripting interface to control applications and override OS settings. Such scripts can automate repetitive tasks, combine features from multiple apps, and create complex workflows to increase usability.

Independently of screen readers, some plugin-based frameworks are designed to make OS-level configuration easy and accessible. For example, Heron et al. [42] proposed 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 an accessibility issue in isolation. For example, WebAnywhere [14] and HearSay [20] are browser plugins that enable screen reading of webpages on public computers that do not have a screen reader installed. Ferres et al. [32] developed 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. [10] developed 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.

The preceding frameworks and plugins require low-level integration with the OS and the host application; however, 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 modify the `HTML` directly, `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 [15, 36, 40, 50, 51, 71, 72, 75, 76]. 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. Examples of some repairs include providing alt-texts for images [72, 75, 76], reordering content navigation [71], performing website-specific customizations [15], and representing web content for aural glancing [36]. 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 OS's native Accessibility API [8, 55] and exposes it to a screen reader plugin. The syntax and semantics of this exposure are screen reader dependent, as specified by the plugin developer guidelines (e.g., for NVDA [5], for JAWS [4]). A screen reader plugin can utilize many functions of the host screen reader, such as obtaining the reference of the focused UI and walking up/down in the DOM-like structure to inject metadata to an anchor 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. In sum, developing a plugin requires domain knowledge of the host application and programming or scripting expertise.

2.5 Online Knowledge Collaboration

The widespread use of the Internet and communication technologies enable collections of individuals with common interests to congregate virtually and pursue shared interests [12, 68]. As a result, communication platforms, ranging from blogs and wikis to groups and Facebook, have emerged as spaces for social interactions, represent new forms of organization [35, 84], and have a wide-ranging impact on product development and knowledge creation [43, 52, 83].

Research in online communities is primarily motivated by an interest in knowledge collaboration [30]. The primary shared objective is "the sharing, transfer, accumulation, transformation, and co-creation of knowledge," which involves the offering of new knowledge and recombining the knowledge of others [30].

Researchers have identified several phenomena in online knowledge collaboration. For example, these collaborations are "loosely coordinated," "self-organizing," and "voluntary" [57], and lack formal structures and hierarchies [26]. Interestingly, unequal participation does not have negative consequences [49]. Faraj et al. [31] described that collaborators in online knowledge collaborations negotiate tensions between changing and retaining established artifacts within systems; they adopt specialized roles, shift production foci, and employ identifiable action patterns[46].

In many online communities, only a few members engage in altruistic behaviors, such as answering questions [33] or moderating communities [24]. Taking part in these altruistic actions can lead to positive self-evaluation of competence because it feels good to help others and the community [78]. Members are also motivated to contribute to online communities by the reputation they gain from their contribution [78], signifying a type of extrinsic motivation [70]. These are consistent with our observations about the forums of blind users and plugin developers.

2.6 Knowledge and Expertise Sharing

Knowledge sharing is an externalization of knowledge in the form of computational or information technology artifacts or repositories [6]. Expertise sharing, in contrast, is "the capability to get the work done or to solve a problem" based on "discussions among knowledgeable actors and less significantly supported by *a priori* externalizations" [67, 77]. Both terms are used in CSCW to emphasize that knowledge is situated in people and location and that the social context is an essential part of using any knowledge [77]. In this work, we consider that the community of blind users shares knowledge, and the community of developers shares expertise.

Table 1. Participant Demographics

ID	Age/ Sex	Screen Reader 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	Congenital	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	Congenital	Student	JAWS, NVDA
P9	40/M	Beginner	Yes	Congenital	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	Congenital	Government employee	NVDA, JAWS
P14	40/F	Expert	No	Congenital	Student disability service	JAWS, NVDA, VoiceOver

LP, light perception.

3 UNDERSTANDING SCREEN READERS' PLUGINS (USERS' PERSPECTIVE)

To understand our RQs from the perspective of blind users, we conducted a user study (IRB approved) with 14 blind participants. Participants' demographics and study details are described in the following.

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 are fluent in English. The participants varied in age from 24 to 52 years ($M = 37$, $SD = 7$). 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 at a university, 2 were in the textile and apparel industry, 1 was 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 and performed remotely via video/audio conferencing tools such as Zoom and phone calls. Two researchers administered the study: one conversed with the participants, whereas 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 [23], 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 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 an individual's expertise. A sampler of questions under RQ1 was 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 do plugins usually provide? (iv) How often do you use a plugin? (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 do you update, install, or uninstall a plugin? (vii) Is there any issue with updating a plugin?

Toward 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 an hour, and participants were compensated with a $30 Amazon or other e-gift card. Each interview culminated with participants making suggestions and recommendations regarding how plugins could be distributed, maintained, and developed.

3.4 Findings: Why Do Blind Users Use Screen Reader Plugins (Users' Perspective)?

The participants mentioned several reasons they use plugins. We categorize these reasons into four themes: blind users need plugins (i) to make inaccessible or partially accessible software accessible, (ii) to modify screen readers' existing features, (iii) to receive audio feedback on keyboard shortcuts or commands, and (iv) to add new shortcuts to screen readers. Overall, plugins make software applications more usable. We elaborate on each of these themes in the following. Note that some participants mentioned plugins as add-ons or scripts.

3.4.1 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 an 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 popular ones. For example, P3, a music enthusiast, who uses a 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—for example, "things are not always labeled properly," and screen readers "do not know what to make of them." Thus, 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. For a richer context, please refer to our later analysis in Section 4.4.

3.4.2 Plugins Can Modify Screen Readers' Existing Feature. Sometimes, the 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 entering or leaving the meeting or someone muting or unmuting 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. Thus, he uses a plugin in NVDA that lets him configure Zoom's notifications.

3.4.3 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.

3.4.4 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 could solve accessibility problems and improve the usability of applications—half (seven) of the participants wished to have more plugins. Of the other half who did not wish for more, three of them were beginners, one was intermediate, and three 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, they did not expect much from plugins. Furthermore, they were financially well off to employ sighted assistants who could provide technical support.

3.5 Findings: How Are Plugins Developed (Users' Perspective)?

3.5.1 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 three 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 they get from reputed developers.

3.5.2 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 reason 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, two 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 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 [5] and JAWS [4]. 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 the extra effort.

3.5.3 Lack of Financial Incentive. Five participants believe that not having financial incentives plays a vital role in the lack of enthusiasm to develop new plugins. P11 believes 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."

In contrast, P3 and P14 mentioned that some developers did ask for financial incentives to develop a plugin. The price tag could range from a few hundred to thousands of dollars, which they could not afford.

3.6 Findings: How Are Plugins Distributed and Deployed (Users' Perspective)?

3.6.1 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 the NVDA community page to be 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.

3.6.2 Challenges in Finding the "Right" Plugin. Not having a central repository makes it difficult for blind users to find the "right" plugin. Unfortunately, this difficulty 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 primarily interested in trying out different plugins from third-party websites. Still, they, too, could sometimes struggle to find the "right" plugin. For example, he shared his experience 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.

3.6.3 Ad Hoc Approaches to Find a Plugin. P8 mentioned a `Telegram`[2]. 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 makes a plugin for her (if needed).

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

Two participants reached out to screen reader's vendors for plugin support. However, their experience was mostly negative. For example, when P2 reached out to Freedom Scientific, the vendor the 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."

3.6.4 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 to 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 saying 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.

[2]https://telegram.org/

3.6.5 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 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 the 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.

3.6.6 Insufficient Plugins for Non-Native English Speakers. Four 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 OCR in non-Latin scripts. P6 stressed the lack of plugins to support non-Latin languages in general. He reached out to Google's regional development head regarding OCR issues but received no response.

3.7 Findings: How Are Plugins Maintained (Users' Perspective)?

3.7.1 Hardly Maintained. One of the recurrent themes throughout our interviews was the maintenance, updates, and compatibility of plugins. All participants complained that the plugins were 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 an 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 and thus reverted 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 works when the OS or application software gets updated. P6 provided an example about a *Railway* service app that broke with JAWS after getting updated.

3.7.2 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 insufficient 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.

3.7.3 Rating System for Plugins. The participants wished to have a rating system for plugins to foster trust. P4 described a hypothetical scenario where users could upvote a helpful plugin. According to him, users should create a poll about a plugin before using it. Then, 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.

3.8 Findings: Usability Issues of Plugins

3.8.1 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. 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 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, rendering those plugins equally useless. Surprisingly, some participants tried plugins with pre-assigned shortcuts plus a description of shortcuts, but 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 configurethe Dual Voice plugin in NVDA but failed, even though he followed the documentation. P8 expressed difficulty configuringthe Table Navigator plugin in NVDA—the documentation instructed them to set an action key, which he failed to find. He described the problem as follows:

> *So, the problem is to find out what the hotkeys are 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.*

3.8.2 Introducing New Shortcuts. We note that blind users need to memorize numerous keyboard shortcuts to interact with a screen reader efficiently. Unfortunately, using a new plugin implies adding a new set of shortcuts to 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 often came up 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." Additionally, 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.

3.9 Findings: 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 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 mentioned 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 complete 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."

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.

4 UNDERSTANDING SCREEN READERS' PLUGINS (COMMUNITY PERSPECTIVE)

Our interview study reveals that plugin are critical to improving the experience of non-visual interaction. However, it also reveals that most blind participants used a small number of plugins, and although they were generally interested in developing their plugins, most of them never attempted to do so. As a result, we realized that our understanding of why blind users use screen readers' plugins (RQ1) and how plugins are developed, deployed, and maintained (RQ2) might not be complete.

This study informed us that NVDA has a community-managed repository (see Section 3.6) that contains more than 150 plugins. Thus, we were curious about what purposes these plugins serve and how they were developed, deployed, and maintained. We scraped online data regarding NVDA and its plugins to gain this complementary perspective and analyzed them manually. The following sections elaborate on our data extraction, analyses, and findings.

4.1 Analysis of Available NVDA Plugins

We scraped 160 plugins for NVDA in June 2020. Seventy-six of them were developed by third parties [65], and 84 were included in NVDA [60]. For each plugin, we extracted the developer-provided short descriptions and supported shortcuts. If available, we reviewed the source code of a plugin, as well as its commit logs, pull requests, and issues number on GitHub [64].

Next, to ensure validity, one of the authors, an expert in developing screen-reading technology, installed third-party plugins and required software (to test application-specific plugins) on a Windows 10 laptop running NVDA version 2020.1, following the documentation and advertised functionalities. This author also set up the NVDA add-on development environment [5] on the same laptop. It took more than 11 hours to complete the setup, which corroborates our earlier finding that developing screen readers' plugins requires substantial technical skills (see Section 3.5.2). The challenges came from finding and installing the compatible packages of Python and SCons[3] libraries, resolving their dependencies and linking errors, defining OS's environment variables, and configuring the runtime environment of NVDA.

Data Analysis. Two researchers manually analyzed scraped data to understand the functionalities of different plugins. They organized functionalities by affinity diagramming [44]. The final diagram contained around 100 affinity notes, moved under 13 affinity groups. We reported these groups by the decreasing order of individual group size (i.e., by the number of affinity notes each

[3]https://scons.org/pages/download.html.

group contains). These numbers (*N*) are shown in parentheses. Each affinity group expands our prior findings on RQ1 (*Why do blind users use screen reader plugins?*). Section 4.4 presents these results.

4.2 Analysis of NVDA Online Forums Data

We scraped data from two NVDA forums (namely, *nvda.groups.io* [66] and *nvda-addons.groups.io* [63]), where NVDA users can post any concerns regarding NVDA, including add-ons or plugins. Sometimes, NVDA developers contribute to these forums, enabling a dynamic interaction between users and developers. Additionally, we scraped "GitHub Issues" [61] of the NVDA repository to understand how NVDA developers collaborate and interact to address issues raised by the users.

We used the *Beautiful Soup* [1] package to scrape 41,333 posts from the preceding two user forums, and *GitHub APIs*[4] to extract 8, 000 issues from GitHub [61]. We cleaned raw forum data by extracting individual `issue_id` or `post_id` (i.e., a unique number attached with each issue or post), `title`, `user_id`, `post_date`, and `post_body`. A single issue or post can have multiple comments in a conversation. Therefore, we first grouped all issues or posts by their IDs, merged them into a single document, and prepended the title. The data extraction and cleaning took place between June 2020 and July 2021.

Data Analysis. Since the volume of extracted data is large, it was not feasible for us to manually analyze all content. Therefore, we randomly sampled 1, 000 posts from the forum data and another 1, 000 issues from the GitHub data that contained plugin-related keywords (e.g., *add-on*, *addon*, *plugin*, *plugin*, and *scripts*). Some forum posts and GitHub issues discussed the same plugin. We identified initial concepts in the sampled data by manual coding. Then, we iterated between inductive and theory-driven approaches to create the coding scheme for the content analysis [48, 80] of individual documents, where each document contained all conversations of a forum post or a GitHub issue. Section 4.5 presents our results.

4.3 Ethical Considerations

Our ethical considerations of using online "public domain" data are informed by the ACM Code of Ethics and Professional Conduct [39], contextual privacy [58], and the inheritance of digital artifacts in social media [22]. For instance, while reporting quotations from community members, we evaluated the potential harm to an individual, the benefits to their community as well as to HCI researchers, and the expectations around how that quotation could be used [58]. Additionally, we were mindful that sometimes using real names or chosen pseudonyms is appropriate, particularly for those who deserve credit for their work [22]. Therefore, instead of anonymizing user names with random digits or paraphrasing quotations, we removed their names but kept their quotations as-is so that the readers can still credit them (by looking up their names through their quotations in search engines).

4.4 Findings: Why Do NVDA Users Use Plugins (Based on Plugin Data)?

Here, we present the findings based on our data analysis (see Section 4) of NVDA's plugins. This analysis expanded our initial understanding of why blind users use plugins to what benefits plugins can offer. Note that some plugins are specific to certain applications, versions, or tasks and thus may not be of interest to many blind users.

4.4.1 Enhancing Usability. We found that plugins are primarily (N = 24) used to enhance the usability of the underlying screen readers, application software, and OS. They modify the default

[4]https://docs.github.com/en/rest/issues.

behaviors of a system or application, provide auditory feedback, and offer new, useful shortcuts. Unsurprisingly, these findings are consistent with our prior findings from the interview study. However, we gained a more nuanced understanding of the role plugins play in enhancing usability, as described in the following.

Protection and Recovery from Unwanted Behavior. Plugins improve usability by preventing unwanted behaviors and helping users recover when such incidents happen. For example, NVDA frequently crashes or freezes. Some plugins allow users to kill NVDA gracefully when frozen, thus averting a system restart that will lose their current work state. When NVDA is busy, some plugins can notify users by providing audio feedback (beep sound). Users can lock their keyboards temporarily to avoid unwanted key presses using plugins. A number of plugins can keep various configurations and audio settings persistent and carry over in different runs (i.e., when a computer restarts), allowing users to recover from crashes. Users can also access the crash report and prior speech history from logs with some plugins to recover their previous work state.

Providing Audio Feedback. Plugins can help users by alerting them during certain keyboard behaviors (e.g., beep on uppercase letters), counting the characters of selected text, and notifying which language is selected for their keyboard (if it supports multi-languages). Users can also use plugins to read clipboard content aloud without pasting it first.

Enhancing Usability of Windows OS Components. The usability of several Windows OS components, such as the clock and calendar, taskbar, and resource monitor, is enhanced by a number of plugins. For example, some plugins make the system clock report times in different time zones; allow users to set/reset alarms and stopwatches with single-key, mnemonic shortcuts (e.g., *S* to *S*tart/*S*top stopwatch, *C* to *C*ancel next alarm); and make the calendar announce important events (e.g., the current day and the number of days left in the current year) upon pressing a function key (e.g., F12). Furthermore, using some plugins, users can easily access elements in Taskbar and SysTray as a list. Some plugins convert the contents of a window into text (e.g., the names of the icons on the desktop) so that users can search by keywords to quickly locate content in that window. Users can also easily access system resources such as CPU, memory, and battery usage shortcuts (e.g., *Shift+1–7*); otherwise, they have to manually find the information by opening the Task Manager and combing through loads of unnecessary information.

4.4.2 Patching Application-Specific Inaccessibility. We noticed that many widely used applications have major accessibility issues, for which plugin-based patches are necessary ($N = 17$). Some plugins improve accessibility by making part of the application accessible, such as the find dialog, search results list, system notifications, toolbar items, context menu, and chat windows. Plugins can also enhance applications by announcing application-specific notifications and status (e.g., used/free space in the Dropbox app), adapting audio feedback (e.g., accessing or silencing a real-time progress bar), and fetching dynamic content on infinite scroll (e.g., Facebook's homepage).

Our user study informed us about the accessibility issues of multimedia players (P1) and music editing software (P3). We found several plugins that improve the accessibility of such software. For example, a plugin for mp3DirectCut allows users to place a marker at the beginning or end of a selected audio clip and manipulate the playback cursor with the keyboard. Similar plugins for other music-editing software notify users about the current audio channel, announce elapsed and remaining time, and allow fast-forwarding of the audio track. Likewise, plugins for media players (e.g., VLC, Winamp) also allow users to control playback (e.g., play, mute, next track) with single-key mnemonic shortcuts and provide audio feedback while fast-forwarding media. In addition, we identified plugins to improve the accessibility of popular productivity software. For example, a plugin for Microsoft Outlook allows users to search in the address book and mark a group of messages as read or unread. A plugin for Calibre e-book manager allows users to search for e-books, announces the available e-books in the library, and reads out metadata of each e-book.

We report applications that are partially accessible and provide a sub-optimal user experience without plugins under the following categories: (i) productivity software (e.g., Outlook, Dropbox, RSS feed), (ii) e-book manager (e.g., Calibre), (iii) multimedia players and editors (e.g., Goldwave, StationPlaylist, mp3DirectCut, VLC player, and Winamp), (iv) instant messaging and video chatting software (e.g., TeamTalk Classic), and (v) peer-to-peer file sharing and bit torrent software (e.g., eMule, and Bit Che).

We further investigated why these applications have more accessibility issues than others and found that some are developed in legacy GUI frameworks (e.g., Win32). As such, we suspect these accessibility issues are attributed to the OS's Accessibility APIs that poorly support legacy frameworks with which these applications are developed. In some cases, applications may invoke too many changes causing screen readers to be unresponsive. For example, 1Password (a popular password manager app) floods property-changed events whenever users move between items in the list. A plugin for this app unregisters property-changed events to prevent NVDA from getting overwhelmed with notifications.

4.4.3 Enhancing Text Accessibility. Thirteen plugins (N = 13) are designed to enhance textual accessibility, such as describing the semantics of phonetic symbols, Unicode symbols, punctuation marks, math equations, and emoticons; differentiating upper/lower case characters, font sizes, and font styles; and accessing the texts in consoles or terminals in logical orders. These plugins sonify symbols and patterns (e.g., phone numbers, email addresses) using earcons [21] or musical tones. Some plugins can also enhance the accessibility of rich text, source code (e.g., Python), and time-series data by announcing the number of chars selected in a sentence fragment, reporting the indentation level or offset of lines and paragraphs, and sonifying with continuously variable pitches. We found a set of plugins that extend the accessibility of math equations by supporting multiple languages in MathPlayer, as well as introducing structural navigation of long math equations and interfacing with braille to display equations.

4.4.4 Multiplexing Audio Synthesizers. Eleven plugins (N = 11) augment the screen reader's built-in audio synthesizer by seamlessly supporting multiple languages or speech technologies. For example, one plugin enables reading mixed-language texts (e.g., English and Chinese) in a way that the built-in synthesizer only reads the Latin scripts, whereas a second synthesizer reads the non-Latin scripts. Another plugin allows a seamless transition between speech synthesis technologies, such as concatenative speech and eloquence. Participants also mentioned using plugins to enable multi-language output in our user study. However, P8 mentioned compatibility issues with such plugins, and P6 mentioned limited support for non-Latin characters.

4.4.5 Masking Input-Output Devices. Ten plugins (N = 10) are designed to mask input devices like keyboard, mouse, and touchscreen and output devices like Bluetooth speakers and braille. For example, inspired by the smartphones' screen readers (e.g., TalkBack and VoiceOver), some plugins bring their touch gestures to touchscreen desktops (e.g., tap and hold to right-click on a UI element).

Others override the default arrow keys to (i) enable hierarchical navigation, where left/right arrow keys move the focus to the next or previous sibling of the currently focused UI element, and up/down arrow keys move the focus to its parent or child element, and (ii) to move the mouse cursor horizontally or vertically by a user-defined unit. Interestingly, this plugin also allows users to label the current mouse position in an application as an important location so that users can retrieve those locations later.

We identified several plugins that can convert characters between different input modalities. For example, one plugin allows users to input Braille characters with the keyboard, and another converts braille input to Unicode braille characters. In addition, a plugin can extend braille output to display Tab and Space characters, auto-scroll, and view two tables simultaneously. A plugin that continuously plays inaudible sound on Bluetooth headphones or speakers is of particular interest because it prevents Bluetooth devices from entering standby mode after a few seconds of inactivity. This fix ensures that when the screen reader speaks again, the first split second of sounds is not lost.

4.4.6 Improving Accessibility of Specific UI Elements. Several plugins (N = 6) make specific UI elements (e.g., *password fields*, *table*, *toolbar*, *context menu*, and *search window*) easy to interact with non-visually. For password fields, some plugins overwrite the default behavior of beeping on each key press or speaking the typed characters as asterisks to not beeping or speaking the typed characters or words as-is.

Some plugins facilitate direct access to a row or column in a table by offering special shortcuts ending with digits to jump to a column/row indexed by that digit in the table. Some enable serial navigation of toolbar elements with arrow keys and the Enter or Space key for simulating a left mouse click on a toolbar item. Finally, for low-vision users to track the screen reader cursor, one plugin highlights the focused item's border with bright-colored (e.g., red, green, and blue) and decorated rectangles (e.g., dotted, solid, dashed).

4.4.7 Simplifying Content Navigation on Webpages. Six plugins (N = 6) offer special shortcuts to simplify content navigation in formatted documents like webpages, word documents, and source codes. These plugins rely on various heuristics—for example, paragraphs are the most important elements in a webpage among all others (e.g., menus, advertisements, images), a paragraph must contain one or more sentences terminated only by punctuation marks (e.g., `.?!`) or spaces (e.g., `\s+`), and paragraphs having the same horizontal offsets or the same font sizes and styles belong to the same hierarchy. When navigating webpage elements by their ARIA roles [45], some plugins force the screen reader to report only a subset of those roles, as specified by the user.

4.4.8 Ensuring Backward Compatibility and Easing Software Updates. Four plugins (N = 4) make certain shortcuts backward compatible if the behavior of these shortcuts is altered in a newer version of NVDA in order to retain a familiar user experience. Additionally, some plugins are designed to ease the ability to check for, download, and apply software updates. This is interesting because, in our user study, participants complained about compatibility and updates of plugins. Only P1 mentioned a plugin that checks the compatibility of installed plugins. Given the small number of such plugins, it is not surprising that no other participants mentioned using them.

4.4.9 Augmenting Clipboard. Three plugins (N = 3) improve the clipboard's usability by reading aloud clipboard content when changed, announcing clipboard operations (e.g., copy, paste, select all, and undo), letting users append text at the clipboard, enabling cloud clipboard, and previewing the paste operation. This is consistent with our user study—P1 reported using such plugins to get audio feedback for clipboard operations. Additionally, one plugin allows users to perform a specialized copy action, such as copying an entire line or word at the cursor location or a fragment between the start (or end) of that line or word and the cursor's current location. Some plugins allow users to review the texts recently spoken by the screen reader and support copying those texts into the clipboard.

4.4.10 Assisting Developers. Three plugins (N = 3) assist blind developers in creating GUIs and web content. Assistance includes speaking out the meta-information of each UI element, such as width, top edge, the distance between the right edge of an element and that of its relative parent, and number of siblings of an element. Additionally, using these plugins, developers can navigate by indentation of lines (e.g., jumping between lines of the same indentation, finding lines having more or less indentation relative to the current line). As well, one plugin is designed to report bugs. It offers shortcuts to inject special tags (e.g., `beginning`, and `ending`) in the screen reader's internal logs so that users can easily extract those log segments and share those with developers on GitHub (e.g., [61]) or with other experienced users on public forums (e.g., [66]). This is insightful because reporting a bug or issue is challenging for blind users without a programming background.

4.4.11 Finding Documentation and Shortcuts. Two plugins (N = 2) help users find appropriate documentation to interact with the system. One plugin provides dynamic tips on interacting with the focused UI element to promote usability. The verbosity of tips is adapted depending on the users' reported expertise level. Another plugin creates a dynamic help menu that contains documentation for each installed plugin. It also provides a list of keyboard shortcuts for all installed plugins in the system, thus reducing the burden of memorizing numerous shortcuts. Our user study revealed that finding proper documentation and memorizing new shortcuts are challenging for plugin users, which these plugins attempt to address.

4.4.12 Making Graphics Accessible via Computer Vision. Images are inaccessible to screen readers. However, two plugins (N = 2) provide a textual description of on-screen images by utilizing online computer vision services or a built-in OCR service. This theme also emerged in our user study. For instance, four participants reported using OCR plugins for scanned documents, and one (P6) mentioned that the ability of computer vision algorithms for non-Latin languages is still primitive.

4.4.13 Making Remote Access Accessible. We found that only one plugin (N = 1, NVDARemote) allows remote connection between two PCs. This is unsurprising because our prior work shows that remote access technologies (e.g., Microsoft Remote Desktop, Citrix, Virtual Machines) are inaccessible by design [17, 18]; these technologies transmit screen content as raw pixels from the guest (i.e., remote computer) to the host (i.e., local computer). Without associated metadata, the screen readers on the host computer cannot access the remote screen of the guest computer (it reads "blank, blank, blank"). Upon further inspection, we found that NVDARemote only transmits textual metadata from the guest to the host. Therefore, it is a workaround that overcomes the deficiencies in current remote access technologies.

4.5 Findings: How Are NVDA Plugins Developed (Community Perspective)?

We now present how NVDA's plugins were developed by the communities of blind users and developers based on our analyses of online data (Section 4.2). Our analyses revealed that plugins are the outcome of complex, tripartite, and lengthy collaborations between individual blind users, the online communities of other blind users, and developers.

4.5.1 Tripartite Collaborations to Create Plugins. Based on our interviews and web data analyses, we put together an interaction diagram in Figure 1 showing how three stakeholders— individual blind users, the community of blind users, and the community of developers—collaborate to create a plugin.

First, a blind user (i.e., the original poster) creates a post on their familiar online forum to seek help from others regarding accessibility or a usability issue with an application or a system. Forum members familiar with the issue respond by providing a solution or directing the poster to a plugin. If no such plugin exists, they recommend the individual create an *issue* on GitHub or reach out to the application vendor. If a GitHub issue is created, developers on GitHub discuss the issue at length and gather details about its reproducibility and severity. They also assess how many users

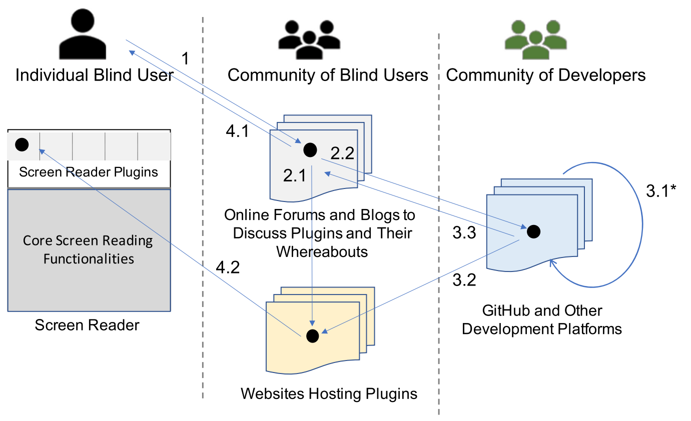

Fig. 1. The complex collaboration among individual blind users, their online communities, and developer communities. Each arrow indicates an interaction. **1:** A blind screen reader user usually creates a post on their familiar online forum to seek help with an application or a system. **2.1:** One or more forum members who are familiar with the issue direct the individual to a plugin. **2.2:** If no such plugin exists, they recommend the individual make a post (e.g., a bug report or issue) on GitHub or vendors' websites. **3.1:** Developers on GitHub discuss an issue (or the bug report) and assess its severity. **3.2:** If the problem is deemed important, they create a plugin or a patch, publish it to a plugin-hosting website(s). **3.3:** Subsequently, they inform the blind community. **4.1:** The original post creator is notified. **4.2:** The poster downloads and uses the plugin, thus completing the collaboration cycle.

can be affected by the issue. If the problem is deemed important, they create a plugin, publish it to a plugin-hosting website(s), and subsequently inform the blind community. The original poster is notified (e.g., receives a reply). They download the plugin from a hosting website or via a shared URL and thus complete the development cycle of a plugin. On average, it takes 3 to 4 months to complete a cycle.

In the following sections, we describe different collaborations in detail.

4.5.2 Collaboration Between an Individual User and the Community of Blind Users. An individual member first creates a post on the blind community when they need a plugin. We identified three different tones for these posts: simple requests, requests with personal stories, and requests with personal commitment.

Simple Requests. Most users politely request a plugin. They often include a use case to justify their need and a thank-you note. The following three quotations exemplify this pattern:

(1) "Is it possible to create an add-on [plugin] which locks the keystroke NVDA+F2 so that every key which will be pressed will be sent through to the application?"
(2) "Diagrams are not readable in PowerPoint. Is there a possibility to access diagrams like in MS Excel?"
(3) "I often wants to copy text from web browser, but not only simple text but as a formatted or preformatted HTML elements or HTML text. And with NVDA, I didn't find how to do it . . . Please can someone look at it and create some addon [plugin] for it? Many thankx."

Established members often respond to these requests by providing useful information or encouraging continued participation. For example, one member provides a solution to the third quote as follows: "You should be able to accomplish this by pressing NVDA+Space to enable focus mode, at least in web applications." Other members respond without providing a solution but to encourage participation. For example, a response to the second quote was as follows: "Have you already contacted the software maker about this? This is always going to be the first part of call to let them know they have a problem."

Requests with Personal Stories. Some users try to create excitement about lesser known apps. They include a motivating use case or personal story to persuade others in the community to create plugins for those apps. For example, a member who uses a dictionary app named *Babylon* informs the community why a plugin for Babylon is needed:

> [Are] any of you guys . . . using Babylon dictionary for translation? This is the most advanced and important dictionary I have ever used. But unfortunately, there are several parts that NVDA is not able to read on it. How about if someone would be able to create an NVDA add-on [plugin] at least to improve the accessibility on the program? . . . I hope that someone will be able to make the dream [happen], because this program is very important, especially in the professional work life and the education field [personal story].

We found that these posts receive mixed responses: some agree with the need, but others point out alternative solutions that are comparable and mainstream.

Requests with Personal Commitment. A few members offer personal commitment—either technical or financial—when requesting a plugin. For example, the following quote shows a member willing to collaborate with the developers to create a plugin:

> Hello All, I'm requesting for an NVDA add-on [plugin] for virtual deejay, a music mixing program for Windows. If not available, I wish to collaborate with a software developer who is willing to partner with me in coming up with such an add-on [plugin].

Sometimes, these members seek only technical assistance to develop a plugin by themselves. For example:

> Hi, I am developing an add-on for web browsing and I need to call some functions to jump to next page element like heading or link [.] without pressing a key on keyboard. [H]ow could I do this?

Usually, community members who are also developers respond to these questions promptly.

Further, we noticed that community members often raise money (ranging from $5 to $100) to fund plugin development (e.g., to pay developers). These fundraising activities complete within days, which indicates a strong social cause (i.e., plugins) and bond among blind users in the community.

We also noticed that some developers make announcements to share their own plugins with the community: "Today, I'm moving audio-themes v6.0 [a plugin] from alpha to beta. All of the issues reported by users have been fixed . . . Please find the add-on package at the following URL [removed]."

However, these announcements are vetted by the community. For example, we found that members warn others about developers who have a history of writing "bad" code, as well as developers who are reliable and trustworthy. For example, a member promotes a recent plugin written by his friend: "This is an add-on component written by my good friend." Another member vouches for "Goug," who "is a musical person and a programmer," who usually recommends reliable "scripts [plugins]," or "[makes] a script" if asked. These are consistent with our findings from the interview.

4.5.3 Collaboration Within the Developer Community. Members of the developer community collaboratively evaluate issues created by the users. They analyze an issue's written description and assess its originality, reproducibility, and problem category. They also analyze similar issues created by other users to estimate an issue's severity and how many blind users could be affected by it. Additionally, they ask other members to point out whether an alternative solution exists for the issue. If fellow developers decide to fix the issue, they identify the required resources and skills and assign responsibility to one or more members who have relevant skills.

We found that the developers' decision to fix an issue is based on several factors: problem description, reproducibility, recentness, and the number of affected users.

Issue Description. Developers are often critical of how an issue is described and decide not to address it. The following statements provide three examples:

(1) "This issue is still a bit vague in scope. I suggest closing it in favour of newer issues that have a clearer scope and goal."
(2) "This issue is getting rather confused. May be we should split it out to multiple ones."
(3) "Closing as will not fix [won't fix]. If there are solid ideas on specific proposals with clearly defined goals, user experience, and feasibility, we can consider these in separate issues."

Reproducibility of Issues. Developers also close an issue because they are unable to reproduce the problem—for example, " I am not able to replicate this." Developers frequently made comments similar to the following one: "[C]losing [a GitHub jargon to close a reported issue] as works for me . . . But we can reopen if this still persists for some users."

Obsolete Issues. Another reason the developers close an issue is that the problem is deemed obsolete or not mainstream (e.g., issues related to Windows XP, the Internet Explorer browser, and Flash technology). For example:

(1) "Closing as won't fix [a tag assigned to many issues as shown in Figure 2] . . . There is no point in us working on additional support for something which is now obsolete."
(2) "[W]hile we are not dropping support for Windows XP, it just is not feasible for us to put resources into fixing XP-specific issues any more. [C]losing as will not fix."
(3) "Is this still relevant since modern computers will not execute programs from USBs? U3 [software] has been obsolete for many years now. [I'd] like to vote for a will not fix."
(4) "IE9 [Internet Explorer browser] is more or less dead, and I thus suggest closing this ticket as a won't fix regardless."
(5) "[G]iven the continuing demise of [F]lash [I] do not think even investigating this is worthwhile. [C]losing."

Alternative Solution Exists. The developers also close an issue because an alternative solution exists for that issue. They also lower the priority of an outstanding issue as soon as a workaround is available. The following three quotes exemplify this scenario:

(1) "Since there does not seem to be interest in this and this ticket is very old, closing for now. There are quite good alternatives out there."
(2) "[H]ighlighting of the current object can be achieved with the 'focus highlight' addon [plugin]. This will probably eventually be integrated into NVDA itself . . . I'm closing this since I believe everything is either already implemented or covered elsewhere."
(3) "I would say P3 [low priority] because there is a workaround."

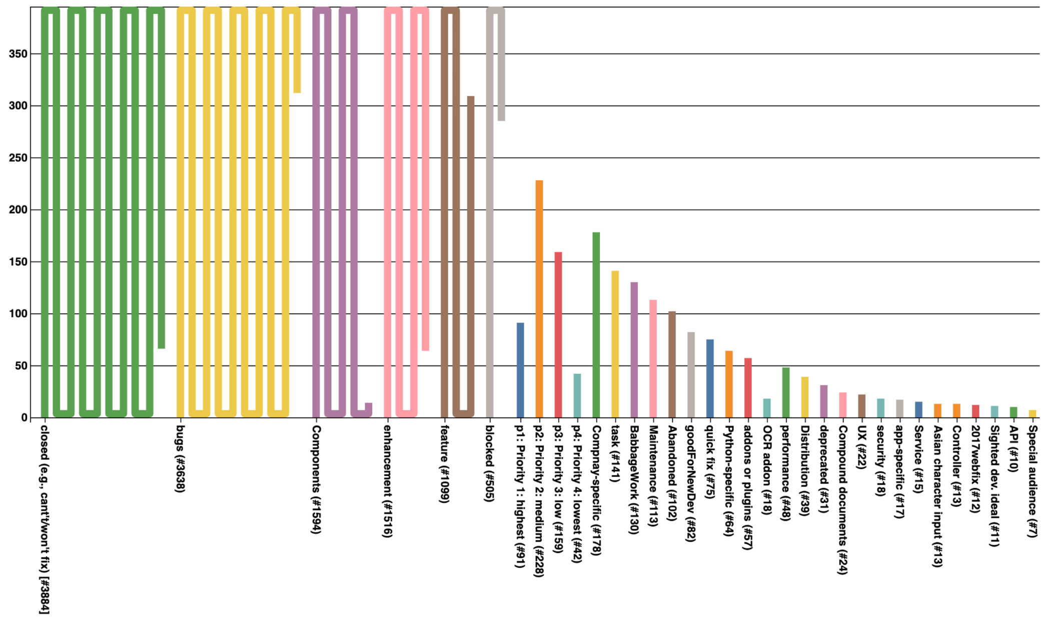

Fig. 2. A wrapped bar chart [47] showing the top 35 categories of *closed* issues on NVDA's GitHub repository (as of April 29, 2021). The number of closed issues in each category is also presented in parentheses of the label (e.g., the *bugs* category has 3,638 closed issues). The wrapped bars enable better large-to-small bar comparison by wrapping large bars over a certain threshold (e.g., 400).

Difficult to Solve Issues. Finally, developers choose not to fix a problem because it is too difficult (technically) or they do not have the required skills:

> [A member] provides a series of seemingly compelling arguments about why this issue is extremely difficult to resolve why it might be controversial to implement in the first place . . . I do not believe too many NVDA users desire to work with in the first place, which requires significant code rewrites . . . and pose several other [UX technical] challenges . . . won't fix [a tag].

Sometimes developers disclose their disabilities and technical skills and seek external help, especially from sighted individuals. For example:

> (1) "This is still an issue and . . . I think I do have the skills to fix this."
> (2) "I can not get sighted help this week. But will do as soon as I can and report back. Somebody else from our [A]rabic speaking followers may do so if possible."

This is interesting because prior work has shown that self-disclosures and making personal references to other members constitute specific rhetorical and discursive practices that build legitimacy and authority [38].

Because of the preceding factors, more than half of the issues created by the user community are immediately closed (without fixing), as shown by the first wrapped bar, tagged with *closed (e.g., can't/won't fix)*, in Figure 2.

4.5.4 *Collaboration Between Blind Users and Developer Communities.*

The Burden of Proof. We found that reporting an issue or requesting a plugin poses a huge burden on blind users because there is no well-established policy or guideline on reporting an issue,

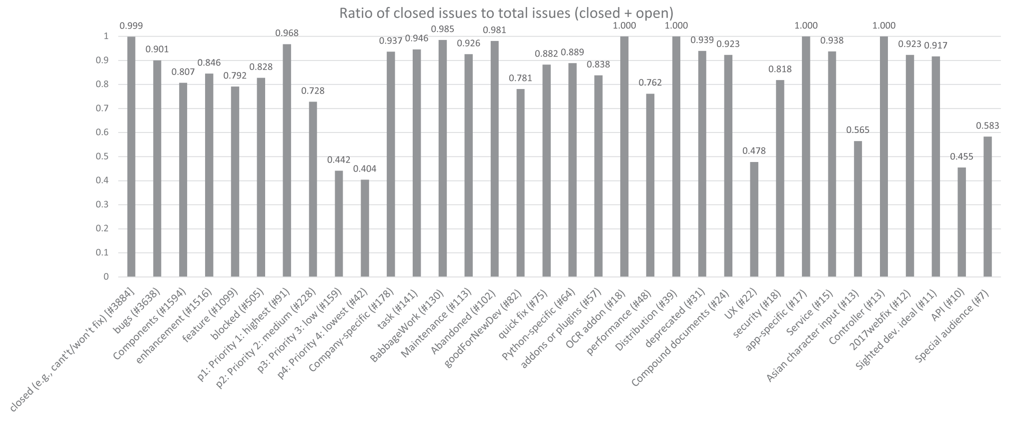

Fig. 3. A bar chart showing the ratios of the number of *closed* issues to the total number of issues of the same top-35 Issue categories on NVDA's GitHub repository (as of April 29, 2021). Note that certain issue categories are more likely to get fixed (i.e., closed) than others (e.g., 90.1% of issues under the *bugs* category are closed, whereas 83.8% of issues under the *addons or plugins* category are closed).

or otherwise sometimes these guidelines[5] are written from the developers' perspective, which is difficult to follow for end users. Therefore, users often face challenges in understanding what information to include and to what extent for the developers to fix the issue. In this regard, we observe that members of the blind user community share resources and tools to collect meta-information, logs, and debug traces for end users so that the developers can consult these materials to diagnose the issue. Users are encouraged to include such metadata when reporting an issue on the developer forum to increase the likelihood of getting it fixed.

The members of the blind community also push the developer community if they (developers) close an important issue without fixing it—they often provide cogent justification and create polls to protest developers' decisions.

Drawing Developers' Attention for Priority. We found tension between the communities of blind users and developers in assigning priority to a reported issue. Members in the user community comment under an issue requesting the developers to assign it a priority (e.g., p1 (highest) to p4 (lowest)) to increase the likelihood of getting it fixed. For example, Figure 3 shows that 96.8% of issues with *p1* priority are resolved (hence closed), compared to only 40.4% of those tagged with *p4* priority. Therefore, it was unsurprising that many comments are priority seeking, similar to this one: "[W]ould you be able to give this a priority?"

The users also attempt to draw developers' attention by framing their issues as "bugs," as shown in the following comment: "It is still occurring . . . Do you not consider it to be a bug?" This is because the developers typically pay more attention to bugs (e.g., 90.1% of issues tagged with *bugs* are fixed in Figure 3).

However, the developers often push back and explain why they cannot assign an issue a priority. For example, one developer replied to a comment on an issue that was claimed to be a bug: "[A]nnoying, but does not prevent work in any way as far as I understand . . . So, I guess we will take a fix, but it is a very low priority for us relative to other work." Another comment shows developers' rationale for not assigning the issue a high priority, "Miranda NG [a chat software] is not a core Windows app. So P2 [higher priority] seems overly eager."

[5]https://github.com/nvaccess/nvda/blob/master/.github/ISSUE_TEMPLATE/bug_report.md.

Despite being contentious at times, we found that the collaborative relationship between the users and developers is largely harmonious.

4.5.5 Leadership in Communities of Blind Users and Developers. We noticed that knowledge collaboration is a key activity in both users and developer communities, and expertise sharing is the primary activity in the developer community. Some participants are recognized for their contributions and emerge as leaders. For example, when members of the developer community debate on an issue, a special member often breaks ties and acts as an arbitrator. Statements from this member often start with an authoritarian tone: "I'm making an executive decision."

The emergence of this type of leadership in online communities where members share knowledge or contribute free software is well documented in the literature [74]. Researchers have identified several task-based behaviors, such as technical contributions and technical communications, as antecedents of lateral authority—that is, a peer-recognized form of leadership [26].

We also found that these leaders are not necessarily the ones who make the most posts or are most sociable; they are the ones who contribute the most to the community's central task. Other work on online knowledge collaboration reported a similar finding [30]. This is interesting because sociability, defined by Simmel [73], as a form of associating with others without ulterior motives or content focus, has long been recognized as a necessary aspect of interpersonal engagement.

5 DESIGN IMPLICATIONS AND DISCUSSION

Our study revealed that plugins could effectively address application-specific accessibility and usability issues. However, only a handful of people develop plugins, and they are well recognized in the blind community. We now report key gaps in current practices and make recommendations to address them in light of our findings. We also outline new directions for future HCI research.

5.1 Raising Awareness About Screen Reader Plugins and Diagnostic Tools

In our interview study, expert participants were 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 that they were already using some of these plugins. We believe this is a general case for most blind users who are not tech-savvy or early in their learning process of screen readers. Some participants who were aware of plugins reported installing malware instead; one reported getting scammed by someone who promised to help him with a plugin. As such, we realize a big value in educating blind users about plugins, their benefits, and potential dangers (e.g., security risks of malware and how to flag suspicious plugins).

We also realize that online forums are of great help for blind users to get support. However, these forums are loosely connected—some blind individuals are members of multiple forums, but most belong to only one forum. Therefore, sharing knowledge across multiple forums is key to keeping blind users informed.

In addition, most blind users are unaware of the dynamics between users' and developers' communities. For example, a problem critical to an individual blind user may be deemed unimportant to developers. Sometimes, an individual does not provide sufficient metadata (e.g., crash reports and logs) to developers to reproduce a problem. Although we found several plugins to capture such metadata, no participants in our interview study (including the expert ones) reported using them. This indicates that there is room for increasing awareness of diagnostic tools and how to report a bug among general blind users.

5.2 Building a Community-Driven Plugin Repository Hosted on a Peer-to-Peer System

We found a pressing need for a community repository to store, distribute, and update plugins for all screen readers. Our data indicate that the community repository of NVDA greatly serves its users, despite having a limited number of plugins and lacking a rating system. Thus, we recommend building one community repository for all available plugins, categorized by screen readers and their versions. In this repository, users can rate a plugin based on its quality, merits, and usefulness. Additionally, we recommend that each plugin have a clear and informative description, possible security issues, known bugs, documentation on shortcuts, and a simple tutorial showcasing its functionalities. We can draw further inspiration from the success of App Store[6] for iOS apps. Like these apps, plugins should receive automatic updates without causing compatibility issues.

However, hosting this community repository can be challenging due to financial costs and the need for technical support. Prior efforts, such as the Accessmonkey framework for collaborative web accessibility improvements [13], did not sustain. Therefore, a promising alternative is to host this repository on a peer-to-peer system (e.g., Blockchain [29, 34]), where individual blind users contribute the storage and computational power of their computers, and the entire community benefits from it. We will investigate this direction in the future.

5.3 Engaging Third-Party Developers and Websites

Since the number of third-party plugin developers is limited and most developers maintain their own websites, blind users often struggle to find specific plugins online. To alleviate this problem, we recommend that the plugin store contains the necessary information (e.g., contact info) associated with the known developers. In addition, it should contain information (e.g., URL) about third-party websites, forums, and GitHub repositories about plugins. Like plugins, developers and websites should have their own ratings. Since some participants are willing to purchase a plugin, a trusted, easy-to-navigate, and accessible transaction system should also be in place so that users can send money to the developers.

5.4 Encouraging Individual Plugin Development

Almost all of our participants were interested in developing their own plugins. However, they reported several issues preventing them from trying to do so. The most common issues they reported were the steep learning curve and the extensive plugin developer's guide (e.g., going through 20- to 30-page documentation). Further, our data analysis showed that high-priority (p1) issues on GitHub are fixed quickly compared to low-priority (p4) ones. However, the priority assignment depends on the judgment of the developers, who often decline users' requests to assign an issue higher priority.

As such, we see a big opportunity to develop an intelligent, easy-to-use tool to generate plugins. This tool can ask users step-by-step questions to learn the expected behavior of an application or system (i.e., the ground truth) and analyze the screen readers' logs to generate a patch. Such a tool will not only empower blind users but also reduce the workload of a handful of plugin developers.

5.5 Plugin-Based Content Delivery

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. Therefore, plugins are here to stay.

However, we note that an unwanted side effect of plugins is that they introduce new shortcuts, which blind users need to memorize, thus posing a learning challenge [17]. Therefore, HCI researchers can investigate how to reduce this learning effort.

Since installing a plugin is easy for blind users, we consider a plugin-like delivery mechanism to be an effective way to combat accessibility issues. Further, HCI researchers who build prototypes

[6]https://www.apple.com/app-store/.

to improve accessibility should consider releasing their prototypes as plugins to quickly reach out to blind users and make a more significant impact.

5.6 Limitations

This work has several limitations. Most participants in the study were experts due to our inclusion criteria (e.g., familiarity with screen readers' plugins). Without this criterion, we were afraid we could not have gained a deeper insight into the ecosystem of plugins. As such, our current findings lacked input from those unfamiliar with screen readers' plugins. However, we believe that the impact of not having their input is small because the plugin literacy among general screen reader users is low. For example, our study revealed that non-expert participants who were aware of plugins were unaware of many aspects (e.g., some plugins are shipped with screen readers). Another limitation is that the data analysis is based on specialized communities of NVDA screen readers. However, we believe our findings are generalizable for other screen readers. Finally, we were unable to code all scraped (online) data due to its sheer volume. As such, we may have overlooked additional insights.

6 CONCLUSION

In this work, we aimed to understand why blind users use screen readers' plugins and how these plugins are created, deployed, and maintained. To that end, we conducted an interview study with 14 blind participants and analyzed plugin-related data scraped from online communities of blind users and developers. Our interview study reveals that plugins can enhance the capabilities of screen readers in making applications more accessible and usable. Although finding the right plugin is challenging, the participants reported that installing a plugin is easy. In addition, all participants stated that plugins could be valuable parts of screen readers when appropriately developed. The study also revealed that once deployed, plugins are hardly maintained, and many become obsolete after some time. Furthermore, plugins have issues with availability, compatibility, unstructured distribution, poor documentation, and security risks. Finally, we found tensions between the plugin users and developers.

Our online data analyses indicated that plugins result from a complex, tripartite collaboration among individual blind users, their online community, and developer communities. The members of the blind user community are supportive of each other. For example, if a user mentions an accessibility or usability issue, the community helps the user by pointing out alternative approaches or solutions. Sometimes, they also share their experience that is similar or relatable to their current problem. Furthermore, the collaboration among the community of blind users plays a critical role in creating plugins—from advising blind users how to submit a plugin request to raising funds to maintaining pressure on the developer community to create a plugin. Moreover, implicit leadership emerges in both user and developer communities. Finally, the developer community has the upper hand in deciding which plugins to develop for the blind community.

To make screen reader plugins more effective, we recommend creating a community repository of plugins hosted on a peer-to-peer infrastructure, a rating system, and a financial model. Furthermore, we envision a system that can provide an easy interface to create plugins by the end users. Making a plugin-based content distribution system to reach blind users is also left for future research. We believe our findings will inspire accessibility researchers and practitioners to investigate potential ways to improve the collaboration between blind users and developer communities, as well as within the blind communities.

A APPENDIX

A.1 A List of Major NVDA Plugins

Table A.1 presents a sampler of major NVDA plugins (N = 77) we analyzed in the article.

Table A.1. Subset of Major NVDA plugins (N = 77) and Their Primary Function (Extracted from NVDA Add-ons Repository, https://addons.nvda-project.org/index.en.html, in July 2021)

Plugin	**Description**
Kill NVDA	Temporarily kills NVDA and restarts if NVDA freezes
Time Zoner	Allows users to configure NVDA to announce the time and data from a list of selected time zones
Phonetic Punctuation	Converts punctuation signs and regular expressions into audio icons
Tony's enhancements	A number of small improvements to NVDA, including enhanced table navigation, dynamic keystroke assignment, and real-time console output
Report Passwords	Announces the text typed in protected controls like passwords instead of asterisks
Synth ring settings selector	Allows users to select which settings should appear on the synth settings ring in NVDA
Debug Helper	Allows users to insert markers (flags) in the NVDA log file using a shortcut; later, these markers can be used to trace relevant logs from the log file for reporting or debugging
Notepad++	Improves the accessibility of Notepad++ by supporting features like setting bookmarks in text, announcing maximum line length, and making the default autocomplete feature accessible
Beep Keyboard	Allows users to configure NVDA to beep on specified keyboard events (e.g., beep for uppercase when `Caps Lock` is on)
Developer Toolkit	Allows blind and visually impaired developers to create GUIs and web contents; enables users to navigate the objects in an interface with keyboards and obtain visual properties like size and position
Add-ons documentation	Adds a menu in NVDA to quickly access the documentation of all the installed plugins (add-ons)
Character Information	Presents the character information in a message such as Unicode name, number, and category
Add-on to count elements of selected text	Announces the number of lines, words, characters, and paragraphs of a selected text using a shortcut
Image Describer	Gives descriptions of images using machine learning methods
Weather Plus	Allows users to access temperature and weather forecast for up to 2 additional days
BrowserNav	Enables quick navigation of browser contents utilizing information such as font size, font style, and vertical alignment
Outlook extended	Improves the accessibility of Microsoft Outlook by adding shortcuts, including announcing the attachment numbers, navigating to the message body, and marking messages as read or unread
AudioChart	Represents time series data in Microsoft Excel as a continuous sound
BluetoothAudio	Prevents Bluetooth audio devices from entering into standby mode by constantly playing an inaudible sound
TextNav	Allows users to find textual contents (paragraphs) in a webpage skipping over menus, advertisements, and other non-textual contents
Calibre	Improves the accessibility of Calibre e-book manager by adding features such as announcing the total number of books and providing meta-information about each book
ToolbarsExplorer	Enables easier navigation and interaction of the items in toolbars with standard keyboard keys (e.g., directional arrow keys) and shortcuts
Input Lock	Allows users to lock their keyboard to prevent accidental key presses
Add-on Updater	Allows users to check whether the installed plugins in NVDA can be updated

(Continued)

Table A.1. Continued

Plugin	Description
Access8Math	Enables users to read, navigate, and write mathematical notations written in MathML
Text Information	Provides information regarding the selected text or the text copied on the clipboard
SentenceNav	Enables users to read and navigate textual contents by sentences
BrailleExtender	Provides various features at the braille level such as reloading braille displays with shortcuts, switching between several input and output braille tables, and using two output braille tables simultaneously
IndentNav	Allows users to navigate textual contents by indentation levels or line offsets
Mozilla Apps Enhancements	Improves the Mozilla browser by supporting features like reading out the contents from the address bar, status bar, and notification bar with shortcuts
sayCurrentKeyboardLanguage	Provides keyboard shortcuts to announce the language of the current keyboard and the default keyboard of the system
Object Location Tones	Indicates the location of the content in focus with a tone during navigation
Enhanced Aria	Allows users to configure the ARIA landmarks to be announced by NVDA while browsing web contents
Lambda Add-on for NVDA	Adds speech and braille support for the LAMBDA (Linear Access to Mathematics for Braille Device and Audio-synthesis) software
Speech History	Allows users to review the most recent 100 items spoken by NVDA and copy the selected items to the clipboard
Speak Passwords	Announces characters instead of asterisks while typing in a password field
Clipspeak	Automatically announces clipboard operations such as cut, copy, and paste
Review Cursor Copier	Provides multiple shortcuts to copy the text under the cursor in different ways (e.g., either the complete line or the word at the cursor location)
Classic Selection	Modifies the behaviors of the default NVDA shortcuts for selecting and copying texts to clipboard
ObjPad	Provides four different ways to use the arrow keys for navigating on-screen or web contents
Switch synth	Enables users to switch between six synthesizers using shortcuts quickly
Report Symbols	Configures NVDA to announce the typed symbols, including non-alphanumeric and blank characters
StationPlaylist	Improves the accessibility of StationPlaylist (an audio streaming software) software by adding several features such as announcing the elapsed/remaining time of a track and moving to previous/next track
Tone Master	Allows users to create music by specifying a sequence of tones with three numeric properties for each tone—pitch, duration, and the gap between two tones
mp3DirectCut	Improves the accessibility of the mp3DirectCut software (an audio editor) by introducing features such as announcing the elapsed/remaining time and placing the cursor at the beginning/end
Crash Hero	Provides information for debugging when NVDA crashes
Tip of the Day	Helps users learn NVDA by providing tips every day
Golden Cursor	Allows users to move the mouse cursor with keyboards and save application-specific mouse positions
Day of the week	Allows users to search dates and find the corresponding day of the week
VLC Media Player	Makes the playback controls of VLC Media Player interactive with keyboard shortcuts and announces elapsed time when forwarding the media
Windows App Essentials	Enhances the accessibility of different Windows modules and controls such as Cortana, Map, and Settings
NVDA Remote Support	Allows users to control a remote computer running NVDA from a local computer
TeamTalk Classic	Improves the accessibility of the TeamTalk Classic software by supporting features like providing information regarding the toolbar items, announcing the state of voice activation, and silencing the progress bar indicating real-time microphone volume
Easy Table Navigator	Enables navigating table cells with directional arrows

(Continued)

Table A.1. Continued

Plugin	Description
Clock and calendar Add-on for NVDA	Allows users to get time and date in specified format and control alarms and stopwatch on Windows
Newfon	A voice synthesizer for Russian, Ukrainian, and three other languages that supports changing language and speech rate
Dual Voice	Configures NVDA to support two separate voice synthesizers—one for Latin and the other for non-Latin scripts
Clip Contents Designer	Supports advanced clipboard operations such as appending text into the clipboard, clearing clipboard contents, and converting contents to HTML
Enhanced Touch Gestures	Supports three-finger and four-finger touch gestures for easier navigation
NV Speech Player	An open source speech synthesizer for NVDA that is responsive and capable of smooth and predictable pronunciation
PC Keyboard Braille Input for NVDA	Allows users to input braille characters with multiple computer keyboard layouts (e.g., English, French, and German)
Bit Che	Improves the Bit Che software by making the search results and the context menu accessible
Read Feeds	Provides easier access to ATOM and RSS feeds such as announcing the title and URL of the current feed, announcing the next/previous feed, and refreshing the feeds
GoldWave	Improves the accessibility of the GoldWave audio editor by announcing the current track position, editing channel, and the total length of the audio file
Emoticons	Provides a human-friendly description of emoticons in text
eMule	Improves the accessibility of eMule file-sharing software by making the contents of the window focusable and navigable with shortcuts
Focus Highlight	Highlights the currently focused item with a colored rectangle
NoBeepsSpeechMode	Excludes beep in the speech mode with a shortcut
Virtual Review	Allows users to review the contents of a window in a text box
placeMarkers	Enables saving and searching specific text strings or bookmarks on webpages or documents in NVDA's browse mode
Control Usage Assistant	Provides users with a short help message regarding how to interact with the currently focused control such as checkboxes and text edit fields
Resource Monitor	Adds new shortcuts to obtain resource information such as CPU load, memory usage, and battery percentage
UnicodeBrailleInput	Allows users to convert text from braille to Unicode braille characters according to an input braille table
OCR	Performs OCR to extract text from an inaccessible object using a shortcut
Extended Winamp	Improves the accessibility of Winamp software by adding features such as setting playback volume and announcing the elapsed time, remaining time, and track length
dropbox	Adds shortcuts to announce Dropbox status and open the Dropbox systray menu
systrayList	Enables easier access and interaction to the items of the systray or the taskbar presenting the items within a list box

A.2 A List of Plugin Groups

Table A.2 presents 13 plugin groups that emerged from our affinity diagramming (updated July 2021), with the group sized indicated by *N*. Each group shows its constituent plugins. A single plugin can appear in multiple groups due to its overlapping functionalities.

Table A.2. Plugin Groups and Their Individual Plugins

Plugin Group	**Individual Plugins**
Enhancing Usability (*N* = 24)	Kill NVDA, Time Zoner, Tony's enhancements, Synth ring settings selector, Beep Keyboard, Add-on to count elements of selected text, Weather Plus, Input Lock, sayCurrentKeyboardLanguage, Speech History, Clipspeak, Crash Hero, Tip of the Day, Day of the week, Clock and calendar Add-on for NVDA, Dual Voice, Clip Contents Designer, Focus Highlight, NoBeepsSpeechMode, Virtual Review, placeMarkers, Control Usage Assistants, Resource Monitor, systrayList
Patching Application-Specific Inaccessibility (*N* = 17)	Notepad++, Outlook extended, calibre, Access8Math, Mozilla Apps Enhancements, StationPlaylist, mp3DirectCut, VLC Media Player, Windows 10 App Essentials, TeamTalk Classic, Bit Che, Reed Feeds, GoldWave, eMule, Resource Monitor, extendedWinamp, dropbox
Enhancing Text Accessibility (*N* = 13)	Phonetic Punctuation, Tony's enhancements, Beep Keyboard, Character Information, Add-on to count elements of selected text, AudioChart, TextNav, Text Information, SentenceNav, IndentNav, Report Symbols, Emoticons, UnicodeBrailleInput
Multiplexing Audio Synthesizers (*N* = 11)	Phonetic Punctuations, Synth ring settings selector, Object Location Tones, Switch synth, Tone Master, Newfon, Dual Voice, NV Speech Player, GoldWave, NoBeepsSpeechMode
Masking Input-Output Devices (*N* = 10)	BluetoothAudio, Input Lock, BrailleExtender, sayCurrentKeyboardLanguage, Lambda Add-on for NVDA, Review Cursor Copier, Golden Cursor, Enhanced Touch Gestures, PC Keyboard Braille Input for NVDA, UnicodeBrailleInput
Improving Accessibility of Specific UI Elements (*N* = 6)	Tony's enhancements, Report Password, ToolbarsExplorer, Speak Password, Easy Table Navigator, Control Usage Assistants
Simplifying Content Navigation on Webpages (*N* = 6)	BrowserNav, TextNav, Enhanced Aria, ObjPad, Easy Table Navigator, Focus Highlight
Ensuring Backward Compatibility and Easing Software Updates (*N* = 4)	Add-ons documentation, Add-on Updater, Clipspeak, Classic Selection
Augmenting Clipboard (*N* = 3)	Clipspeak, Classic Selection, Clip Contents Designer
Assisting Developers (*N* = 3)	Debug Helper, Developer Toolkit, Crash Hero
Finding Documentation and Shortcuts (*N* = 2)	Add-ons documentation, Control Usage Assistant
Making Graphics Accessible via Computer Vision (*N* = 2)	Image Describer, OCR
Making Remote Access Accessible (*N* = 1)	NVDA Remote Support

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful feedback.

REFERENCES

[1] Beautiful Soup. n.d. Beautiful Soup 4.4.0 Documentation. Retrieved February 23, 2023, from https://beautiful-soup-4.readthedocs.io/en/latest/.

[2] Freedom Scientific. 2018. What's New in JAWS 2018 Screen Reading Software. Retrieved September 20, 2018 from https://www.freedomscientific.com/downloads/JAWS/JAWSWhatsNew.
[3] NV Access. 2020. NV Access. Retrieved September 20, 2018 from https://www.nvaccess.org/.
[4] Freedom Scientific. 2021. JAWS Script Developer Guide: Introduction. Retrieved February 11, 2022 from https://support.freedomscientific.com/Content/Documents/Other/ScriptManua l/01-0_Introduction.htm.
[5] GitHub. 2021. NVDA Add-on Developer Guide. Retrieved February 11, 2022 from https://github.com/nvdaaddons/devguide/wiki/NVDA%20Add-on%20Development%20Guide.
[6] Mark S. Ackerman, Juri Dachtera, Volkmar Pipek, and Volker Wulf. 2013. Sharing knowledge and expertise: The CSCW view of knowledge management. *Computer Supported Cooperative Work* 22, 4-6 (2013), 531–573.
[7] Addons.mozilla.org (AMO). 2020. Add-ons for Firefox. Retrieved June 29, 2020 from https://addons.mozilla.org/en-US/firefox/.
[8] Apple. 2011. NSAccessibility. Retrieved June 29, 2020 from https://developer.apple.com/documentation/appkit/accessibility_for_macos/nsaccessibility.
[9] Apple Inc. 2020. VoiceOver. Retrieved February 11, 2022 from https://www.apple.com/accessibility/osx/voiceover/.
[10] 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 (CHI'15)*. ACM, New York, NY, 3043–3052. https://doi.org/10.1145/2702123.2702589
[11] Eric Bergman and Earl Johnson. 1995. Towards accessible human-computer interaction. *Advances in Human-Computer Interaction* 5, 1 (1995), 87–114.
[12] Hossein Bidgoli. 2007. *The Handbook of Computer Networks, Distributed Networks, Network Planning, Control, Management, and New Trends and Applications*. Vol. 3. Wiley.
[13] Jeffrey P. Bigham. 2007. Accessmonkey: Enabling and sharing end user accessibility improvements. *ACM SIGACCESS Accessibility and Computing* 89 (Sept. 2007), 3–6. https://doi.org/10.1145/1328567.1328568
[14] 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'10)*. ACM, New York, NY, Article 15, 4 pages. https://doi.org/10.1145/1805986.1806007
[15] 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'07)*. 25–34.
[16] 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 (ASSETS'17)*. ACM, New York, NY, 110–119. https://doi.org/10.1145/3132525.3132531
[17] 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, New York, NY, 5862–5868. https://doi.org/10.1145/3025453.3025731
[18] Syed Masum Billah, Donald E. Porter, and I. V. Ramakrishnan. 2016. Sinter: Low-bandwidth remote access for the visually-impaired. In *Proceedings of the 11th European Conference on Computer Systems*. ACM, New York, NY, Article 27, 16 pages. https://doi.org/10.1145/2901318.2901335
[19] Dorian Birsan. 2005. On plug-ins and extensible architectures. *Queue* 3, 2 (2005), 40–46.
[20] 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 *Proceedings of the International World Wide Web Conference (WWW'10)*. 1233–1236.
[21] Stephen A. Brewster, Peter C. Wright, and Alistair D. N. Edwards. 1993. An evaluation of earcons for use in auditory human-computer interfaces. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. ACM, New York, NY, 222–227. https://doi.org/10.1145/169059.169179
[22] Jed R. Brubaker, Lynn S. Dombrowski, Anita M. Gilbert, Nafiri Kusumakaulika, and Gillian R. Hayes. 2014. Stewarding a legacy: Responsibilities and relationships in the management of post-mortem data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 4157–4166.
[23] A. Bryman and R. G. Burgess. 1994. *Analyzing Qualitative Data*. Routledge. https://books.google.com/books?id=KQkotSd9YWkC.
[24] Brian Butler, Lee Sproull, Sara Kiesler, and Robert Kraut. 2002. Community effort in online groups: Who does the work and why. *Leadership at a Distance: Research in Technologically Supported Work* 1 (2002), 171–194.
[25] Vinton G. Cerf. 2012. Why is accessibility so hard? *Communications of the ACM* 55, 11 (2012), 7.
[26] Linus Dahlander and Siobhan O'Mahony. 2011. Progressing to the center: Coordinating project work. *Organization Science* 22, 4 (2011), 961–979.
[27] GitHub. n.d. UI Automation Can Cause Unbounded Memory Consumption. Retrieved November 3, 2020 from https://github.com/dotnet/winforms/issues/3182.

[28] 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 (UIST'11)*. ACM, New York, NY, 225–234. https://doi.org/10.1145/2047196.2047226

[29] Chris Elsden, Arthi Manohar, Jo Briggs, Mike Harding, Chris Speed, and John Vines. 2018. Making sense of blockchain applications: A typology for HCI. In *Proceedings of the 2018 Chi Conference on Human Factors in Computing Systems*. 1–14.

[30] S. Faraj, S. Jarvenpaa, and A. Majchrzak. 2011. Knowledge collaboration in online communities. *Organization Science* 22 (2011), 1224–1239.

[31] Samer Faraj, Srinivas Kudaravalli, and Molly Wasko. 2015. Leading collaboration in online communities. *MIS Quarterly* 39, 2 (2015), 393–412.

[32] 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 (ASSETS'07)*. ACM, New York, NY, 67–74. https://doi.org/10.1145/1296843.1296857

[33] Danyel Fisher, Marc Smith, and Howard T. Welser. 2006. You are who you talk to: Detecting roles in Usenet newsgroups. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*. IEEE, Los Alamitos, CA, 59b.

[34] Michael Fröhlich, Franz Waltenberger, Ludwig Trotter, Florian Alt, and Albrecht Schmidt. 2022. Blockchain and cryptocurrency in human computer interaction: A systematic literature review and research agenda. *arXiv preprint arXiv:2204.10857* (2022).

[35] Janet Fulk and Gerardine DeSanctis. 1995. Electronic communication and changing organizational forms. *Organization Science* 6, 4 (1995), 337–349.

[36] 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 and Accessibility (ASSETS'14)*. ACM, New York, NY, 67–74. https://doi.org/10.1145/2661334.2661363

[37] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI'04)*. ACM, New York, NY, 93–100. https://doi.org/10.1145/964442.964461

[38] Jolene Galegher, Lee Sproull, and Sara Kiesler. 1998. Legitimacy, authority, and community in electronic support groups. *Written Communication* 15, 4 (1998), 493–530.

[39] D. W. Gotterbarn, Bo Brinkman, Catherine Flick, Michael S. Kirkpatrick, Keith Miller, Kate Vazansky, and Marty J. Wolf. 2018. *ACM Code of Ethics and Professional Conduct*. ACM, New York, NY. https.www.acm.org/binaries/content/assets/about/acm-code-of-ethics-booklet.pdf.

[40] Shuai Hao, Bin Liu, Suman Nath, William G. J. 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.

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

[42] 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 (EICS'13)*. ACM, New York, NY, 33–42. https://doi.org/10.1145/2494603.2480316

[43] Helena Holmström and Ola Henfridsson. 2006. Improving packaged software through online community knowledge. *Scandinavian Journal of Information Systems* 18, 1 (2006), 2.

[44] K. Holtzblatt and H. Beyer. 1997. *Contextual Design: Defining Customer-Centered Systems*. Elsevier Science. https://books.google.com/books?id=JxQaQgOONGIC.

[45] Michael Cooper and Joseph Scheuhammer. n.d. WAI-ARIA 1.0 Authoring Practices. Retrieved February 11, 2022 from http://www.w3.org/TR/wai-aria-practices/.

[46] Gerald C. Kane, Jeremiah Johnson, and Ann Majchrzak. 2014. Emergent life cycle: The tension between knowledge change and knowledge retention in open online coproduction communities. *Management Science* 60, 12 (2014), 3026–3048.

[47] Alireza Karduni, Ryan Wesslen, Isaac Cho, and Wenwen Dou. 2020. Du Bois wrapped bar chart: Visualizing categorical data with disproportionate values. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI'20)*. ACM, New York, NY, 1–12. https://doi.org/10.1145/3313831.3376365

[48] Klaus Krippendorff. 2018. *Content Analysis: An Introduction to Its Methodology*. SAGE.

[49] George Kuk. 2006. Strategic interaction and knowledge sharing in the KDE developer mailing list. *Management Science* 52, 7 (2006), 1031–1042.

[50] 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, New York, NY, 1719–1728.

[51] 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 (CHI'07)*. ACM, New York, NY, 943–946. https://doi.org/10.1145/1240624.1240767

[52] Ann Majchrzak, Christian Wagner, and Dave Yates. 2013. The impact of shaping on knowledge reuse for organizational improvement with wikis. *MIS Quarterly* (2013), 455–469.

[53] Microsoft. n.d. Microsoft Active Accessibility: Architecture. Retrieved March 15, 2015 from https://msdn.microsoft.com/en-us/library/windows/desktop/dd373592(v=vs.85).aspx.

[54] Microsoft. 2020. Business Apps: Microsoft AppSource. Retrieved June 29, 2020 from https://appsource.microsoft.com/en-us/marketplace/apps?src=office&product=office.

[55] Microsoft. 2020. UI Automation Overview. Retrieved February 11, 2022 from http://msdn.microsoft.com/en-us/library/ms747327.aspx.

[56] Farhani Momotaz, Md. Touhidul Islam, Md. Ehtesham-Ul-Haque, and Syed Masum Billah. 2021. Understanding screen readers' plugins. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS'21)*. ACM, New York, NY, 1–10. https://doi.org/10.1145/3441852.3471205

[57] J. Y. Moon and L. Sproull. 2000. Essence of distributed work: The case of the Linux kernel. *First Monday* 5, 11 (Nov. 2000). DOI: https://doi.org/10.5210/fm.v5i11.801

[58] Helen Nissenbaum. 2009. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, Stanford, CA.

[59] GitHub. n.d. Issues: nvaccess/nvda. Retrieved February 11, 2022 from https://github.com/nvaccess/nvda/issues?q=label%3Abug%2Fcrash.

[60] GitHub. 2020. nvda/source/appModules at Master: nvaccess/nvda. Retrieved June 29, 2020 from https://github.com/nvaccess/nvda/tree/master/source/appModules.

[61] GitHub. 2020. Issues: nvaccess/nvda. Want to Contribute to nvaccess/nvda? Retrieved June 29, 2020 from https://github.com/nvaccess/nvda/issues.

[62] GitHub. n.d. NVDA Causing CPU Usage Spikes When Some Particular Applications Are Running. https://github.com/nvaccess/nvda/issues/1228.

[63] NVDA Groups. 2021. NVDA Add-ons Central. Retrieved January 20, 2021 from https://nvda-addons.groups.io/g/nvda-addons.

[64] GitHub. 2020. nvaccess/nvda: NVDA, the Free and Open Source Screen Reader for Microsoft Windows. Retrieved June 29, 2020 from https://github.com/nvaccess/nvda.

[65] NVDA. 2020. Welcome to the NVDA Community Add-ons Website. Retrieved June 29, 2020 from https://addons.nvda-project.org/index.en.html.

[66] NV Access. 2020. nvdanvda.groups.io: Home. Retrieved July 19, 2020 from https://nvda.groups.io/g/nvda.

[67] Volkmar Pipek, Volker Wulf, and Aditya Johri. 2012. Bridging artifacts and actors: Expertise sharing in organizational ecosystems. *Computer Supported Cooperative Work* 21, 2-3 (2012), 261–282.

[68] Jenny Preece. 2000. *Online Communities: Designing Usability and Supporting Sociability*. John Wiley & Sons.

[69] Robert Reimann, Alan Cooper, David Cronin, and Chris Noessel. 2014. *About Face: The Essentials of Interaction Design* (4th ed.). Wiley.

[70] Paul Resnick and Richard Zeckhauser. 2002. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. In *The Economics of the Internet and E-commerce*. Emerald Group Publishing Ltd.

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

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

[73] Georg Simmel. 1949. The sociology of sociability. *American Journal of Sociology* 55, 3 (1949), 254–261.

[74] Lee Sproull. 2011. Prosocial behavior on the net. *Daedalus* 140, 4 (2011), 140–153.

[75] 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 (ASSETS'08)*. 193–200.

[76] 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 (ASSETS'09)*. 195–202.

[77] Mark S. Ackerman, Volkmar Pipek, and Volker Wulf. 2003. *Sharing Expertise: Beyond Knowledge Management*. MIT Press, Cambridge, MA.

[78] Molly McLure Wasko and Samer Faraj. 2005. Why should I share? Examining social capital and knowledge contribution in electronic networks of practice. *MIS Quarterly* 29, 1 (2005), 35–57.

[79] WebAIM. 2020. Screen Reader User Survey #3 Results. Retrieved February 11, 2022 from http://webaim.org/projects/screenreadersurvey3/.

[80] Robert Philip Weber. 1990. *Basic Content Analysis* (2nd ed.). SAGE.

[81] 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 (OOPSLA Companion'08)*. ACM, New York, NY, 893–894. https://doi.org/10.1145/1449814.1449895

[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 *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS'19)*. ACM, New York, NY, 96–104. https://doi.org/10.1145/3308561.3353806

[83] Youngjin Yoo, Ola Henfridsson, and Kalle Lyytinen. 2010. Research commentary—The new organizing logic of digital innovation: An agenda for information systems research. *Information Systems Research* 21, 4 (2010), 724–735.

[84] Raymond F. Zammuto, Terri L. Griffith, Ann Majchrzak, Deborah J. Dougherty, and Samer Faraj. 2007. Information technology and the changing fabric of organization. *Organization Science* 18, 5 (2007), 749–762.

Received 2 July 2022; revised 14 January 2023; accepted 21 January 2023