Heriot-Watt University

Master Class Report

User Interface Design for Vision Impaired Users

Screen Readers and Other Assistive Technologies

James McLean
Software Engineering MEng

Supervised by
Andrew Ireland

Purpose

The purpose of this document is to serve as the basis for a masterclass on digital accessibility. The masterclass will comprise a presented lecture on the subject matter and a practical exercise to reinforce the learned material.

Overview

This document presents a study into digital accessibility, covering HCI, keyboard navigation, and screen readers (as well as other assistive technologies).

Contents

Introduction

Computers are an inescapable necessity of modern society, permeating almost every aspect of our lives. Common tasks of information gathering previously done through various means; by telephone; written by hand; face to face; etc, are now done purely through computers. Given this move towards digital methods, those who rely on assistive technologies will become disenfranchised if computer systems fail to support those technologies.

While, not a widely considered aspect of *digital poverty*, lack of support could certainly be considered to fall under the Digital Poverty Alliance's definition:

> "The inability to interact with the online world fully, when, where and how an individual needs to."[1]

Due to this inevitability, we must consider how we use and interact with computers so everyone can benefit from the efficiency improvement they bring. This is especially important for users with disabilities which impact the way in which they use computers - tasks which are now computer reliant may not previously have been hindered by these disabilities.

For example, a person with a vision impairment may previously have used a telephone for various tasks that have now been moved online. If the online services do not support the necessary assistive technologies effectively, these tasks can become impossible.

An important tool for vision impaired users is the *Screen Reader*, an assistive technology which creates an audio description of the user interface. Creating a user interfact (UI) which can be effectively navigated with a screen reader is crucial for users who cannot rely on a graphical output.

When done properly, computer accessibility has the potential to provide greater independence to users with disabilities; when done poorly, it it is likely to make them more reliant on others.

Chapter 1

How do we interact with computers?

Before we can look at the technical side of computer accessibility, we must first briefly consider high level concepts of Human-Computer Interaction (HCI) - the study of the design of computer interfaces.

First and foremost, *User Interface design* - this focuses on the design of software UIs as well as interaction peripherals. HCI considerations include layout, navigation, visual design, and interaction patterns [2].

Leading on from this, we must also think about *User Experience (UX)*. This examines how positive (or negative) the experience of using an interface is. These two concepts form the high level backbone of HCI.

Another HCI concept we will consider is *conceptual modelling*. A conceptual model is an abstract representation of the mental model a user forms around using a system. When designing an interface, we need to understand the user's understanding of the interface. This is important when getting into the world of screen readers as for most visual users it is difficult to fully appreciate what makes for a good audio-based UI.

Finally, we must look at the components involved in constructing an HCI model:

- Users: The human users of the system - for accessibility, constraints of their ability to use a computer is the primary focus.
- System: The computer system in question - hardware interface as well as software interface.
- World: The environment the interaction takes place in - we must consider any challenges posed by the context and surroundings of the

interaction.

Now we have covered the fundamentals of interaction design, we can begin to look at the underpinning theory of digital accessibility.

Chapter 2

How can we be accessible?

An important consideration when creating UIs is "Design for all"[3]. This simply means the interface can be used effectively by all user types, regardless of ability. This could be as basic as use of easy to understand text for users who may not speak the language fluently, or a simple interface for users with below average computer literacy skills. One of the most important components of this is how we navigate the UI - many users cannot use traditional input/control methods due to disability. Figure 2.1 shows the user types that rely on keyboard navigation and screen readers.

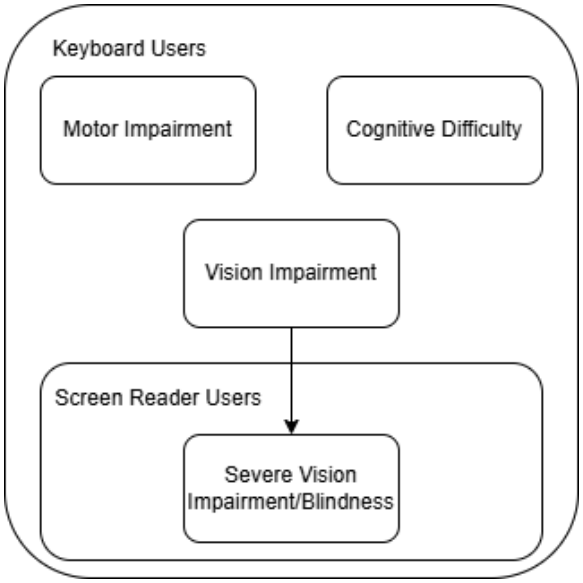


Figure 2.1: A taxonomy of user types with different disabilities.

2.1 Keyboard navigation

The modern user interface is primarily designed around two input devices; keyboard and mouse. The keyboard, the more established of the pair, fulfils the requirement for explicit data entry. Seeing as it developed naturally as an abstraction of the typewriter, it builds on decades of refinement to provide an intuitive input method.

In 1964 Doug Englbart created the first computer mouse, a wheel based device in a wooden enclosure. However, the mouse would not become a standard peripheral until the mid-eighties with the development of more advanced graphical interfaces [4]. These new UIs, focussed on providing a more intuitive user experience, standardised the common relationship we know today; mouse for navigation, keyboard for data input.

However, the mouse demands two expectations of its user; fine motor skills and the ability to track the cursor. Users with motor impairments may not be able to accurately manipulate the mouse when trying to use a mouse-based interface. Users with cognitive difficulties or vision impairments may now be able to follow the cursor on the screen.

What we need is an alternative which is static and visually bold. Since we are eliminating the mouse, we are left with just the keyboard for navigation and data input. The keyboard can fulfil this purpose very well as the keys offer a discrete form control and do not require high precision to operate.

The most obvious would be the arrow keys which have been a common alternative for cursor control since the eighties. However, as previously outlined, we are trying to move away from the cursor as well as the mouse. In order to be accessible for non-cursor users, the UI must be based around item focus and selection - the focus of the current item must be visually bold so it is clear to follow.

The standard method of keyboard control the tab key. This involves linearly cycling through menu items using tab and reversing using shift+tab. Arrow keys are then used as a secondary control scheme for inside elements - e.g. navigating between items in a selected drop-down.

The main reason for wanting a linear system is for the simplicity of using other alternative input devices. As previously established, non-cursor users are unable to use mice or similar precise navigation systems, however, this group also includes users who will also be unable to use a standard keyboard. There are a number of peripherals which can utilise a linear navigation system, for example sip-and-puff devices.

Sip-and-puff systems consist of a straw or mouthpiece which the user can blow and suck to interact with a computer. This matches the tab-

based navigation method well as the two distinct actions can be mapped to forwards and back. However, it is important to keep in mind that devices such as this are highly customisable to the user so this is not an exact specification - though it is common for these peripherals to utilise keyboard navigation in some form. Sip-and-puff devices can be calibrated to recognise pressure of input - how hard the user blows or sucks - for a further level of interaction [5].

2.1.1 An example of keyboard navigation

Figure 2.2 shows a storyboard of the keyboard navigation order of a login screen. Each element is focused in an intuitive left-to-right; top-to-bottom manner.[1]

Figure 2.2: Login screen keyboard navigation.

2.2 Screen readers

Above we have established that keyboard navigation helps users who are unable to use a cursor-based interaction method; either through a keyboard or

[1] The order which is most intuitive will depend cultural norms.

another assistive peripheral, but this style of navigation can also be utilised for more than just the user types outlined so far.

As previously mentioned not all non-cursor users will have a motor impairment, some may have cognitive difficulties or a vision impairment which may make it difficult to follow a cursor. Users with more severe vision impairments or blindness will need an alternative for system output as well as input.

Arguably the most common computer interface component is the screen. Not only is it an output for a system incorporating a keyboard and/or mouse, or another form of controller, but also a full input/output device in the case of touch screens. Just how reliant we are on a visual representation of computer data is a rare consideration. But how do users unable to view a screen interact with a computer?

The main technology employed by vision impaired users is the screen reader. A screen reader is a piece of software that simply describes a visual interface through audio. This can range from an initial description of the full interface, though to more detailed descriptions of individual elements and dynamic content.

Screen readers rely on keyboard navigation, notifying users of focussed elements and giving further details available only though visual context - disabled elements, stateful elements, required fields, etc. This makes good keyboard navigation even more important - certain liberties that may not necessarily cause issue for users who can work from the visual context of the UI, such as the order of element navigation, can lead to problems for screen reader users.

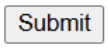
But intuitive keyboard navigation is only half the story; in order to have a suitable replacement for the computer screen we need to abstract the visual element effectively. Descriptions must be clear and concise, presenting essential information to the user. In a web-form for example, when a field is focussed, the information given by the screen reader should follow a structure such as:

[field type][field name][required]

"Text field, full name, required."

For a button, the description should follow a structure such as:

[action] button



"Submit button."

When entering a menu the description should follow a structure such as:

[menu type] with [x] items - [menu items]

"Navigation with four items - home link, news, link, contact link, about link."

2.2.1 A mental model for screen readers

When considering a mental model for a screen reader, we need to think about how we can write assistive descriptions which provide a functionally equivalent experience to using the visual interface.

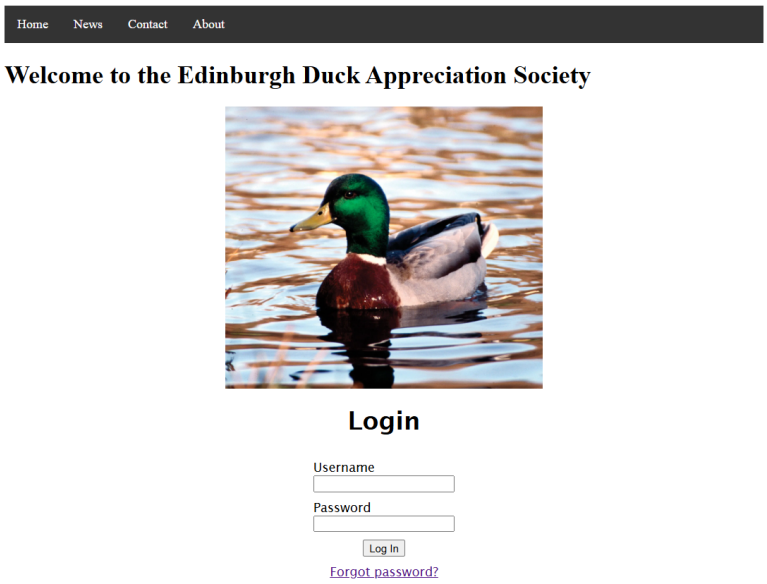


Figure 2.3: An example webpage.

Figure 2.3 shows an example webpage. Mapping a mental model to the visual UI is fairly simple, the processes to operate the page will be intuitive

to anyone familiar with basic computer literacy skills. However, to consider how a screen reader user may interpret the page is another matter.

A possible good screen reader description of the page is as follows:

"Navigation bar with four links: Home, News, Contact, About - heading level one: Welcome to the Edinburgh Duck Appreciation Society - Image: A mallard duck in water - heading level one: Login - Text edit: Username, blank - Text edit: Password, blank - Log in button - Forgot password? link"

Now this is a fairly simple page, but it gives an idea of what we should be expecting. It is not enough just to add descriptions based on what we see, we need to think about the effectiveness of pure description alone.

There is, of course, no defined standard and each screen reader will generate descriptions in a slightly different way, but these examples give an idea of what a good description should sound like. Screen readers, as will all assistive technologies, are highly customisable, allowing the user to set parameters such as accents, talking speed and even verbosity - this is why use of concise descriptions along with proper use of metadata is so important, the screen reader needs to know what information is relevant to its settings when generating a description.

If good descriptions are not provided pages can become unusable for screen reader users. For example, the fields within a form should have an associated label, if a label is not referenced the screen reader would announce the following field thusly:

"Text edit, blank"

The field is announced, but there is no indication of what the field is for or its expected input. If the label is correctly connected, the screen reader would announce it as:

"Text edit: Username, blank"

A common situation where this is an issue is multi-line address fields. If there is only one address label (see figure 2.4) then only that first field will have a descriptive screen reader announcement.

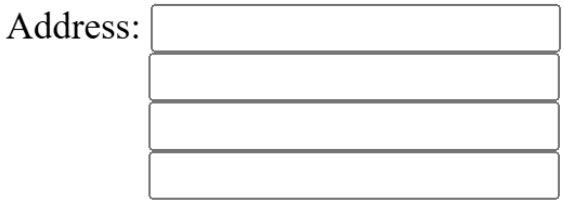


Figure 2.4: A poor example of multi-line address fields.

We sort this by giving each address line a label to make it clearer to the user (see figure 2.5).

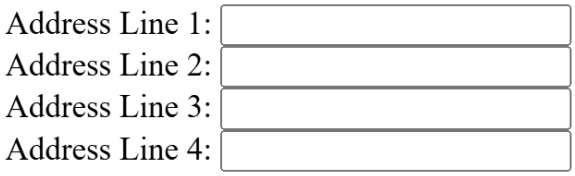


Figure 2.5: A good example of multi-line address fields.

Another common example of poor description is when a form has a "Name" field. This does not seem like major issue on the surface, but it is very unclear. It is uncertain as to whether the expected input is the user's first name, surname, or full name. For screen-based users, this may be obvious from the context - they can presume based on whether the form also contains a "Surname" field. For screen reader users, this context is not so easy to gather. If their assumption is wrong, they need to take the time to go back and amend. The user should be able to fill out the form accurately without needing a second pass.

This shows the importance of considering conceptual models when designing a screen reader compatible UI - we need to think about how we can best support the user's understanding of the system.

2.3 Braille Readers

An assistive technology adjacent to screen readers are *Braille Readers*. These devices consist of input and control keys, and a refreshable Braille display to dynamically provide page descriptions using a similar system to a screen reader.

Braille readers rely on many of the same accessibility concepts as screen readers, they just provide output in a different format.

Chapter 3

How can we implement accessiblity features?

Having now outlined the importance of accessibility support, and the technologies and strategies we can utilise, we must now consider the implementation.

Accessibility support can be developed in various ways depending on the technology/framework being used to develop the UI. For example, The .NET UI framework Windows Forms offers accessibility metadata in the properties for most elements, as well as features such as high contrast mode [6]. This is all handled automatically by the framework, so all developers need do is fill out the metadata appropriately.

For web pages there exists the Web Content Accessibility Guidelines (WCAG), a specification on accessibility maintained by the World Wide Web Consortium (W3C). This provides a thorough set of expectations for webpages, along with complete guidance on implementation through HTML, CSS and JavaScript. The prevalence of web based interfaces through web applications, and desktop applications through frameworks such as Electron, means WCAG can be directly applied across a great range of software. For this reason it will be the primary focus of this chapter.

3.1 Web Content Accessibility Guidelines

WCAG version 1.0 was released by W3C in 1999 as a set of standards for web pages [7]. It primarily focuses on HTML features which can be utilised to support accessible technologies. Each guideline has a compliance level; A, AA or AAA. To reach a certain level of compliance, a webpage must meet

all of that level's criteria, with AAA compliance being the most desirable.

In 2008, WCAG 2.0 was released. It built upon the previous version, by introducing four principles [8]:

- Perceivable - Information and user interface components must be presentable to users in ways they can perceive.
- Operable - User interface components and navigation must be operable.
- Understandable - Information and the operation of user interface must be understandable.
- Robust - Content must be robust enough that it can be interpreted reliably by a wide variety of user agents[1], including assistive technologies.

These principles are used to group guidelines so developers have an idea of the key areas they should focus on.

Further updates have come in the form of 2.1 in 2018, and 2.2 in 2023. These built upon the previous version to better reflect how the use of the web has changed since 2008; more dynamic webpages, move towards mobile devices, etc.

In 2021 a complete overhaul to the standard, WCAG 3, was announced. The most recent working draft (at time of writing) was released in December 2024 and contains a bare bones overview of the guidelines, with most branded with a "Needs additional research" editor's note [10].

3.2 How can we achieve WCAG compliance?

The WCAG document provides a specification to work from, but it is important to understand how use it. As previously stated, guidelines are categorised by four principles; perceivable, operable, understandable, robust. As well as this, they have a compliance level; A, AA, and AAA[2].

To achieve a conformance level, a website must satisfy all of that level's success criteria. Most large websites provide an "accessibility statement"

[1] "User agents" are defined by W3C as, "software that people use to access web content, including desktop graphical browsers, voice browsers, mobile phone browsers, multimedia players, plug-ins, and some assistive technologies." [9]

[2] This (as with the rest of this section) refers to WCAG 2.2 and prior. WCAG 3 will use a new system, but as of the latest update (December 2024) the precise details are unknown.

which outlines the targeted conformance level, as well as the approach taken to maintaining that level. AAA conformance is rare, with most widely used web services only reaching partial AA conformance.

An interesting example of this is gov.uk, a website widely praised for its WCAG focussed design. In its accessibility statement, it declares twenty instances of non-compliance[3][11]. This goes to show how difficult it is to maintain full compliance on a large scale.

3.3 Adding accessibility features to web-based interfaces

3.3.1 ARIA

Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA or just ARIA) are a collection of roles and attributes which extend standard HTML to provide further accessibility support. To use ARIA constructs properly, there are five rules to follow [12]:

1. Do not use ARIA unless absolutely necessary - always use standard HTML/CSS features where possible.
2. Do not change native semantics, unless you really have to - avoid making completely custom elements with ARIA attributes.
3. All interactive ARIA controls must be usable with the keyboard.
4. Do not use role="presentation" or aria-hidden="true" on a focusable element - these attributes will cause the element to be completely ignored by assistive technologies.
5. All interactive elements must have an accessible name - this name must provide all context derivable from the visual interface.

Particular ARIA attributes to consider are:

- *role* - This attribute allows further contextual information to be provided for an element. It is useful for breaking the page into sections.
- *aria-expanded* - For elements such as drop-down menus which have expanded and collapsed states.

[3] As of early 2025.

- *aria-labelledby* - For input fields to reference the relevant label.
- *aria-label* - Similar to previous, but used when it is not possible to reference the label.

3.3.2 Breaking things down

A good starting point for making a webpage more screen reader friendly is breaking the page down into sections.

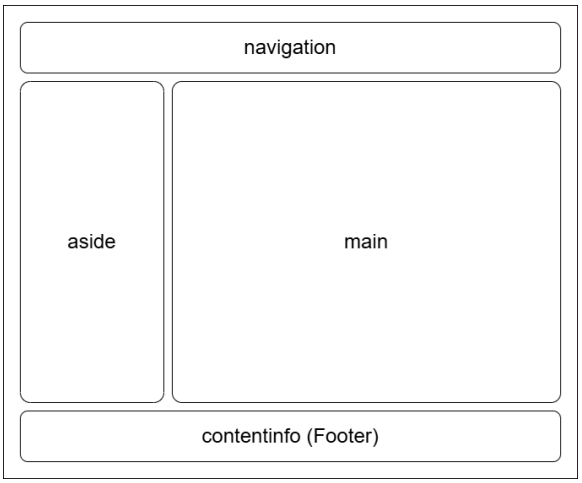


Figure 3.1: A webpage broken down by section.

Figure 3.1 shows a breakdown of a typical web page with some common page sections. To specify these sections in the page design, the *role* attribute is applied to elements to create a landmark. In this case, the top section, possibly a nav-bar, is given the role value "navigation". The bottom element, the footer, is given role value "contentinfo".

The primary section between these is the main content section which has the role "main" - this section is the central focus of the page. Finally, beside the main section is a section with the role "aside" - this section could provide secondary information or act as a side menu.

These roles are used by the screen reader to describe the page structure.

3.3.3 Forms and inputs

As mentioned in 2.2, correctly connecting an input element to its relevant label is crucial for an effective page description.

Two of the ARIA attributes mentioned in 3.3.1 can be used to achieve this; *aria-labeledby* and *aria-label*.

A standard HTML input field follows a structure such as:

```
<label for="name-field">Full Name:</label>
<input id="name-field">
```

The label *is* referenced with the *for* attribute, however this reference is used for extracting the raw form data after submission. For some assistive technologies, this may not be enough.

We can add the *aria-labeledby* attribute to guarantee this connection:

```
<label for="name-field">Full Name:</label>
<input id="name-field" aria-labeledby="name-field">
```

Another benefit of aria-labeledby is that it can be used to reference multiple label elements. It can also be used by with non-form elements if appropriate.

It may be that the label does not provide all the necessary information, or cannot be connected to with *aria-labeledby*. This would be where the *aria-label* attribute would come into play. The following is an example where not all the relevant info is available through the label.

```
<label for="name-field">Full Name:</label>
<span>Enter in block caps.</span>
<input id="name-field" aria-label="Full name, enter
    in block caps">
```

3.3.4 Further considerations

Skip to content

A *Skip to Content* button is a feature which makes navigating multiple pages of a website much faster for keyboard and screen reader users. Because there will be a consistent structure across pages, users will have to navigate through the same menus every time they move to a new page.

This can be bypassed by a skip to content button (see figure 3.2). This is a the first focusable element in the page, and is hidden when not focused. When selected, focus is moved to the main section of the page. Its target is defined using landmarks, so multiple options can be available for jumping to different sections.

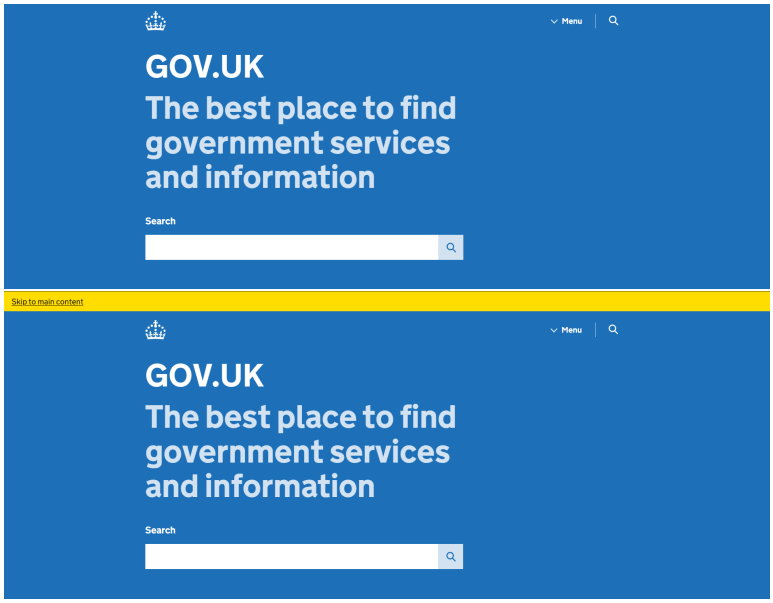


Figure 3.2: Skip to content button hidden and shown on gov.uk.

Tabindex

Precedence of focus can be defined using the *tabindex* attribute. Elements with a lower *tabindex* value will be focused before ones with a higher value.

As precedence is generally inferred from the order of elements in the DOM, the most common use of *tabindex* is to make non-focusable elements focusable. This is done by setting the value to "0".

However, it is important to keep in mind that it is heavily advised to avoid making non-focusable elements focusable as it can make the page more confusing to navigate [13].

3.4 Testing and evaluation

There are two main approaches that can be taken to developing accessible features in a web page; ad-hoc and post-hoc.

3.4.1 Ad-hoc

The slower, but potentially more effective approach is to add accessibility features during the main development of a webpage or web-application. This

means adding the attributes when elements are created and incorporating testing into the main testing procedure.

This approach relies primarily on the developer, and their knowledge and experience with the WCAG spec.

3.4.2 Post-hoc

The faster and more commonly employed approach is a more post-hoc method, evaluating and improving the accessibility as a secondary part of the development process.

To aid in the evaluation process, browser extensions can be used to scan the webpage for WCAG non-compliance and other HCI errors. This can also be available as an automated service, scanning pages periodically and alerting developers of any issues that arise.

3.4.3 Which is better?

As with so many things, a mixture of the two approaches is probably ideal. The focus either way will be informed by the wider development methodology of the project. Generally the ad-hoc approach promotes accessibility in the design process, but the chances of getting it right straight away are highly unlikely. Striking a balance between the two is key.

Compliance by construction can be employed to support either approach, taking the responsibility of accessibility feature out of the hands of the developer. Web component libraries are heavily used in web-application development and these will provide some level of WCAG compliance by default.

Well Formed is a tool which can be used to generate WCAG compliant form elements. Forms are designed using a visual interface, and come with a basic CSS file.

Final thoughts

When developing accessible user interfaces, we need to take a user-centred approach. This means considering not just the visual design, but also the interaction methods used. Applications and websites cannot rely on a traditional keyboard and mouse based input system or a visual output system alone. Linear keyboard controllable applications can support a multitude of assistive technologies, including screen readers, Braille readers, sip and puff devices, etc.

Screen readers and Braille readers also rely on clear and effective element descriptions for generating usable page descriptions. For websites and web applications, WAI-ARIA can be used to support this where standard HTML cannot and following the WCAG specification can help with constructing a set of requirements to guide development.

Finally, finding the right approach to implementing accessibility features can be crucial to the form the final product takes. Implementing during main development could lead to a better final result as accessibility will inform the design of the application to a greater degree. Taking an accessibility phase at the end of a major piece of development may be faster if there are no major problems with the design.

Making use of accessible by construction tools such as Well Formed for create web forms, as well as the wide array of WCAG compliant component libraries, can reduce accessibility concern considerably.

In our increasingly digital world, designing applications an websites in an accessible way is crucial to support the independence of people with disabilities.

Further reading

W3Cx: Introduction to Web Accessibility - W3C

`https://www.edx.org/learn/web-accessibility/the-world-wide-web-consortium-w3c-introduction-to-web-accessibility`

WCAG 2.2 (latest published version) - W3C

`https://www.w3.org/TR/WCAG22/`

WCAG 3 (latest published version) - W3C

`https://www.w3.org/TR/wcag-3.0/`

A Generator for Accessible HTML Forms - James McLean

`https://www.jmcl.xyz/dissertation`

Bibliography

[1] Digital Poverty Alliance. *How we define digital poverty.* URL: https://digitalpovertyalliance.org/. (accessed: 06.03.2025).

[2] Faimina Pathan, Ashish Deharkar, and Lowlesh Yadav. "HUMAN-COMPUTER INTERACTION (HCI)". In: *International Research Journal of Modernization in Engineering Technology and Science* 6 (July 2024), pp. 2582–5208.

[3] Martin Schrepp and Patrick Fischer. "A GOMS Model for Keyboard Navigation in Web Pages and Web Applications". In: vol. 4061. July 2006, pp. 287–294. ISBN: 978-3-540-36020-9. DOI: 10.1007/11788713_43.

[4] Doug Engelbart Institute. *Historic Firsts: The Mouse.* URL: https://dougengelbart.org/content/view/162/. (accessed: 19.01.25).

[5] Bureau of Internet Accessibility. *Understanding Assistive Technologies: What Are Sip-and-Puff Systems?* URL: https://www.boia.org/blog/understanding-assistive-technologies-what-are-sip-and-puff-systems. (accessed: 01.02.2025).

[6] DCtheGeek adego brittmsantos brittmsantos. *Walkthrough: Creating an Accessible Windows-based Application.* URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/advanced/walkthrough-creating-an-accessible-windows-based-application?view=netframeworkdesktop-4.8. (accessed: 08.02.2025).

[7] Pete Bruhn. *WCAG Version History.* URL: https://accessibleweb.com/wcag/wcag-version-history/. (accessed: 10.02.2025).

[8] W3C. *Introduction to Understanding WCAG 2.0.* URL: https://www.w3.org/TR/UNDERSTANDING-WCAG20/intro.html. (accessed: 11.02.2025).

[9] Shadi Abou Zahra, ed. *Accessibility Principles*. URL: https://www.w3.org/WAI/fundamentals/accessibility-principles/. (accessed: 11.02.2025).

[10] WC3. *W3C Accessibility Guidelines (WCAG) 3.0*. URL: https://www.w3.org/TR/2024/WD-wcag-3.0-20241212/. (accessed: 13.02.2025).

[11] GOV.UK. *Accessibility statement for www.gov.uk*. URL: https://www.gov.uk/help/accessibility-statement. (accessed: 13.02.2025).

[12] David MacDonald Steve Faulkner, ed. *Using ARIA*. URL: https://www.w3.org/TR/using-aria/. (accessed: 06.03.2025).

[13] Mozilla. *tabindex - Accessibility Concerns*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex#accessibility_concerns. (accessed: 06.03.2025).