# WAS Study Artifacts v1.0

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.1.1 (Non-text Content)

- Every non-text element requires a text alternative.

- People who are blind or deafblind cannot access images directly; they rely on alternative text, which is read by a screen reader or converted to refreshable braille.

- The alternative text must:

• Be programmatically determinable (attached to the element in markup).

• Be meaningful (conveying the author’s intent, not just a literal description).

• Be concise (ideally ≤150 characters; long or complex images may need a supplemental long description).

Best Practices:

- Use `alt` attribute for `<img>` whenever possible.

- Avoid phrases like “image of” or “graphic of” — screen readers announce this automatically.

- If the image conveys complex information, provide both short alt text and a longer description (on-page, linked, expandable, or dialog-based).

- For purely decorative/redundant images, use `alt=""`, `role="presentation"`, or implement as CSS background images so assistive technologies ignore them.

Examples:

Bad Example: `<img src="stamp\_final\_notice.png" alt="">`

- This removes important information (text in image “Final Notice”).

Good Example: `<img src="stamp\_final\_notice.png" alt="Final Notice">`

- Conveys the essential purpose of the image.

Special Cases:

- \*\*Logos\*\*: alt text should convey the brand or company name.

- \*\*Actionable images (buttons, links)\*\*: alt text must describe the action or destination (e.g., “Search” not “Magnifying glass”).

- \*\*Form inputs type="image"\*\*: alt text must describe the function (e.g., “Submit”).

- \*\*Animated images\*\*: provide a way to stop/pause if >5 seconds, and never flash more than 3 times per second.

- \*\*Images of text\*\*: avoid unless essential (e.g., logo); prefer real text.

- \*\*Complex graphics (charts, diagrams, paintings)\*\*: use concise alt text plus a long description that conveys details and author intent.

Key WCAG Mapping:

- SC 1.1.1: Non-text Content.

- SC 1.4.5: Images of Text (discourages text in images unless essential).

- SC 2.2.2: Pause, Stop, Hide (for animated images).

- SC 2.3.1: Three Flashes or Below Threshold.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.1.1 (Non-text Content), 1.4.1 (Use of Color), 1.4.3 (Contrast), 2.1.1 (Keyboard)

Complex Images

- Some images (e.g., detailed diagrams, paintings, maps) require more than short alt text.

- Strategy:

• Provide concise alt text (≤150 characters) summarizing purpose.

• Supply a long description when necessary (on-page, linked, expandable, or dialog-based).

- Long description content depends on author’s intent: literal, symbolic, instructional, etc.

- Example: A map showing a walking route → short alt: “Walking directions from Starbucks to Insomnia Cookies”; long description provides step-by-step text directions.

Scalable Vector Graphics (SVG)

- Advantages: scalable without loss of clarity, customizable colors, supports accessibility features directly.

- Best practices:

• Assign `role="img"`.

• Provide alternative text via `<title>` or ARIA (`aria-label`, `aria-labelledby`).

• For complex SVGs, provide long descriptions.

• Decorative SVGs: hide with `aria-hidden="true"`.

• Avoid embedding via `<object>` or `<iframe>`.

• Ensure sufficient color contrast (≥4.5:1 small text; ≥3:1 large/bold text).

• Eliminate or minimize inline `<text>` inside SVG, or reference it in alt/long descriptions.

- Animated SVGs:

• Do not flash >3 times per second.

• Should not auto-play >5 seconds; allow user control (pause/stop).

- Interactive SVGs:

• Must be fully keyboard- and touch-accessible.

• Provide visible focus indicators.

• Expose name, role, value of interactive elements.

Icon Fonts

- Function like images, but are technically fonts.

- Informative/actionable icon fonts without visible text:

• Assign `role="img"`.

• Provide alt text via `aria-label` or `aria-labelledby`.

- Decorative/redundant icon fonts: hide with `aria-hidden="true"`.

- Pros/cons:

• Icon fonts: lightweight, scale well, customizable color, but limited to one color.

• SVG icons: can be multi-colored but rely on image support.

HTML5 Canvas

- Used to draw graphics with JavaScript.

- Accessibility concern: content inside `<canvas>` is not inherently accessible.

- Best practices:

• Assign `role="img"`.

• Provide concise alt text with `aria-label` or fallback text linked via `aria-labelledby`.

• For complex graphics, include long descriptions (e.g., accessible tables linked with `aria-describedby`).

• Ensure a background fill (for high contrast mode).

• Avoid text inside canvas; use SVG if text needed.

• Ensure custom canvas controls are keyboard accessible (`onkeydown`, `onkeyup`).

• Use ARIA roles (`role="button"`, `aria-label="Play"`) for interactive canvas elements.

Key WCAG Mapping:

- SC 1.1.1: Non-text Content.

- SC 1.4.3: Contrast (min contrast ratios).

- SC 2.1.1: Keyboard (keyboard operability for interactive content).

- SC 2.2.2: Pause, Stop, Hide (control of animations).

- SC 2.3.1: Three Flashes or Below Threshold.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.2.x (Time-based Media), 1.4.2 (Audio Control), 2.2.2 (Pause, Stop, Hide)

Multimedia (Audio/Video)

- Four primary methods make audio/video accessible:

• Captions — synchronized text for spoken words and key sounds.

• Transcripts — standalone text of spoken words and important visuals.

• Audio descriptions — narration of key visual details not conveyed in audio.

• Sign language interpretation — synchronized visual translation of spoken language.

- Prerecorded multimedia:

• MUST provide captions.

• SHOULD include transcripts (essential for deafblind users).

• MUST provide audio description if visual content is essential.

- Live multimedia:

• MUST provide live captions for spoken content (e.g., stenographer or CART service).

• Audio description optional for live content, but recommended.

- Auto-play:

• Audio/video that starts automatically can interfere with screen readers and comprehension.

• WCAG SC 1.4.2 requires a mechanism to pause, stop, or mute audio playing longer than 3 seconds.

Objects and Plug-ins

- `<object>` elements can embed PDFs, media, or plug-ins.

- MUST provide meaningful alternative text via `aria-label` or `aria-labelledby`.

- Historical plug-ins (Flash, Silverlight, Java) are deprecated and generally inaccessible. Use HTML5 technologies instead.

- If plug-ins are unavoidable:

• Must expose accessibility information through APIs.

• Must meet the same accessibility requirements as other content.

Documents

- HTML is the most accessible format (built-in semantic structure, broad device support).

- For non-HTML documents (Word, PowerPoint, PDF, EPUB):

• Apply accessibility principles: meaningful titles, heading structure, alt text for images, data table headers, color contrast, readable fonts.

• PDFs: must be tagged PDF for accessibility.

• EPUB: adhere to HTML accessibility principles and EPUB Accessibility specification (EPUB 3 preferred).

• Provide an HTML alternative version when possible for maximum accessibility.

Key WCAG Mapping:

- SC 1.2.2: Captions (Prerecorded).

- SC 1.2.3: Audio Description or Media Alternative.

- SC 1.2.4: Captions (Live).

- SC 1.2.5: Audio Description (Prerecorded).

- SC 1.4.2: Audio Control.

- SC 2.2.2: Pause, Stop, Hide.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Summary & Checklist (Deque Content 1)

Summary

- Every non-text element must have a meaningful alternative (alt text, transcripts, captions, audio descriptions, or sign language as appropriate).

- Alternative text must be programmatically determinable, meaningful, and concise.

- Complex images require both concise alt text and a longer description.

- Decorative/redundant images should be hidden from assistive technologies.

- SVG, Canvas, and Icon Fonts can be made accessible with proper roles, labels, and keyboard support.

- Multimedia must include captions, transcripts, and audio descriptions; avoid or control auto-play.

- Objects and plug-ins must have alternative text; avoid legacy plug-ins.

- Non-HTML documents must follow accessibility principles; tagged PDF and EPUB 3 are required for full accessibility.

Checklist

Images

- Informative images must have concise, meaningful alt text.

- Decorative/redundant images must have null alt text or be hidden with role="presentation".

- Actionable images must describe purpose or action.

- Complex images require alt text plus long description.

- Images of text should be avoided unless essential.

- Animated images must provide pause/stop controls and not flash more than 3 times per second.

SVG

- Assign role="img".

- Provide alt text via title, aria-label, or aria-labelledby.

- Use long descriptions for complex SVGs.

- Hide decorative SVGs with aria-hidden="true".

- Ensure color contrast is sufficient.

- Avoid embedding via object or iframe.

- Ensure keyboard and touch accessibility for interactive SVGs.

Icon Fonts

- Assign role="img" and provide alt text for informative/actionable icons without text.

- Hide decorative/redundant icons with aria-hidden="true".

Canvas

- Assign role="img".

- Provide alt text via aria-label, aria-labelledby, or fallback text.

- Supply long descriptions for complex content.

- Ensure background fill for high contrast.

- Make custom canvas controls keyboard accessible.

Multimedia

- Prerecorded multimedia: captions required, transcripts recommended, audio descriptions required when visuals are essential.

- Live multimedia: captions required for spoken content.

- Avoid auto-play or provide pause/stop/mute control.

Documents and Objects

- Provide meaningful alternative text for <object> elements.

- Prefer HTML for accessibility.

- PDFs must be tagged.

- EPUB must conform to EPUB Accessibility specification (EPUB 3 preferred).

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.1 (Bypass Blocks), 2.4.3 (Focus Order), 2.4.6 (Headings and Labels)

Semantic Structure and Navigation (Introductory Concepts)

- Semantic structure enables assistive technologies to convey meaningful information about a page’s content and layout.

- Headings, lists, landmarks, and proper semantic tags provide a navigable framework for screen reader and keyboard users.

- Without semantic structure, users may need to read an entire page linearly, which is inefficient and confusing.

Key Elements

- Page Titles: Each page must have a unique, meaningful <title> element that describes purpose or topic.

- Headings:

• Use true heading elements (<h1>, <h2>, etc.) instead of styled text.

• Maintain logical hierarchy; do not skip levels.

• Use concise, descriptive text in headings.

- Landmarks:

• Use HTML5 landmarks (<header>, <nav>, <main>, <footer>) or ARIA landmark roles.

• Minimize duplicate landmarks; when repeated, give unique labels with aria-label or aria-labelledby.

- Navigation:

• Provide skip links or methods to bypass repetitive blocks of content.

• Maintain consistent navigation order across pages.

- Reading and Focus Order:

• Ensure source code order matches logical reading and visual order.

• tabindex values should not be used to force custom orders, except in rare controlled cases.

Examples

Bad Example: Large headings styled with <div> and CSS only → screen readers do not recognize them as headings.

Good Example: <h2>About Our Services</h2> → provides both styling and semantic meaning.

WCAG Mapping

- SC 2.4.1: Bypass Blocks (skip navigation).

- SC 2.4.2: Page Titled.

- SC 2.4.3: Focus Order.

- SC 2.4.6: Headings and Labels.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.2 (Page Titled), 3.1.1 (Language of Page), 3.1.2 (Language of Parts)

Page Titles and Language

Page Titles

- Every page must include a <title> element with descriptive text.

- Titles must be accurate and meaningful, not generic (e.g., “Home” or “Untitled”).

- Page titles should reflect page purpose and match or closely align with the main heading (H1).

- Unique information should appear first in the title.

- For pages that are part of a workflow or multi-step process, titles should indicate current step.

Language of Page

- The primary language of the page must be specified in the <html lang=""> attribute.

- The language value must be valid (e.g., lang="en", lang="fr").

- Screen readers use this to select correct pronunciation rules.

Language of Parts

- Inline changes in language must be identified with a valid lang attribute on the specific element.

- Examples: <p lang="es">Hola</p> within an English page.

- This ensures correct pronunciation and braille output.

Examples

Bad Example: <title>Untitled Document</title>

Good Example: <title>Order Form – Step 2: Shipping Information</title>

Bad Example: <html>

Good Example: <html lang="en">

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships), 2.4.1 (Bypass Blocks)

Landmarks

- Landmarks identify key regions of a page, allowing screen reader and keyboard users to navigate efficiently.

- HTML5 landmarks include: <header>, <nav>, <main>, <footer>, <aside>.

- ARIA roles can supplement or replace landmarks when necessary: role="banner", role="navigation", role="main", role="contentinfo", role="complementary".

- Use landmarks consistently across pages.

Best Practices

- Every page should include a <main> region, containing the central content.

- Avoid more than one <main>, <banner>, or <contentinfo> landmark per page.

- If multiple landmarks of the same type are needed (e.g., two navigation regions), give each a unique, programmatically determinable label using aria-label or aria-labelledby.

- Keep the total number of landmarks to a minimum, consistent with the page’s structure.

- All visible text should be inside a landmark.

Accessibility Considerations

- Landmarks provide orientation and allow screen reader users to jump directly to regions.

- Proper landmark use improves efficiency, especially on content-rich or repetitive pages.

Examples

Bad Example: Entire page wrapped in <div> elements without landmarks → forces linear reading.

Good Example: <main id="content">…</main>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships), 2.4.6 (Headings and Labels)

Headings

- Headings organize content into a logical hierarchy that is perceivable by both sighted and non-sighted users.

- Use proper heading elements (<h1>–<h6>) to represent structure, not visual styling with <div> or <span>.

- Headings must be accurate, informative, and concise.

- Headings should follow a logical outline:

• Begin main content with a single <h1>.

• Use subheadings in correct order (<h2>, <h3>, etc.).

• Avoid skipping levels (e.g., <h1> directly to <h3>).

- Avoid using heading tags solely for visual effect when the text is not a true heading.

Best Practices

- Ensure headings describe purpose of the section.

- Keep heading text brief but meaningful.

- Match or closely align the <h1> text with the page <title>.

- Provide unique headings for different sections on the same page.

Examples

Bad Example: <div class="bigbold">Our Services</div>

Good Example: <h2>Our Services</h2>

Bad Example: <h1>Contact Us</h1><h3>Mailing Address</h3>

Good Example: <h1>Contact Us</h1><h2>Mailing Address</h2>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.4 (Link Purpose), 3.2.4 (Consistent Identification)

Links

- All links must be semantically coded with <a href="…">.

- Link text must clearly describe the purpose or destination.

- Users should be able to understand a link’s function without needing surrounding context.

- Avoid vague link text such as “click here” or “read more.”

- Links must be visually distinguishable from surrounding text, not just by color but also by underline or other visual cue.

- Links and controls with the same function across pages must be consistently identified.

Special Cases

- Links to external sites may indicate they open outside the current site.

- Links that open a new tab or window should inform the user (e.g., visually or via text “opens in a new window”).

- Links to non-HTML documents (PDF, Word, etc.) should indicate file type and size.

- Links embedded in images must use meaningful alt text that conveys purpose of the link.

Examples

Bad Example: <a href="report.pdf">Click here</a>

Good Example: <a href="report.pdf">2024 Annual Report (PDF)</a>

Bad Example: Image link <a href="home.html"><img src="home.png" alt="house"></a>

Good Example: <a href="home.html"><img src="home.png" alt="Home Page"></a>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.1 (Bypass Blocks), 2.4.5 (Multiple Ways), 3.2.3 (Consistent Navigation)

Navigation Between Pages

- Provide methods to bypass repetitive navigation so users can reach main content quickly.

- Techniques include:

• Skip links — a visible or focus-visible link at the top of the page that jumps to main content.

• Landmarks — use <header>, <nav>, <main>, <footer> or equivalent ARIA roles.

- Skip links should be the first focusable element, visible on keyboard focus if not always visible.

- Consistency matters:

• Repeated navigation menus must appear in the same relative order across pages.

• Navigation links must not change order unexpectedly.

- Provide multiple ways to locate pages within a site (e.g., navigation menus, site map, search feature).

Best Practices

- Clearly indicate current page in navigation menus, both visually and programmatically.

- Use ARIA attributes or visual styling to mark active links.

- Ensure skip links work correctly even when page layouts change with responsive design.

Examples

Bad Example: No skip link, forcing screen reader users to tab through long menus on every page.

Good Example: <a href="#maincontent" class="skip">Skip to main content</a>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.1 (Bypass Blocks), 2.4.3 (Focus Order)

Navigation Within Pages

- Provide mechanisms for users to move efficiently within a single page.

- Common techniques:

• Skip navigation links — allow jumping directly to the main content.

• Table of contents at the top of long pages — reflecting the heading structure.

- Skip links must be keyboard functional and the first or nearly first focusable item on the page.

- Skip links must be visible at all times or visible on keyboard focus.

- Reading and focus order must be logical and intuitive:

• Screen readers follow source code order.

• Keyboard users follow tab/focus order.

• Avoid tabindex values greater than 0, which can create confusing navigation paths.

Paginated Views

- Users should be informed which page/view is currently active.

- Indicate active page both visually and programmatically.

- Example: aria-current="page" on the active pagination link.

Single-Key Shortcuts

- If a single-character keyboard shortcut is used, it must be:

• Turn-off-able,

• Remappable, or

• Only active when the associated component is focused.

- This prevents accidental activation for speech recognition users or those with motor impairments.

Examples

Bad Example: Pagination links with no indication of current page → users cannot orient themselves.

Good Example: <a href="/page2" aria-current="page">2</a>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships)

Tables

- Use semantic markup for tabular data, not for layout purposes.

- Every data table should include:

• <caption> or an accessible name to describe the table’s purpose.

• Properly marked header cells (<th>) for rows and/or columns.

- Data cells (<td>) must be programmatically associated with the correct headers.

Simple Tables

- Use <th scope="col"> and <th scope="row"> to associate headers with cells.

- Keep tables as simple as possible to avoid complex associations.

Complex Tables

- For grouped headers or multi-level associations, use the headers and id attributes.

- Provide a long description if needed for clarity.

- Avoid splitting one logical table into nested or separate tables.

Layout Tables

- Should not be used for purely visual layout.

- If absolutely necessary:

• Use only <td> (no <th>, <caption>).

• Mark with role="presentation" so assistive tech does not interpret as data.

Examples

Bad Example: Using <table> to arrange a page layout → screen readers misinterpret structure.

Good Example: <table><caption>Quarterly Sales</caption><tr><th scope="col">Region</th><th scope="col">Q1</th>…</tr>…</table>

WCAG Mapping

- SC 1.3.1: Info and Relationships.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships)

Lists

- Lists must use semantic HTML elements:

• Ordered lists: <ol>

• Unordered lists: <ul>

• Definition lists: <dl> with <dt> (term) and <dd> (definition)

- Avoid using lists solely for layout or visual indentation.

- Each list item must be contained in <li>, <dt>, or <dd> elements as appropriate.

- Lists help screen readers convey structure, number of items, and relationships between items.

Best Practices

- Use ordered lists (<ol>) when sequence or priority matters.

- Use unordered lists (<ul>) when order is not important.

- Use definition lists (<dl>) for term–definition pairs, not for styling.

- Keep list structures clean and avoid nesting beyond necessity.

Examples

Bad Example: Using <p> with dashes for a bulleted list.

Good Example: <ul><li>First item</li><li>Second item</li></ul>

Bad Example: Using <dl> for navigation menu.

Good Example: <ul><li><a href="home.html">Home</a></li>…</ul>

WCAG Mapping

- SC 1.3.1: Info and Relationships.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.4.2 (Page Titled), 1.3.1 (Info and Relationships)

Iframes

- Iframes are used to embed external content within a page.

- Each iframe must have a non-empty title attribute.

• The title must be accurate, descriptive, and unique within the page.

• The title should summarize the purpose of the embedded content.

- The source page within an iframe must also have a meaningful <title>.

- Hidden or decorative iframes should be hidden from assistive technologies using aria-hidden="true".

Best Practices

- Ensure the heading hierarchy of the iframe content fits logically within the parent page hierarchy, if possible.

- Avoid unlabeled iframes; otherwise, screen readers will announce only “frame” with no context.

- Use iframes sparingly, as they can complicate navigation for screen reader users.

Examples

Bad Example: <iframe src="ads.html"></iframe>

Good Example: <iframe src="ads.html" title="Sponsored advertisement"></iframe>

Bad Example: Multiple iframes titled “widget.”

Good Example: <iframe src="weather.html" title="Weather forecast"></iframe>

<iframe src="news.html" title="Top news headlines"></iframe>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships), 1.3.2 (Meaningful Sequence)

Other Semantic Elements

<strong> and <em>

- <strong> should be used to indicate importance.

- <em> should be used to indicate emphasis.

- Both convey meaning programmatically and visually.

- Do not rely on visual styling alone to convey emphasis.

<blockquote> and <q>

- <blockquote> should be used for extended quotations.

- <q> should be used for inline quotations.

- Avoid using either element purely for styling.

<code> and <pre>

- <code> is used to mark inline code snippets.

- <pre> is used to preserve formatting of blocks of code or preformatted text.

- Both should be used when presenting actual code, not for styling.

<del> and <ins>

- <del> is used to indicate deleted or removed content.

- <ins> is used to indicate inserted or new content.

- Critical changes marked this way should also be described in text for clarity.

<mark>

- <mark> highlights text that is relevant to the user’s current context or search.

- Important highlights should also be explained in accompanying text for non-visual users.

Parsing and Validity

- All elements must have complete start and end tags.

- IDs must be unique within a page.

- Parent-child element relationships must be valid according to HTML specification.

- Deprecated markup should not be used.

Examples

Bad Example: Using <blockquote> for indentation.

Good Example: <blockquote cite="author">Extended quotation here</blockquote>

Bad Example: <span class="bold">Important</span>

Good Example: <strong>Important</strong>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 4.1.1 (Parsing), 4.1.2 (Name, Role, Value)

Parsing and Validity

- Markup must be valid and follow the rules of the language specification.

- Elements must have complete start and end tags.

- Elements must be nested according to specification (no overlapping tags).

- IDs must be unique across the page.

- Names of landmarks, tables, iframes, and other block elements should be unique when multiple instances exist.

- Deprecated markup (e.g., <font>, <center>) should not be used.

- Content should be free of duplicate attributes, improperly closed elements, or structural errors.

Best Practices

- Run automated validators (e.g., W3C HTML Validator) to check parsing and structural validity.

- Ensure attributes are properly closed and quoted.

- Use modern, standards-based HTML and CSS rather than deprecated tags or proprietary features.

- Verify ARIA attributes are valid and used according to WAI-ARIA specification.

- Ensure every interactive element exposes its accessible name, role, and value correctly.

Examples

Bad Example: <div id="nav"></div><div id="nav"></div> → duplicate IDs.

Good Example: <div id="mainnav"></div><div id="footernav"></div>

Bad Example: <font color="red">Warning</font> → deprecated markup.

Good Example: <span class="warning">Warning</span> styled with CSS.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.1.1 (Non-text Content)

Images, Canvas, SVG, and Other Non-Text Content

General Rule

- All non-text content must be represented by text alternatives so it can be conveyed through assistive technologies.

Image Alt Text

- Informative images:

• Must have programmatically determinable alternative text.

• Alt text must be meaningful and concise (≤150 characters).

• Should not include “image of” or “graphic of.”

- Decorative or redundant images:

• Use null alt text (alt=""), role="presentation", or CSS background.

- Actionable images (links, buttons, controls):

• Alt text must describe purpose or function (e.g., “Search” not “Magnifying glass”).

- Form inputs type="image":

• Must have alt text describing the button’s function (e.g., “Submit”).

- Animated images:

• Must provide a way to pause/stop if they auto-play more than 5 seconds.

• Must not flash more than 3 times per second.

- Complex images:

• Require short alt text plus a long description (on-page, expandable, or linked).

- Images of text:

• Should not be used unless essential (logos, brand marks).

• Real text is preferred for scalability and customization.

SVG (Scalable Vector Graphics)

- Assign role="img" to SVG elements.

- Provide alternative text via <title>, aria-label, or aria-labelledby.

- Complex SVGs require long descriptions.

- Decorative SVGs should be hidden with aria-hidden="true".

- Ensure color contrast of at least 4.5:1 (small text) or 3:1 (large/bold text).

- Avoid embedding SVG via <object> or <iframe>.

Canvas

- Content inside <canvas> is not accessible by default.

- Assign role="img" and provide alt text with aria-label, aria-labelledby, or fallback text.

- For complex graphics, associate a long description (aria-describedby).

- Ensure a background fill for high-contrast support.

- Do not put text inside canvas; use SVG if text is needed.

- Custom canvas controls must be keyboard accessible and expose name/role/value.

Other Non-Text Media

- Objects: All <object> elements must have alt text via aria-label or aria-labelledby.

- Silverlight/Flash/Java: Deprecated; if unavoidable, must expose accessibility APIs and follow same principles.

- Documents (Word, PDF, EPUB):

• Must follow accessibility guidelines (titles, headings, alt text, table headers, contrast, fonts).

• PDFs must be tagged PDFs.

• EPUB should conform to EPUB Accessibility specification (EPUB 3 preferred).

• Provide HTML alternative when possible.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.4.x (Distinguishable)

Visual Design and Colors

General Principles

- Information must not be conveyed by color alone.

- Visual content must have sufficient contrast to be perceivable by users with low vision or color blindness.

- Designs should accommodate multiple viewport sizes (desktop, tablet, mobile) and zoom levels.

Color Use

- Any information conveyed by color must also be available via text or another indicator.

- Link text must be distinguishable from surrounding text by more than color alone.

- Interactive elements should have clear visual focus indicators beyond color.

- Actionable elements (buttons, controls) must have a minimum contrast ratio of 3:1 against adjacent backgrounds.

Contrast

- Small text (under 18pt regular or 14pt bold) requires at least 4.5:1 contrast ratio.

- Large text (18pt regular or 14pt bold and larger) requires at least 3:1 contrast ratio.

- Focus indicators, UI boundaries, and essential graphics must meet 3:1 contrast ratio minimum.

- Higher contrast ratios (7:1 for normal text, 4.5:1 for large text) are recommended for enhanced readability.

Layout and Typography

- Blocks of content should be visually separated with spacing, margins, or padding.

- Labels should be visually adjacent to their form controls.

- Maintain a single primary visual focus in layout.

- Font and text styling should support readability:

• Avoid full justification (ragged right is easier to read).

• Limit line length to ~80 characters (40 for CJK languages).

• Line spacing at least 1.5x within paragraphs.

• Paragraph spacing at least 1.5x line spacing.

• Fonts should be user-customizable.

CSS-Generated and Hidden Content

- CSS-generated content should not convey essential information unless paired with a text alternative.

- Decorative CSS content should be hidden from assistive technologies with aria-hidden="true".

- Hidden or inactive content must not be read by screen readers until it is visible and active.

- Hover/focus-triggered additional content must be dismissible, hoverable, and persistent.

Examples

Bad Example: Using only red text to indicate an error.

Good Example: Error messages in red text plus an error icon and text label.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.4.4 (Resize Text), 1.4.10 (Reflow), 1.4.12 (Text Spacing), 1.3.4 (Orientation)

Responsive Design and Zoom

General Principles

- Responsive design benefits both mobile users and users with low vision who zoom in on content.

- When screen real estate is limited, content should adapt and simplify rather than requiring horizontal scrolling.

Responsive Design

- Forms, images, tables, objects, UI components, and text should reflow to fit the viewport without causing horizontal scrolling.

- Features of the content may be simplified or eliminated at small viewports to preserve usability.

- Navigation menus should adapt to single-column or simplified structures for small screens and zoom states.

Zoom

- Text must remain functional and readable when magnified up to 200%.

- Spacing between letters, words, lines, and paragraphs must be adjustable without loss of content or functionality (SC 1.4.12).

- Content must not require both horizontal and vertical scrolling at 320px width (for vertical scrolling content) or 256px height (for horizontal scrolling content), unless essential.

- Users must be able to zoom in and out on mobile devices.

Magnification Quality

- Text should resize without blurring or distortion.

- Icons and graphics should scale cleanly.

- Layout must remain usable at various zoom levels.

Orientation

- Content must not lock to portrait or landscape orientation unless essential (e.g., check scanning apps, games requiring orientation).

Target Size

- Click/touch targets should be large enough (≥44px by 44px) and spaced apart so they can be used easily with a finger.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.2.x (Time-based Media), 1.4.2 (Audio Control), 2.2.2 (Pause, Stop, Hide), 2.3.1 (Three Flashes)

Multimedia, Animations, and Motion

Accessible Multimedia Methods

- Captions: synchronized text for dialogue and important sounds.

- Transcripts: text versions of speech and visual information, essential for deafblind users.

- Audio descriptions: narration of important visual details not present in audio.

- Sign language interpretation: visual translation for users who know sign language.

Additional Considerations

- Clear audio is necessary; background sounds should not obscure speech.

- Avoid flashing content (no more than 3 flashes per second).

- Minimize motion that can cause dizziness or disorientation for people with vestibular disorders.

- Media players must be keyboard accessible and expose names, roles, values of controls.

- Auto-play must not interfere with screen readers and must provide pause/stop/mute.

Captions and Transcripts

- Prerecorded multimedia must include captions.

- Live multimedia with dialogue must include real-time captions.

- Audio-only content must have transcripts.

- Video-only content must have audio descriptions or text transcripts.

- Captions and transcripts should identify speakers, capture important background sounds, and remain synchronized.

Audio Descriptions

- Required for prerecorded multimedia with important visual information.

- Extended audio descriptions should be provided if standard descriptions cannot cover all necessary detail.

Sign Language

- May be provided for prerecorded or live multimedia.

- Useful but not a substitute for captions.

Animations and Motion

- Parallax effects should be minimized.

- Background videos/animations must not convey essential content unless fully controllable by the user.

- Auto-playing background media must stop after 5 seconds or have controls to pause/stop.

- Text and essential graphics over moving backgrounds must maintain minimum contrast ratios (4.5:1 for small text, 3:1 for large/bold text).

- Interaction-triggered animations (e.g., hover effects) must not trap focus and should not distract from primary tasks.

Flashing Content

- Content must not flash more than 3 times per second.

- Exceptions only if flashes are small, low-contrast, and below general and red flash thresholds.

Best Practices

- Provide users with clear media controls.

- Avoid motion that causes distraction or health risks.

- Keep visual design usable in reduced-motion environments (respect prefers-reduced-motion CSS setting).

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 2.1.1 (Keyboard), 2.5.x (Pointer Gestures, Target Size, Input Modalities)

Device-Independent User Input Methods

General Principles

- All functionality must be operable using multiple input methods (keyboard, mouse, touch, assistive technologies).

- Do not rely on a single type of input for critical tasks.

- Ensure input methods remain available and usable even when assistive technologies are active (e.g., screen readers on mobile overriding swipe gestures).

Keyboard Access

- Every interactive element must be focusable and operable with the keyboard alone.

- Focus order must be logical and intuitive.

- Focus must always be visible.

- Avoid keyboard traps: users must be able to move focus away from any element using standard keys.

- Provide proper keyboard event handling (Enter/Space in addition to mouse clicks).

Touch Access

- Touch targets must be large enough (≥44x44px recommended).

- Provide sufficient spacing between adjacent interactive elements to avoid accidental activation.

- Custom gestures (swipe, pinch) must have alternative methods; gestures may not work when a screen reader is active.

- All functionality must be available with simple taps/clicks.

Mouse and Pointer Access

- Pointer interactions must mirror keyboard equivalents.

- Hover styles should match keyboard focus styles.

- Do not require precision pointing for essential tasks.

Alternative Inputs

- Voice input, switch devices, and screen readers must be able to operate all controls.

- Single-key shortcuts must be turn-off-able, remappable, or limited to when the relevant control is focused.

Best Practices

- Always support keyboard as a baseline input method.

- Ensure parity across keyboard, mouse, and touch input.

- Respect user preferences for reduced motion or alternate interaction modes.

Examples

Bad Example: Custom slider operable only by drag-and-drop mouse interaction.

Good Example: Slider with draggable handle, arrow key support, and input box for manual value entry.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 1.3.1 (Info and Relationships), 3.3.2 (Labels or Instructions), 3.3.3 (Error Suggestion), 3.3.4 (Error Prevention)

Form Labels, Instructions, and Validation

General Principles

- All form controls must have programmatically associated labels.

- Labels and instructions must be visible and adjacent to their form controls.

- Label text must be descriptive, concise, and explain purpose of the field.

- Instructions for completing forms must be clear, available before submission, and persistent.

Labels

- Use <label> elements explicitly associated with inputs via the for attribute.

- Group related fields using <fieldset> and <legend>.

- Place labels close to fields for easy association (especially important for screen magnifier users).

- Do not rely solely on placeholder text as a label.

Instructions

- Provide instructions in text, not just visually (e.g., “Required fields are marked with \*”).

- Include constraints, formats, or requirements (e.g., “Password must be at least 8 characters”).

Error Messages

- Must identify which field has an error and describe the problem.

- Error text must be programmatically associated with the field (e.g., aria-describedby).

- Errors should be announced to screen reader users when they occur.

Validation

- Provide real-time validation feedback where possible, in accessible form.

- Errors triggered by moving focus must be spoken by screen readers.

- Provide suggestions for correcting errors (e.g., “Enter email in format name@example.com”).

Best Practices

- Support Autofill where appropriate.

- Ensure error prevention on critical transactions (e.g., confirmation step for purchases).

- Design error messages that are visible, perceivable, and specific.

Examples

Bad Example: <input type="text" placeholder="Name">

Good Example: <label for="fullname">Full Name</label><input id="fullname" type="text">

Bad Example: Error only indicated by red border.

Good Example: <span id="email-error">Email address is required</span>

<input type="email" aria-describedby="email-error">

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 4.1.3 (Status Messages), 2.4.3 (Focus Order), 2.1.1 (Keyboard)

Dynamic Updates, AJAX, and Single-Page Applications

General Principles

- Modern web apps often update content dynamically without full page reloads.

- These updates must remain perceivable and operable for users of assistive technologies.

Dynamic Content

- When new content is inserted into the DOM, ensure it appears in a logical reading order.

- Use ARIA live regions to announce updates that are important but not focus-triggered.

- Avoid sudden content shifts that disorient users.

Focus Management

- After dynamic actions (form submission, modal dialog opening, AJAX updates), move focus appropriately.

- Ensure that focus does not disappear or land illogically.

- When dismissing dialogs or overlays, return focus to the triggering element.

- Avoid focus traps inside modals or dynamic components.

Single-Page Applications (SPAs)

- Ensure screen readers are informed when views change (update page title, use ARIA landmarks).

- Manage keyboard focus across view changes.

- Use ARIA roles, states, and properties to reflect component state (expanded, collapsed, selected, etc.).

- Provide clear feedback for success/failure of user actions (alerts, status messages).

Best Practices

- Keep updates minimal and predictable.

- Provide visible and programmatic indicators when content changes.

- Ensure forms and widgets give immediate, accessible feedback after submission or interaction.

Examples

Bad Example: AJAX form updates silently, leaving screen reader users unaware of the result.

Good Example: <div role="status" aria-live="polite">Form submitted successfully</div>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → WCAG 2.2 Success Criterion 4.1.3 (Status Messages), 2.4.3 (Focus Order), 2.1.1 (Keyboard)

Dynamic Updates, AJAX, and Single-Page Applications

General Principles

- Modern web apps often update content dynamically without full page reloads.

- These updates must remain perceivable and operable for users of assistive technologies.

Dynamic Content

- When new content is inserted into the DOM, ensure it appears in a logical reading order.

- Use ARIA live regions to announce updates that are important but not focus-triggered.

- Avoid sudden content shifts that disorient users.

Focus Management

- After dynamic actions (form submission, modal dialog opening, AJAX updates), move focus appropriately.

- Ensure that focus does not disappear or land illogically.

- When dismissing dialogs or overlays, return focus to the triggering element.

- Avoid focus traps inside modals or dynamic components.

Single-Page Applications (SPAs)

- Ensure screen readers are informed when views change (update page title, use ARIA landmarks).

- Manage keyboard focus across view changes.

- Use ARIA roles, states, and properties to reflect component state (expanded, collapsed, selected, etc.).

- Provide clear feedback for success/failure of user actions (alerts, status messages).

Best Practices

- Keep updates minimal and predictable.

- Provide visible and programmatic indicators when content changes.

- Ensure forms and widgets give immediate, accessible feedback after submission or interaction.

Examples

Bad Example: AJAX form updates silently, leaving screen reader users unaware of the result.

Good Example: <div role="status" aria-live="polite">Form submitted successfully</div>

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Accessibility in the Design and Development Lifecycle

Design Considerations

- Accessibility should be integrated at the earliest stages of design, not added later.

- Designers, developers, and QA must collaborate to ensure consistent application of accessibility principles.

Key Design Factors

- Page Titles: must be descriptive and meaningful.

- Headings: should create a logical hierarchy and reflect document structure.

- Navigation: must provide skip mechanisms and consistent order across pages.

- Links: link text must clearly indicate purpose.

- Color and Contrast: visual information must not rely solely on color; maintain WCAG minimum contrast ratios.

- Responsive Layout: support reflow and zoom without loss of content or functionality.

- Forms: labels, instructions, and error feedback must be visible, programmatically associated, and descriptive.

- Multimedia: captions, transcripts, and audio descriptions must be provided.

- Dynamic Content: updates must be perceivable and focus managed properly.

- Controls: favor standard HTML controls when possible; custom controls must follow WAI-ARIA best practices.

Inclusive Design Practices

- Design for users with varied disabilities, including vision, hearing, motor, and cognitive impairments.

- Consider user strategies and coping mechanisms (keyboard vs. mouse, screen reader navigation, zoom use).

- Provide redundant cues (text, icons, sound) so no single sensory channel is required.

- Anticipate assistive technology use (screen readers, magnifiers, voice control).

Best Practices

- Document accessibility requirements early in project specifications.

- Use role-based accessibility checklists (for designers, developers, QA).

- Apply automated and manual testing throughout development, not just at the end.

- Iterate design choices based on feedback from users with disabilities.

Examples

Bad Example: Designing color-only indicators for errors without contrast testing.

Good Example: Designing error states that combine color, icon, text description, and ARIA alerts.

Domain II: Identifying Accessibility Issues in Web Solutions

Subsection: Accessibility Quality Assurance → Scope of Testing

The Scope of the Test

General Principles

- Accessibility testing must cover more than isolated components; scope should include full pages, templates, and widgets.

- Defining scope early ensures critical issues are found before launch.

Entire Pages and Templates

- Test complete pages, not just fragments.

- Include templates to ensure consistent accessibility across repeated layouts.

- Evaluate page structure, navigation, landmarks, headings, and reading/focus order.

- Confirm all required elements are present in every template (skip links, page title, ARIA landmarks).

Page Content and Widgets

- Test interactive content such as menus, dialogs, tabs, carousels, and custom controls.

- Confirm widgets provide programmatically determinable roles, names, and states.

- Ensure widgets are keyboard accessible and usable with screen readers.

- Verify that widgets integrate into the page’s overall focus and reading order.

Best Practices

- Scope testing to include static content, interactive controls, multimedia, forms, and dynamic updates.

- Consider both visual and non-visual rendering of content.

- Define boundaries: testing should cover common templates, key workflows, and representative samples of large content sets.

Examples

Bad Example: Testing only a homepage banner image for alt text.

Good Example: Testing homepage template, banner, navigation menus, forms, and all interactive widgets.

Domain II: Identifying Accessibility Issues in Web Solutions

Subsection: Testing with Assistive Technologies and Tools → Basic Testing Routine

A Basic Testing Routine

Purpose

- Provides a repeatable method for identifying common accessibility issues.

- Combines automated tools with manual checks for comprehensive coverage.

Step 1: Run Automated Checks

- Use tools like axe DevTools or browser-based validators.

- Automated checks catch common issues (missing alt text, low contrast, form label absence).

- Limitations: automation detects only ~30% of WCAG issues; manual testing remains necessary.

Step 2: Screen Reader Checks

- Use screen readers (NVDA, JAWS, VoiceOver, TalkBack).

- Confirm logical reading order, headings, landmark navigation, link clarity, and form labeling.

- Test custom widgets and dynamic updates with a screen reader active.

Step 3: Keyboard Accessibility

- Navigate entire site using only keyboard (Tab, Shift+Tab, arrow keys, Enter, Space).

- Confirm all functionality is operable without a mouse.

- Ensure visible focus indicators, logical focus order, and no keyboard traps.

- Test with screen reader active to ensure compatibility.

Step 4: Touch Device Accessibility

- Test mobile/touch interactions, ensuring touch targets are large and spaced.

- Confirm that functionality works both with and without screen reader enabled.

- Ensure custom gestures have alternative controls.

Step 5: Check Color and Contrast

- Verify that information is not conveyed by color alone.

- Test contrast ratios (4.5:1 small text, 3:1 large/bold).

- Check link visibility and distinguishability.

Step 6: Test Alt Text

- Ensure alt text is meaningful, concise, and context-appropriate.

- Confirm decorative images are hidden from assistive technologies.

Step 7: Test Multimedia

- Check captions, transcripts, and audio descriptions.

- Verify auto-play has pause/stop controls.

Step 8: Test Landmarks, Headings, and Links

- Confirm landmarks are present and labeled.

- Check heading hierarchy for logic and clarity.

- Ensure link text is descriptive.

Step 9: Test Forms

- Labels must be visible and programmatically associated.

- Error handling must be clear, perceivable, and specific.

- Validation feedback must be accessible.

Step 10: Test Dynamic Content

- Ensure AJAX and SPA updates are announced and focus managed.

- Verify status messages are read by screen readers.

Best Practices

- Combine automated and manual testing.

- Follow a consistent routine across projects.

- Involve users with disabilities when possible for real-world validation.

Domain III: Remediating Accessibility Issues in Web Solutions

Subsection: Severity, Prioritization, and Remediation → Bug/Issue Management

Bug/Issue Management

Purpose

- Accessibility findings must be documented clearly and tracked to resolution.

- Good issue management ensures fixes are understood, prioritized, and implemented efficiently.

Writing Effective Accessibility Bug Tickets

- Clearly describe the issue in plain language.

- Include steps to reproduce.

- Identify the specific WCAG success criteria violated.

- Describe the user impact (who is affected, how severe the barrier is).

- Provide screenshots, code snippets, or recordings when helpful.

- Suggest remediation guidance or link to relevant documentation.

Prioritizing Accessibility Bugs

- Consider multiple factors:

• Severity: how much the issue blocks access for users with disabilities.

• Frequency: how often the issue occurs across the site or workflow.

• Legal Risk: whether the issue represents non-conformance with required standards.

• Effort: feasibility of remediation vs. alternatives.

- Critical issues (blocking users from completing core tasks) should be resolved first.

- Cosmetic or low-impact issues can be scheduled after critical fixes.

Bug Tracking

- Use standard issue-tracking tools (Jira, GitHub Issues, etc.).

- Ensure accessibility bugs are tagged consistently.

- Assign ownership to specific roles or teams.

- Track remediation progress and verify fixes with retesting.

Best Practices

- Write reports so even those with little accessibility background can act on them.

- Communicate findings constructively, focusing on user impact.

- Retest after fixes are implemented.

- Integrate accessibility defects into the same workflow as other bugs (not a separate silo).

Examples

Bad Example: “Page is not accessible.”

Good Example: “Login form: Password field has no programmatically associated label. Violates WCAG 1.3.1. Screen reader users cannot identify the field’s purpose. Add <label for='password'>Password</label> to fix.”

Domain II: Identifying Accessibility Issues in Web Solutions

Subsection: Testing Tools (Automated and Manual) → Automated Testing Tools

Automated Testing Tools

Purpose

- Automated tools provide fast initial checks for accessibility issues.

- They are not comprehensive and must be supplemented with manual testing.

Common Tools

- axe DevTools (Deque): browser extension for automated WCAG testing.

- WAVE (WebAIM): browser extension and API for detecting common issues.

- Lighthouse (Google): automated performance and accessibility analysis.

- Pa11y: open-source accessibility testing tool.

- Tenon, Siteimprove, and other enterprise solutions for ongoing monitoring.

Types of Automated Testing

- In-browser extensions: quick checks during development.

- Unit testing integrations: test accessibility in component/unit test frameworks.

- Spider or crawler tools: scan large sites for recurring issues.

- Continuous monitoring: scheduled scans to detect regressions after updates.

Strengths

- Quickly finds missing alt attributes, low contrast, missing form labels, invalid ARIA, and structural errors.

- Scales across many pages efficiently.

- Useful for regression testing and baseline conformance.

Limitations

- Cannot determine whether alt text is meaningful.

- Cannot verify correct heading hierarchy or logical focus order.

- Cannot evaluate usability or user experience barriers.

- Typically detects only 20–30% of possible WCAG failures.

Best Practices

- Integrate automated checks into development and QA workflows.

- Train developers to run automated scans early in coding process.

- Treat automated results as a starting point, not full coverage.

- Always combine with manual testing for a complete evaluation.

Domain I–III: Creating, Identifying, and Remediating Accessible Web Solutions

Subsection: Summary

Summary

- Accessibility must be integrated across the entire development lifecycle, from design through testing and remediation.

- Non-text content requires meaningful text alternatives; complex graphics and media need extended descriptions, captions, transcripts, or audio descriptions.

- Semantic structure (titles, headings, landmarks, lists, tables, iframes) ensures content is programmatically determinable and navigable.

- Visual design must maintain sufficient color contrast, avoid relying on color alone, and adapt responsively to zoom and different devices.

- Multimedia and motion content must provide captions, transcripts, audio descriptions, and controls to pause or stop playback. Flashing and excessive motion must be avoided.

- All functionality must be operable via keyboard, mouse, touch, and alternative inputs. No single input method should be required.

- Forms must have visible, programmatically associated labels, clear instructions, and accessible error handling.

- Dynamic content and single-page applications must announce updates, manage focus properly, and avoid disorienting changes.

- Testing scope should cover entire pages, templates, and widgets, not just isolated elements.

- A basic accessibility testing routine combines automated tools with manual checks for screen readers, keyboard use, touch, color contrast, alt text, multimedia, forms, and dynamic updates.

- Bug/issue management requires clear documentation, WCAG mapping, user impact description, prioritization, and integration into normal defect tracking systems.

- Automated tools are valuable for catching common issues quickly, but manual testing is essential for full coverage.

Key Takeaway

- Accessibility is a shared responsibility across roles and phases.

- Effective accessibility programs combine standards compliance, inclusive design, testing strategies, and remediation workflows.

- The goal is not only technical conformance but creating usable, inclusive experiences for people with disabilities.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Non-Text Content and Alternative Text Principles

- Every non-text element requires a meaningful text alternative.

- Blind users rely on screen readers to read alt text; deafblind users access it through braille displays.

- The technique for providing alternatives depends on the format (HTML <img>, SVG markup, HTML5 <canvas>, or other), but the principle is the same: provide an equivalent text substitute.

- Other content also requires alternatives:

• Video requires captions, transcript, and may require audio descriptions.

• Audio requires at least a transcript.

• Plug-ins and non-HTML documents (Word, PDF, PowerPoint, EPUB) must use accessibility APIs.

- HTML versions of non-HTML content are often the most broadly accessible across devices.

Course framing (Deque curriculum context):

- Intended audience: developers, QA testers.

- Level: advanced (HTML, CSS, JavaScript knowledge recommended).

- Prerequisites: Accessibility fundamentals, UX design, semantic structure/navigation.

- Guidelines reference: WCAG 2.2 Level AA.

- Terminology:

• MUST = required.

• SHOULD = strongly recommended.

• MAY = optional.

- IAAP certification relevance: aligned with WAS exam; course offers 1.5 CAECs.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Image Alternative Text (Alt Text Basics)

- Good alternative text must be:

• Programmatically determinable (readable by assistive technologies).

• Meaningful (accurately conveys the author’s intent/purpose).

• Concise (about 150 characters or less).

- If an image is purely decorative or redundant:

• Hide it from assistive technologies (alt="", role="presentation", or CSS background).

• Note: Office documents (Word, PowerPoint) lack a decorative image flag, so every image requires alternative text there.

- Complex or information-rich images may also need a supplementary long description.

- Goal: provide an equivalent experience for users who are blind, without overwhelming them with unnecessary detail.

Accessible Name Calculation for <img>:

- Order of precedence:

1. aria-labelledby (overrides all).

2. aria-label (if aria-labelledby is absent).

3. alt attribute (standard method).

4. title attribute (read only if no other method is present; support varies across screen readers).

- Screen readers may handle the title attribute differently:

• Some (VoiceOver) treat it as an additional description alongside other text.

• Others (NVDA, JAWS) only read title if no accessible name exists.

Bad practices:

- Redundant phrasing like "image of" or "graphic of" (screen readers already announce the element type).

- Overly long or literal alt text that overwhelms users.

Good practices:

- Tailor alt text to the image’s purpose, not just its visual details.

- Provide long descriptions separately when necessary.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Informative Images

- Informative images convey content and therefore MUST have programmatically determinable alt text.

- Alt text must:

• Be meaningful (communicate the author’s intent and purpose of the image).

• Be concise (about 150 characters).

• Avoid redundant identifiers like “image of” or “graphic of.”

- Without alt text, screen readers may announce the file name, which is usually unhelpful.

Authoring approach:

- Ask: Why is this non-text content here? What information is it conveying? What purpose does it serve?

- Alt text must substitute for the image’s purpose, not just its literal content.

Good Example: Informative image with descriptive alt text

<img src="sewing\_machine\_singer.jpg" alt="A Singer model antique sewing machine">

(Screen reader announces: “Graphic, A Singer model antique sewing machine.”)

Bad Example: No alt text

<img src="sewmach2.jpg">

(Screen reader announces the file name, e.g., “graphic sewmach2 dot J P G.”)

Context-specific alt text:

- A sunset photo used in a photography guide may need alt text like: “Camera set to expose the sunset, underexposing the land and seagull silhouette.”

- Logos: alt should identify the brand (e.g., alt="Deque University"). Including the word "logo" may be optional depending on context.

Bad practices:

- Vague alt text (“An ink stamp”) that omits key details.

- Empty alt text on informative images (causes the image to be ignored).

- Redundant phrasing like “photo of” in alt text.

Conciseness rule:

- While alt text has no strict character limit, practical guidance is ~150 characters.

- Avoid excessive descriptions; use long description techniques when more detail is required.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Decorative or Redundant Images

- Images that do not convey content, are decorative, or duplicate information already provided in text MUST be hidden from assistive technologies.

- Techniques:

• Use null alt attribute: alt=""

• Use ARIA role="presentation"

• Implement as CSS background images

- Do not omit the alt attribute entirely. If left off, screen readers often announce the file name, which is confusing and unhelpful.

Good Example: Redundant image with null alt text

<a href="https://dequeuniversity.com">

<img src="home-icon.png" alt="">

Home Page

</a>

(Screen reader reads only “link Home Page.”)

Bad Example: Redundant image with duplicate alt text

<a href="https://dequeuniversity.com">

<img src="home-icon.png" alt="Home Page">

Home Page

</a>

(Screen reader reads “link Home Page Home Page.”)

Rule of thumb:

- Do not duplicate adjacent text in the image alt attribute.

- Decorative images should not waste users’ time by being announced when they add no new information.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Actionable Images (Links, Buttons, Controls)

- Actionable images (used as links, buttons, or controls) MUST have meaningful alt text.

- Alt text must describe the purpose, destination, or action — not the image’s literal appearance.

Examples:

- Good: <img src="search\_icon.png" alt="Search"> (describes the action of the button).

- Bad: <img src="search\_icon.png" alt="Magnifying glass"> (describes the graphic, not the function).

Guidelines:

- Avoid phrases like “image of” or “link to” in alt text.

• Screen readers already announce the element type (e.g., “link,” “graphic”).

- Alt text for actionable images is usually short (often one or two words).

- For form input images (type="image"), alt text MUST indicate the input’s purpose.

• Example: <input type="image" src="submit.png" alt="Submit">

Conciseness:

- Alt text for actionable elements should be as brief as possible, while still accurately communicating the action.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Form Inputs type="image"

- Form inputs that use type="image" MUST have alt text.

- Alt text must clearly convey the purpose or result of the action.

- Keep it concise — often a single word is sufficient.

Good Example:

<input type="image" src="submit.png" alt="Submit">

(Screen reader announces “Submit button.”)

Bad Example:

<input type="image" src="submit.png">

(Screen reader may announce only “button,” or the file name, leaving users without purpose context.)

Notes:

- Avoid decorative or vague descriptions. Always tie alt text to the function (e.g., “Search,” “Next,” “Login”).

- Remember that screen readers already announce the control type (button), so only the meaningful label is needed in the alt text.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Animated Images

- Animated images must be handled carefully to avoid accessibility barriers.

- Requirements:

• Provide a method to pause, stop, or hide any prerecorded video-only content that auto-plays for more than 5 seconds.

• Animated images MUST NOT flash or flicker more than 3 times per second (to prevent seizures in users with photosensitive epilepsy).

- Overly distracting animations can interfere with both users with disabilities and those without.

Good Example:

- An animated banner that loops for 3 seconds, then stops, with a visible “Pause” button available.

Bad Example:

- An endlessly looping GIF with no pause or stop mechanism.

Notes:

- If animation is needed, ensure it supports the content rather than distracting from it.

- Consider whether SVG or CSS animations might offer more accessible control compared to GIFs or other legacy formats.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Complex Images

- Complex images (e.g., diagrams, charts, paintings, maps) often require more than simple alt text.

- Best practice:

• Provide short, meaningful alt text (≤150 characters).

• Supplement with a long description to convey full detail.

Options for long descriptions:

- Place description in surrounding page content.

- Provide a link to a separate page with the description.

- Use a button to open a dialog or expand/collapse the description.

- Use aria-describedby to associate the image with an extended description elsewhere.

Author’s intent matters:

- Literal vs symbolic purpose should guide what is described.

- Example: A historical painting may require context about symbolism, not just visual elements.

- Example: A scientific diagram may need precise, literal detail for interpretation.

Bad practice:

- Using excessively long alt text inside the alt attribute.

- Omitting critical descriptive content needed to understand the image’s purpose.

Rule:

- Always provide alt text, and when the image conveys more than alt text can reasonably cover, add a structured long description.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Images of Text

- Real text should be used instead of images of text whenever possible.

- Problems with images of text:

• Do not scale well and can become blurry under magnification.

• Cannot be customized for font, size, color, or background.

- Exceptions:

• Essential cases (e.g., logos).

• If text is fully customizable (e.g., using SVG).

Requirements:

- Informative text must not be embedded in images unless customization is possible or essential.

- Decorative or branding-related text (e.g., logos) may remain as images but still require alt text that conveys meaning.

Good Example:

- Logo with alt="Deque University" (conveys brand identity).

Bad Example:

- Promotional banner with important event details as an image, without text alternative. Users relying on screen readers or zoom cannot access it.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → CSS Background Images

- CSS background images are not directly announced to screen readers and cannot have alt attributes.

- Accessibility approaches:

• Decorative/redundant CSS images: do not provide a text alternative (let them be ignored).

• Informative or actionable CSS images: provide a programmatically determinable text alternative in the HTML (e.g., aria-label, visible text).

- Avoid relying on CSS background images for important content or controls, since they are harder to make accessible.

Bad Example:

- Using a CSS background image of a “Search” icon as the only visual indicator for a button, without any accessible label. Screen readers provide no information.

Good Example:

- Using a CSS background image purely for decoration, with the actionable text label “Search” present in the HTML.

Notes:

- Do not use visibility:hidden or display:none for accessible alternatives, since these hide content from all users (including assistive tech).

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Image Maps

- Client-side image maps require two layers of alternative text:

• The img element must have programmatically determinable alt that meaningfully describes the image map as a whole.

• Each actionable area element must have programmatically determinable alt that meaningfully describes its destination or action.

- Conciseness guideline: keep each alt to about 150 characters where feasible.

- Prefer client-side image maps; avoid server-side image maps when equivalent functionality is possible with client-side techniques.

Good Example: Client-side map with alt on img and each area

<img src="solar\_system.jpg" alt="Solar System" usemap="#Map1">

<map name="Map1">

<area shape="rect" href=".../Mercury" alt="Mercury">

<area shape="rect" href=".../Venus" alt="Venus">

<area shape="rect" href=".../Earth" alt="Earth">

</map>

Bad Example: Missing alt on areas (actions unlabeled)

<img src="campus\_map.jpg" alt="Campus map with building links" usemap="#campus">

<map name="campus">

<area shape="rect" href=".../library">

<area shape="rect" href=".../lab">

</map>

Authoring notes:

- The img alt should describe the overall graphic or purpose of the map (for example, “Solar system diagram with links to each planet”).

- Each area alt should describe the specific target or action (for example, “Library” or “Admissions office”).

- Do not include redundant words like “link to” or “image of”; assistive technologies already announce roles.

- Validate that keyboard focus is visible for each area and that the tab order is logical.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Advanced Alt Text and Extended Descriptions

- Simple alt text is sufficient for most images, but some require extended description.

- Complex or symbolic images (e.g., historical paintings, diagrams, maps) often exceed the 150-character best practice limit for alt text.

- Strategy:

• Provide concise alt text (≤150 characters) to identify the image.

• Add a long description to fully convey intent, symbolism, or detailed information.

Techniques for long descriptions:

- Place the description in page content near the image.

- Use a link to a separate page with a long description.

- Provide a button that opens or expands a description.

- Associate description with the image via aria-describedby.

Examples:

1. Illustrative painting (“American Progress,” 1872 by John Gast):

• Short alt: “Painting titled American Progress, 1872.”

• Long description: Explain symbolism (westward expansion, settlers, Native Americans retreating, cultural meaning).

2. Ecology diagram (five stages of forest regrowth):

• Short alt: “Diagram of 5 stages of forest regrowth, described below.”

• Long description: Step-by-step explanation of each stage (grasses → annual plants → shrubs → oak trees → mature forest).

3. Map with directions:

• Short alt: “Walking directions from Starbucks to Insomnia Cookies, details below.”

• Long description: Text-based navigation with street names and distances.

Guiding principles:

- Always ask: What is the author’s purpose? Is the image literal, symbolic, instructional?

- Provide enough detail for a blind or low-vision user to fully understand the intended meaning.

- Do not overload alt attributes with long text — use structured long descriptions instead.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → SVG (Scalable Vector Graphics)

- SVG = XML-based vector graphics that scale without losing clarity.

- Accessibility benefits:

• Lossless scaling (important for users with low vision using magnification).

• Colors and styles can be customized via CSS or user settings.

• Accessibility features can be embedded directly in markup.

Accessible authoring patterns:

1. SVG as <img src>

- Use for simple graphics.

- Best practice:

• <img src="graphic.svg" role="img" alt="Meaningful description">

• Always include role="img" due to VoiceOver bug.

• Alternative text should be concise and meaningful.

2. Inline SVGs

- Use when more complex graphics require CSS/JavaScript manipulation.

- Best practice:

• <svg role="img" aria-labelledby="titleID descID">

<title id="titleID">Short meaningful title</title>

<desc id="descID">More detailed description</desc>

</svg>

• <title> provides short alternative text.

• <desc> provides extended description.

• aria-labelledby ensures programmatic association.

3. Decorative SVGs

- Hide from assistive tech:

• <svg aria-hidden="true">

4. Embedded SVGs

- Avoid <object> or <iframe> for SVG embedding (screen reader compatibility issues).

- Prefer <img src> or inline SVG.

Complex SVGs

- Provide both short alt and long description.

- Options for long description:

• Visible description in page content.

• Expand/collapse region or modal dialog.

• aria-describedby linking to extended description.

Text inside SVG

- Minimize use of <text> elements.

- If text is essential, reference it in aria-labelledby or long description.

- Avoid large amounts of inline text that screen readers would announce as one continuous block.

Color Contrast in SVG

- Always include a solid background color behind text or key visuals to ensure visibility in high contrast modes.

Animated SVGs

- Use JavaScript or CSS, not <animate> element (deprecated).

- Must not flash more than 3 times per second.

- Must not auto-play longer than 5 seconds without user activation.

- Must allow users to pause/stop motion.

Interactive SVGs

- Requirements:

• Fully keyboard accessible (tab-focusable, visible focus indicators, logical tab order).

• Provide equivalent touch interactions.

• Communicate name, role, and value of controls (e.g., role="button", aria-label="Play").

• Manage focus logically during interactions.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Icon Fonts

- Icon fonts use symbols/glyphs in place of letters/numbers.

- Benefits:

• Lightweight, scalable, and render clearly when zoomed.

• Easy to customize color with CSS.

• Widely supported across devices.

Accessibility requirements:

Informative icon fonts

- Treat them as images.

- Use role="img" to expose them as graphics to assistive tech.

- Provide a meaningful text alternative via aria-label or aria-labelledby.

Actionable icon fonts

- If functioning as links, buttons, or controls, they must have meaningful alt text equivalents.

- Best practice:

• <span class="icon-facebook" role="img" aria-label="Facebook"></span>

• Use aria-labelledby if visible text is already present.

Decorative or redundant icon fonts

- Hide from assistive technologies using aria-hidden="true".

- Prevents assistive tech from announcing meaningless or redundant characters.

Bad practice:

- Using CSS clip or hidden text hacks for alt text (unreliable, discouraged).

Comparison: SVG vs Icon Fonts

- Both are vector-based and scale well.

- Icon fonts: single color, still render if images are disabled, limited flexibility.

- SVG icons: can be multi-color, better semantic accessibility, but require image support.

Guideline:

- Choose format based on project needs, but always provide programmatically determinable names, roles, and values for meaningful or actionable icons.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → HTML5 Canvas

- <canvas> defines a drawable region manipulated with JavaScript (used for graphics, games, animations).

- Accessibility issue: content drawn inside <canvas> is not inherently accessible to assistive tech.

Accessible authoring practices:

1. Basic text alternatives

- Always assign role="img" to the <canvas>.

- Provide alt text using aria-label, aria-labelledby, or fallback text inside <canvas>…</canvas>.

- Ensure fallback text is programmatically associated with the canvas element.

2. Complex alternatives

- Provide short alt text (≤150 characters).

- Provide a long description (using aria-describedby, expandable text, dialog, or link to external page).

- Example: describe chart data in a data table associated with the canvas.

3. Low vision accessibility

- Ensure all <canvas> graphics have a background fill (prevents disappearance in high contrast mode).

- Avoid placing critical text inside <canvas> — use SVG if text must scale cleanly with magnification.

4. Keyboard accessibility

- If <canvas> contains interactive controls:

• Ensure all controls are keyboard-operable (support Enter/Space for activation).

• Provide visible focus indicators.

• Match hover styles with keyboard focus styles.

• Use appropriate ARIA roles and labels (e.g., role="button", aria-label="Play").

Guideline:

- Prefer SVG for interactive content when possible (has semantic elements and better built-in accessibility).

- If using <canvas>, explicit ARIA techniques are required to expose content and functionality to assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Multimedia, Objects, and Documents

Multimedia accessibility

- Four main methods for accessibility:

• Captions — synchronized text for spoken content in video.

• Transcripts — full text of spoken words and key visual information.

• Audio descriptions — narration of important visual details not in the audio.

• Sign language interpretation — video of interpreter synchronized with audio.

- Requirements:

• Prerecorded video with audio MUST have captions.

• Prerecorded audio MUST have a transcript.

• Live multimedia MUST have live captions.

• Video with critical visual information MUST have audio descriptions.

- Auto-play media: must not interfere with screen reader output; users must be able to pause/stop playback.

Object element accessibility

- <object> was originally for plug-ins (PDFs, videos, audio, images).

- Rule: Every <object> MUST have meaningful alt text.

- Best practice: use aria-label or aria-labelledby.

Document accessibility

- HTML is the most accessible format: mature features, works in browsers without extra software.

- Provide HTML versions of non-HTML documents when possible.

- Non-HTML formats (Word, PowerPoint, PDF, EPUB) MUST follow accessibility principles:

• Use document title and semantic structure (e.g., headings, styles).

• Provide alternative text for images.

• Mark table headers correctly.

• Ensure sufficient color contrast.

• Use readable fonts.

- PDF: must be tagged; may require manual tagging for complex documents.

- EPUB: must follow HTML accessibility principles; EPUB 3 preferred; also follow EPUB Accessibility specification.

Key guideline:

- Always provide text-based alternatives (captions, transcripts, tagged structure).

- Prefer HTML-based formats where possible for maximum accessibility across devices.

Domain I: Creating Accessible Web Solutions

Subsection: Guidelines, Principles, and Techniques → Summary and Checklist (Images, SVG, Canvas, Multimedia, Documents)

General principle

- All non-text content MUST have a meaningful text equivalent or alternative representation.

Checklist:

Image Alt Text

- Informative images: MUST have programmatically determinable alt text that conveys the author’s intent. Keep concise (~150 characters).

- Decorative/redundant images: MUST be hidden (alt="", role="presentation", or CSS background).

- Actionable images: MUST describe the purpose/action (e.g., alt="Search"), not the visual.

- Form inputs type="image": MUST have alt text conveying the input’s function.

- Complex images: MUST have alt text + long description if details exceed concise alt limits.

- Images of text: avoid unless essential (e.g., logos); otherwise use real text.

- CSS background images: informative/actionable ones MUST have equivalent text in HTML.

- Image maps: <img> must have alt; each <area> must have alt describing destination/action. Avoid server-side maps.

SVG

- <img src="file.svg" role="img" alt="..."> for simple cases.

- Inline <svg> role="img" with <title> (short alt) and <desc> (extended description).

- Decorative SVGs: aria-hidden="true".

- Avoid embedding via <object> or <iframe>.

- Provide color contrast and background fill.

- Animated SVGs: no flashing >3Hz, no auto-play >5s, must be pausable.

- Interactive SVGs: fully keyboard- and touch-accessible; expose name, role, value of controls.

Icon Fonts

- Informative: role="img" + aria-label/aria-labelledby.

- Actionable: MUST have meaningful label.

- Decorative/redundant: aria-hidden="true".

HTML5 Canvas

- Assign role="img" to <canvas>.

- Provide alt text via aria-label/aria-labelledby or fallback text.

- Complex graphics: add long descriptions (aria-describedby or linked content).

- Must have background fill.

- Custom controls must be keyboard-accessible and expose correct role/label.

Multimedia

- Prerecorded video: captions + transcript + audio description if needed.

- Live multimedia: live captions.

- Audio-only: transcript required.

- Sign language interpretation may be provided as supplemental access.

- Auto-play must not disrupt assistive technologies.

Objects and Documents

- <object> elements: MUST have alternative text.

- Non-HTML documents (Word, PDF, EPUB, etc.) MUST follow accessibility principles.

- Prefer HTML versions of documents for maximum accessibility.

- PDFs: MUST be tagged.

- EPUB: follow HTML principles, use EPUB 3, meet EPUB Accessibility guidelines.

Core guideline:

- Provide multiple equivalent alternatives to ensure information and functionality are perceivable, operable, and understandable to all users.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Introduction

- This module extends multimedia accessibility concepts beyond captions, transcripts, audio descriptions, and sign language interpretation.

- Focus areas include:

• Clear audio: Background sounds must not interfere with spoken words.

• Seizure prevention: Avoid flashing visual effects that may induce seizures.

• Motion sensitivity: Limit excessive motion in video or animations that may cause dizziness or disorientation in people with vestibular disorders.

• Media player accessibility: Ensure keyboard access and correct communication of names, roles, values, and states of controls.

• Auto-play concerns: Prevent media from automatically playing in ways that interfere with screen reader use.

- Intended audience: web developers and QA testers.

- Technical level: easy/non-technical.

- Prerequisites: accessibility fundamentals and prior modules (semantic structure, images, colors, responsive design, basic multimedia).

- Guidelines: aligns with WCAG 2.2, Level AA.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

- IAAP Certification: course counts for 0.50 CAECs toward CPACC, WAS, or CPWA.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Multimedia Accessibility Decision Matrix

- Purpose: Provides guidance on which media alternatives are appropriate depending on media type (audio-only, video-only, multimedia) and compliance level (WCAG A, AA, AAA).

- Time-based media categories:

• Audio-only (e.g., podcast, radio recording)

• Video-only (e.g., silent movie, how-to video without narration)

• Multimedia (synchronized audio + video, e.g., movie, commercial, narrated how-to)

- Alternative methods:

• Captions = synchronized text alternative for speech and non-speech audio.

• Transcripts = text description of visual + auditory info.

• Audio descriptions (AD) = narration describing important visuals.

• Sign language = visual translation of spoken language.

- Applicability:

• Not all alternatives apply to every media type (e.g., video-only content doesn’t require captions; audio-only doesn’t require AD).

• Media can be live or pre-recorded.

- Regulatory alignment:

• MUST = required by WCAG/Section 508/CVAA.

• SHOULD = strongly encouraged for broader accessibility (e.g., deaf-blind users).

• MAY = optional, goes “above and beyond.”

- Matrix use: helps map content type + compliance level to correct accessibility alternative(s).

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Media Player Accessibility

- Context: HTML5 video largely replaced older plugins (Flash, QuickTime, RealMedia, etc.), making accessible media players more feasible. Accessibility, however, varies by browser and screen reader support.

- Recommendation: Use players designed/tested for accessibility.

• Example: Able Player (open-source) – supports captions, transcripts, audio descriptions, and sign language, with fully accessible controls.

• Other accessible players: OzPlayer, Nomensa media player, PayPal media player.

- Accessibility Requirements:

• Keyboard Accessibility → all functions must be operable via keyboard; logical tab order; visible focus indicators; no mouse-only actions; logical focus management during interactions; provide instructions for nonstandard controls.

• Screen Reader Accessibility → controls MUST correctly communicate name, role, and value.

- Names: e.g., "Play", "Pause", "Volume".

- Roles: e.g., `role="button"`, `role="slider"`.

- Values: dynamic updates (e.g., “50% volume”).

- Use ARIA attributes when using custom controls; follow ARIA Authoring Practices.

• Captions, Transcripts, and Audio Descriptions:

- SHOULD be available via distinct controls/icons (captions, transcript, audio description, sign language).

- Features may be provided either within the player or via alternate methods (e.g., embedded captions, separate transcript).

- Web captioning strongly preferred for customization.

- Example: Able Player integrates multiple accessibility features.

• Customizability:

- SHOULD allow caption customization (font, size, color, background, opacity).

- SHOULD remember user preferences (via cookies or account storage).

- SHOULD allow full-screen video for low-vision users.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Background Sounds

- Accessibility Impact:

• Background sounds can interfere with comprehension for people who are hard of hearing, use screen readers, have cognitive disabilities, or are easily distracted.

• Auto-playing background audio can prevent screen reader users from hearing navigation cues and make it difficult to find/disable the sound.

- Guidelines for Media:

• In prerecorded audio-only and multimedia content:

- Background sounds SHOULD be eliminated during narration/dialog OR

- Background sounds MUST be at least 20 dB lower than the foreground (≈4x quieter), except for occasional sounds ≤2 seconds OR

- Users MUST have a method to turn them off.

• In live audio-only and live multimedia:

- Same principles apply, though not mandated by WCAG, but strongly recommended where possible.

- Practical Considerations:

• Background music may enhance content but MUST NOT overpower speech/narration.

- Best practice: play music only during breaks or at reduced volume.

• Background noises (applause, traffic, wind, machinery, chatter) distract from dialogue.

- Reduce during recording; use professional equipment and dedicated microphones.

• Overlapping dialogue complicates comprehension, especially for hearing aid users.

- Encourage one speaker at a time during recordings.

• For live content, audio technicians should plan and manage background audio levels when feasible (e.g., news reporting).

- Guidelines for Web Pages:

• Any audio that auto-plays for more than 3 seconds MUST provide a mechanism to:

- Pause, stop, or mute audio, OR

- Control its volume independently of system volume.

• Best practice: audio should be user-initiated, or auto-play should stop after 3 seconds.

• Controls MUST be easy to locate, appear near the top of the page, be keyboard operable, and clearly labeled.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Seizure-Inducing Flashes

- Accessibility Risk:

• Flashing content (bright strobe, intense patterns, rapidly changing images) can trigger seizures in people with photosensitive epilepsy.

• Flashing can also distract or disorient users with cognitive or vestibular disabilities.

- WCAG Conformance:

• Content MUST NOT flash more than 3 times per second.

• Exception: if the combined area of flashing is small, the contrast is low, and the flash is not red.

• Guidelines apply to full pages, embedded objects, ads, banners, video, and animations.

- Testing Methods:

• Tools: Photosensitive Epilepsy Analysis Tool (PEAT), Trace Research and Development Center tool.

• Manual review: check animations, video sequences, ads, games, or auto-updating components for strobing or intense flashing.

- Design Best Practices:

• Avoid flashing altogether whenever possible.

• Replace problematic effects with safer alternatives:

- Subtle animations

- Smooth fades

- Slow transitions

• Warn users before unavoidable flashing content and provide a way to skip or disable it.

- Real-World Issues:

• Auto-updating ads and video promotions often introduce flashes without site owner awareness.

• Games and entertainment media frequently contain rapid effects—require careful evaluation.

• Emergency alert banners with strobe or intense blink patterns can be inaccessible.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Motion Sensitivity

- Accessibility Risk:

• Motion effects can cause nausea, dizziness, headaches, or disorientation for users with vestibular disorders.

• Examples: parallax scrolling, auto-updating carousels, large animated transitions, or motion-triggered gestures.

- WCAG Conformance:

• Content that moves, blinks, scrolls, or auto-updates for more than 5 seconds MUST provide a mechanism to pause, stop, or hide it.

• Motion animation triggered by user interaction (e.g., hover, scroll) MUST be able to be disabled unless essential.

• Users SHOULD have the option to reduce or eliminate motion effects.

- Testing Methods:

• Use operating system accessibility settings (e.g., “Reduce Motion” on iOS/macOS).

• Check for controls to pause/stop motion or animations on the site.

• Test with assistive technologies and verify that user preferences carry through.

- Design Best Practices:

• Prefer static alternatives when animation is not essential.

• Avoid parallax and auto-playing carousels unless critical.

• Provide clear “pause/stop” buttons for all moving content.

• Support user system settings for reduced motion via CSS media query: `prefers-reduced-motion`.

• Use smooth, subtle animations instead of rapid or exaggerated motion.

- Real-World Issues:

• Marketing banners often auto-rotate or slide with no user control.

• Games and visual demos may use strong camera movement or panning effects.

• Background parallax and floating UI elements may conflict with user motion settings.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Auto-Play Concerns

- Accessibility Risk:

• Auto-playing media can disorient users and interfere with screen readers, making it hard to navigate or interact with the page.

• Unexpected audio can create confusion, stress, or distraction for users with cognitive or sensory disabilities.

- WCAG Conformance:

• Any audio that plays automatically for more than 3 seconds MUST provide a mechanism to pause, stop, or mute it, OR allow independent volume control.

• Auto-playing video with audio SHOULD be avoided; if present, provide user controls to disable or pause playback.

• Auto-play MAY be acceptable when critical (e.g., emergency alerts), but only with accessible controls.

- Testing Methods:

• Verify that auto-play does not start unexpectedly on page load.

• Confirm that controls to stop or mute auto-play are clearly visible and keyboard operable.

• Test with screen readers to ensure audio does not override navigation or output.

- Design Best Practices:

• Default to user-initiated playback rather than auto-play.

• If auto-play is unavoidable:

- Limit to ≤3 seconds.

- Provide prominent pause/stop/mute controls at the top of the page.

- Ensure controls are labeled and operable with keyboard and assistive tech.

• Use silent auto-play for video previews if needed (common in social media feeds).

• Clearly indicate when media is about to begin playing.

- Real-World Issues:

• News sites and ad networks often embed auto-playing videos without accessible controls.

• Social media platforms frequently use silent auto-play for scrolling feeds—preferred to avoid disruption.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia, Animations, and Motion → Summary

- Accessible multimedia requires more than captions and transcripts; it must also address:

• Background sounds that compete with dialogue or narration.

• Seizure-inducing flashes that must be avoided or controlled.

• Motion sensitivity concerns, where animations or scrolling effects may cause dizziness or nausea.

• Media player accessibility, ensuring full keyboard and screen reader compatibility, as well as customizable captions and accessible controls.

• Auto-play behavior, which should be limited, user-controlled, and never disruptive to screen readers.

- Regulatory and Standards Alignment:

• WCAG 2.2 Levels A and AA set the baseline requirements.

• Section 508 and CVAA add further compliance obligations in the U.S.

• Global standards (e.g., EN 301 549) extend requirements internationally.

- Best Practices:

• Always provide user controls to pause, stop, or adjust multimedia.

• Favor user-initiated playback over auto-play.

• Support system-level accessibility preferences (e.g., reduced motion).

• Test with real assistive technologies and diverse user needs.

- Key Takeaway: Accessible multimedia balances creative design with user control, ensuring that all people—including those with sensory, cognitive, or vestibular disabilities—can engage with and benefit from media content.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Introduction

- Forms are a critical component of web accessibility because they are used for communication, transactions, and data collection.

- Accessibility problems in forms create significant barriers for users with disabilities, particularly those relying on screen readers, keyboard navigation, or voice input.

- Accessible forms benefit all users by improving clarity, error prevention, and usability.

- Intended audience: web developers, designers, and QA testers.

- Technical level: moderate.

- Prerequisites: understanding of semantic HTML, labels, focus management, and landmarks.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA) form the baseline.

• Section 508 and EN 301 549 harmonize with WCAG requirements for forms.

• Additional guidance from WAI-ARIA Authoring Practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Labels and Instructions

- Importance of Labels:

• Labels provide essential context for screen reader users and help all users understand what information is required.

• Every form control MUST have an accessible name that is programmatically associated with it.

- Guidelines:

• Use the <label> element with the “for” attribute linked to the form control’s “id.”

• Place labels adjacent to their controls (above or to the left) for clarity.

• Placeholder text SHOULD NOT be used as a replacement for labels; it disappears when users start typing and is not always reliably announced by assistive technology.

• Group related controls (e.g., radio buttons, checkboxes) with a <fieldset> and <legend> for context.

• Provide clear instructions and indicate required vs. optional fields.

• Use plain language; avoid jargon or ambiguous instructions.

- Examples:

Bad Example:

<input type="text" placeholder="First Name">

(No label, placeholder only.)

Good Example:

<label for="fname">First Name</label>

<input type="text" id="fname" name="fname">

- Error Prevention:

• Instructions SHOULD be available before input, not only after submission.

• Required fields MUST be indicated (e.g., text “required” or symbol like \*).

• Instructions MUST be visible and programmatically associated with inputs where relevant.

- Assistive Technology:

• Screen readers announce labels tied to inputs, helping users understand each field’s purpose.

• Voice input software relies on labels to activate form fields via voice commands.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Input Types

- Native HTML5 Input Types:

• Offer built-in accessibility, validation, and semantics when used correctly.

• Examples: type="email", type="tel", type="url", type="number", type="date", type="time", type="search".

• Benefits: screen readers announce input purpose; mobile devices display optimized keyboards; browsers apply built-in validation.

- Guidelines:

• Use semantic input types whenever possible instead of generic type="text".

• Ensure accessible names are provided for all inputs, regardless of type.

• Avoid overriding native behavior unless necessary.

• Provide clear error messages if built-in validation fails.

• Where native types lack broad support (e.g., date/time pickers), provide accessible fallback.

- Examples:

Good Example:

<label for="email">Email Address</label>

<input type="email" id="email" name="email" required>

(Announced as “Edit, Email Address, required.” On mobile, email keyboard appears.)

Bad Example:

<label for="email">Email Address</label>

<input type="text" id="email" name="email">

(Loses semantic benefit, less usable on mobile devices.)

- ARIA Usage:

• ARIA roles and properties SHOULD NOT replace native input types.

• Use ARIA only when native HTML cannot provide the needed functionality.

- Best Practices:

• Keep controls simple and consistent.

• Test on desktop and mobile devices.

• Verify input types with multiple browsers and assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Instructions and Help Text

- Purpose:

• Provide users with guidance on how to complete forms correctly and efficiently.

• Help text reduces errors and supports users with cognitive or language-related disabilities.

- Guidelines:

• Place instructions near relevant fields so they are easy to find.

• Instructions MUST be programmatically associated with the field if they are essential.

• Supplemental instructions MAY appear before the form, within the form, or inline.

• Use clear, simple language; avoid technical jargon.

• Identify required vs. optional fields consistently.

- Examples:

• Inline Help:

<label for="phone">Phone Number</label>

<input type="tel" id="phone" aria-describedby="phoneHelp">

<small id="phoneHelp">Include area code.</small>

• Pre-Form Instructions:

“All fields marked with \* are required.”

- Accessibility Considerations:

• Screen readers use aria-describedby to announce help text linked to inputs.

• Instructions MUST remain visible; avoid disappearing text that only appears on hover/focus.

• Avoid relying solely on color or symbols for meaning (e.g., red asterisk without explanation).

- Error Prevention:

• Anticipate user confusion and provide examples when format is strict (e.g., date: MM/DD/YYYY).

• Instructions SHOULD be available both visually and programmatically.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Validation and Error Handling

- Purpose:

• Ensure users understand when errors occur and how to correct them.

• Provide clear, accessible feedback for all users, including screen reader and keyboard users.

- Guidelines:

• Errors MUST be identified and described to the user in text.

• Error messages MUST be programmatically associated with the corresponding input fields.

• Use aria-describedby or similar methods to link error messages to inputs.

• Do not rely solely on color, icons, or visual changes to convey errors.

• Provide suggestions for corrections when possible.

- Error Prevention:

• Validate input formats in real time where appropriate (without blocking progress).

• Provide examples of correct formats near fields with strict requirements.

• Clearly identify required fields to avoid unnecessary errors.

- Examples:

Bad Example:

<input type="email" id="email">

(On submit: “Error” appears in red only, no context for screen reader users.)

Good Example:

<label for="email">Email Address</label>

<input type="email" id="email" aria-describedby="emailError">

<span id="emailError">Enter a valid email address, e.g., name@example.com</span>

- Accessibility Considerations:

• Error messages SHOULD appear immediately after the input field they relate to.

• Users MUST be able to navigate back to erroneous fields easily (focus management).

• Screen readers should announce both the error and the field label together.

• Errors SHOULD NOT be announced only via modal dialogs or alerts without field association.

- Best Practices:

• Offer confirmation messages after successful submission.

• Use plain, friendly language for errors (avoid technical jargon).

• Ensure forms are tested with multiple assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Error Recovery and Confirmation

- Purpose:

• Help users recover from mistakes without losing progress.

• Provide reassurance when forms are submitted successfully.

- Guidelines for Error Recovery:

• Preserve user input when errors occur—MUST NOT clear entire form on submit failure.

• Clearly identify each error and provide specific instructions for correction.

• Place error messages near the corresponding field and ensure they are programmatically tied.

• Provide a summary of all errors at the top of the form for quick orientation.

• Allow users to navigate directly from summary errors to fields in question.

- Guidelines for Confirmation:

• Provide a confirmation page or inline message after successful submission.

• Confirmation MUST be clear, concise, and accessible to screen readers.

• Confirmation SHOULD include a summary of submitted data for verification (unless privacy/security concerns prevent).

• Provide next steps or follow-up information when relevant.

- Examples:

• Error Recovery:

“You entered 555-123. Phone numbers must be 10 digits. Please re-enter.”

• Confirmation:

“Your application has been submitted successfully. A confirmation email has been sent to you.”

- Accessibility Considerations:

• Screen reader users rely on clear feedback to know if submission was successful.

• Keyboard focus SHOULD move to error messages or confirmation notices automatically.

• Confirmation notices MUST remain visible until dismissed or the user moves forward.

• Avoid using only color or icons for error/confirmation feedback.

- Best Practices:

• Use polite, supportive language when pointing out errors.

• For multi-step forms, preserve state when users move backward or forward.

• Test error recovery and confirmation with multiple assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Keyboard and Screen Reader Accessibility

- Keyboard Accessibility:

• All form controls MUST be operable via keyboard alone.

• Logical tab order MUST follow the visual reading order.

• Focus MUST be visible and clearly indicate which element is active.

• Custom widgets (e.g., date pickers, combo boxes) MUST provide keyboard equivalents for all functions.

• Keyboard traps (where focus cannot escape) MUST be avoided.

- Screen Reader Accessibility:

• Each form control MUST have an accessible name and role announced correctly.

• Labels MUST be explicitly associated with form controls.

• Error messages and help text MUST be programmatically tied to the relevant fields.

• Grouping structures (fieldset + legend) provide essential context for related inputs.

• Use ARIA sparingly—only when native semantics are insufficient.

- Testing Strategies:

• Navigate through form with only the Tab, Shift+Tab, arrow keys, and Enter/Space.

• Test with at least one screen reader (e.g., NVDA, JAWS, VoiceOver).

• Verify that screen readers announce labels, roles, and instructions clearly.

• Confirm focus moves logically and predictably across the form.

- Best Practices:

• Keep tab order short and intuitive.

• Avoid using tabindex values > 0 (can cause confusion).

• Provide skip links for long forms to improve navigation efficiency.

• Ensure dynamic error or confirmation messages are announced (via ARIA live regions if needed).

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Summary

- Accessible forms require careful attention to labels, instructions, input types, error handling, and confirmation.

- Key elements include:

• Labels and instructions MUST be programmatically associated with inputs.

• Use semantic HTML5 input types to improve usability and accessibility.

• Provide clear, persistent help text where needed.

• Errors MUST be identified in text and tied to the relevant fields.

• Preserve user input and support error recovery without data loss.

• Offer clear, accessible confirmation messages after successful submission.

• Ensure full keyboard operability and logical focus order.

• Screen readers MUST announce form fields, instructions, and errors correctly.

• Group related fields with fieldset and legend for context.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define the baseline requirements.

• Section 508 and EN 301 549 harmonize requirements globally.

• ARIA Authoring Practices provide guidance when custom controls are necessary.

- Best Practices:

• Always use native HTML form controls where possible.

• Favor simplicity and consistency in form design.

• Provide error prevention through clear instructions and examples.

• Test with keyboard-only navigation and multiple screen readers.

- Key Takeaway: Accessible forms improve usability for all users, reduce errors, and ensure equal access for people with disabilities.

Domain I: Creating Accessible Web Solutions

Subsection: Data Tables → Introduction

- Purpose:

• Data tables present structured information that users need to navigate and interpret accurately.

• Poorly coded tables create barriers for screen reader users and those relying on keyboard navigation.

• Accessible tables ensure that headers, relationships, and data are conveyed programmatically.

- Intended audience: web developers and QA testers.

- Technical level: moderate.

- Prerequisites: semantic HTML knowledge, landmarks, headings, and forms.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible table structures.

• ARIA Authoring Practices provide additional techniques for complex tables.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Data Tables → Simple Tables

- Purpose:

• Simple tables organize information into rows and columns with clear header associations.

• Screen readers rely on proper markup to communicate relationships between headers and data cells.

- Guidelines:

• Use <table> for tabular data only (not layout).

• Define headers with <th> elements, not <td>.

• Scope attribute (scope="col" or scope="row") SHOULD be used to associate headers with their respective rows or columns.

• Keep tables simple with minimal nesting and avoid merged cells where possible.

• Provide captions (<caption>) to describe the table’s purpose.

• Avoid leaving data cells blank; if empty, indicate meaning (e.g., “N/A”).

- Example (Good):

<table>

<caption>Quarterly Sales Results</caption>

<tr>

<th scope="col">Quarter</th>

<th scope="col">Sales</th>

</tr>

<tr>

<td>Q1</td>

<td>$50,000</td>

</tr>

</table>

- Example (Bad):

<table>

<tr><td>Quarter</td><td>Sales</td></tr>

<tr><td>Q1</td><td>$50,000</td></tr>

</table>

(Lacks semantic headers, harder for screen readers.)

- Accessibility Considerations:

• Captions provide context when multiple tables exist on a page.

• Screen readers announce headers when moving through data cells.

• Logical reading order MUST be preserved for both visual and non-visual users.

- Best Practices:

• Keep tables short and simple.

• Use lists instead of tables when tabular relationships are unnecessary.

• Test table navigation with keyboard and screen readers.

Domain I: Creating Accessible Web Solutions

Subsection: Data Tables → Complex Tables

- Purpose:

• Complex tables contain multiple levels of headers or irregular structures that require explicit markup to maintain accessibility.

• Examples: tables with merged cells, multiple header rows, or multi-level categories.

- Guidelines:

• Use id and headers attributes to explicitly associate data cells with the correct headers.

• Avoid unnecessary complexity—simplify table structure whenever possible.

• Provide captions (<caption>) and, if needed, a summary to describe how the table is organized.

• Split complex tables into multiple simpler tables if the structure becomes too difficult to interpret.

- Example (Multi-Level Headers):

<table>

<caption>Regional Sales Report</caption>

<tr>

<th id="region" rowspan="2">Region</th>

<th id="q1" colspan="2">Q1</th>

<th id="q2" colspan="2">Q2</th>

</tr>

<tr>

<th id="q1sales">Sales</th>

<th id="q1profit">Profit</th>

<th id="q2sales">Sales</th>

<th id="q2profit">Profit</th>

</tr>

<tr>

<td headers="region">North</td>

<td headers="q1 q1sales">$50,000</td>

<td headers="q1 q1profit">$5,000</td>

<td headers="q2 q2sales">$55,000</td>

<td headers="q2 q2profit">$6,000</td>

</tr>

</table>

- Accessibility Considerations:

• Screen readers can correctly announce multi-level relationships when headers and ids are used properly.

• Complex tables without explicit associations may be confusing or unusable.

• Nested tables SHOULD be avoided, as they complicate navigation and interpretation.

- Best Practices:

• Provide alternate representations of the data (e.g., charts or summaries) when possible.

• Ensure consistent use of headers, ids, and scopes.

• Test with multiple screen readers to confirm that associations are clear.

Domain I: Creating Accessible Web Solutions

Subsection: Data Tables → Responsive Tables

- Purpose:

• Responsive tables adapt to smaller screen sizes (e.g., mobile devices) without losing accessibility or usability.

• Poorly implemented responsive designs can break header associations or hide important data.

- Guidelines:

• Maintain proper semantic markup (<table>, <th>, <td>, scope, headers/id) regardless of screen size.

• Do NOT convert tables into non-semantic structures (e.g., lists or divs) for mobile display.

• Use CSS techniques to make tables scrollable horizontally when necessary.

• Consider alternative layouts (e.g., stacking cells under headers) but preserve accessibility relationships.

- Techniques:

• Horizontal scrolling with overflow-x: auto is often the simplest and most reliable.

• For stacked layouts, programmatically associate data with headers using aria-labelledby or aria-describedby.

• Provide a summary or caption to explain responsive behavior if content display differs between desktop and mobile.

- Accessibility Considerations:

• Screen reader users rely on consistent header associations; ensure they are preserved after layout changes.

• Keyboard users MUST be able to scroll horizontally or navigate through restructured tables.

• Avoid hiding essential data for smaller screens; if hidden, provide alternative access.

- Best Practices:

• Test responsive tables on multiple devices and orientations.

• Ensure zooming does not break table layout.

• Provide alternate views (e.g., downloadable CSV or accessible charts) for large datasets.

Domain I: Creating Accessible Web Solutions

Subsection: Data Tables → Summary

- Accessible tables ensure that data is presented clearly and relationships are conveyed programmatically.

- Key requirements:

• Use semantic HTML (<table>, <th>, <td>) only for tabular data, not layout.

• Provide captions to describe table purpose.

• Use scope, headers, and ids to maintain header associations.

• Keep tables as simple as possible; avoid merged or nested cells where feasible.

• Complex tables MUST include explicit associations between headers and data.

• Responsive designs MUST preserve accessibility and prevent data loss.

• Avoid hiding essential data on smaller screens.

• Test navigation with screen readers and keyboard-only input.

- Regulatory Alignment:

• WCAG 2.2 (Levels A and AA) defines table accessibility requirements.

• Section 508 and EN 301 549 harmonize international compliance.

• ARIA techniques supplement but SHOULD NOT replace semantic HTML.

- Best Practices:

• Split complex tables into simpler ones where possible.

• Provide alternative representations of data (charts, summaries, CSV downloads).

• Favor horizontal scrolling over structural changes that break semantics.

• Always verify table usability on both desktop and mobile devices.

- Key Takeaway: Accessible tables preserve meaning, structure, and usability for all users, ensuring equitable access to structured information.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Introduction

- Purpose:

• Images play a key role in communication, branding, and conveying information on the web.

• Without proper alternatives, images can create barriers for users with visual impairments or those relying on assistive technologies.

- Intended audience: web developers, designers, and content authors.

- Technical level: easy to moderate.

- Prerequisites: knowledge of semantic HTML, ARIA basics, and general accessibility principles.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible image usage.

• WAI-ARIA provides additional guidance for complex image descriptions.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Images → Informative Images

- Purpose:

• Informative images convey meaningful information that is not already available in text.

• Users with visual impairments rely on alternative text to understand the content and function of these images.

- Guidelines:

• Provide concise, descriptive alt text that communicates the essential meaning of the image.

• Avoid phrases like “image of” or “picture of”—screen readers already announce images.

• Alt text SHOULD be unique and not duplicate nearby text unless redundancy is intentional.

• If text appears in the image, replicate it in alt text unless it is decorative or redundant.

• Keep alt text brief—generally under 125 characters, but long enough to capture meaning.

• For logos, alt text SHOULD identify the organization (e.g., “Deque Systems logo”).

- Examples:

• Good: <img src="piechart.png" alt="Pie chart showing 60% mobile users and 40% desktop users">

• Bad: <img src="piechart.png" alt="chart">

- Accessibility Considerations:

• Screen readers announce alt text in place of the image.

• Informative icons (e.g., magnifying glass for search, envelope for email) MUST have alt text or an accessible name.

• Decorative flourishes, borders, or spacer images SHOULD use empty alt (alt="") to avoid clutter.

- Best Practices:

• Ensure alt text is accurate, concise, and contextual.

• Use consistent style for icons and symbols across the site.

• Test with screen readers to verify that meaning is preserved.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Decorative Images

- Purpose:

• Decorative images add visual appeal but do not convey essential information.

• For accessibility, they MUST be ignored by assistive technologies so they do not add noise for screen reader users.

- Guidelines:

• Use empty alt attributes (alt="") for purely decorative images.

• Decorative images SHOULD NOT be given titles or ARIA labels.

• CSS background images are often appropriate for decoration instead of HTML <img>.

• Ensure decorative images do not duplicate meaningful information already conveyed in text.

- Examples:

• Good: <img src="border.png" alt="">

• Bad: <img src="border.png" alt="decorative border">

• Good: background-image in CSS for styling.

- Accessibility Considerations:

• Screen readers skip images with empty alt attributes, allowing smoother navigation.

• If a decorative image is incorrectly given alt text, it can confuse or annoy users.

• Background images in CSS are not read by screen readers, making them suitable for non-essential visuals.

- Best Practices:

• Only use decorative images when they add aesthetic value.

• Ensure decorative images do not distract from or obscure informative content.

• When in doubt, ask whether removing the image would reduce meaning—if not, it is decorative.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Functional Images

- Purpose:

• Functional images are used as interactive elements—links, buttons, or controls.

• Alt text MUST describe the action or destination, not the visual appearance.

- Guidelines:

• Provide alt text that communicates the function (e.g., “Search,” “Submit order,” “Go to home page”).

• Do not use file names or vague labels like “button” or “link.”

• If text already appears within the image, alt text MUST replicate that text.

• Ensure functional images have accessible names through alt text, aria-label, or visible text.

• Avoid redundancy when combining text and images inside links/buttons; one accessible label is sufficient.

- Examples:

• Good: <a href="index.html"><img src="home.png" alt="Home"></a>

• Bad: <a href="index.html"><img src="home.png" alt="house icon"></a>

• Good: <input type="image" src="search.png" alt="Search">

- Accessibility Considerations:

• Screen readers announce the alt text as the control’s accessible name.

• Functional images without alt text result in “unlabeled button” or “link” announcements.

• Consistent labeling improves usability for voice input users.

- Best Practices:

• Always prioritize describing the function over the image itself.

• Test functional images with screen readers and keyboard navigation.

• Replace image buttons with native HTML controls (e.g., <button>) where possible.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Complex Images

- Purpose:

• Complex images convey detailed information that cannot be summarized in short alt text.

• Examples include charts, graphs, maps, diagrams, and infographics.

- Guidelines:

• Provide a short alt attribute describing the general purpose of the image.

• Provide a longer description elsewhere on the page or via a linked resource.

• Use aria-describedby to associate the image with a nearby text description when needed.

• For charts and graphs, replicate data in an accessible format such as a data table.

• For maps, include textual alternatives for important information (e.g., list of locations with addresses).

- Examples:

• Good: <img src="chart.png" alt="Bar chart showing sales by quarter" aria-describedby="chartDesc">

<div id="chartDesc">Sales were highest in Q2 ($75,000), followed by Q3 ($60,000), Q1 ($50,000), and Q4 ($45,000).</div>

• Bad: <img src="chart.png" alt="complex chart">

- Accessibility Considerations:

• Screen readers announce alt text and, when provided, linked longer descriptions.

• Without extended descriptions, users miss critical information conveyed by visuals.

• Text descriptions benefit all users, especially when images fail to load.

- Best Practices:

• Always provide redundant accessible formats for complex visuals.

• Ensure descriptions are accurate, concise, and easy to locate.

• Provide downloadable accessible data formats (CSV, HTML tables) where applicable.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Groups of Images

- Purpose:

• Groups of related images may collectively convey information (e.g., photo sequences, sets of icons, image galleries).

• Accessibility requires providing context for the group as well as for individual images if needed.

- Guidelines:

• Provide a group-level description that summarizes the overall purpose.

• Each image within the group SHOULD have its own alt text if it adds unique meaning.

• If images are purely decorative within the group, they SHOULD use empty alt (alt="").

• Use figure and figcaption elements to associate images with captions when appropriate.

• For icon sets, ensure each icon has a unique accessible name if used individually.

- Examples:

• Gallery Example:

<figure>

<img src="sunrise.jpg" alt="Sunrise over mountains">

<figcaption>Morning view of the Rocky Mountains</figcaption>

</figure>

• Icon Set Example:

<img src="print.png" alt="Print">

<img src="save.png" alt="Save">

<img src="share.png" alt="Share">

- Accessibility Considerations:

• Screen readers read alt text for each image in sequence unless grouped with descriptive captions.

• Group-level descriptions prevent redundancy and improve clarity for users.

• Decorative elements inside groups must not interfere with navigation or meaning.

- Best Practices:

• Favor concise, meaningful descriptions for individual images.

• Use captions to provide context and reduce repetitive alt text.

• Test galleries, carousels, or image groups with keyboard and screen readers.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Image Maps

- Purpose:

• Image maps use a single image divided into clickable regions (hotspots), each linking to a different destination or action.

• Without proper markup, screen reader and keyboard users cannot identify or navigate hotspots.

- Guidelines:

• Use the <map> element with <area> elements to define hotspots.

• Each <area> MUST have alt text describing its purpose or destination.

• Avoid using image maps for essential navigation when simpler alternatives (lists of links, menus) are available.

• If the same functionality can be provided with standard HTML, prefer that approach.

• For decorative or redundant image maps, alt text SHOULD be empty.

- Example:

<img src="worldmap.png" usemap="#worldmap" alt="World map with regional links">

<map name="worldmap">

<area shape="rect" coords="34,44,270,350" href="northamerica.html" alt="North America">

<area shape="rect" coords="290,172,333,250" href="europe.html" alt="Europe">

</map>

- Accessibility Considerations:

• Screen readers announce each hotspot’s alt text as a link.

• Keyboard users must be able to tab through all hotspots in logical order.

• If regions overlap or are too small, they may be difficult for users to access.

- Best Practices:

• Provide a text-based alternative for critical navigation.

• Ensure hotspot regions are large enough for users with motor impairments.

• Test with screen readers and keyboard navigation to confirm functionality.

Domain I: Creating Accessible Web Solutions

Subsection: Images → Summary

- Accessible images ensure that visual content is perceivable and usable by people with disabilities.

- Key requirements:

• Informative images MUST have concise, descriptive alt text.

• Decorative images MUST use empty alt (alt="") or CSS backgrounds to be ignored by assistive technology.

• Functional images MUST describe their purpose or action, not appearance.

• Complex images MUST include extended descriptions or data alternatives (tables, text summaries).

• Groups of images MUST have both group-level and individual descriptions where meaningful.

• Image maps MUST include alt text for each hotspot and preserve keyboard navigation.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA provide baseline image accessibility requirements.

• Section 508 and EN 301 549 harmonize international compliance.

• WAI-ARIA provides supplementary techniques for advanced use cases.

- Best Practices:

• Always ask: “If this image were removed, would meaning be lost?” If yes, provide alt text or description.

• Keep alt text short, specific, and contextual.

• Avoid redundancy when text nearby already conveys the same information.

• Provide alternative data formats for complex visuals (CSV, HTML tables).

• Test with screen readers to confirm alt text and descriptions are announced as expected.

- Key Takeaway: Accessible image practices balance simplicity and accuracy, ensuring that all users can understand visual content regardless of ability.

Domain I: Creating Accessible Web Solutions

Subsection: Links → Introduction

- Purpose:

• Links are fundamental for navigation and interaction on the web.

• Inaccessible links can prevent users from understanding purpose, navigating effectively, or accessing content.

- Intended audience: web developers, designers, and content authors.

- Technical level: easy.

- Prerequisites: knowledge of semantic HTML and text alternatives.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible links.

• WAI-ARIA may supplement but MUST NOT replace native link semantics.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Links → Link Purpose and Text

- Purpose:

• Users must be able to determine the purpose of a link from its text alone or its context.

• Clear link purpose benefits screen reader users, keyboard users, and people with cognitive disabilities.

- Guidelines:

• Link text MUST be descriptive and convey destination or action.

• Avoid vague text such as “click here,” “read more,” or “more info.”

• Links SHOULD make sense out of context (e.g., in a screen reader’s links list).

• When context is necessary, ensure it is programmatically tied (e.g., with aria-describedby).

• Links MUST NOT rely only on surrounding visual cues (color, icons).

• Use consistent labeling for the same destinations across the site.

- Examples:

• Good: <a href="report.pdf">2025 Accessibility Report (PDF)</a>

• Bad: <a href="report.pdf">Click here</a>

• Good (with context): <a href="profile.html" aria-describedby="userContext">View profile</a>

<span id="userContext">of Jane Doe</span>

- Accessibility Considerations:

• Screen readers often present a list of links; vague text makes navigation difficult.

• Voice input users rely on link names to activate them by speech.

• Cognitive users benefit from predictable and descriptive link text.

- Best Practices:

• Use the shortest text possible while still being descriptive.

• Avoid duplicating identical link text for different destinations.

• Indicate file type and size in link text when relevant (e.g., “Annual Report (PDF, 2 MB)”).

Domain I: Creating Accessible Web Solutions

Subsection: Links → Link States and Visual Indicators

- Purpose:

• Users need to identify links visually and understand their states (default, hover, focus, visited).

• Proper indicators support users with visual, cognitive, and motor disabilities.

- Guidelines:

• Links MUST be visually distinguishable from surrounding text (e.g., underlines, contrast).

• Color alone MUST NOT be the only method of differentiation.

• Ensure sufficient contrast between link text and background (WCAG requires at least 4.5:1 for normal text, 3:1 for large text).

• Focus states MUST be visible for keyboard users.

• Hover and active states SHOULD provide additional clarity without causing motion sensitivity.

• Visited links SHOULD appear visually distinct to aid navigation.

- Examples:

• Good: a { color: #0645AD; text-decoration: underline; }

• Bad: a { color: #333; text-decoration: none; }

- Accessibility Considerations:

• Screen reader users rely on context; visual users rely on consistent indicators.

• Colorblind users need redundant indicators like underlines or bolding.

• Visible focus indicators are essential for users navigating by keyboard.

- Best Practices:

• Maintain consistent link styling throughout the site.

• Ensure focus outlines are not disabled by CSS (avoid outline: none).

• Provide adequate hit areas for users with motor impairments.

Domain I: Creating Accessible Web Solutions

Subsection: Links → Skip Links and In-Page Links

- Purpose:

• Skip links and in-page links allow users to bypass repetitive content and jump to key sections.

• They are critical for keyboard and screen reader users who otherwise must navigate through long menus or repeated elements.

- Guidelines:

• Provide a visible or focusable “Skip to main content” link at the top of the page.

• Skip links MUST be keyboard operable and appear when focused.

• In-page links (anchors) MUST have descriptive text indicating the destination (e.g., “Jump to form section”).

• Ensure target sections have appropriate landmarks or headings to provide meaningful context.

• Avoid using vague targets like <a href="#top">Back to top</a> without clear context.

- Examples:

• Good: <a href="#maincontent" class="skip-link">Skip to main content</a>

<main id="maincontent">…</main>

• Bad: <a href="#content">Click here</a>

- Accessibility Considerations:

• Screen readers announce skip links as part of navigation, allowing users to move efficiently.

• Keyboard users save time by jumping directly to main sections.

• Landmarks (<main>, <nav>, <header>, <footer>, <aside>) enhance the effectiveness of skip and in-page links.

- Best Practices:

• Ensure skip links are visible when tabbed to but not distracting visually when unfocused.

• Provide multiple skip options on long or complex pages (e.g., skip to navigation, skip to search).

• Test with keyboard and screen readers to verify correct focus movement.

Domain I: Creating Accessible Web Solutions

Subsection: Links → Summary

- Accessible links ensure users can navigate and interact with content effectively.

- Key requirements:

• Link text MUST be descriptive and convey purpose without relying on surrounding context.

• Avoid vague labels like “click here” or “read more.”

• Links MUST be visually distinct from regular text, not relying on color alone.

• Focus indicators MUST be visible for keyboard users.

• Visited links SHOULD appear visually distinct to aid orientation.

• Skip links and in-page links MUST be provided to allow users to bypass repetitive content.

• Landmarks and headings SHOULD complement skip links for efficient navigation.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define link accessibility requirements.

• Section 508 and EN 301 549 provide harmonized global standards.

- Best Practices:

• Keep link text short, specific, and consistent.

• Indicate file type/size in links when relevant (e.g., PDF, DOCX, 2 MB).

• Ensure skip links are visible on focus and correctly move focus to the target.

• Maintain consistent link styling across the site.

• Test with keyboard-only navigation and screen readers.

- Key Takeaway: Accessible links balance clarity, consistency, and efficiency, supporting navigation for all users, especially those with disabilities.

Domain I: Creating Accessible Web Solutions

Subsection: Headings → Introduction

- Purpose:

• Headings provide structure, organization, and hierarchy to web content.

• They allow users—especially screen reader and keyboard users—to navigate quickly and understand page layout.

• Inaccessible heading structures can confuse users and make content difficult to scan.

- Intended audience: web developers, designers, and content authors.

- Technical level: easy.

- Prerequisites: knowledge of semantic HTML and landmarks.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible heading structures.

• WAI-ARIA provides supplemental guidance for dynamic content.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Headings → Structure and Hierarchy

- Purpose:

• Proper heading structure communicates the outline of a page.

• Users of screen readers and other assistive technologies rely on headings for orientation and navigation.

- Guidelines:

• Use <h1> through <h6> elements to represent page hierarchy.

• A page SHOULD have a single <h1> that represents the main topic or purpose.

• Headings MUST follow a logical nesting order (e.g., <h2> under <h1>, <h3> under <h2>).

• Do NOT skip heading levels unnecessarily (e.g., jumping from <h1> to <h4>).

• Do NOT use headings purely for visual styling; use CSS for appearance.

• Headings MUST be descriptive and concise, reflecting the section content.

- Examples:

• Good:

<h1>Annual Accessibility Report</h1>

<h2>Executive Summary</h2>

<h2>Findings</h2>

<h3>WCAG Compliance</h3>

• Bad:

<h1>Report</h1>

<h4>Summary</h4>

<h2>Details</h2>

- Accessibility Considerations:

• Screen readers allow users to navigate by headings; incorrect hierarchy disrupts this flow.

• Headings improve readability and scanning for all users, including those with cognitive disabilities.

• Multiple <h1> elements are allowed in HTML5, but a single clear <h1> is best for accessibility.

- Best Practices:

• Outline the page structure before coding.

• Keep headings short, specific, and free of jargon.

• Test with screen readers to ensure logical navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Headings → Visual Presentation

- Purpose:

• Headings must be visually distinct to support all users, including those with cognitive or learning disabilities.

• Clear visual presentation reinforces the structural hierarchy for sighted users.

- Guidelines:

• Use CSS to style headings so they stand out from body text (e.g., larger font size, bold, spacing).

• Maintain consistency in heading styles across the site.

• Ensure sufficient contrast between headings and background (WCAG requires at least 4.5:1 for normal text, 3:1 for large text).

• Do not rely solely on color to distinguish headings from text.

• Avoid using ALL CAPS for long headings, as it reduces readability.

- Accessibility Considerations:

• Users with low vision or dyslexia benefit from clear, distinct, and consistent heading styles.

• Visual separation helps users scan and orient themselves in content.

• Poor styling (e.g., headings indistinguishable from body text) reduces usability for everyone.

- Best Practices:

• Match visual styling to logical structure (e.g., <h1> largest, <h2> smaller, etc.).

• Use white space and margins to separate headings from surrounding text.

• Do not fake headings with bold or enlarged body text—use semantic heading tags.

• Test readability on multiple devices and screen sizes.

Domain I: Creating Accessible Web Solutions

Subsection: Headings → Summary

- Accessible headings provide structure, orientation, and clarity for all users.

- Key requirements:

• Use semantic HTML (<h1>–<h6>) to define structure, not just visual styling.

• A page SHOULD include one clear <h1> to represent the main purpose.

• Headings MUST follow logical nesting without skipping levels.

• Headings MUST be descriptive and concise.

• Headings MUST be visually distinct from surrounding text with consistent styling.

• Ensure adequate color contrast between headings and background.

- Accessibility Benefits:

• Screen reader users can navigate efficiently by headings.

• Cognitive and low-vision users benefit from clear, consistent hierarchy and presentation.

• Well-structured headings improve readability, scannability, and SEO.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define heading accessibility requirements.

• Section 508 and EN 301 549 align internationally with these standards.

- Best Practices:

• Outline page structure before implementation.

• Style headings consistently with CSS to reflect hierarchy.

• Avoid using bold body text or visual tricks as a substitute for headings.

• Test with screen readers and sighted users to verify both structure and readability.

- Key Takeaway: Proper headings enhance usability, navigation, and comprehension, ensuring equitable access to content.

Domain I: Creating Accessible Web Solutions

Subsection: Landmarks → Introduction

- Purpose:

• Landmarks define key regions of a web page, helping users navigate efficiently.

• They allow screen reader users to jump directly to sections such as navigation, main content, and search.

• Without landmarks, users must navigate linearly, which can be time-consuming and confusing.

- Intended audience: web developers and designers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of semantic HTML5 elements and ARIA basics.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible page structures.

• WAI-ARIA provides additional support for custom landmarks.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Landmarks → Types of Landmarks

- Purpose:

• Landmarks provide programmatic regions that assistive technology users can navigate quickly.

• HTML5 elements and ARIA roles define the primary landmark types.

- Native HTML5 Landmarks:

• <header> → introductory content, typically site or section header.

• <nav> → navigation links.

• <main> → primary page content (MUST appear only once per page).

• <aside> → tangentially related content (e.g., sidebar, related links).

• <footer> → page or section footer.

• <form> → input region (SHOULD be labeled).

• <section> → thematic grouping of content (SHOULD have an accessible name).

- ARIA Landmarks (when native HTML is not available):

• role="banner" → equivalent of <header>.

• role="navigation" → equivalent of <nav>.

• role="main" → equivalent of <main>.

• role="complementary" → equivalent of <aside>.

• role="contentinfo" → equivalent of <footer>.

• role="form" → equivalent of <form>.

• role="region" → equivalent of <section>, MUST include accessible name (aria-label or aria-labelledby).

• role="search" → identifies a search region.

- Accessibility Considerations:

• Use native HTML5 landmarks where possible; fall back to ARIA only if necessary.

• Each landmark SHOULD have a unique label when multiple of the same type exist (e.g., “Main navigation,” “Footer links”).

• Avoid overuse—too many landmarks may overwhelm users.

- Best Practices:

• Provide at least one <main> landmark per page.

• Label forms and regions clearly (e.g., “Contact Form,” “Related Articles”).

• Test navigation using screen reader landmark shortcuts.

Domain I: Creating Accessible Web Solutions

Subsection: Landmarks → Best Practices and Common Mistakes

- Best Practices:

• Use native HTML5 elements (<main>, <nav>, <header>, <footer>, <aside>, <section>, <form>) instead of ARIA roles whenever possible.

• Ensure each page has exactly one <main> landmark to identify the primary content.

• Provide unique labels when multiple landmarks of the same type exist (e.g., “Primary Navigation,” “Footer Navigation”).

• Use <section> or role="region" only when content grouping needs an accessible name and navigation benefit.

• Keep landmark usage consistent across the site for predictable navigation.

- Common Mistakes:

• Using landmarks inconsistently across pages, confusing users.

• Nesting multiple <main> landmarks on a single page.

• Overusing <section> without accessible names, creating meaningless regions.

• Omitting labels for forms or regions, leaving screen reader users with generic names like “form” or “region.”

• Applying ARIA landmarks redundantly to native HTML elements (e.g., <nav role="navigation">).

- Accessibility Considerations:

• Screen reader users rely on landmark shortcuts to move through content efficiently.

• Consistent labeling improves usability for users with cognitive and memory-related disabilities.

• Keyboard users benefit indirectly from logical, consistent structure.

- Testing Strategies:

• Use screen reader shortcut keys to navigate by landmarks.

• Verify that landmarks are announced with clear and unique labels.

• Check that page outline makes sense and avoids redundant or missing regions.

Domain I: Creating Accessible Web Solutions

Subsection: Landmarks → Summary

- Accessible landmarks provide structure and navigation shortcuts for assistive technology users.

- Key requirements:

• Use native HTML5 landmarks (<main>, <nav>, <header>, <footer>, <aside>, <section>, <form>) whenever possible.

• Only one <main> landmark MUST appear per page.

• Landmarks of the same type SHOULD have unique labels (e.g., “Main Navigation,” “Footer Links”).

• <section> or role="region" SHOULD only be used when content requires an accessible name.

• Avoid redundant ARIA roles when native semantics already exist.

• Keep landmark usage consistent across pages.

- Accessibility Benefits:

• Screen reader users can navigate directly to landmarks via shortcuts.

• Cognitive and low-vision users benefit from consistent structure and labeling.

• Consistent use of landmarks improves orientation and reduces navigation effort.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for programmatic page structure.

• Section 508 and EN 301 549 align globally with these standards.

- Best Practices:

• Outline and label page regions before coding.

• Test with screen reader landmark navigation to ensure clarity.

• Keep the number of landmarks manageable to avoid overwhelming users.

• Ensure all forms and regions are labeled meaningfully.

- Key Takeaway: Proper landmark usage provides clear structure, efficient navigation, and consistency, making web content more accessible and user-friendly.

Domain I: Creating Accessible Web Solutions

Subsection: Lists → Introduction

- Purpose:

• Lists communicate groups of related items in a structured way.

• When coded properly, lists are announced by screen readers with the number of items and relationships.

• Inaccessible lists confuse users and reduce comprehension, especially for screen reader and keyboard users.

- Intended audience: developers, designers, and content authors.

- Technical level: easy.

- Prerequisites: knowledge of semantic HTML and headings.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible use of lists.

• WAI-ARIA techniques provide supplemental guidance but SHOULD NOT replace semantic HTML.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Lists → Ordered and Unordered Lists

- Purpose:

• Ordered (<ol>) and unordered (<ul>) lists communicate structured groups of items.

• Screen readers announce list type, number of items, and allow navigation through items.

- Guidelines:

• Use <ul> for unordered collections and <ol> when sequence or hierarchy matters.

• Each list item MUST be wrapped in <li>.

• Avoid using lists purely for layout or visual indentation.

• Nested lists SHOULD be used only when logical and meaningful.

- Examples:

• Good (unordered):

<ul>

<li>Apples</li>

<li>Oranges</li>

<li>Bananas</li>

</ul>

• Good (ordered):

<ol>

<li>Preheat oven</li>

<li>Mix ingredients</li>

<li>Bake for 30 minutes</li>

</ol>

• Bad:

<p>\* Apples<br>\* Oranges<br>\* Bananas</p>

(Improper structure, not announced as a list.)

- Accessibility Considerations:

• Screen readers announce “list with X items” before reading.

• Improperly coded lists read as plain text, losing structure.

• Lists aid cognitive comprehension by grouping related items.

- Best Practices:

• Always use semantic list markup rather than styling text to look like a list.

• Keep lists concise and avoid unnecessary nesting.

• Use descriptive headings before long lists for clarity.

Domain I: Creating Accessible Web Solutions

Subsection: Lists → Definition Lists

- Purpose:

• Definition lists (<dl>) pair terms with their descriptions.

• They are useful for glossaries, metadata, or name-value relationships.

• When coded correctly, screen readers announce the term and its description in context.

- Guidelines:

• Use <dl> to wrap the entire list.

• Use <dt> for the term (definition term).

• Use <dd> for the description (definition description).

• Do not misuse <dl> for layout purposes.

• Keep associations clear—each <dt> may have one or more <dd> elements.

- Example:

<dl>

<dt>CPU</dt>

<dd>Central Processing Unit, the primary component of a computer that processes instructions.</dd>

<dt>RAM</dt>

<dd>Random Access Memory, short-term data storage used by active processes.</dd>

</dl>

- Accessibility Considerations:

• Screen readers announce “term” and “definition,” providing context.

• Misusing <dl> for layout causes confusion and meaningless announcements.

• Ensure terms and definitions are concise and paired logically.

- Best Practices:

• Use definition lists only when a clear name–value relationship exists.

• Avoid long or complex definitions—supplement with links if necessary.

• Provide headings for large glossaries to improve navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Lists → Summary

- Accessible lists provide structure, grouping, and clarity for users.

- Key requirements:

• Use semantic list markup (<ul>, <ol>, <dl>) for all lists.

• <li> MUST wrap each item in ordered and unordered lists.

• <dt> and <dd> MUST be used for definition lists.

• Do not simulate lists with plain text and symbols (e.g., \* or -).

• Avoid using lists purely for layout or decoration.

• Nest lists only when relationships are meaningful.

- Accessibility Benefits:

• Screen readers announce list type and number of items.

• Properly coded lists improve navigation, comprehension, and usability.

• Users with cognitive disabilities benefit from structured grouping of related items.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define semantic requirements for lists.

• Section 508 and EN 301 549 align with these requirements.

- Best Practices:

• Keep lists concise and relevant.

• Use headings or labels for long lists to improve orientation.

• Test lists with screen readers to ensure structure is preserved.

- Key Takeaway: Lists must be coded semantically to preserve meaning and accessibility, enhancing comprehension and navigation for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Introduction

- Purpose:

• Navigation is essential for orienting users and helping them move efficiently through a site.

• Inaccessible navigation can create barriers for screen reader, keyboard, and cognitive users.

- Intended audience: developers, designers, and content authors.

- Technical level: moderate.

- Prerequisites: knowledge of semantic HTML, landmarks, and links.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible navigation.

• WAI-ARIA provides supplemental techniques for custom navigation components.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Navigation Mechanisms

- Purpose:

• Multiple navigation mechanisms improve orientation and usability for diverse users.

• Users with disabilities benefit from predictable, consistent, and flexible navigation options.

- Guidelines:

• Provide at least two navigation mechanisms for locating content (e.g., menu + search, site map + breadcrumbs).

• Navigation MUST be consistent across pages in terms of structure, order, and labeling.

• Menus SHOULD use semantic markup (<nav>, <ul>, <li>, <a>) for clarity.

• Avoid JavaScript-only navigation without accessible fallback.

• Provide descriptive labels for all navigation links.

- Types of Navigation Mechanisms:

• Primary navigation (main menu)

• Secondary navigation (sub-menus, sidebars)

• Breadcrumb trails

• Site maps

• Search functionality

• Pagination and next/previous links

• Skip links and in-page anchors

- Accessibility Considerations:

• Screen reader users rely on consistent navigation order to build mental models.

• Keyboard users require logical tab order and visible focus states.

• Cognitive users benefit from predictable navigation and redundant options.

- Best Practices:

• Use headings and landmarks to supplement navigation.

• Provide skip links to bypass repetitive navigation.

• Ensure search functionality is accessible and supports autocomplete carefully.

• Keep navigation concise, avoiding clutter and overload.

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Menus

- Purpose:

• Menus provide structured access to site sections and must be accessible to all users.

• Poorly coded menus can block navigation for screen reader or keyboard users.

- Guidelines:

• Use semantic markup (<nav>, <ul>, <li>, <a>) to build menus.

• All menu items MUST be keyboard operable, including sub-menus.

• Visible focus indicators MUST be provided for keyboard navigation.

• Expandable menus SHOULD use ARIA roles and states (aria-haspopup, aria-expanded, aria-controls).

• Use logical tab order; focus MUST move predictably through menu items.

• Provide clear, descriptive link text for each menu item.

• Avoid hover-only activation; ensure menus open via keyboard actions (Enter, Space, Arrow keys).

- Example (Accessible Menu Structure):

<nav>

<ul>

<li><a href="home.html">Home</a></li>

<li><a href="about.html">About Us</a></li>

<li>

<button aria-haspopup="true" aria-expanded="false">Services</button>

<ul>

<li><a href="design.html">Design</a></li>

<li><a href="development.html">Development</a></li>

</ul>

</li>

</ul>

</nav>

- Accessibility Considerations:

• Screen readers announce ARIA states for expandable menus.

• Keyboard users rely on arrow key navigation for sub-menus.

• Mobile users require large touch targets and consistent menu structures.

- Best Practices:

• Test menus with screen readers, keyboard, and touch devices.

• Keep menu structures simple and shallow to reduce complexity.

• Ensure ARIA attributes are updated dynamically as menus expand/collapse.

• Provide fallback options if JavaScript fails.

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Breadcrumbs

- Purpose:

• Breadcrumbs provide users with orientation and a way to move back through a site’s hierarchy.

• They are especially helpful for users with cognitive disabilities or those navigating deep site structures.

- Guidelines:

• Use an ordered list (<ol>) to represent breadcrumbs semantically.

• Each breadcrumb item SHOULD be a link except the current page, which MUST be plain text or marked with aria-current="page".

• Breadcrumbs MUST appear consistently across the site in the same location.

• Separators (e.g., “>”) SHOULD be implemented via CSS or aria-hidden="true" to avoid redundancy for screen readers.

- Example:

<nav aria-label="Breadcrumb">

<ol>

<li><a href="home.html">Home</a></li>

<li><a href="products.html">Products</a></li>

<li><a href="electronics.html">Electronics</a></li>

<li aria-current="page">Televisions</li>

</ol>

</nav>

- Accessibility Considerations:

• Screen readers announce breadcrumb navigation clearly when marked with aria-label.

• aria-current ensures users know their current location in the hierarchy.

• Breadcrumbs reduce cognitive load by providing consistent orientation.

- Best Practices:

• Keep breadcrumb labels concise and descriptive.

• Ensure breadcrumbs supplement but do not replace primary navigation.

• Test with screen readers to confirm correct announcement of hierarchy.

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Pagination

- Purpose:

• Pagination allows users to navigate through multi-page content such as search results or article series.

• Accessible pagination ensures screen reader, keyboard, and cognitive users can efficiently move between pages.

- Guidelines:

• Use a list structure (<ul> or <ol>) for pagination controls.

• Each pagination control MUST be a link or button with descriptive text (e.g., “Next page,” “Page 2”).

• The current page MUST be indicated using aria-current="page" or visually distinct styling.

• Provide both “Previous” and “Next” controls for sequential navigation.

• Do not rely solely on symbols (e.g., “<” or “>”) without accessible names.

- Example:

<nav aria-label="Pagination">

<ul>

<li><a href="page1.html">1</a></li>

<li aria-current="page">2</li>

<li><a href="page3.html">3</a></li>

<li><a href="page3.html" rel="next">Next</a></li>

</ul>

</nav>

- Accessibility Considerations:

• Screen readers announce pagination controls as a list of links.

• aria-current clarifies which page is active.

• Keyboard users need logical tab order and visible focus indicators.

• Users with cognitive disabilities benefit from predictable labels and navigation flow.

- Best Practices:

• Keep pagination concise (avoid displaying dozens of page numbers at once).

• Provide “First” and “Last” controls for large result sets.

• Ensure pagination works consistently across screen sizes and devices.

• Test pagination with screen readers and keyboard-only navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Navigation → Summary

- Accessible navigation ensures users can move efficiently, stay oriented, and reach content regardless of disability.

- Key requirements:

• Provide at least two navigation mechanisms (e.g., menu + search, breadcrumbs + site map).

• Menus MUST be semantic, keyboard operable, and screen reader accessible.

• Breadcrumbs MUST use ordered lists, with the current page indicated using aria-current.

• Pagination MUST provide descriptive labels and indicate the current page.

• Navigation MUST be consistent across all pages in structure, order, and labeling.

• Skip links and landmarks SHOULD supplement navigation for efficiency.

- Accessibility Benefits:

• Screen reader users can navigate predictably with consistent structures.

• Keyboard users can reach all navigation elements with visible focus.

• Cognitive users benefit from multiple navigation options and clear orientation.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require multiple navigation mechanisms and accessible structures.

• Section 508 and EN 301 549 harmonize with these requirements.

- Best Practices:

• Keep navigation clear, concise, and uncluttered.

• Ensure labels are descriptive, consistent, and free of jargon.

• Test menus, breadcrumbs, and pagination with screen readers and keyboards.

• Provide redundant navigation methods for complex sites.

- Key Takeaway: Accessible navigation combines consistency, flexibility, and clarity, ensuring users of all abilities can explore and understand site content.

Domain I: Creating Accessible Web Solutions

Subsection: Color and Contrast → Introduction

- Purpose:

• Color and contrast affect readability, usability, and accessibility for users with low vision, color blindness, or cognitive disabilities.

• Inaccessible color choices can render text, controls, and critical information unusable.

- Intended audience: designers, developers, and content authors.

- Technical level: easy to moderate.

- Prerequisites: knowledge of CSS styling and WCAG color contrast requirements.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessible color and contrast practices.

• WAI-ARIA provides additional support for visual indicators.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Color and Contrast → Contrast Requirements

- Purpose:

• Adequate color contrast ensures text and important visual elements are perceivable by users with low vision or color deficiencies.

• WCAG defines minimum contrast ratios to ensure legibility.

- Guidelines:

• Text and images of text MUST have a contrast ratio of at least:

- 4.5:1 for normal text.

- 3:1 for large text (18pt or 14pt bold and above).

• Non-text UI components (buttons, icons, form fields) and graphical objects MUST have a contrast ratio of at least 3:1 against adjacent colors.

• Text over gradients, images, or patterns MUST maintain required contrast in all areas.

• Placeholder text MUST meet contrast requirements if it conveys important information.

• Contrast requirements apply to both default and hover/focus states.

- Testing Methods:

• Automated tools: Axe, WAVE, Lighthouse, Colour Contrast Analyser.

• Manual checks with eyedropper and contrast ratio calculators.

• Test across multiple devices, browsers, and system settings.

- Accessibility Considerations:

• Users with low vision or color blindness rely on sufficient contrast to distinguish text and controls.

• Inadequate contrast can make essential content invisible or difficult to use.

• High contrast modes in operating systems may override site colors—design should not break in these modes.

- Best Practices:

• Use high-contrast color palettes by default.

• Avoid relying on subtle shades or low contrast for aesthetics.

• Test early in the design process to prevent costly rework.

Domain I: Creating Accessible Web Solutions

Subsection: Color and Contrast → Use of Color

- Purpose:

• Color alone must not be the sole method for conveying information, prompting action, or distinguishing elements.

• Users with color blindness or low vision may not perceive color differences.

- Guidelines:

• Information conveyed by color MUST also be available through text, icons, patterns, or other indicators.

• Error states (e.g., red borders) MUST include text or icons describing the error.

• Required fields MUST use text (e.g., “required”) or symbols with explanations, not just color.

• Graphs and charts SHOULD use patterns, textures, or labels in addition to color coding.

• Links MUST be distinguishable from surrounding text by more than color alone (e.g., underline).

• Color indicators MUST meet contrast requirements for visibility.

- Examples:

• Bad (color only): <input style="border: 2px solid red;">

(Relies only on red border for error state.)

• Good: <input aria-describedby="error1" style="border: 2px solid red;">

<span id="error1">Error: Please enter a valid email.</span>

- Accessibility Considerations:

• Users with red-green color blindness may miss error messages relying only on color.

• Users with cognitive disabilities benefit from redundant cues such as text labels or icons.

- Best Practices:

• Combine color with text, shapes, or patterns.

• Label graph data points or provide accessible data tables as alternatives.

• Test with grayscale or color-blindness simulators to confirm usability.

Domain I: Creating Accessible Web Solutions

Subsection: Color and Contrast → Visual Indicators and Focus

- Purpose:

• Visual indicators such as focus outlines, hover styles, and active states help users understand interaction points.

• Removing or weakening these indicators harms accessibility for keyboard and low-vision users.

- Guidelines:

• Focus indicators MUST be visible and distinct from the default state.

• Color contrast between focus indicator and background MUST be at least 3:1.

• Hover and active states SHOULD provide additional styling (e.g., underline, bold, background change).

• Do NOT remove outlines with CSS (e.g., outline: none) unless replaced with an accessible alternative.

• Links MUST be visually distinguishable in all states.

• Buttons, form fields, and custom controls MUST include clear focus and hover indicators.

- Examples:

• Bad: a:focus { outline: none; }

• Good: a:focus { outline: 2px solid #005fcc; }

- Accessibility Considerations:

• Keyboard-only users rely entirely on focus indicators to track navigation.

• Users with cognitive disabilities benefit from consistent visual cues.

• Low-vision users need high-contrast, highly visible indicators.

- Best Practices:

• Ensure focus styles are consistent across all browsers and devices.

• Use more than color alone for state changes (e.g., underline + color).

• Test keyboard navigation on every interactive element.

• Keep indicator styles consistent across site components.

Domain I: Creating Accessible Web Solutions

Subsection: Color and Contrast → Summary

- Accessible color and contrast ensure that content and interactive elements are perceivable and usable by all users.

- Key requirements:

• Text contrast MUST meet WCAG minimums (4.5:1 normal, 3:1 large).

• Non-text UI elements and graphics MUST meet at least 3:1 contrast.

• Information MUST NOT rely on color alone—provide text, patterns, or icons as alternatives.

• Focus, hover, and active states MUST have clear, visible indicators.

• Outlines MUST NOT be removed unless replaced with accessible alternatives.

- Accessibility Benefits:

• Users with low vision, color blindness, or cognitive disabilities can access content without barriers.

• Clear indicators help keyboard and screen reader users track interactions.

• Strong color contrast improves usability for all users, even in bright light or on poor displays.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define color and contrast requirements.

• Section 508 and EN 301 549 mandate compliance internationally.

- Best Practices:

• Design with high contrast from the start instead of retrofitting.

• Use redundant indicators (icons, labels, patterns) in addition to color.

• Test with color-blindness simulators, grayscale, and high-contrast modes.

• Validate color contrast with automated tools and manual testing.

- Key Takeaway: Accessible use of color and contrast ensures critical information is perceivable, interactive elements are clear, and navigation is inclusive for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Introduction

- Purpose:

• Multimedia (audio, video, animations) is widely used for communication and engagement.

• Without accessibility features, multimedia can exclude users with hearing, vision, or cognitive disabilities.

- Intended audience: developers, designers, content creators, and QA testers.

- Technical level: moderate.

- Prerequisites: knowledge of HTML5 media elements and ARIA basics.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible multimedia.

• WAI-ARIA and W3C Media Accessibility User Requirements provide supplemental guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Captions

- Purpose:

• Captions provide a synchronized text alternative for spoken dialogue and important sounds in video/audio content.

• They are essential for users who are deaf or hard of hearing, and they also benefit users in noisy or quiet environments.

- Guidelines:

• Captions MUST include all spoken dialogue and relevant non-speech sounds (e.g., [laughter], [music]).

• Captions MUST be synchronized with the audio.

• Captions SHOULD be accurate, complete, and properly punctuated.

• Auto-generated captions MUST be reviewed and corrected for accuracy.

• Provide closed captions (toggleable) rather than only open captions (always visible) unless required.

• Captions MUST be available for both prerecorded and live content (live captions may be slightly delayed).

- Accessibility Considerations:

• Captions support users who cannot hear or who have difficulty processing audio.

• Captions also benefit users who are non-native speakers or who process information better visually.

• Captions support search and indexing of multimedia content.

- Best Practices:

• Ensure captions do not obscure important visual content.

• Provide options to customize caption appearance (font size, color, background).

• Test captions for synchronization and readability across devices.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Transcripts

- Purpose:

• Transcripts provide a complete text alternative for audio and video content.

• They are essential for users who are deaf-blind and beneficial for those who prefer reading or need searchable text.

- Guidelines:

• Transcripts MUST include all spoken content and relevant non-speech information (e.g., [applause], [dog barking]).

• Transcripts SHOULD be provided for both audio-only and video content.

• Transcripts MUST be easy to locate and access near the media player.

• Interactive transcripts (synchronized with playback) are strongly recommended when feasible.

• Transcripts SHOULD identify speakers when multiple people are involved.

- Accessibility Considerations:

• Transcripts support users who cannot access audio or video at all.

• They allow content to be searchable, indexable, and easier to translate.

• Transcripts benefit users in environments where playing media is not possible (quiet zones, low bandwidth).

- Best Practices:

• Provide downloadable transcripts in accessible formats (HTML, TXT, DOCX).

• Keep transcripts accurate, well-formatted, and easy to read.

• Use plain language and consistent speaker labeling.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Audio Description

- Purpose:

• Audio description provides spoken narration of important visual information in video content.

• It ensures users who are blind or have low vision can understand on-screen actions, settings, and context.

- Guidelines:

• Audio description MUST convey essential visual information not available through dialogue.

• Extended audio description SHOULD be provided when pauses in dialogue are insufficient.

• For prerecorded media, audio descriptions MUST be synchronized with video playback.

• Provide text-based descriptions as an alternative when audio description is not possible.

• Avoid redundancy—only describe information not already clear from dialogue.

- Examples of Information to Describe:

• Actions (e.g., “She opens the envelope and smiles.”)

• Settings (e.g., “A crowded city street at night.”)

• Visual details essential to understanding (e.g., “The warning light flashes red.”)

- Accessibility Considerations:

• Blind and low-vision users rely on audio description to access visual-only information.

• Audio description enhances comprehension for users with cognitive disabilities.

• Optional audio description tracks provide flexibility without disrupting other users.

- Best Practices:

• Integrate audio description into production workflows early.

• Provide both standard and extended versions where possible.

• Test with blind users to ensure descriptions are useful and not distracting.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Media Players

- Purpose:

• Media players must provide accessible controls and support captions, transcripts, and audio descriptions.

• Inaccessible players prevent users with disabilities from operating or understanding multimedia content.

- Guidelines:

• All controls (play, pause, volume, captions, fullscreen) MUST be keyboard operable.

• Controls MUST have accessible names and roles (using native HTML or ARIA where needed).

• Captions and audio description tracks MUST be supported.

• Focus MUST be visible and move predictably between controls.

• Provide customization options for captions (font size, color, background).

• Auto-play SHOULD be avoided; if used, users MUST be able to stop/pause immediately.

- Accessibility Considerations:

• Screen readers must announce controls clearly.

• Keyboard users rely on logical tab/arrow navigation for operation.

• Users with cognitive disabilities benefit from simple, consistent interfaces.

• Mobile users need large, accessible touch targets.

- Best Practices:

• Use native HTML5 <video> and <audio> elements where possible.

• Ensure third-party players meet WCAG requirements.

• Provide alternative download links when streaming controls are inaccessible.

• Test with multiple assistive technologies, browsers, and devices.

Domain I: Creating Accessible Web Solutions

Subsection: Multimedia → Summary

- Accessible multimedia ensures that audio and video content is usable by people with hearing, vision, and cognitive disabilities.

- Key requirements:

• Captions MUST include all spoken dialogue and important non-speech sounds.

• Transcripts MUST be provided for audio and video, including non-speech information.

• Audio descriptions MUST convey essential visual information in video.

• Media players MUST support captions, transcripts, audio descriptions, and be fully keyboard accessible.

• Auto-play MUST NOT interfere with user control; users MUST be able to pause/stop easily.

- Accessibility Benefits:

• Deaf and hard-of-hearing users can follow content via captions and transcripts.

• Blind and low-vision users gain access to visual information via audio description.

• All users benefit from transcripts for searchability, translation, and reference.

• Keyboard and screen reader users can operate accessible players independently.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define multimedia accessibility requirements.

• Section 508 and EN 301 549 mandate multimedia accessibility internationally.

- Best Practices:

• Integrate captions, transcripts, and descriptions into production workflows.

• Avoid reliance on auto-generated captions without human review.

• Provide alternative download options when players are inaccessible.

• Test multimedia with multiple assistive technologies and user groups.

- Key Takeaway: Accessible multimedia requires captions, transcripts, audio descriptions, and operable players, ensuring inclusive access to rich media experiences.

Domain I: Creating Accessible Web Solutions

Subsection: Content Structure and Semantics → Introduction

- Purpose:

• Proper content structure and semantics ensure that information is conveyed meaningfully to all users, including those using assistive technologies.

• Inaccessible structures confuse navigation, disrupt reading order, and reduce comprehension.

- Intended audience: developers, designers, and content authors.

- Technical level: moderate.

- Prerequisites: knowledge of HTML, headings, landmarks, and ARIA basics.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require semantic content structures.

• WAI-ARIA provides supplemental techniques for complex or dynamic content.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Content Structure and Semantics → Semantic HTML

- Purpose:

• Semantic HTML provides meaning and context to content beyond visual presentation.

• Assistive technologies rely on semantic elements to announce roles, states, and relationships.

- Guidelines:

• Use native HTML elements for their intended purpose (e.g., <button> for actions, <a> for links, <table> for tabular data).

• Avoid using generic containers (<div>, <span>) when semantic elements exist.

• Headings (<h1>–<h6>) MUST reflect document structure and hierarchy.

• Lists (<ul>, <ol>, <dl>) MUST represent groups of related items.

• Forms MUST use <label>, <fieldset>, <legend> for semantic relationships.

• Sectioning elements (<header>, <main>, <footer>, <article>, <section>, <aside>, <nav>) SHOULD be used to provide landmarks and structure.

• Use <figure> and <figcaption> for images requiring captions.

• Use <blockquote>, <cite>, <abbr>, <time>, and other semantic elements when applicable.

- Accessibility Considerations:

• Screen readers announce semantics, enabling efficient navigation and orientation.

• Semantic markup improves interoperability with voice input, braille, and AI tools.

• Using semantic HTML reduces the need for ARIA, lowering complexity and error risk.

- Best Practices:

• Always prefer semantic HTML over ARIA for accessibility.

• Validate markup with tools such as the W3C Validator.

• Test with screen readers to confirm semantic elements are conveyed correctly.

• Maintain consistency in use of semantics across the site.

Domain I: Creating Accessible Web Solutions

Subsection: Content Structure and Semantics → ARIA Usage

- Purpose:

• Accessible Rich Internet Applications (ARIA) attributes provide roles, states, and properties to enhance accessibility when native HTML is insufficient.

• ARIA should supplement—not replace—semantic HTML.

- Guidelines:

• Follow the principle: “Use native HTML first, ARIA if necessary.”

• ARIA MUST NOT override or duplicate native semantics (e.g., <button role="button"> is redundant).

• Required ARIA attributes (aria-label, aria-labelledby, aria-describedby) MUST provide meaningful names and descriptions.

• Widgets (accordions, modals, sliders) MUST use ARIA roles and states when no native HTML equivalent exists.

• Use aria-live for dynamic content updates that MUST be announced by screen readers.

• Use aria-hidden="true" only when content should be excluded from assistive technologies.

• Avoid misuse of role="presentation" or role="none" on meaningful elements.

- Accessibility Considerations:

• Incorrect ARIA use can harm accessibility more than help.

• Screen readers rely on ARIA states (expanded, selected, checked) to communicate dynamic behavior.

• Overuse of ARIA increases maintenance complexity and testing requirements.

- Best Practices:

• Apply ARIA only when no semantic HTML alternative exists.

• Keep ARIA markup consistent and minimal.

• Test with multiple screen readers to confirm ARIA is announced correctly.

• Reference WAI-ARIA Authoring Practices for design patterns and examples.

Domain I: Creating Accessible Web Solutions

Subsection: Content Structure and Semantics → Reading and Navigation Order

- Purpose:

• The reading and navigation order must match the visual and logical structure of the page.

• Inconsistent or illogical order disrupts comprehension and navigation for screen reader and keyboard users.

- Guidelines:

• DOM order MUST reflect logical reading order (left to right, top to bottom in most languages).

• Visual presentation using CSS (flex, grid, positioning) MUST NOT break logical reading order.

• Tab order MUST follow a logical sequence through interactive elements.

• Use tabindex sparingly; avoid tabindex values greater than 0 unless absolutely necessary.

• Landmarks and headings SHOULD supplement navigation order for clarity.

• Hidden elements (e.g., aria-hidden, display:none) MUST be excluded from navigation.

- Accessibility Considerations:

• Screen readers read DOM order, not visual layout.

• Keyboard users navigate in tab order; incorrect sequencing creates barriers.

• Cognitive users rely on predictable navigation flow.

- Best Practices:

• Check reading order by navigating with a screen reader.

• Test keyboard tabbing order visually to ensure logical flow.

• Avoid excessive use of tabindex; prefer natural DOM flow.

• Keep interactive elements grouped logically and consistently.

Domain I: Creating Accessible Web Solutions

Subsection: Content Structure and Semantics → Summary

- Accessible content structure ensures that information is conveyed logically and consistently across technologies.

- Key requirements:

• Semantic HTML MUST be used to define meaning and relationships (e.g., headings, lists, tables, forms).

• ARIA SHOULD supplement, not replace, native HTML.

• Reading and navigation order MUST match the logical and visual order of content.

• Tab order MUST follow a predictable, logical sequence without unnecessary overrides.

• Landmarks and headings SHOULD be used to enhance navigation and comprehension.

- Accessibility Benefits:

• Screen reader users can rely on consistent semantics for orientation and navigation.

• Keyboard users benefit from logical focus and tab order.

• Cognitive and low-vision users gain clarity through predictable structures.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA include requirements for semantic structure, ARIA use, and logical order.

• Section 508 and EN 301 549 align with these international standards.

- Best Practices:

• Use semantic HTML whenever possible, resorting to ARIA only when needed.

• Keep reading and navigation order consistent with design and intent.

• Validate structure with automated tools and test manually with assistive technologies.

• Document coding standards for semantics and enforce them during QA.

- Key Takeaway: Proper structure and semantics ensure that content is accessible, navigable, and meaningful for all users, regardless of technology.

Domain I: Creating Accessible Web Solutions

Subsection: Language → Introduction

- Purpose:

• Correctly identifying and coding language ensures that assistive technologies can render content accurately.

• Screen readers use language metadata to apply correct pronunciation, voice, and braille translation.

• Inaccessible or missing language declarations cause mispronunciation and confusion.

- Intended audience: developers, designers, content authors.

- Technical level: easy.

- Prerequisites: knowledge of HTML attributes and document structure.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible language coding.

• WAI-ARIA provides support for dynamic language switching.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Language → Page Language

- Purpose:

• Identifying the default language of a page allows screen readers and other assistive technologies to apply correct pronunciation and processing.

• Missing or incorrect language declarations can cause mispronunciation and comprehension issues.

- Guidelines:

• Use the lang attribute on the <html> element to declare the default language of the page.

• The language code MUST follow BCP 47 (e.g., lang="en", lang="fr-CA").

• Only one default language SHOULD be declared per page.

• For multilingual sites, ensure each page has the correct lang attribute set.

- Examples:

• Good: <html lang="en">

• Good: <html lang="es">

• Bad: <html> (no lang attribute)

- Accessibility Considerations:

• Screen readers switch voices based on the declared language.

• Braille output depends on accurate language settings.

• Search engines and translation tools also rely on language declarations.

- Best Practices:

• Always set the lang attribute, even for English content.

• Validate language codes to ensure compliance with BCP 47.

• Test screen reader pronunciation to confirm correct rendering.

Domain I: Creating Accessible Web Solutions

Subsection: Language → Changes in Language

- Purpose:

• Inline changes in language must be identified so assistive technologies can adjust pronunciation, braille output, and interpretation.

• Without proper marking, users may not understand foreign words, names, or phrases.

- Guidelines:

• Use the lang attribute on the specific element where the language changes.

• Mark short phrases, single words, or passages written in a different language.

• Do not mark technical terms, proper names, or words that have been fully adopted into the page’s primary language.

• Nested lang attributes SHOULD override the page-level default for the marked content only.

- Examples:

• Good: <p>The motto of the European Union is <span lang="fr">“Unis dans la diversité.”</span></p>

• Good: <p>She ordered a <span lang="it">cappuccino</span> at the café.</p>

• Bad: <p>They said bonjour to the guests.</p> (Missing lang attribute for “bonjour.”)

- Accessibility Considerations:

• Screen readers use correct voice and pronunciation when language is marked.

• Braille translation depends on accurate language tagging.

• Multilingual users benefit from clarity and accurate reading flow.

- Best Practices:

• Mark all meaningful language changes, even for short words or phrases.

• Validate with assistive technologies to ensure correct switching occurs.

• Avoid over-marking terms where the meaning is already clear in context.

Domain I: Creating Accessible Web Solutions

Subsection: Language → Summary

- Accessible language coding ensures that assistive technologies render content with correct pronunciation, braille translation, and meaning.

- Key requirements:

• The default page language MUST be identified using the lang attribute on the <html> element.

• Language codes MUST follow BCP 47 standards (e.g., lang="en-US", lang="fr-CA").

• Changes in language within content MUST be marked with lang attributes on specific elements.

• Only meaningful language changes SHOULD be marked; do not overuse.

- Accessibility Benefits:

• Screen readers switch pronunciation automatically based on declared language.

• Braille output and translation tools rely on correct tagging.

• Multilingual content becomes understandable and inclusive.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require language declaration at both page and inline levels.

• Section 508 and EN 301 549 align globally with these requirements.

- Best Practices:

• Always declare a page-level language.

• Validate language codes and test with multiple screen readers.

• Mark inline language changes consistently and accurately.

• Avoid over-marking where unnecessary.

- Key Takeaway: Correct use of language attributes ensures accurate pronunciation, comprehension, and translation, supporting accessibility and usability for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Introduction

- Purpose:

• Forms enable user interaction, input, and submission of data.

• Inaccessible forms prevent users with disabilities from completing essential tasks such as registration, checkout, or communication.

- Intended audience: developers, designers, and content authors.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of semantic HTML, labels, ARIA, and form validation.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible forms.

• WAI-ARIA provides additional techniques for dynamic or custom form components.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Labels and Instructions

- Purpose:

• Labels and instructions ensure that users understand the purpose of each form field and how to complete it.

• Inaccessible or missing labels create barriers for screen reader and cognitive users.

- Guidelines:

• Every form control MUST have an associated label (<label> element, aria-label, or aria-labelledby).

• Labels MUST clearly describe the purpose of the input (e.g., “Email address” instead of “Enter here”).

• Place labels adjacent to their form controls for clarity.

• Placeholder text MUST NOT replace labels; it should be supplemental.

• Instructions (e.g., format requirements) MUST be provided in text near the relevant fields.

• Group related form controls with <fieldset> and <legend>.

- Examples:

• Good: <label for="email">Email address</label>

<input type="email" id="email" name="email">

• Bad: <input type="email" placeholder="Enter email">

- Accessibility Considerations:

• Screen readers announce labels as part of form controls.

• Users with cognitive disabilities rely on clear, consistent instructions.

• Placeholder-only labels disappear when typing, reducing accessibility.

- Best Practices:

• Keep labels short, clear, and consistent.

• Indicate required fields explicitly (e.g., “Email address (required)”).

• Provide format examples (e.g., “MM/DD/YYYY”) outside the input field.

• Test with screen readers to ensure labels are read correctly.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Keyboard and Focus

- Purpose:

• All form fields and controls must be fully operable by keyboard alone.

• Predictable and visible focus ensures that users can navigate and interact with forms efficiently.

- Guidelines:

• Every form control MUST be reachable and operable using Tab, Shift+Tab, Enter, and Space.

• Focus order MUST follow a logical sequence that matches the visual layout.

• Visible focus indicators MUST be provided for all interactive elements.

• Do NOT remove default focus outlines without providing an accessible replacement.

• Custom controls MUST support keyboard interactions (arrow keys, space, enter).

• Tabindex SHOULD be avoided unless required to fix logical focus order.

- Examples:

• Good: <input type="text" id="name"> (natively focusable with Tab)

• Bad: <div role="textbox"> (requires ARIA and tabindex for keyboard support)

- Accessibility Considerations:

• Screen readers rely on logical focus order for comprehension.

• Keyboard-only users must see where focus is at all times.

• Disrupted focus order confuses both keyboard and assistive technology users.

- Best Practices:

• Test form navigation using keyboard only (no mouse).

• Ensure focus indicators are highly visible with at least 3:1 contrast.

• Group related fields logically for smoother navigation.

• Use native HTML controls wherever possible to ensure built-in accessibility.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Error Identification and Validation

- Purpose:

• Users must be informed clearly when errors occur in form submission and provided with guidance to correct them.

• Inaccessible error handling can block task completion, leading to frustration and exclusion.

- Guidelines:

• Errors MUST be identified in text, not just with color or styling.

• Error messages MUST be programmatically associated with the relevant fields (aria-describedby, id).

• Validation messages MUST be specific and helpful (e.g., “Password must be at least 8 characters”).

• Required fields MUST be indicated before submission, not only after errors.

• Errors SHOULD be summarized at the top of the form in addition to inline feedback.

• Use aria-live regions to announce errors dynamically when possible.

- Examples:

• Good:

<label for="email">Email address</label>

<input type="email" id="email" aria-describedby="emailError">

<span id="emailError">Error: Please enter a valid email address.</span>

• Bad:

<input type="email" style="border:2px solid red">

- Accessibility Considerations:

• Screen reader users rely on programmatic associations to connect errors with inputs.

• Color-blind users may miss color-only error indicators.

• Cognitive users need simple, clear instructions to resolve errors.

- Best Practices:

• Provide inline error messages as well as a summary for long forms.

• Phrase error messages positively and constructively.

• Validate both client-side (for speed) and server-side (for reliability).

• Test error handling with screen readers and keyboard navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Accessible Form Controls

- Purpose:

• Form controls must be designed and coded to be perceivable, operable, and understandable by all users.

• Inaccessible custom controls prevent users with disabilities from completing tasks.

- Guidelines:

• Use native HTML controls whenever possible (<input>, <select>, <textarea>, <button>).

• Custom controls MUST include appropriate roles, states, and properties using ARIA when no native option exists.

• Labels MUST be programmatically associated with controls.

• Controls MUST provide clear focus indicators and be fully keyboard operable.

• Group related inputs with <fieldset> and <legend>.

• Ensure sufficient contrast for text, borders, and focus outlines on controls.

- Examples:

• Good (native select):

<label for="country">Country</label>

<select id="country">

<option>United States</option>

<option>Canada</option>

</select>

• Bad (custom dropdown without ARIA):

<div role="listbox">…</div>

- Accessibility Considerations:

• Screen readers announce native form controls with minimal additional markup.

• ARIA-enhanced custom controls require thorough testing to ensure correct announcement.

• Users with motor impairments rely on predictable, consistent behavior across controls.

- Best Practices:

• Prefer native HTML elements for reliability and lower maintenance.

• Test custom controls with multiple screen readers and browsers.

• Document ARIA patterns used for custom widgets.

• Ensure all controls provide error feedback and follow logical tab order.

Domain I: Creating Accessible Web Solutions

Subsection: Forms → Summary

- Accessible forms ensure that users can complete tasks such as registration, checkout, and communication without barriers.

- Key requirements:

• Every form control MUST have a clear, programmatically associated label.

• Instructions and format requirements MUST be visible and not rely on placeholder text alone.

• Forms MUST be fully operable by keyboard with logical and visible focus.

• Errors MUST be identified in text, programmatically linked to fields, and explained clearly.

• Custom controls MUST implement ARIA roles, states, and properties when no native equivalent exists.

• Group related inputs with <fieldset> and <legend>.

- Accessibility Benefits:

• Screen reader users hear labels, instructions, and errors announced correctly.

• Keyboard users can navigate smoothly through all inputs.

• Cognitive users benefit from clear instructions and constructive error feedback.

• All users gain from predictable, consistent form design.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define form accessibility requirements.

• Section 508 and EN 301 549 mandate accessible form design globally.

- Best Practices:

• Prefer native HTML controls over custom components.

• Provide inline error messages and an error summary for complex forms.

• Validate on both client and server side for speed and reliability.

• Test with screen readers, keyboard-only navigation, and different devices.

- Key Takeaway: Accessible forms combine semantic markup, keyboard operability, clear instructions, and robust error handling to ensure equitable access for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Animations and Motion → Introduction

- Purpose:

• Animations, motion, and auto-updating content can enhance user experience but may also cause accessibility issues.

• Users with vestibular disorders, attention limitations, or cognitive disabilities may experience discomfort, distraction, or disorientation.

- Intended audience: designers, developers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of CSS animations, JavaScript timers, and WCAG time-based media requirements.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require controls for motion and updates.

• WAI-ARIA practices provide guidance for dynamic content handling.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Animations and Motion → Auto-updating Content

- Purpose:

• Auto-updating content (e.g., news tickers, stock feeds, chat windows) can disorient users if not controlled.

• Users must have the ability to pause, stop, or adjust updates.

- Guidelines:

• Content that updates automatically MUST provide user controls to pause, stop, or hide updates.

• Updates MUST be announced programmatically when relevant using aria-live regions.

• Avoid constant, rapid updates that overwhelm screen reader users.

• Auto-refreshing pages SHOULD be avoided; provide manual refresh options instead.

• For chat and log windows, new messages MUST NOT steal focus or disrupt reading order.

- Examples:

• Good: <div aria-live="polite">Latest update: Market closes at 3 PM</div>

• Bad: <marquee>Breaking News...</marquee>

- Accessibility Considerations:

• Screen readers may repeatedly interrupt speech when content updates too frequently.

• Users with cognitive disabilities may lose focus if content shifts unexpectedly.

• Auto-updates without controls can cause motion sickness or distraction.

- Best Practices:

• Provide settings for update speed and frequency.

• Use aria-live="polite" for non-urgent updates, aria-live="assertive" only for urgent ones.

• Ensure updates are logged so users can review missed content.

Domain I: Creating Accessible Web Solutions

Subsection: Animations and Motion → Animations and Motion Effects

- Purpose:

• Animations and motion can enhance engagement but may trigger vestibular disorders, distraction, or seizures.

• Users must be able to reduce or disable motion effects when needed.

- Guidelines:

• Motion effects that autoplay for more than 5 seconds MUST provide controls to pause, stop, or hide them.

• Motion-triggered interactions (e.g., parallax, tilt-to-scroll) MUST have accessible alternatives.

• Use prefers-reduced-motion media query to respect user system preferences.

• Avoid flashing content that exceeds WCAG seizure thresholds (no more than 3 flashes per second).

• Animations SHOULD be subtle and not interfere with readability or task completion.

- Examples:

• Good:

@media (prefers-reduced-motion: reduce) {

\* {

animation: none !important;

transition: none !important;

}

}

• Bad:

<div class="spinning-banner">Spinning nonstop</div>

- Accessibility Considerations:

• Users with vestibular disorders may experience nausea or dizziness from motion.

• Seizure-prone users can be harmed by flashing content.

• Cognitive users may lose focus with distracting animations.

- Best Practices:

• Provide user settings to disable or minimize motion effects.

• Keep transitions brief, subtle, and purposeful.

• Test with prefers-reduced-motion enabled to confirm proper fallback.

• Avoid unnecessary parallax or looping animations.

Domain I: Creating Accessible Web Solutions

Subsection: Animations and Motion → Summary

- Accessible handling of animations, motion, and auto-updating content ensures usability and comfort for all users.

- Key requirements:

• Auto-updating content MUST include controls to pause, stop, or adjust updates.

• Animations longer than 5 seconds MUST provide user controls to stop or hide them.

• Motion effects MUST respect user system settings such as prefers-reduced-motion.

• Flashing content MUST NOT exceed WCAG seizure thresholds (no more than 3 flashes per second).

• Motion-triggered interactions MUST have accessible alternatives.

- Accessibility Benefits:

• Users with vestibular disorders avoid dizziness and nausea.

• Seizure-prone users are protected from harmful flashing.

• Cognitive users maintain focus without distraction.

• Screen reader users receive updates in a controlled, non-disruptive manner.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for motion and time-based media.

• Section 508 and EN 301 549 mandate compliance internationally.

- Best Practices:

• Use subtle, purposeful animations that enhance usability.

• Respect user preferences for reduced motion.

• Test with screen readers, keyboard navigation, and motion sensitivity settings.

• Provide user settings to disable or limit motion.

- Key Takeaway: Accessible animations and motion respect user control, minimize harm, and ensure inclusive experiences without sacrificing engagement.

Domain I: Creating Accessible Web Solutions

Subsection: Time Limits → Introduction

- Purpose:

• Time limits can create significant barriers for users with disabilities, especially those needing more time for reading, understanding, or input.

• Accessible design requires that users have sufficient time and control over time-restricted interactions.

- Intended audience: developers, designers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of session management, timers, and interactive content.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible time-based interactions.

• WAI-ARIA guidelines provide support for dynamic timer controls.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Time Limits → User Control of Time Limits

- Purpose:

• Users must be able to extend, adjust, or disable time limits that affect task completion.

• Without controls, timeouts can prevent users with disabilities from finishing important processes.

- Guidelines:

• Time limits MUST provide users with options to turn off, adjust, or extend the limit to at least 10 times the default duration.

• Session timeouts for security MUST warn users before expiration and provide a way to extend.

• Automatic logouts SHOULD save user input so progress is not lost.

• CAPTCHAs and time-sensitive challenges MUST provide alternatives without strict time pressure.

• Games and quizzes MAY include time limits, but accessible alternatives or accommodations MUST be offered.

- Examples:

• Good: “Your session will expire in 2 minutes. Click here to extend.”

• Bad: Session expires without warning or option to extend.

- Accessibility Considerations:

• Users with motor impairments may type slowly and need more time.

• Users with cognitive or reading disabilities may need additional time to understand instructions.

• Users with assistive technologies may require extended input time.

- Best Practices:

• Provide warnings before timeouts and easy ways to extend.

• Save user progress automatically during timeouts or logouts.

• Allow custom settings for time limits where possible.

• Test workflows with extended timers and assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Time Limits → Exceptions and Considerations

- Purpose:

• Some time limits are essential for functionality, security, or fairness and cannot be removed entirely.

• Even in these cases, accessible accommodations must be provided.

- Guidelines:

• Security-related timeouts (e.g., banking, healthcare) MAY enforce strict time limits but MUST warn users before expiration and allow reauthentication without losing data.

• Real-time events (e.g., auctions, live polling) MAY include fixed limits but SHOULD provide alternatives where possible (e.g., proxy bidding).

• Essential time limits MUST be clearly communicated before starting the task.

• Any exceptions MUST be documented and justified as essential to the activity.

- Accessibility Considerations:

• Users with disabilities need fair opportunities to complete time-sensitive tasks.

• Warnings and progress-saving features reduce the negative impact of unavoidable timeouts.

• Providing alternatives ensures inclusion without compromising essential requirements.

- Best Practices:

• Warn users early and clearly about unavoidable time limits.

• Provide options to resume or retry tasks without penalty.

• Allow proxy or delegated participation when strict timing is required.

• Document exceptions in accessibility conformance reports.

Domain I: Creating Accessible Web Solutions

Subsection: Time Limits → Summary

- Accessible time limit management ensures that users with disabilities can complete tasks without undue pressure.

- Key requirements:

• Users MUST be able to turn off, extend, or adjust time limits to at least 10 times the default duration.

• Session timeouts MUST provide advance warnings and allow extension or reauthentication without data loss.

• Time limits essential for security, fairness, or real-time events MAY remain but MUST include accommodations and clear communication.

• CAPTCHAs and timed challenges MUST have accessible alternatives.

- Accessibility Benefits:

• Users with motor, cognitive, and reading disabilities gain fair opportunities to finish tasks.

• Screen reader and keyboard users avoid being locked out by inflexible timers.

• Progress-saving features reduce frustration for all users.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for time-based interactions.

• Section 508 and EN 301 549 mandate compliance globally.

- Best Practices:

• Provide clear warnings and extendable timeouts.

• Save user progress automatically during sessions.

• Clearly explain exceptions and document them in conformance reports.

• Test time-limited workflows with assistive technologies and diverse users.

- Key Takeaway: Accessible time limits balance security and fairness with user control, ensuring tasks remain achievable for people of all abilities.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication → Introduction

- Purpose:

• Authentication processes (e.g., logins, CAPTCHAs, MFA) must be accessible so all users can securely access content and services.

• Inaccessible authentication can completely block users with disabilities from accounts, purchases, or critical services.

- Intended audience: developers, designers, security teams.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of authentication flows, form design, and assistive technology interaction.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for accessible authentication.

• W3C guidelines address CAPTCHA and cognitive accessibility considerations.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication → Cognitive Function Tests

- Purpose:

• Authentication MUST NOT rely solely on cognitive function tests such as puzzles, memory tasks, or pattern recognition.

• Users with cognitive, learning, or memory-related disabilities may be unable to complete such tasks.

- Guidelines:

• CAPTCHAs based only on puzzles, distorted text, or image recognition MUST have accessible alternatives.

• Authentication SHOULD allow password managers and assistive technologies to fill credentials.

• Avoid requiring users to memorize or transcribe codes without alternatives.

• Provide multiple authentication methods (e.g., email link, SMS, authenticator app, biometric).

• MFA (multi-factor authentication) MUST include at least one accessible option.

- Examples:

• Bad: “Select all images with traffic lights.” (inaccessible for blind users)

• Good: “Click the link we emailed you to sign in.” (accessible alternative)

- Accessibility Considerations:

• Users with dyslexia or memory impairments may fail visual or text-based puzzles.

• Blind users cannot solve image-based CAPTCHAs without alternatives.

• Alternatives must balance security with accessibility.

- Best Practices:

• Use device-based or token-based authentication instead of puzzles.

• Offer biometric or email/SMS codes as alternatives.

• Follow W3C guidance on accessible authentication without cognitive tests.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication → CAPTCHA and Alternatives

- Purpose:

• CAPTCHAs are commonly used to prevent automated access but often create barriers for users with disabilities.

• Accessible alternatives must be provided to ensure fair access.

- Guidelines:

• CAPTCHAs MUST NOT rely on a single sensory modality (e.g., vision only, hearing only).

• Provide at least one accessible alternative such as:

- Audio CAPTCHA for users who cannot see images.

- Text-based logic questions that do not require memory or math.

- Email or SMS verification links.

- Device-based verification (e.g., authenticator apps, biometrics).

• reCAPTCHA v2 and v3 SHOULD be configured for accessibility, ensuring screen reader compatibility and avoiding excessive puzzle use.

• CAPTCHAs MUST NOT block assistive technology input or password managers.

- Examples:

• Bad: Distorted text CAPTCHA requiring sighted interpretation only.

• Good: “Check your email and click the verification link.”

- Accessibility Considerations:

• Blind and low-vision users often cannot complete image-based CAPTCHAs.

• Deaf users may be excluded by audio-only CAPTCHAs.

• Cognitive disabilities make solving puzzles or math problems inaccessible.

- Best Practices:

• Use accessible authentication methods instead of CAPTCHAs whenever possible.

• Offer multiple alternatives and allow the user to choose.

• Test CAPTCHAs with assistive technologies before deployment.

• Document the accessibility of authentication workflows.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication → Multi-Factor Authentication (MFA)

- Purpose:

• Multi-factor authentication (MFA) strengthens security but can exclude users if methods are inaccessible.

• Accessible MFA ensures that users with disabilities can securely log in without barriers.

- Guidelines:

• MFA MUST offer at least one accessible method (e.g., email link, SMS, authenticator app, biometric).

• Avoid requiring vision-only methods (e.g., QR code scanning without alternatives).

• Voice call or audio-only MFA MUST include text-based alternatives for deaf users.

• Authentication apps MUST support screen readers and large text.

• Biometrics SHOULD be offered as optional but not required, since not all users can use them.

• Security tokens or FIDO2 devices SHOULD support tactile or auditory feedback for accessibility.

- Examples:

• Good: Offering choice between SMS, email, or authenticator app.

• Bad: Only supporting QR code scanning for MFA setup.

- Accessibility Considerations:

• Blind users may be unable to scan QR codes.

• Deaf users cannot rely on voice call verification.

• Users with motor impairments may struggle with hardware tokens lacking accessible design.

- Best Practices:

• Provide multiple MFA methods and allow users to choose.

• Ensure authenticator apps work with screen readers and magnifiers.

• Test MFA flows with diverse users and assistive technologies.

• Document supported accessible MFA options in user guidance.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication → Summary

- Accessible authentication ensures that all users can securely access accounts, services, and protected content without being blocked by inaccessible tests or processes.

- Key requirements:

• Authentication MUST NOT rely solely on cognitive function tests (e.g., puzzles, memory, pattern recognition).

• CAPTCHAs MUST provide at least one accessible alternative not dependent on a single sensory modality.

• MFA MUST include at least one accessible method (e.g., email, SMS, authenticator app, biometric).

• Password managers and assistive technologies MUST be supported in login fields.

• Security-related exceptions MUST still provide accommodations to avoid excluding users.

- Accessibility Benefits:

• Users with cognitive, vision, hearing, or motor disabilities can complete authentication independently.

• Screen reader and keyboard users avoid barriers from inaccessible CAPTCHA or MFA.

• Inclusive authentication improves usability for all users while maintaining strong security.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA include criteria for accessible authentication.

• Section 508 and EN 301 549 require accessible authentication methods globally.

- Best Practices:

• Provide multiple authentication options and allow user choice.

• Use accessible alternatives to CAPTCHA whenever possible.

• Test authentication flows with assistive technologies and users with disabilities.

• Document supported accessible methods in user guidance and conformance reports.

- Key Takeaway: Secure authentication can be both strong and accessible when designed with multiple inclusive options, avoiding reliance on cognitive or sensory barriers.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Gestures → Introduction

- Purpose:

• Pointer gestures (e.g., swiping, dragging, pinching) are widely used on touchscreens but may exclude users with motor disabilities.

• Accessible design ensures gestures are operable with simple alternatives like taps or clicks.

- Intended audience: developers, designers, UX teams.

- Technical level: moderate.

- Prerequisites: knowledge of touchscreen interactions, event handling, and input device diversity.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for pointer accessibility.

• W3C Pointer Events specification provides guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Gestures → Alternatives to Complex Gestures

- Purpose:

• Complex gestures such as swiping, dragging, or multi-finger actions can be difficult or impossible for users with motor disabilities.

• Accessible alternatives ensure that all interactions can be performed with a single pointer input.

- Guidelines:

• All functionality available through complex gestures MUST also be available via a simple tap, click, or keyboard input.

• Multi-finger gestures (e.g., pinch-to-zoom) MUST have single-finger or on-screen control alternatives (e.g., zoom buttons).

• Drag-and-drop interactions MUST support keyboard alternatives (e.g., move item with arrow keys and spacebar).

• Touch targets MUST be large enough to support precise input.

• Gesture alternatives SHOULD be visually discoverable and consistently placed.

- Examples:

• Bad: Only supporting swipe left/right to dismiss items.

• Good: Providing both swipe gesture and visible “Dismiss” button.

- Accessibility Considerations:

• Users with motor impairments may not be able to perform swiping or pinching.

• Users relying on assistive input devices (e.g., switch controls, voice input) need simple, discrete alternatives.

• Consistent alternative controls reduce cognitive effort and increase usability.

- Best Practices:

• Provide clear and obvious alternative controls for all gesture-based actions.

• Test with keyboard, screen readers, and switch devices.

• Design for “fat finger” use with at least 44x44px touch targets.

• Document gesture alternatives in help and onboarding materials.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Gestures → Alternatives to Complex Gestures

- Purpose:

• Complex gestures such as swiping, dragging, or multi-finger actions can be difficult or impossible for users with motor disabilities.

• Accessible alternatives ensure that all interactions can be performed with a single pointer input.

- Guidelines:

• All functionality available through complex gestures MUST also be available via a simple tap, click, or keyboard input.

• Multi-finger gestures (e.g., pinch-to-zoom) MUST have single-finger or on-screen control alternatives (e.g., zoom buttons).

• Drag-and-drop interactions MUST support keyboard alternatives (e.g., move item with arrow keys and spacebar).

• Touch targets MUST be large enough to support precise input.

• Gesture alternatives SHOULD be visually discoverable and consistently placed.

- Examples:

• Bad: Only supporting swipe left/right to dismiss items.

• Good: Providing both swipe gesture and visible “Dismiss” button.

- Accessibility Considerations:

• Users with motor impairments may not be able to perform swiping or pinching.

• Users relying on assistive input devices (e.g., switch controls, voice input) need simple, discrete alternatives.

• Consistent alternative controls reduce cognitive effort and increase usability.

- Best Practices:

• Provide clear and obvious alternative controls for all gesture-based actions.

• Test with keyboard, screen readers, and switch devices.

• Design for “fat finger” use with at least 44x44px touch targets.

• Document gesture alternatives in help and onboarding materials.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Gestures → Path-Based Gestures

- Purpose:

• Path-based gestures (e.g., drawing shapes, scribbles, or signatures) can create accessibility barriers for users with motor or cognitive impairments.

• Accessible design ensures these gestures have simple, alternative methods.

- Guidelines:

• Any functionality requiring a path-based gesture MUST have an alternative input method (e.g., button press, keyboard input, form field).

• Signature fields SHOULD allow typed name entry or uploading a saved signature file as alternatives.

• Drawing gestures MUST NOT be the only way to complete an action.

• Alternatives MUST be clearly labeled and functionally equivalent.

- Examples:

• Bad: Drawing a checkmark on the screen to confirm an action with no alternative.

• Good: Providing a “Confirm” button alongside gesture-based input.

- Accessibility Considerations:

• Users with tremors, limited dexterity, or motor disabilities may not be able to draw precise paths.

• Cognitive disabilities may make path-based tasks confusing or impossible.

• Screen readers and voice input users cannot complete drawing gestures.

- Best Practices:

• Always offer button or keyboard alternatives for path-based gestures.

• Provide multiple signature options (typed, uploaded, drawn).

• Test with switch control and alternative input devices.

• Keep gesture-based features optional rather than required.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Gestures → Summary

- Accessible pointer gestures ensure that touchscreen and gesture-based interactions are usable by all users, including those with motor and cognitive disabilities.

- Key requirements:

• All functionality available via complex gestures (swipe, drag, pinch) MUST also be available through simple taps, clicks, or keyboard input.

• Path-based gestures (e.g., drawing, scribbling, signature) MUST have alternative methods such as buttons, typed input, or file uploads.

• Touch targets MUST be large enough (minimum 44x44px) to support users with limited precision.

• Alternatives MUST be clearly labeled, discoverable, and functionally equivalent.

- Accessibility Benefits:

• Users with motor impairments can complete tasks without relying on complex or path-based gestures.

• Users of switch devices, voice input, and screen readers gain equal access to gesture-based functionality.

• Cognitive users benefit from predictable, simple alternatives.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for pointer accessibility.

• Section 508 and EN 301 549 align with these international standards.

- Best Practices:

• Always design with simple input alternatives in addition to gestures.

• Test with diverse input methods, including switch control and voice commands.

• Ensure gesture alternatives are documented in help/onboarding.

• Keep gestures optional to avoid exclusion.

- Key Takeaway: Pointer gesture accessibility ensures inclusive interaction by providing simple, reliable alternatives for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Target Size and Input Modalities → Introduction

- Purpose:

• Interactive elements must have adequate size and spacing to be easily activated by users with limited precision, motor impairments, or touch input.

• Providing accessible target sizes improves usability for all users, especially on mobile devices.

- Intended audience: developers, designers, UX teams.

- Technical level: easy to moderate.

- Prerequisites: knowledge of HTML, CSS, and responsive design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for target size and input accessibility.

• W3C Mobile Accessibility guidelines provide supplemental best practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Target Size and Input Modalities → Target Size

- Purpose:

• Ensuring adequate size of interactive elements helps users with limited dexterity, low vision, or touch input activate controls without error.

• Small or closely packed targets increase the risk of mistakes and frustration.

- Guidelines:

• Targets MUST have a minimum size of 24x24 CSS pixels (WCAG 2.2 Level AA).

• A target size of 44x44 CSS pixels is strongly recommended (aligns with mobile accessibility guidelines).

• Targets MUST have sufficient spacing so they do not overlap or cause accidental activation.

• Exceptions MAY apply when:

- The target is part of inline text.

- The target is controlled by the user agent and not author-defined.

- A larger target size would interfere with the essential design.

- Examples:

• Good: A “Submit” button sized at 44x44px with padding around it.

• Bad: A small 12x12px icon link with no surrounding space.

- Accessibility Considerations:

• Users with motor impairments need larger, well-spaced targets for accurate input.

• Mobile users benefit from larger touch targets due to finger size and screen variability.

• Users with low vision rely on larger targets to avoid precision difficulties.

- Best Practices:

• Use padding around icons and links to increase target size without affecting visual design.

• Maintain consistent sizing across all interactive elements.

• Test on mobile devices and with users who have limited dexterity.

• Avoid clustering too many interactive elements close together.

Domain I: Creating Accessible Web Solutions

Subsection: Target Size and Input Modalities → Multiple Input Modalities

- Purpose:

• Interactive elements should support multiple input methods (mouse, keyboard, touch, voice, assistive technologies).

• Ensuring compatibility with diverse input modalities makes content usable by the widest range of users.

- Guidelines:

• All interactive functionality MUST be available through both pointer (mouse/touch) and keyboard input.

• Voice input users MUST be able to activate controls via accessible names that match visible labels.

• Custom widgets MUST support multiple input methods, including touch and keyboard.

• Drag-and-drop interactions MUST provide alternatives (e.g., keyboard arrow keys, buttons).

• Pointer-specific gestures MUST NOT be the only means of interaction.

- Examples:

• Good: A “Delete” button operable by mouse click, keyboard Enter/Space, and voice command.

• Bad: A drag-only slider with no keyboard alternative.

- Accessibility Considerations:

• Users with motor impairments may depend on switch devices or voice input.

• Blind users rely on keyboard and screen reader navigation instead of mouse.

• Supporting multiple input methods reduces reliance on precision and speed.

- Best Practices:

• Ensure visible labels match programmatic names to support voice input.

• Provide keyboard shortcuts for commonly used functions.

• Test widgets across mouse, keyboard, touch, and voice input.

• Document supported input modalities for custom controls.

Domain I: Creating Accessible Web Solutions

Subsection: Target Size and Input Modalities → Summary

- Accessible target sizes and support for multiple input modalities ensure that users can interact with content accurately and efficiently.

- Key requirements:

• Targets MUST be at least 24x24 CSS pixels; 44x44px is strongly recommended.

• Targets MUST have sufficient spacing to avoid accidental activation.

• All functionality MUST be available via mouse, keyboard, touch, and voice input.

• Custom widgets MUST provide equivalent alternatives for drag-and-drop or gesture-based actions.

• Visible labels MUST match programmatic names to support voice input.

- Accessibility Benefits:

• Users with motor impairments can activate controls without precision struggles.

• Mobile and touch users benefit from larger targets and spacing.

• Voice and switch control users gain full access when labels and names align.

• Cognitive users experience more predictable, reliable interactions.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for target size and input modalities.

• Section 508 and EN 301 549 mandate compliance internationally.

- Best Practices:

• Increase touch targets with padding instead of enlarging icons.

• Maintain consistent sizing across all controls.

• Test with mouse, keyboard, touch, and voice input methods.

• Document exceptions when larger targets are not feasible.

- Key Takeaway: Adequate target size and multiple input modalities reduce barriers, ensuring reliable, inclusive interaction for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Motion Actuation → Introduction

- Purpose:

• Motion actuation allows users to trigger functionality through device motion (e.g., shaking, tilting, orientation changes).

• Without accessible alternatives, motion-based controls can exclude users with motor disabilities or those who cannot physically manipulate devices.

- Intended audience: developers, designers, UX teams.

- Technical level: moderate.

- Prerequisites: knowledge of device sensors, motion APIs, and event handling.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 address input accessibility, including motion actuation.

• W3C Pointer Events and Device Orientation specifications provide implementation guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Motion Actuation → Alternatives to Motion Actuation

- Purpose:

• Users must not be required to perform physical motion (e.g., shaking, tilting, device rotation) as the only way to trigger functionality.

• Accessible alternatives ensure inclusivity for users with motor disabilities or those in restricted environments.

- Guidelines:

• Any action triggered by motion MUST also be available through a standard control (e.g., button, keyboard shortcut).

• Motion-based inputs (e.g., “shake to undo”) MUST NOT be the sole method of performing critical tasks.

• Users MUST be able to disable motion-based interactions if they cause issues.

• Alternative controls SHOULD be easy to discover and placed near the primary interaction.

- Examples:

• Bad: Only supporting “shake to refresh.”

• Good: Providing both “shake to refresh” and a visible “Refresh” button.

- Accessibility Considerations:

• Users with motor impairments may not be able to physically perform motion gestures.

• Users in vehicles or fixed positions cannot always tilt or move devices.

• Accidental motion can trigger unwanted actions if not supplemented by standard controls.

- Best Practices:

• Always pair motion interactions with visible, accessible controls.

• Provide settings to disable motion triggers.

• Test features with motor-impaired users and switch input devices.

• Clearly document motion alternatives in help and onboarding.

Domain I: Creating Accessible Web Solutions

Subsection: Motion Actuation → Summary

- Accessible motion actuation ensures that device motion (e.g., shaking, tilting, rotating) does not create barriers for users with disabilities.

- Key requirements:

• Any motion-triggered action MUST also be available through standard controls (buttons, links, keyboard).

• Users MUST be able to disable motion-based inputs if they cause difficulty or accidental activation.

• Motion gestures MUST NOT be the only method of completing critical actions.

• Alternatives MUST be easy to discover and functionally equivalent.

- Accessibility Benefits:

• Users with motor impairments gain full access without relying on physical movement.

• Users in fixed or restricted environments can still complete tasks.

• Reduces accidental triggers for all users, improving usability.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for motion actuation.

• Section 508 and EN 301 549 align with these standards globally.

- Best Practices:

• Pair motion actuation with clear, visible alternative controls.

• Provide an option in settings to disable or adjust motion triggers.

• Test with users who cannot perform physical gestures.

• Clearly explain available alternatives in help and onboarding materials.

- Key Takeaway: Motion actuation can enhance interactivity, but it must always include accessible, discoverable alternatives to ensure inclusivity.

Domain I: Creating Accessible Web Solutions

Subsection: Device and Orientation Accessibility → Introduction

- Purpose:

• Content and functionality must be usable regardless of device orientation (portrait or landscape).

• Restricting orientation creates barriers for users who cannot rotate their devices due to motor, vision, or mounting constraints.

- Intended audience: developers, designers, UX teams.

- Technical level: easy to moderate.

- Prerequisites: knowledge of responsive design, CSS, and device testing.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include orientation accessibility requirements.

• W3C Mobile Accessibility guidelines provide best practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Device and Orientation Accessibility → Orientation Flexibility

- Purpose:

• Users must be able to access and operate content in both portrait and landscape orientations.

• Restricting orientation excludes users who cannot rotate devices due to disability, mounting, or environmental factors.

- Guidelines:

• Content MUST support both portrait and landscape orientations.

• Orientation MUST NOT be locked unless essential (e.g., check-scanning apps, games requiring a specific orientation).

• If orientation is restricted, the restriction MUST be clearly communicated to the user.

• Interfaces SHOULD adapt fluidly to orientation changes using responsive design.

- Examples:

• Bad: An app that only works in portrait mode with no option for landscape.

• Good: A form that adapts seamlessly whether in portrait or landscape.

- Accessibility Considerations:

• Users with motor impairments may have devices mounted in fixed positions.

• Low-vision users may prefer landscape for magnification and larger text.

• Screen reader and switch device users benefit from consistent layouts across orientations.

- Best Practices:

• Design layouts that reflow gracefully in both orientations.

• Test on a range of devices to ensure flexibility.

• Communicate orientation requirements clearly when they are essential.

• Use responsive frameworks to support adaptive design.

Domain I: Creating Accessible Web Solutions

Subsection: Device and Orientation Accessibility → Summary

- Device and orientation accessibility ensures that content works across portrait and landscape modes, preventing exclusion of users who cannot physically rotate their devices.

- Key requirements:

• Content MUST support both portrait and landscape orientations.

• Orientation MUST NOT be locked unless essential to functionality.

• If restriction is necessary, it MUST be clearly communicated.

• Layouts SHOULD adapt fluidly through responsive design.

- Accessibility Benefits:

• Users with mounted devices or motor impairments can still access content.

• Low-vision users can choose orientation that maximizes readability.

• Flexible layouts improve usability for all users, especially on mobile.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require support for multiple orientations.

• Section 508 and EN 301 549 align with these global standards.

- Best Practices:

• Design responsively for both orientations from the start.

• Test across devices, screen sizes, and accessibility settings.

• Provide clear instructions if orientation restrictions are unavoidable.

• Avoid assuming users can freely rotate their device.

- Key Takeaway: Orientation flexibility is essential for equitable access, ensuring content adapts to users instead of forcing users to adapt to content.

Domain I: Creating Accessible Web Solutions

Subsection: Content Reflow and Responsiveness → Introduction

- Purpose:

• Content must reflow and remain usable without loss of information or functionality when resized, zoomed, or viewed on small screens.

• Inaccessible reflow forces horizontal scrolling, overlapping elements, or content loss, creating barriers for low-vision and mobile users.

- Intended audience: developers, designers, UX teams.

- Technical level: moderate.

- Prerequisites: knowledge of CSS, responsive design, and accessibility testing.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate reflow and responsive accessibility.

• W3C Mobile Accessibility guidelines provide supplemental best practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Content Reflow and Responsiveness → Reflow Requirements

- Purpose:

• Users must be able to zoom and resize content without losing information or functionality.

• Reflow ensures usability on small screens and for users with low vision who enlarge text.

- Guidelines:

• Content MUST reflow without requiring two-dimensional scrolling when zoomed up to 400%.

• Horizontal scrolling MUST NOT be required for text except for content such as tables, images, or maps where unavoidable.

• Text MUST reflow in a single column when zoomed.

• Responsive layouts SHOULD adapt seamlessly across screen sizes and orientations.

• CSS media queries and flexible units (em, rem, %) SHOULD be used to support reflow.

- Examples:

• Bad: A fixed-width layout that forces horizontal scrolling at 200% zoom.

• Good: A responsive design that reflows text into one column at 400% zoom.

- Accessibility Considerations:

• Users with low vision rely on zoom and text resizing.

• Mobile users benefit from responsive reflow across screen sizes.

• Cognitive users maintain comprehension when layouts are predictable and uncluttered.

- Best Practices:

• Test reflow at 200% and 400% zoom to ensure compliance.

• Use relative units for fonts, spacing, and containers.

• Avoid fixed pixel heights/widths that prevent reflow.

• Design mobile-first, then enhance for larger screens.

Domain I: Creating Accessible Web Solutions

Subsection: Content Reflow and Responsiveness → Responsive Design Considerations

- Purpose:

• Responsive design ensures content adapts gracefully across devices, screen sizes, and orientations.

• Accessibility requires that responsive layouts remain functional and understandable for all users.

- Guidelines:

• Content MUST adapt to different viewport sizes without loss of information or functionality.

• Interactive elements MUST remain usable at all breakpoints (buttons, menus, forms).

• Text MUST remain readable without requiring horizontal scrolling or excessive zoom.

• Navigation SHOULD remain consistent across viewports with logical adaptation (e.g., hamburger menus).

• Layout changes MUST NOT disrupt reading order or keyboard focus.

- Examples:

• Bad: A menu that disappears on small screens without an accessible replacement.

• Good: A navigation bar that collapses into an accessible, labeled hamburger menu.

- Accessibility Considerations:

• Screen reader users rely on consistent navigation structures across breakpoints.

• Low-vision users depend on readable text without horizontal scrolling.

• Keyboard users must be able to access all responsive elements without traps.

- Best Practices:

• Test across a range of devices and screen sizes.

• Ensure hamburger and collapsible menus are fully keyboard accessible and labeled.

• Use flexible grids and fluid layouts to support scaling.

• Validate reading order with screen readers at all breakpoints.

Domain I: Creating Accessible Web Solutions

Subsection: Content Reflow and Responsiveness → Summary

- Accessible reflow and responsiveness ensure that content adapts to zoom, resizing, and different screen sizes without loss of usability.

- Key requirements:

• Content MUST reflow without requiring two-dimensional scrolling at up to 400% zoom.

• Text MUST reflow in a single column, except where unavoidable (tables, images, maps).

• Layouts MUST remain functional and consistent across devices and orientations.

• Responsive design SHOULD preserve reading order, focus, and navigation.

- Accessibility Benefits:

• Low-vision users can zoom in without losing content or forcing horizontal scrolling.

• Mobile and tablet users gain seamless usability across screen sizes.

• Cognitive users benefit from predictable, uncluttered layouts.

• All users gain from flexible, mobile-friendly design.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for reflow and responsiveness.

• Section 508 and EN 301 549 mandate compliance globally.

- Best Practices:

• Use relative units (em, rem, %) instead of fixed pixels.

• Test layouts at 200% and 400% zoom and across multiple devices.

• Ensure responsive menus and navigation remain accessible.

• Validate reading order and focus at all breakpoints.

- Key Takeaway: Reflow and responsive design ensure that content remains accessible and functional across zoom levels, screen sizes, and orientations.

Domain I: Creating Accessible Web Solutions

Subsection: Visual Presentation and Text Customization → Introduction

- Purpose:

• Users must be able to adjust visual presentation (fonts, spacing, colors) to meet their accessibility needs.

• Inaccessible presentation settings can block users with low vision, dyslexia, or cognitive disabilities from effectively reading content.

- Intended audience: developers, designers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of CSS, typography, and accessibility testing.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessible visual customization.

• W3C guidance on adaptable text provides additional best practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Visual Presentation and Text Customization → Text Spacing and Customization

- Purpose:

• Users must be able to adjust text spacing, line height, and presentation without loss of content or functionality.

• Customization supports users with low vision, dyslexia, and cognitive processing differences.

- Guidelines:

• Content MUST remain functional and readable when users apply the following minimum text spacing:

- Line height (line spacing): 1.5 times font size

- Spacing following paragraphs: 2 times font size

- Letter spacing (tracking): 0.12 times font size

- Word spacing: 0.16 times font size

• Text MUST NOT overlap, truncate, or lose content when these spacing values are applied.

• Users MUST be able to override font family, size, and color through custom stylesheets or browser settings.

• Avoid fixed heights or pixel-based layouts that prevent spacing adjustments.

- Examples:

• Bad: Text that overlaps or cuts off when line spacing is increased.

• Good: Flexible layout where text expands without breaking.

- Accessibility Considerations:

• Users with dyslexia benefit from increased spacing between letters and words.

• Low-vision users rely on larger fonts and line spacing to reduce eye strain.

• Cognitive users gain clarity and comprehension with adaptable presentation.

- Best Practices:

• Use relative units (em, rem, %) instead of fixed pixels for spacing and fonts.

• Test content with custom stylesheets and browser overrides.

• Provide controls or themes for user-adjustable spacing and fonts.

• Validate that no content or functionality is lost with adjusted spacing.

Domain I: Creating Accessible Web Solutions

Subsection: Visual Presentation and Text Customization → Color and Contrast Customization

- Purpose:

• Users must be able to adjust colors and contrast without losing information or functionality.

• Inaccessible color schemes or fixed styles can exclude users with low vision, color blindness, or cognitive disabilities.

- Guidelines:

• Text and interactive elements MUST maintain sufficient contrast against the background (minimum 4.5:1 for normal text, 3:1 for large text).

• Users MUST be able to override author-defined colors using browser or system settings.

• Information MUST NOT be conveyed by color alone; redundant cues (e.g., icons, text) MUST be provided.

• High-contrast mode and dark mode SHOULD be supported.

• Focus indicators and interactive states MUST remain visible in all themes.

- Examples:

• Bad: Red and green used to indicate “error” and “success” with no text labels.

• Good: Error messages displayed with text and icon, in addition to color.

- Accessibility Considerations:

• Users with color blindness may not distinguish red/green differences.

• Low-vision users rely on high-contrast modes for readability.

• Cognitive users benefit from consistent, non-distracting color schemes.

- Best Practices:

• Support OS-level high-contrast and dark mode settings.

• Test with user-defined stylesheets and custom color schemes.

• Ensure hover, focus, and active states remain distinct in all modes.

• Avoid relying on gradients or background images for essential contrast.

Domain I: Creating Accessible Web Solutions

Subsection: Visual Presentation and Text Customization → Summary

- Accessible visual presentation ensures users can adjust spacing, fonts, colors, and contrast without losing content or functionality.

- Key requirements:

• Content MUST remain functional when users apply WCAG minimum text spacing adjustments.

• Users MUST be able to override fonts, colors, and styles through system or browser settings.

• Text and interactive elements MUST meet contrast requirements (4.5:1 for normal text, 3:1 for large text).

• Information MUST NOT rely on color alone for meaning.

- Accessibility Benefits:

• Users with dyslexia benefit from increased spacing and adjustable fonts.

• Low-vision users gain from larger fonts, custom colors, and high contrast.

• Color-blind users understand information through redundant cues.

• All users benefit from customizable, comfortable reading experiences.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define spacing, color, and contrast customization requirements.

• Section 508 and EN 301 549 mandate compliance internationally.

- Best Practices:

• Use relative units for fonts and spacing to allow user scaling.

• Support OS-level high-contrast and dark mode preferences.

• Provide multiple themes or style options where possible.

• Test with custom stylesheets and assistive technology overrides.

- Key Takeaway: Flexible text and color customization ensures content adapts to user needs, improving readability, comprehension, and inclusivity.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Order and Consistency → Introduction

- Purpose:

• Content order and consistency ensure predictable navigation, comprehension, and interaction across a website or application.

• Inconsistent layouts or unpredictable order can confuse users, especially those with cognitive or vision disabilities.

- Intended audience: developers, designers, content authors.

- Technical level: easy to moderate.

- Prerequisites: knowledge of HTML structure, CSS layout, and navigation design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require consistent, logical content order.

• W3C usability guidance reinforces predictability as a key accessibility principle.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Order and Consistency → Logical Content Order

- Purpose:

• Logical content order ensures that information is presented in a meaningful sequence for both sighted and non-sighted users.

• Assistive technologies rely on DOM order to determine reading and navigation flow.

- Guidelines:

• DOM order MUST match the intended logical and visual reading order.

• CSS positioning (flex, grid, absolute) MUST NOT disrupt logical sequence.

• Headings, landmarks, and lists MUST reinforce logical structure.

• Hidden elements (display:none, aria-hidden) MUST NOT interfere with navigation.

• Tab order MUST follow a predictable and consistent path.

- Examples:

• Bad: Using CSS to visually place the sidebar above the main content while DOM order places it after.

• Good: Sidebar coded before main content in DOM when visually placed on top.

- Accessibility Considerations:

• Screen reader users follow DOM order, not visual design.

• Keyboard users depend on logical tab order for efficient navigation.

• Cognitive users benefit from predictable sequences across pages.

- Best Practices:

• Validate order by navigating with screen readers and keyboard only.

• Keep related items grouped logically in the DOM.

• Minimize tabindex usage; rely on natural DOM flow.

• Test across responsive breakpoints to ensure order remains logical.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Order and Consistency → Consistent Navigation and Identification

- Purpose:

• Consistency in navigation and identification ensures users can reliably locate content and functions across pages.

• Inconsistent naming, placement, or layout confuses users with cognitive or vision disabilities.

- Guidelines:

• Navigation menus MUST appear in a consistent order across pages.

• Repeated components (headers, footers, sidebars) MUST remain consistent in placement.

• Icons and controls MUST use consistent labels and symbols across the site.

• Interactive elements with the same function MUST have the same accessible name on all pages.

• Variations in navigation MAY be allowed when necessary (e.g., step-by-step processes) but MUST be clearly explained.

- Examples:

• Bad: Search box labeled “Find” on one page and “Search” on another.

• Good: Consistently labeled “Search” across all pages.

- Accessibility Considerations:

• Screen reader users rely on predictable landmarks and consistent naming.

• Cognitive users benefit from familiarity and reduced cognitive load.

• Keyboard users navigate more efficiently with consistent menus.

- Best Practices:

• Define navigation and labeling standards in a design system.

• Audit content regularly for consistent terminology and layout.

• Use templates to enforce consistency across pages.

• Provide clear explanations if navigation flow changes (e.g., checkout process).

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Order and Consistency → Summary

- Accessible content order and consistency ensure that users can navigate, understand, and interact with content predictably.

- Key requirements:

• DOM order MUST match logical and visual reading order.

• CSS positioning MUST NOT disrupt the logical sequence.

• Navigation, landmarks, and repeated components MUST remain consistent across pages.

• Icons, controls, and labels MUST use consistent identification for the same functions.

• Tab order MUST follow a predictable, logical sequence.

- Accessibility Benefits:

• Screen reader users gain accurate reading and navigation flow.

• Keyboard users navigate more efficiently with consistent tab order.

• Cognitive users benefit from reduced confusion and predictable layouts.

• All users gain from consistent interaction patterns across the site.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require logical order and consistent navigation.

• Section 508 and EN 301 549 reinforce these requirements globally.

- Best Practices:

• Validate logical order with screen readers and keyboard testing.

• Enforce consistency through design systems and templates.

• Regularly audit navigation, labels, and components for consistency.

• Communicate clearly when deviations are necessary (e.g., process steps).

- Key Takeaway: Logical order and consistent design reduce barriers, making navigation and comprehension easier for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Names and Labels → Introduction

- Purpose:

• Accessible names and labels ensure that interactive elements and content are properly announced by assistive technologies.

• Without accessible names, users relying on screen readers, voice input, or switch devices cannot understand or activate controls.

- Intended audience: developers, designers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of HTML attributes, ARIA, and assistive technology interaction.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible names and labels.

• WAI-ARIA Authoring Practices provide implementation guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Names and Labels → Programmatic Association

- Purpose:

• Programmatic association ensures that names and labels are correctly conveyed to assistive technologies.

• Without programmatic association, screen readers may not announce controls properly.

- Guidelines:

• Every form control and interactive element MUST have a programmatically associated name or label.

• Use the <label> element for form fields, linked via for/id.

• Use aria-label or aria-labelledby when native labels are not possible.

• Decorative icons MUST be hidden with aria-hidden="true".

• Images used as controls MUST have meaningful alt text or aria-labels.

• Accessible names SHOULD match or include visible labels.

- Examples:

• Good:

<label for="email">Email address</label>

<input id="email" type="email">

• Good: <button aria-label="Close menu">✖</button>

• Bad: <input type="text" placeholder="Email"> (placeholder is not a label)

- Accessibility Considerations:

• Screen readers announce accessible names, guiding interaction.

• Voice input users rely on matching visible and programmatic names.

• Users with cognitive disabilities benefit from clear, consistent labeling.

- Best Practices:

• Prefer native labels over ARIA when possible.

• Ensure accessible names match visible text to support speech recognition.

• Test with screen readers to confirm correct label announcements.

• Avoid relying on placeholder text as the only label.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Names and Labels → Name, Role, Value

- Purpose:

• Assistive technologies must be able to determine the name, role, and value of interactive elements.

• Without this information, users cannot understand or operate controls effectively.

- Guidelines:

• All interactive elements MUST expose:

- Name: the accessible identifier (label or aria-label).

- Role: the type of control (button, checkbox, link).

- Value: the current state or setting (checked, expanded, progress level).

• Use native HTML controls where possible, as they expose name, role, and value automatically.

• For custom controls, use ARIA attributes (role, aria-\*) to define name, role, and value.

• Changes in value MUST be communicated programmatically (e.g., aria-checked updates).

- Examples:

• Good (native): <input type="checkbox" id="subscribe">

<label for="subscribe">Subscribe to newsletter</label>

• Good (custom): <div role="checkbox" aria-checked="false" tabindex="0">Subscribe</div>

• Bad: <div>Subscribe</div> (no role, name, or state)

- Accessibility Considerations:

• Screen readers announce name, role, and value to convey meaning.

• Keyboard and switch users rely on accurate state announcements for navigation.

• Consistent exposure of these attributes improves usability across assistive technologies.

- Best Practices:

• Prefer native HTML controls before using ARIA.

• Update aria-\* attributes dynamically when state changes.

• Test custom widgets with multiple screen readers to confirm correct announcements.

• Keep accessible names concise and consistent with visible labels.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Names and Labels → Summary

- Accessible names and labels ensure that interactive elements are properly announced and operable with assistive technologies.

- Key requirements:

• Every control MUST have a programmatically associated name or label.

• Accessible names SHOULD match visible text to support voice input.

• Interactive elements MUST expose name, role, and value programmatically.

• Decorative icons MUST be hidden from assistive technologies.

• Placeholder text MUST NOT serve as the only label.

- Accessibility Benefits:

• Screen reader users understand the purpose and state of controls.

• Voice input users can activate controls by speaking visible labels.

• Cognitive users benefit from clear, consistent identification of controls.

• All users gain from predictable, labeled interactions.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require programmatic names, roles, and values.

• Section 508 and EN 301 549 mandate accessible naming practices globally.

- Best Practices:

• Prefer native HTML labels over ARIA when possible.

• Keep accessible names short, clear, and aligned with visible text.

• Test with screen readers and voice input technologies.

• Ensure aria-\* attributes update dynamically for custom controls.

- Key Takeaway: Exposing accurate names, roles, and values ensures that assistive technologies can convey interactive elements reliably, supporting usability and inclusion.

Domain I: Creating Accessible Web Solutions

Subsection: Focus Appearance and Visibility → Introduction

- Purpose:

• Focus appearance and visibility ensure that users navigating with keyboards or assistive technologies can always identify which element is active.

• Without visible focus, users may become lost or unable to complete tasks.

- Intended audience: developers, designers, QA testers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of CSS, keyboard navigation, and interactive elements.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require visible, accessible focus indicators.

• W3C usability guidelines reinforce the importance of strong focus visibility.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Focus Appearance and Visibility → Visible Focus Indicators

- Purpose:

• Users navigating with a keyboard or assistive technology must always know which element has focus.

• Removing or obscuring focus indicators creates significant barriers for navigation.

- Guidelines:

• All interactive elements MUST display a visible focus indicator when focused.

• Focus indicators MUST have sufficient contrast (at least 3:1 against adjacent colors).

• Authors MUST NOT remove or disable default browser focus outlines without providing accessible replacements.

• Custom focus styles SHOULD be consistent across all interactive elements.

• Focus MUST remain visible even during animations or transitions.

- Examples:

• Bad: button:focus { outline: none; }

• Good: button:focus { outline: 3px solid #000; }

- Accessibility Considerations:

• Keyboard-only users rely on visible focus for orientation.

• Low-vision users require strong contrast and clear indicators.

• Screen reader users benefit when visual focus aligns with programmatic focus.

- Best Practices:

• Enhance default focus styles rather than removing them.

• Use thick, high-contrast outlines that work across backgrounds.

• Test focus visibility on different devices, resolutions, and themes.

• Validate focus alignment with both keyboard and screen reader navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Focus Appearance and Visibility → Focus Consistency and Management

- Purpose:

• Consistent and predictable focus behavior ensures users do not lose context while navigating.

• Improper focus management can disorient users, especially those relying on screen readers or keyboards.

- Guidelines:

• Focus MUST move logically through content in the expected sequence.

• Focus MUST NOT be trapped within components without an explicit way to exit.

• Modals and dialogs MUST trap focus inside while open, then return focus to the triggering element when closed.

• Dynamic content (e.g., expanding menus, pop-ups) MUST receive focus when activated.

• Focus MUST NOT be forced to unexpected elements unless initiated by the user.

- Examples:

• Bad: Opening a modal but leaving focus behind on the page.

• Good: Focus automatically moves to the first interactive element in the modal, then returns to the triggering button upon close.

- Accessibility Considerations:

• Keyboard and switch users rely on consistent focus order to navigate efficiently.

• Screen reader users depend on focus alignment with programmatic reading order.

• Cognitive users benefit from predictable, non-disruptive focus transitions.

- Best Practices:

• Use tabindex sparingly; prefer natural DOM order.

• Test modals, menus, and pop-ups with both keyboard and screen readers.

• Return focus to logical elements after dynamic updates.

• Document focus management rules in the design system.

Domain I: Creating Accessible Web Solutions

Subsection: Focus Appearance and Visibility → Summary

- Accessible focus appearance and management ensure that users navigating with keyboards, screen readers, or assistive technologies can maintain orientation and complete tasks efficiently.

- Key requirements:

• Focus indicators MUST be visible with at least 3:1 contrast.

• Authors MUST NOT remove focus outlines without providing accessible alternatives.

• Focus MUST follow a logical sequence without trapping users.

• Modals and dialogs MUST trap focus while open, then return it to the triggering element.

• Dynamic content MUST receive focus when activated, but focus MUST NOT be forced unexpectedly.

- Accessibility Benefits:

• Keyboard-only users always know their current position.

• Screen reader users benefit from alignment of programmatic and visual focus.

• Cognitive users gain predictability from consistent focus management.

• All users experience smoother, more reliable navigation.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for focus visibility and management.

• Section 508 and EN 301 549 mandate focus accessibility globally.

- Best Practices:

• Enhance default focus indicators rather than removing them.

• Test focus visibility and order with keyboard and assistive technologies.

• Return focus to logical elements after dynamic interactions.

• Document focus rules in design systems to ensure consistency.

- Key Takeaway: Strong, consistent focus appearance and management prevent disorientation, enabling efficient navigation for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Cancellation → Introduction

- Purpose:

• Pointer cancellation ensures that users do not accidentally trigger actions before completing intended input.

• Preventing premature activation reduces errors, especially for users with motor or cognitive disabilities.

- Intended audience: developers, designers, QA testers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of pointer events, input devices, and JavaScript interaction handling.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate safe and predictable input handling.

• W3C Pointer Events specification provides implementation guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Cancellation → Input Handling Requirements

- Purpose:

• Input handling must prevent unintended actions from being triggered by incomplete or accidental pointer interactions.

• Users should have control over final activation of actions.

- Guidelines:

• Actions triggered by pointer input MUST NOT be completed on down-event alone (mousedown, touchstart).

• Activation SHOULD occur on up-event (mouseup, touchend), allowing users to cancel by moving away.

• Drag-and-drop actions MUST allow cancellation by moving the pointer away before releasing.

• Context menus or additional options SHOULD be available for complex interactions.

• Input handling MUST NOT require fine motor precision beyond basic pointer control.

- Examples:

• Bad: A button that immediately submits a form on mousedown.

• Good: A button that submits only on mouseup or touchend, allowing cancellation if the user drags away.

- Accessibility Considerations:

• Users with tremors or motor impairments may press down unintentionally and need the ability to cancel.

• Touch users may slip or change their mind before lifting their finger.

• Consistent, forgiving input handling reduces frustration for all users.

- Best Practices:

• Always trigger primary actions on up-event, not down-event.

• Provide clear visual feedback during interaction (e.g., hover or active states).

• Allow cancellation by dragging away or pressing Escape.

• Test with touchscreens, mice, and alternative input devices.

Domain I: Creating Accessible Web Solutions

Subsection: Pointer Cancellation → Summary

- Pointer cancellation ensures users can control actions without triggering them accidentally, improving reliability and reducing errors.

- Key requirements:

• Actions MUST NOT complete on down-event alone.

• Activation SHOULD occur on up-event, giving users a chance to cancel.

• Drag-and-drop MUST allow cancellation before release.

• Input handling MUST avoid requiring fine motor precision.

- Accessibility Benefits:

• Users with motor impairments avoid unintended activations caused by tremors or slips.

• Touchscreen users can change their mind before lifting their finger.

• Cognitive users benefit from predictable, forgiving input handling.

• All users experience smoother, less error-prone interactions.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for pointer cancellation.

• Section 508 and EN 301 549 align with these standards globally.

- Best Practices:

• Trigger actions on up-event rather than down-event.

• Provide clear visual feedback during pointer interaction.

• Allow easy cancellation through drag-away or Escape key.

• Test input handling across mouse, touch, and alternative devices.

- Key Takeaway: Accessible pointer cancellation provides users with the ability to confirm intent before activation, ensuring accuracy and inclusivity.

Domain I: Creating Accessible Web Solutions

Subsection: Error Prevention and Confirmation → Introduction

- Purpose:

• Error prevention and confirmation ensure that users can avoid mistakes or recover easily when errors occur.

• Without safeguards, users with cognitive, motor, or vision disabilities may lose progress or perform unintended actions.

- Intended audience: developers, designers, QA testers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of forms, validation, and interaction design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require error prevention in forms and processes.

• W3C usability guidance highlights the importance of confirmations and undo.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Error Prevention and Confirmation → Preventing User Errors

- Purpose:

• Preventing errors is more effective than forcing users to recover after mistakes.

• Accessible error prevention reduces frustration and improves task success for all users.

- Guidelines:

• Forms MUST include clear labels, instructions, and examples before submission.

• Required fields MUST be identified programmatically and visually.

• Input constraints (format, length, type) SHOULD be enforced with inline guidance before submission.

• Auto-complete and input assistance SHOULD be provided where appropriate (e.g., address, date).

• High-risk actions (e.g., deleting data, submitting payments) MUST provide confirmation dialogs or undo options.

• Defaults SHOULD be safe and reversible.

- Examples:

• Bad: Submitting a form that rejects an email after submission without prior guidance.

• Good: Inline hint text “Enter a valid email address (example@domain.com)” with validation before submission.

- Accessibility Considerations:

• Users with cognitive disabilities benefit from clear instructions and reduced chance of error.

• Screen reader users need programmatically conveyed constraints and required indicators.

• Motor-impaired users rely on error prevention to reduce repeated input.

- Best Practices:

• Provide inline validation with accessible ARIA alerts or live regions.

• Use safe defaults (e.g., “Cancel” as the default in critical dialogs).

• Allow users to review and edit information before final submission.

• Test forms and processes with assistive technologies to ensure error prevention works effectively.

Domain I: Creating Accessible Web Solutions

Subsection: Error Prevention and Confirmation → Confirmation and Recovery

- Purpose:

• When errors occur, users must be able to confirm, undo, or recover without losing progress.

• Accessible recovery reduces frustration and prevents permanent mistakes.

- Guidelines:

• High-risk actions (e.g., submitting payments, deleting content) MUST require confirmation before execution.

• Undo options SHOULD be provided whenever possible.

• Error messages MUST be clear, specific, and programmatically associated with the relevant field or action.

• Recovery instructions MUST be provided in plain language with actionable steps.

• System errors (e.g., timeouts, server failures) MUST allow retry or recovery without data loss.

- Examples:

• Bad: Immediate deletion of files with no confirmation or recovery option.

• Good: Confirmation dialog “Are you sure you want to delete this file?” with Undo option.

- Accessibility Considerations:

• Users with cognitive disabilities benefit from confirmation dialogs and plain-language instructions.

• Screen reader users require programmatic association of error messages with the correct input field.

• Motor-impaired users rely on undo to correct accidental activations without repeating complex processes.

- Best Practices:

• Provide undo for destructive actions whenever feasible.

• Ensure error messages are concise, specific, and announced to assistive technologies.

• Maintain user progress during errors (e.g., preserve form input after a timeout).

• Use accessible modals or alerts for confirmation dialogs.

Domain I: Creating Accessible Web Solutions

Subsection: Error Prevention and Confirmation → Summary

- Error prevention and confirmation ensure users can avoid mistakes, confirm critical actions, and recover from errors without losing progress.

- Key requirements:

• Forms MUST provide clear labels, instructions, and inline validation.

• Required fields MUST be identified programmatically and visually.

• High-risk actions MUST require confirmation or provide undo.

• Error messages MUST be specific, actionable, and associated with relevant inputs.

• System errors MUST allow retry or recovery without data loss.

- Accessibility Benefits:

• Cognitive users benefit from clear guidance and confirmation before committing to actions.

• Screen reader users receive accurate error associations and recovery instructions.

• Motor-impaired users can undo accidental activations without repeating complex steps.

• All users experience fewer errors and smoother recovery processes.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for error prevention and recovery.

• Section 508 and EN 301 549 mandate these practices globally.

- Best Practices:

• Use plain language for instructions and error messages.

• Provide undo whenever feasible for destructive actions.

• Preserve user progress during errors or timeouts.

• Test error handling with assistive technologies.

- Key Takeaway: Error prevention, confirmation, and recovery ensure inclusive, forgiving experiences where users can act confidently without fear of permanent mistakes.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Help and Support → Introduction

- Purpose:

• Accessible help and support ensure users can find assistance, guidance, or clarification when needed.

• Without accessible help, users with disabilities may be unable to complete tasks or resolve issues independently.

- Intended audience: developers, designers, content authors, support teams.

- Technical level: easy to moderate.

- Prerequisites: knowledge of content authoring, navigation, and accessibility principles.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible help mechanisms.

• W3C usability guidance emphasizes providing assistance for complex tasks.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Help and Support → Providing Accessible Help

- Purpose:

• Help content and support features must be accessible so all users can understand and act on assistance provided.

• Inaccessible help creates barriers for users who most need guidance.

- Guidelines:

• Help content (FAQs, instructions, tooltips) MUST be perceivable and operable with assistive technologies.

• Inline instructions and contextual help SHOULD be provided near complex inputs (e.g., forms).

• Tooltips and help popovers MUST be keyboard accessible and dismissible.

• Live support options (chat, phone, email) MUST be accessible across modalities.

• Documentation SHOULD be available in multiple formats (text, audio, video with captions).

- Examples:

• Bad: Help icons that only display instructions on mouse hover.

• Good: Help icons that are focusable, provide instructions on click or keyboard activation, and are announced by screen readers.

- Accessibility Considerations:

• Screen reader users need help content to be programmatically associated with relevant inputs.

• Cognitive users benefit from clear, step-by-step instructions placed contextually.

• Users with motor impairments require keyboard operability and accessible chat support.

- Best Practices:

• Provide contextual help at the point of need rather than requiring navigation away.

• Ensure help icons and links have descriptive labels (e.g., “Help with password requirements”).

• Offer human support options in addition to self-service documentation.

• Test all help features with screen readers and keyboard-only navigation.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Help and Support → Consistency and Availability

- Purpose:

• Help and support must be consistently available across pages and interactions so users can reliably find assistance.

• Inconsistent or hidden help resources create confusion and barriers for users with disabilities.

- Guidelines:

• Help options (FAQs, contact links, support widgets) MUST be available consistently across all pages where complex tasks occur.

• Placement of help links or icons SHOULD be consistent in location (e.g., always top-right or footer).

• Labels for help resources MUST remain consistent (e.g., always “Help” rather than varying terms).

• Support documentation SHOULD remain accessible offline or printable when possible.

• Complex workflows MUST include embedded help at each critical step.

- Examples:

• Bad: A checkout process with no help option until the final payment screen.

• Good: A form that provides inline help at each step and consistent access to live chat.

- Accessibility Considerations:

• Cognitive users rely on predictable locations for help resources.

• Screen reader users benefit from consistent labeling and programmatic landmarks for help.

• Users with motor impairments require quick access to help without navigating long distances.

- Best Practices:

• Establish a consistent location for help and support links across the site.

• Provide multiple contact channels (chat, phone, email) to support diverse needs.

• Ensure live support platforms (chatbots, agents) are fully accessible.

• Regularly audit help content for accessibility and consistency.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Help and Support → Summary

- Accessible help and support ensure users can find guidance and complete tasks independently, reducing abandonment and frustration.

- Key requirements:

• Help content MUST be perceivable and operable with assistive technologies.

• Tooltips and contextual help MUST be keyboard accessible and dismissible.

• Help resources MUST be consistently available and labeled across pages.

• Live support options MUST be accessible across multiple modalities.

- Accessibility Benefits:

• Screen reader users gain programmatic access to contextual help.

• Cognitive users benefit from consistent, clear, and predictable assistance.

• Motor-impaired users access help via keyboard or alternative input methods.

• All users benefit from reliable support options when tasks are complex.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require accessible help mechanisms.

• Section 508 and EN 301 549 mandate accessible support globally.

- Best Practices:

• Place help options in consistent, predictable locations.

• Provide inline guidance at points of complexity (e.g., forms, workflows).

• Offer multiple channels of support (chat, email, phone) that are all accessible.

• Test help systems with screen readers, keyboard-only navigation, and users with cognitive needs.

- Key Takeaway: Accessible help and support empower users to complete tasks with confidence, providing essential guidance without creating new barriers.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Security → Introduction

- Purpose:

• Authentication and security processes must be accessible so that users with disabilities can log in, verify identity, and complete secure actions without barriers.

• Inaccessible security features (CAPTCHA, MFA, timeouts) can block users entirely.

- Intended audience: developers, security engineers, UX designers.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of authentication flows, MFA, CAPTCHA, and secure session management.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible authentication processes.

• W3C Accessible Authentication guidance provides additional best practices.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Security → Barriers in Authentication

- Purpose:

• Many authentication methods create barriers for users with disabilities if not designed inclusively.

• Identifying these barriers helps guide accessible alternatives.

- Common Barriers:

• CAPTCHAs relying only on vision or hearing (e.g., distorted text, audio puzzles).

• MFA that requires scanning QR codes or responding to voice calls only.

• Time-limited logins or sessions without accommodations for extended input time.

• Password requirements that depend on memory or complex cognitive tasks.

• Security questions based on personal memory recall, which may not be accessible.

- Accessibility Considerations:

• Blind or low-vision users struggle with image-based CAPTCHAs and QR code scanning.

• Deaf users cannot complete audio-only MFA.

• Cognitive disabilities may make memory-based tasks and complex requirements inaccessible.

• Motor impairments can make short timeouts or fine-precision inputs difficult.

- Best Practices:

• Provide at least one accessible alternative for every authentication method.

• Avoid relying on a single sensory modality.

• Offer flexible timeouts or extendable sessions.

• Replace memory-based authentication with device-based or multi-factor options.

• Test all authentication processes with assistive technologies.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Security → Accessible Authentication Methods

- Purpose:

• Authentication methods must include accessible options so all users can securely log in and verify identity.

- Guidelines:

• CAPTCHAs MUST have at least one accessible alternative (e.g., logic puzzle, checkbox, audio + visual options).

• Authentication MUST support password managers and assistive technologies for credential entry.

• MFA MUST provide at least one accessible option (e.g., SMS, email, authenticator app with screen reader support).

• Biometrics MAY be offered but MUST NOT be the only authentication method.

• Timeouts MUST allow extension or recovery without losing data.

• Security questions SHOULD be replaced with accessible, non-memory-dependent alternatives.

- Examples:

• Bad: Login requires solving distorted text CAPTCHA with no alternative.

• Good: Login allows “Email me a link” option in addition to password entry.

- Accessibility Considerations:

• Blind users need alternatives to image-only CAPTCHA and QR codes.

• Deaf users require text-based MFA alternatives to voice calls.

• Cognitive users benefit from simplified authentication (e.g., emailed links).

• Motor-impaired users need more time to complete inputs without session loss.

- Best Practices:

• Provide multiple authentication methods and let users choose.

• Use device-based authentication (tokens, security keys) with accessible feedback.

• Ensure authenticator apps and biometric methods work with screen readers.

• Document accessible options clearly in user guidance.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Security → Security and Accessibility Balance

- Purpose:

• Security requirements must be met without creating accessibility barriers.

• Balancing security and accessibility ensures users are protected while retaining full access.

- Guidelines:

• Security measures MUST NOT rely solely on inaccessible methods (e.g., image CAPTCHA, voice-only verification).

• MFA MUST include at least one accessible method that meets the same security level.

• Biometric authentication SHOULD be supplemented with alternative methods for users unable to use them.

• Session timeouts MUST allow warning and extension for users needing more time.

• Authentication flows SHOULD be designed with both usability and security in mind.

- Examples:

• Bad: Bank login requiring only biometric authentication (fingerprint) with no alternative.

• Good: Bank login offering fingerprint, SMS code, or emailed link as options.

- Accessibility Considerations:

• Security-only focus without accessibility excludes users from critical services.

• Accessible alternatives must maintain equal protection levels, not weaker fallbacks.

• Consistency across secure flows reduces cognitive and usability barriers.

- Best Practices:

• Involve accessibility experts in security design decisions.

• Test authentication and security flows with users of assistive technology.

• Provide transparent documentation on accessible options.

• Monitor for new security methods and evaluate accessibility impact before adoption.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Security → Summary

- Accessible authentication and security ensure all users can log in, verify identity, and perform secure actions without barriers.

- Key requirements:

• CAPTCHAs MUST provide accessible alternatives.

• MFA MUST include at least one accessible option (e.g., SMS, email, accessible authenticator app).

• Password and credential entry MUST support assistive technologies and password managers.

• Biometrics MUST NOT be the only authentication method.

• Session timeouts MUST allow extension and recovery.

- Accessibility Benefits:

• Blind and low-vision users gain alternatives to visual CAPTCHAs and QR codes.

• Deaf users benefit from text-based MFA options.

• Cognitive users avoid reliance on memory-based security questions.

• Motor-impaired users gain flexibility with extended timeouts and alternative input methods.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for accessible authentication.

• Section 508 and EN 301 549 mandate inclusive security practices globally.

- Best Practices:

• Provide multiple accessible authentication methods and allow user choice.

• Balance strong security with usability by testing across assistive technologies.

• Replace memory- and vision-dependent tasks with inclusive alternatives.

• Document accessible options in support and onboarding materials.

- Key Takeaway: Security must not exclude — accessible authentication balances protection with inclusion, ensuring equitable access for all users.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Timeouts and Interruptions → Introduction

- Purpose:

• Timeouts and interruptions must be managed accessibly so users do not lose progress or become disoriented.

• Without accommodations, users with cognitive, motor, or vision disabilities may be disproportionately affected by time-limited interactions.

- Intended audience: developers, designers, UX and QA teams.

- Technical level: moderate.

- Prerequisites: knowledge of session management, JavaScript timers, and interaction design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for accessible time limits.

• W3C usability guidance emphasizes warnings, extensions, and recovery.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Timeouts and Interruptions → Time Limit Requirements

- Purpose:

• Time-limited interactions must include accessible warnings, extensions, and recovery options.

• Prevents users from losing progress due to disabilities or situational constraints.

- Guidelines:

• Users MUST be warned before a session or task times out.

• Users MUST be given sufficient time to extend or save their work.

• Timeouts MUST allow extension of at least 10 times the original limit (WCAG guidance).

• Critical data (e.g., form input, documents) MUST be preserved during timeouts and recoverable after re-authentication.

• Timeouts MUST NOT be used as the sole security mechanism without alternatives.

• Users SHOULD be able to disable non-essential timeouts.

- Examples:

• Bad: Banking session logs out after 2 minutes of inactivity with no warning.

• Good: Banking session warns after 10 minutes, offers “Extend session,” and saves entered data.

- Accessibility Considerations:

• Cognitive users may need more time to process or complete inputs.

• Motor-impaired users may type slowly or use alternative input devices.

• Low-vision users may require more time to navigate interfaces.

- Best Practices:

• Provide clear, accessible warnings with sufficient notice.

• Preserve form data and session state across timeouts.

• Offer adjustable or disable-able timeouts for non-critical tasks.

• Test timeout flows with users of assistive technology.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Timeouts and Interruptions → Interruptions and Notifications

- Purpose:

• Interruptions and notifications must be managed so they do not disorient users or block task completion.

• Uncontrolled interruptions can create barriers for users with cognitive, motor, or vision disabilities.

- Guidelines:

• Critical notifications (e.g., security alerts, errors) MUST be announced programmatically and remain until dismissed.

• Non-critical notifications (e.g., marketing pop-ups) MUST be dismissible and SHOULD NOT take focus automatically.

• Interruption dialogs MUST be accessible by keyboard and screen reader.

• Users MUST be able to resume their task without losing progress after dismissing interruptions.

• Frequent or repetitive notifications MUST allow user control (e.g., “Don’t show again”).

- Examples:

• Bad: An auto-refresh popup that steals focus and clears user input.

• Good: A security alert displayed in a modal, announced to assistive tech, dismissible by keyboard, and preserving user data.

- Accessibility Considerations:

• Screen reader users rely on ARIA live regions for notification announcements.

• Cognitive users require control over frequency and intrusiveness of interruptions.

• Motor-impaired users must be able to dismiss dialogs via keyboard or alternative input.

- Best Practices:

• Use ARIA live regions for status updates and alerts.

• Keep notifications persistent until dismissed by the user.

• Avoid automatic focus shifts unless user-initiated.

• Allow users to customize or suppress non-essential notifications.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Timeouts and Interruptions → Summary

- Accessible timeouts and interruptions ensure users can complete tasks without losing progress or being disoriented by unexpected events.

- Key requirements:

• Users MUST be warned before timeouts and given options to extend.

• Timeouts MUST preserve user data and allow recovery after re-authentication.

• Notifications MUST be programmatically announced, dismissible, and non-disruptive.

• Interruptions MUST NOT take focus automatically unless user-initiated.

• Users MUST have control over frequency and persistence of non-critical interruptions.

- Accessibility Benefits:

• Cognitive users gain more time to complete tasks without unexpected disruption.

• Motor-impaired users avoid losing progress due to slow input or inaccessible interruptions.

• Screen reader users reliably receive announcements of critical alerts.

• All users benefit from predictable, controlled interaction patterns.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for timeouts and interruptions.

• Section 508 and EN 301 549 mandate compliance internationally.

- Best Practices:

• Provide accessible warning dialogs before timeouts.

• Preserve form data and progress across session interruptions.

• Use ARIA live regions for notifications and alerts.

• Test timeout and interruption flows with assistive technologies.

- Key Takeaway: Accessible timeouts and interruptions protect user progress, prevent disorientation, and maintain trust in digital interactions.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Animations and Motion → Introduction

- Purpose:

• Animations and motion must be designed accessibly to avoid triggering vestibular disorders, distractions, or barriers to comprehension.

• Inaccessible animations can cause dizziness, nausea, or cognitive overload for some users.

- Intended audience: developers, designers, UX teams.

- Technical level: moderate.

- Prerequisites: knowledge of CSS animations, JavaScript, and user preference media queries.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for accessible animations.

• W3C guidance on reduced motion and multimedia accessibility applies.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Animations and Motion → Motion Sensitivity Considerations

- Purpose:

• Motion can cause dizziness, nausea, or distraction for users with vestibular disorders or cognitive conditions.

• Accessible design must minimize unnecessary motion and respect user preferences.

- Guidelines:

• Content that flashes, blinks, or auto-animates MUST NOT exceed WCAG thresholds (no more than 3 flashes per second).

• Animations triggered automatically (e.g., parallax, auto-scroll) MUST be avoidable or disabled.

• Users MUST be able to disable non-essential motion through system settings (prefers-reduced-motion) or in-app controls.

• Essential animations (e.g., loading indicators) MUST be subtle and non-disruptive.

• Animations SHOULD reinforce meaning or feedback, not distract.

- Examples:

• Bad: A background with constant auto-scrolling motion.

• Good: A simple fade-in transition when opening a modal, which can be disabled by user preference.

- Accessibility Considerations:

• Vestibular disorders can be triggered by parallax, zoom, or panning effects.

• Cognitive disabilities may worsen with distracting or excessive motion.

• Users with ADHD benefit from reduced motion environments.

- Best Practices:

• Respect prefers-reduced-motion CSS media query in all animations.

• Provide a user-facing setting to disable or minimize motion.

• Use minimal, purpose-driven transitions instead of decorative motion.

• Test with reduced motion enabled to confirm usability.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Animations and Motion → Reduced Motion and Alternatives

- Purpose:

• Users must have the ability to reduce or disable motion effects that are not essential to functionality.

• Providing reduced motion alternatives prevents barriers for users with vestibular disorders, cognitive conditions, or attention difficulties.

- Guidelines:

• Websites and apps MUST respect system-level reduced motion preferences (prefers-reduced-motion).

• Non-essential animations (e.g., parallax, auto-scrolling, transitions) MUST be disabled or minimized when reduced motion is enabled.

• Alternatives SHOULD be provided (e.g., fade or instant appearance instead of zoom or slide).

• Motion that conveys information (e.g., progress indicators) MUST have an accessible non-motion equivalent (e.g., text label, percentage).

• Decorative animations SHOULD default to minimal styles unless user opts in.

- Examples:

• Bad: Parallax scrolling effects that cannot be turned off.

• Good: Parallax replaced with a static image when reduced motion is enabled.

- Accessibility Considerations:

• Vestibular disorder users rely on reduced motion to avoid nausea or dizziness.

• Cognitive users benefit from simplified visuals with fewer distractions.

• Screen reader users rely on text-based equivalents when motion is disabled.

- Best Practices:

• Always use the prefers-reduced-motion media query for CSS animations.

• Provide application-level settings for motion reduction beyond system preferences.

• Replace motion with static or fading alternatives rather than removing context entirely.

• Test with reduced motion enabled on multiple devices and OS settings.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Animations and Motion → Summary

- Accessible animations and motion ensure users are not excluded or harmed by unnecessary visual effects.

- Key requirements:

• Flashing content MUST NOT exceed 3 flashes per second.

• Non-essential motion MUST be avoidable, disabled, or minimized.

• System-level preferences (prefers-reduced-motion) MUST be respected.

• Essential animations MUST have non-motion alternatives (e.g., text progress).

• Motion SHOULD support meaning, not distract from it.

- Accessibility Benefits:

• Users with vestibular disorders avoid nausea and dizziness.

• Cognitive users benefit from reduced distractions and simplified visuals.

• ADHD users gain focus in low-motion environments.

• All users enjoy smoother, purpose-driven animation experiences.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define thresholds for flashing and motion.

• Section 508 and EN 301 549 require reduced motion compliance.

- Best Practices:

• Use prefers-reduced-motion in CSS and JavaScript animations.

• Provide in-app settings for reduced motion beyond OS-level controls.

• Replace animated effects with subtle fades or static alternatives.

• Test with reduced motion enabled and with real users sensitive to motion.

- Key Takeaway: Animations should enhance, not exclude—accessible motion respects user preferences, avoids harm, and maintains clarity.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Status Messages and Alerts → Introduction

- Purpose:

• Status messages and alerts must be communicated accessibly so users are aware of important updates without losing context.

• Inaccessible alerts may go unnoticed or disrupt tasks for users with disabilities.

- Intended audience: developers, designers, QA testers.

- Technical level: moderate.

- Prerequisites: knowledge of ARIA live regions, semantic HTML, and assistive technology behavior.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 include requirements for accessible status updates.

• WAI-ARIA guidelines define roles and practices for alerts and live regions.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Status Messages and Alerts → Implementing Status Messages

- Purpose:

• Status messages must be conveyed programmatically so assistive technologies can announce them without disrupting user tasks.

• Proper implementation ensures users do not miss critical updates.

- Guidelines:

• Status messages MUST be coded using semantic HTML or ARIA roles (status, alert, log, progressbar).

• Messages that do not take focus SHOULD use aria-live regions (polite or assertive).

• Error messages MUST be programmatically associated with their relevant input fields.

• Success, warning, and error states MUST include text and not rely on color alone.

• Messages SHOULD remain visible long enough for users to perceive them.

- Examples:

• Good: <div role="status" aria-live="polite">Form saved successfully.</div>

• Good: <div role="alert">Error: Please enter a valid email address.</div>

• Bad: Inline error text without ARIA or programmatic association.

- Accessibility Considerations:

• Screen reader users rely on ARIA live regions for real-time announcements.

• Cognitive users benefit from clear, concise, and persistent status messages.

• Low-vision users need sufficient contrast and visibility for message text.

- Best Practices:

• Use semantic roles before ARIA where possible (e.g., <output>, <progress>).

• Choose aria-live="polite" for non-critical updates; use "assertive" sparingly for urgent alerts.

• Ensure all status messages are text-based and programmatically exposed.

• Test with multiple screen readers to confirm correct announcement behavior.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Status Messages and Alerts → Summary

- Accessible status messages ensure users are informed of important updates without losing context or missing critical information.

- Key requirements:

• Status messages MUST be conveyed using semantic roles or ARIA (status, alert, log, progressbar).

• Error messages MUST be programmatically linked to relevant input fields.

• Messages MUST NOT rely on color alone for meaning.

• Messages SHOULD remain visible long enough to be perceived.

• aria-live regions MUST be used for dynamic updates.

- Accessibility Benefits:

• Screen reader users receive timely announcements of updates.

• Cognitive users benefit from clear, persistent, and simple status messaging.

• Low-vision users gain visibility with high-contrast, text-based alerts.

• All users experience smoother interactions with clear feedback.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for accessible status messages.

• Section 508 and EN 301 549 mandate compliance for alerts and updates.

- Best Practices:

• Use semantic HTML roles before applying ARIA.

• Match urgency to aria-live setting (polite vs. assertive).

• Keep language concise and avoid technical jargon in alerts.

• Test with screen readers to confirm announcement accuracy.

- Key Takeaway: Accessible status messages ensure all users are aware of system feedback, reducing confusion and supporting task completion.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop Functionality → Introduction

- Purpose:

• Drag-and-drop functionality must be accessible to users who cannot operate a mouse or perform complex pointer gestures.

• Without accessible alternatives, users with motor, vision, or cognitive disabilities may be unable to complete tasks involving drag-and-drop.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of JavaScript, ARIA, keyboard interactions, and pointer events.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible alternatives to drag-and-drop.

• WAI-ARIA Authoring Practices provide guidance for implementing accessible drag-and-drop.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop Functionality → Introduction

- Purpose:

• Drag-and-drop functionality must be accessible to users who cannot operate a mouse or perform complex pointer gestures.

• Without accessible alternatives, users with motor, vision, or cognitive disabilities may be unable to complete tasks involving drag-and-drop.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of JavaScript, ARIA, keyboard interactions, and pointer events.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible alternatives to drag-and-drop.

• WAI-ARIA Authoring Practices provide guidance for implementing accessible drag-and-drop.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop Functionality → Keyboard and Alternative Input Support

- Purpose:

• Drag-and-drop interactions must be operable using keyboards and alternative input devices to ensure accessibility.

- Guidelines:

• Every drag-and-drop feature MUST provide an equivalent keyboard interaction.

• Keyboard users MUST be able to pick up, move, and drop items using keys (e.g., Enter/Space to pick up, arrow keys to move, Enter/Space to drop).

• Screen reader users MUST receive announcements when items are selected, moved, and dropped.

• Alternative input devices (e.g., switch controls, voice input) MUST be able to operate drag-and-drop through accessible commands.

• Dragging MUST allow cancellation before drop without triggering unintended actions.

- Examples:

• Bad: A file manager where files can only be rearranged by mouse drag.

• Good: A file manager that allows selecting an item with Enter, moving it with arrow keys, and dropping it with Enter.

- Accessibility Considerations:

• Motor-impaired users may not be able to perform drag gestures precisely.

• Screen reader users need programmatic announcements to track movement and drop targets.

• Voice input users must be able to issue commands like “Move item to position 3.”

- Best Practices:

• Follow WAI-ARIA practices for draggable and droppable elements.

• Provide clear visual and programmatic feedback for drag state and drop targets.

• Ensure cancellation is always possible (e.g., pressing Escape).

• Test with keyboard-only navigation and screen readers.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop Functionality → Programmatic Announcements and Feedback

- Purpose:

• Users must receive clear, programmatic feedback during drag-and-drop interactions to maintain context and control.

- Guidelines:

• Drag start, movement, and drop events MUST be announced to assistive technologies.

• Drop targets MUST be programmatically identified and announced when items are moved over them.

• Success or failure of a drop MUST be communicated with clear status messages.

• Visual indicators (e.g., highlighted drop zones) MUST be paired with programmatic announcements.

• Announcements SHOULD be concise and provide essential context only.

- Examples:

• Good: Screen reader announces “Item picked up. Move with arrow keys. Over folder ‘Documents’. Press Enter to drop.”

• Bad: Silent drag-and-drop with no programmatic feedback.

- Accessibility Considerations:

• Screen reader users depend on announcements to understand drag state and available drop zones.

• Cognitive users benefit from consistent and simple feedback language.

• Motor-impaired users need confirmation that actions were completed successfully.

- Best Practices:

• Use ARIA drag-and-drop roles and states (aria-grabbed, aria-dropeffect).

• Provide both visual and auditory cues for drag state and drop success.

• Keep messages brief but clear (avoid information overload).

• Test drag-and-drop workflows with multiple screen readers and input devices.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop Functionality → Summary

- Accessible drag-and-drop ensures users with disabilities can perform reordering, file movement, and other interactions without barriers.

- Key requirements:

• Drag-and-drop MUST have an equivalent keyboard interaction.

• Screen readers MUST announce drag start, movement, drop targets, and results.

• Visual indicators MUST be paired with programmatic feedback.

• Users MUST be able to cancel a drag without committing unintended actions.

- Accessibility Benefits:

• Motor-impaired users can move items without relying on fine mouse control.

• Screen reader users receive essential announcements for drag state and targets.

• Cognitive users benefit from consistent, simple feedback and predictable actions.

• All users gain from clearer, more flexible interaction methods.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require accessible alternatives for drag-and-drop.

• Section 508 and EN 301 549 mandate compliance for interactive functionality.

- Best Practices:

• Follow WAI-ARIA Authoring Practices for drag-and-drop roles and states.

• Provide visual and auditory cues for movement and success.

• Keep instructions concise and reinforce them during interaction.

• Test with screen readers, keyboard-only workflows, and alternative input devices.

- Key Takeaway: Drag-and-drop must not exclude — accessible alternatives, announcements, and flexible inputs ensure usability for everyone.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components → Introduction

- Purpose:

• Custom widgets and components must be designed with accessibility in mind, ensuring they expose correct roles, states, and behaviors to assistive technologies.

• Without accessible implementation, custom controls can block users relying on screen readers, keyboards, or alternative inputs.

- Intended audience: developers, UI engineers, designers.

- Technical level: advanced.

- Prerequisites: knowledge of ARIA, semantic HTML, and widget behavior patterns.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessibility of custom controls.

• WAI-ARIA Authoring Practices provide detailed implementation guidance.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components → Roles, States, and Properties

- Purpose:

• Custom widgets must expose correct roles, states, and properties so assistive technologies can interpret and interact with them.

- Guidelines:

• Every custom component MUST have an appropriate role (native HTML role or ARIA role).

• Interactive elements MUST expose current states (e.g., expanded/collapsed, checked/unchecked) via aria-\* attributes.

• Relationships between elements MUST be programmatically conveyed (e.g., aria-controls, aria-labelledby).

• Names and roles MUST remain consistent across the application.

• Redundant ARIA MUST be avoided when native HTML already provides semantics.

- Examples:

• Good: A custom accordion with role="button" on headers and aria-expanded="true/false".

• Good: A custom tab system with role="tab", role="tabpanel", and aria-controls linking them.

• Bad: A clickable <div> with no role or ARIA attributes.

- Accessibility Considerations:

• Screen reader users depend on roles and states to understand custom components.

• Keyboard users require consistent behavior aligned with native elements.

• Cognitive users benefit from predictable, descriptive names and roles.

- Best Practices:

• Prefer native HTML semantics whenever possible before adding ARIA.

• Keep roles and states synchronized with user interaction dynamically.

• Validate ARIA implementation using automated tools and manual screen reader testing.

• Document custom widget behaviors in the design system for consistency.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components → Keyboard and Interaction Support

- Purpose:

• Custom widgets must provide full keyboard operability and predictable interaction patterns.

• Without proper support, users relying on keyboards, switch devices, or voice input may be excluded.

- Guidelines:

• Every interactive component MUST be operable by keyboard alone.

• Standard interaction patterns (e.g., arrow keys for menus, tabs, sliders) MUST be followed consistently.

• Focus MUST move predictably within and between widgets.

• Widgets MUST allow escape or exit via keyboard (e.g., Escape to close menus).

• Alternative inputs (switches, voice) MUST be able to activate and navigate custom controls.

- Examples:

• Good: A custom slider operable with left/right arrows and reporting value changes programmatically.

• Good: A modal dialog that traps focus and closes with Escape.

• Bad: A custom dropdown requiring mouse clicks only.

- Accessibility Considerations:

• Keyboard-only users rely on logical tab and arrow key navigation.

• Screen reader users need accurate role and state announcements aligned with keyboard actions.

• Motor-impaired users depend on efficient keyboard or alternative input paths.

- Best Practices:

• Follow ARIA Authoring Practices for keyboard interaction models.

• Test with keyboard-only navigation before screen reader validation.

• Keep interaction consistent with platform and browser expectations.

• Provide clear documentation of expected interaction patterns.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components → Testing and Validation

- Purpose:

• Custom widgets require rigorous testing to ensure accessibility across assistive technologies and input methods.

• Automated checks alone are insufficient; manual testing is essential.

- Guidelines:

• All custom components MUST be tested with screen readers (NVDA, JAWS, VoiceOver).

• Keyboard-only navigation MUST be validated across components.

• Roles, states, and properties MUST be verified dynamically during interaction.

• Automated tools (e.g., axe, Lighthouse, WAVE) SHOULD be used to catch common ARIA errors.

• User testing with assistive technology users SHOULD be conducted for complex widgets.

- Examples:

• Good: Testing a custom date picker with NVDA and confirming correct announcements of roles and values.

• Bad: Relying only on automated linting without real-world screen reader testing.

- Accessibility Considerations:

• Different screen readers may interpret ARIA implementations differently.

• Keyboard support must be consistent across browsers and OS platforms.

• Cognitive users benefit when widgets are tested for clarity and predictable behavior.

- Best Practices:

• Include accessibility testing in QA workflows for all custom components.

• Maintain regression tests for ARIA roles and keyboard support.

• Document testing outcomes and fixes for future reference.

• Engage real users with disabilities for usability validation of complex widgets.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components → Summary

- Accessible custom widgets and components ensure users can interact with non-standard controls reliably and inclusively.

- Key requirements:

• Custom widgets MUST expose correct roles, states, and properties via native HTML or ARIA.

• Widgets MUST be fully operable with keyboard and alternative inputs.

• Focus management MUST be predictable and consistent.

• Automated and manual testing MUST validate accessibility across AT and browsers.

- Accessibility Benefits:

• Screen reader users receive accurate role and state announcements.

• Keyboard-only and motor-impaired users can navigate and operate controls.

• Cognitive users benefit from consistent, predictable interactions.

• All users experience reliable, inclusive functionality.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require custom controls to meet accessibility standards.

• Section 508 and EN 301 549 mandate accessible implementation of interactive elements.

- Best Practices:

• Prefer native HTML elements before creating custom widgets.

• Follow WAI-ARIA Authoring Practices for roles, states, and keyboard models.

• Test thoroughly with multiple screen readers and browsers.

• Document widget behaviors in the design system to ensure consistency.

- Key Takeaway: Custom does not mean inaccessible—accessible widgets expose semantics, provide full input support, and undergo robust testing to serve all users.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Personalization and Adaptability → Introduction

- Purpose:

• Content personalization and adaptability allow users to adjust interfaces to meet their accessibility needs without breaking functionality.

• Inaccessible designs that restrict customization (e.g., fixed layouts, locked styles) prevent users from creating usable experiences.

- Intended audience: developers, designers, content authors.

- Technical level: moderate.

- Prerequisites: knowledge of CSS, responsive design, and assistive technology settings.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require support for personalization and adaptability.

• W3C guidance highlights adaptability as a core accessibility principle.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Personalization and Adaptability → User-Controlled Presentation

- Purpose:

• Users must be able to adapt presentation (fonts, colors, spacing, layout) without losing content or functionality.

• Preventing personalization creates barriers for low-vision, dyslexic, and cognitive users.

- Guidelines:

• Text resizing up to 200% MUST NOT cause loss of content or functionality.

• Users MUST be able to override author styles with custom stylesheets or browser settings.

• Color schemes MUST support high contrast modes and dark mode without breaking usability.

• Line spacing, letter spacing, and text alignment adjustments MUST be supported.

• Layouts MUST remain functional when zoomed or reflowed.

- Examples:

• Bad: Fixed pixel layouts that clip text when zoomed.

• Good: Responsive layouts that reflow text and preserve readability at 200% zoom.

- Accessibility Considerations:

• Low-vision users rely on zoom and high-contrast settings.

• Dyslexic users benefit from custom fonts, spacing, and simplified layouts.

• Cognitive users may need larger text and uncluttered layouts.

- Best Practices:

• Use relative units (em, rem, %) instead of fixed pixels.

• Test with browser zoom, custom stylesheets, and OS high-contrast settings.

• Avoid fixed containers or overflow clipping for text content.

• Support prefers-color-scheme for dark and light mode compatibility.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Personalization and Adaptability → Adaptable Content and Alternatives

- Purpose:

• Content must adapt to different user needs and technologies without losing meaning or usability.

• Providing alternatives ensures that critical information is always perceivable.

- Guidelines:

• Content MUST remain usable across different orientations, viewport sizes, and input methods.

• Essential information MUST NOT be conveyed by style alone (e.g., color, shape, or position).

• Alternatives (text, captions, transcripts) MUST be provided for media content.

• Interactive controls MUST adapt to user settings (e.g., larger buttons with zoom).

• Content SHOULD adapt gracefully across simplified or personalized views.

- Examples:

• Bad: Using only color (red/green) to indicate stock performance.

• Good: Providing both color and text labels such as “Stock Up” / “Stock Down.”

- Accessibility Considerations:

• Blind users rely on text alternatives for images and graphics.

• Deaf users depend on captions or transcripts for audio/video.

• Low-vision and motor-impaired users need adaptable layouts and larger control targets.

• Cognitive users benefit from simplified, uncluttered content modes.

- Best Practices:

• Provide semantic markup that supports flexible adaptation.

• Pair visual indicators with textual equivalents.

• Ensure interactive elements scale properly under zoom and personalization.

• Test adaptability with responsive tools, screen readers, and user-controlled styles.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Content Personalization and Adaptability → Summary

- Accessible personalization and adaptability ensure users can tailor digital experiences to their needs without breaking functionality.

- Key requirements:

• Text resizing up to 200% MUST NOT cause loss of content or functionality.

• Users MUST be able to override styles with custom settings or stylesheets.

• Color schemes MUST support high-contrast and dark mode.

• Alternatives MUST be provided for media and non-text content.

• Layouts and controls MUST adapt to user settings and zoom levels.

- Accessibility Benefits:

• Low-vision users gain flexibility with zoom, contrast, and font adjustments.

• Dyslexic and cognitive users benefit from custom fonts, spacing, and simplified layouts.

• Deaf and blind users access alternatives for media and graphics.

• All users experience more resilient and customizable interfaces.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for reflow, text resizing, and adaptability.

• Section 508 and EN 301 549 mandate adaptability and user personalization globally.

- Best Practices:

• Use semantic HTML and relative units (em, rem, %).

• Support prefers-color-scheme and other system-level accessibility settings.

• Provide alternatives for all non-text content.

• Test with zoom, custom styles, and assistive technologies.

- Key Takeaway: Personalization and adaptability empower users to control how they experience content, ensuring accessibility across diverse needs.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Consistent Identification and Navigation → Introduction

- Purpose:

• Consistent identification and navigation ensure that users can predict and recognize repeated components across an interface.

• Inconsistent labeling or placement of elements creates confusion, especially for users with cognitive or screen reader needs.

- Intended audience: developers, designers, content authors.

- Technical level: easy to moderate.

- Prerequisites: knowledge of navigation design, labeling, and interface patterns.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require consistency for repeated elements.

• W3C usability principles emphasize predictability and familiarity.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Consistent Identification and Navigation → Consistent Labels and Roles

- Purpose:

• Repeated elements must be labeled and identified consistently across pages and contexts.

• Inconsistent labels or roles confuse users and increase cognitive load.

- Guidelines:

• Components that perform the same function MUST use the same label and role.

• Icons MUST have consistent alternative text across all occurrences.

• Navigation links MUST retain consistent naming across pages (e.g., “Contact Us” not “Reach Out” elsewhere).

• Interactive controls SHOULD NOT change roles or accessible names between contexts.

• Labels SHOULD remain aligned with visible text for speech input users.

- Examples:

• Bad: A search function labeled “Search” on one page and “Find” on another.

• Good: Search function consistently labeled “Search” across the entire site.

- Accessibility Considerations:

• Screen reader users rely on consistent labeling for recognition.

• Cognitive users benefit from predictable labels that reduce confusion.

• Voice input users depend on consistent visible labels for accurate command execution.

- Best Practices:

• Establish a design system with standardized labels and roles.

• Audit navigation and controls regularly for consistency.

• Keep labels short, clear, and descriptive.

• Document consistent patterns in developer and content authoring guidelines.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Consistent Identification and Navigation → Consistent Navigation Structure

- Purpose:

• Navigation menus and structures must remain consistent across pages to support predictable user experiences.

• Unexpected changes in navigation order or placement disorient users, especially those relying on screen readers or cognitive predictability.

- Guidelines:

• Primary navigation MUST appear in the same location and order across all pages.

• Secondary navigation (breadcrumbs, sidebars) MUST retain consistent structure.

• Reordering or renaming navigation elements SHOULD be avoided unless functionally required.

• Landmarks (e.g., <nav>, <header>, <footer>) MUST remain consistent across pages.

• Dynamic navigation elements (e.g., expanding menus) MUST retain logical structure when opened.

- Examples:

• Bad: “About Us” link appears in different positions on different pages.

• Good: Navigation structure remains consistent, with “About Us” always in the same location.

- Accessibility Considerations:

• Screen reader users rely on predictable landmarks and navigation order.

• Cognitive users benefit from stable, predictable menu structures.

• Motor-impaired users save effort when navigation consistency reduces exploration.

- Best Practices:

• Define and enforce consistent navigation in the design system.

• Use templates or components to maintain uniform placement.

• Test navigation consistency across pages with both visual and screen reader checks.

• Provide orientation aids like breadcrumbs in multi-level navigation systems.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Consistent Identification and Navigation → Summary

- Consistent identification and navigation ensure predictability, reduce confusion, and support efficient task completion for all users.

- Key requirements:

• Components performing the same function MUST use consistent labels and roles.

• Icons and navigation links MUST have consistent names and alternative text.

• Navigation structures and landmarks MUST remain stable across pages.

• Dynamic elements MUST retain logical and predictable behavior.

- Accessibility Benefits:

• Screen reader users navigate more efficiently with predictable labels and landmarks.

• Cognitive users benefit from familiarity and reduced memory load.

• Voice input users rely on consistent labels for accurate voice commands.

• All users experience smoother navigation and stronger orientation.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for consistent identification and navigation.

• Section 508 and EN 301 549 require consistency in repeated components.

- Best Practices:

• Document and enforce consistency in the design system.

• Audit interfaces regularly for consistent labeling and navigation patterns.

• Provide orientation aids (breadcrumbs, consistent headers/footers).

• Test with assistive technologies and real users for predictable navigation flow.

- Key Takeaway: Consistency in labels, roles, and navigation reduces confusion, supports efficiency, and ensures inclusive user experiences.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Error Identification and Feedback → Introduction

- Purpose:

• Error identification and feedback ensure users can detect, understand, and correct mistakes efficiently.

• Inaccessible error handling leaves users stuck, frustrated, or unable to complete tasks.

- Intended audience: developers, designers, content authors, QA testers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of forms, validation, ARIA associations, and error design patterns.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessible error identification.

• WAI-ARIA guidelines support accessible associations for error handling.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Error Identification and Feedback → Error Detection and Association

- Purpose:

• Errors must be programmatically identified and associated with relevant fields so all users can detect and correct them.

• Without proper association, screen reader users may not know where the error occurred.

- Guidelines:

• Error messages MUST be programmatically linked to their corresponding input fields using aria-describedby or similar methods.

• Required fields MUST be indicated both visually and programmatically.

• Errors MUST be announced to assistive technologies when they appear.

• Multiple errors SHOULD be summarized at the top of the form with links to each field.

• Error messages MUST be concise, specific, and actionable.

- Examples:

• Bad: “Invalid entry” message displayed with no programmatic link to the input.

• Good: “Error: Email address is required” linked to the email field and announced to screen readers.

- Accessibility Considerations:

• Screen reader users rely on programmatic associations to locate and understand errors.

• Cognitive users benefit from clear, specific, and easy-to-follow error text.

• Motor-impaired users require efficient navigation to fields with errors.

- Best Practices:

• Use aria-invalid on fields with errors.

• Provide error summaries with anchor links for quick navigation.

• Write plain-language, actionable error messages.

• Test error identification with multiple screen readers and browsers.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Error Identification and Feedback → Error Feedback and Recovery

- Purpose:

• Users must receive clear feedback about errors and guidance to recover without losing progress.

• Effective recovery prevents abandonment and frustration.

- Guidelines:

• Error feedback MUST describe what went wrong and how to fix it.

• Recovery instructions MUST be provided in plain language.

• Error messages SHOULD remain visible until resolved by the user.

• Users MUST NOT lose previously entered data when errors occur.

• Critical actions (e.g., payment submission) MUST allow confirmation or correction before finalization.

• Inline error messages SHOULD appear near the relevant field, not only at the top of the form.

- Examples:

• Bad: Clearing all entered data when one field fails validation.

• Good: Preserving data and showing “Password must contain at least 8 characters, including one number.”

- Accessibility Considerations:

• Cognitive users rely on plain, actionable instructions for recovery.

• Screen reader users need errors to be programmatically tied to inputs for immediate correction.

• Motor-impaired users benefit from preserved data, reducing re-entry effort.

- Best Practices:

• Preserve form input during errors or page reloads.

• Keep feedback concise, specific, and accessible to assistive technologies.

• Provide confirmation dialogs before critical actions.

• Test error recovery flows with real users and assistive technology.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Error Identification and Feedback → Summary

- Accessible error identification and feedback ensure users can detect, understand, and recover from mistakes without losing progress.

- Key requirements:

• Error messages MUST be programmatically linked to the relevant fields.

• Errors MUST be announced to assistive technologies when they appear.

• Feedback MUST describe the issue and provide actionable recovery steps.

• Data MUST be preserved during errors to avoid re-entry.

• Summaries SHOULD provide quick navigation to errors.

- Accessibility Benefits:

• Screen reader users gain direct, programmatic associations between errors and inputs.

• Cognitive users benefit from clear, plain-language recovery guidance.

• Motor-impaired users avoid retyping content due to preserved input.

• All users experience more forgiving, supportive interactions.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA define requirements for error identification and feedback.

• Section 508 and EN 301 549 mandate accessible error handling globally.

- Best Practices:

• Use aria-invalid and aria-describedby for field-level error associations.

• Keep error messages concise, specific, and solution-focused.

• Preserve all entered data during validation and error handling.

• Test with assistive technologies and across real-world user scenarios.

- Key Takeaway: Accessible error handling reduces frustration, supports recovery, and ensures users can complete tasks with confidence.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Target Size and Spacing → Introduction

- Purpose:

• Target size and spacing ensure that interactive elements are large enough and spaced appropriately for users with motor or vision impairments.

• Small or tightly packed controls create barriers for users with limited dexterity or screen magnification.

- Intended audience: developers, designers, UX testers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of CSS, responsive design, and input methods.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require sufficient target sizes for accessibility.

• W3C usability guidance emphasizes larger touch targets for mobile accessibility.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Target Size and Spacing → Target Size Requirements

- Purpose:

• Interactive elements must meet minimum size requirements to ensure they are usable by people with motor impairments, low vision, or alternative input devices.

- Guidelines:

• Targets MUST be at least 24x24 CSS pixels (WCAG 2.2 Level AA).

• Targets SHOULD be at least 44x44 CSS pixels (recommended for mobile/touch).

• Spacing between adjacent targets MUST prevent accidental activation.

• Critical functions (e.g., Delete, Submit) MUST provide sufficient target area and spacing.

• Overlapping or hidden hit areas MUST be avoided.

- Examples:

• Bad: Tiny icons (10x10 pixels) used as buttons with no spacing.

• Good: Buttons sized 44x44 pixels with adequate spacing between them.

- Accessibility Considerations:

• Motor-impaired users require larger, well-spaced targets to reduce errors.

• Low-vision users using magnification benefit from larger clickable areas.

• Touchscreen users gain usability and reduced accidental taps.

- Best Practices:

• Design with mobile-first principles using larger default target sizes.

• Increase padding rather than relying solely on icon size.

• Test on touch devices, screen magnifiers, and with switch input.

• Apply consistent target sizing across the interface.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Target Size and Spacing → Spacing and Placement

- Purpose:

• Adequate spacing and thoughtful placement of interactive elements reduce accidental activation and support efficient use.

- Guidelines:

• Targets MUST have sufficient spacing between them to prevent errors.

• High-risk targets (e.g., Delete, Confirm) MUST include extra spacing or confirmation.

• Targets SHOULD NOT be placed too close to screen edges where accidental activation is more likely.

• Placement MUST remain consistent across the interface to support predictability.

• Hit areas SHOULD be visually and programmatically aligned to the target.

- Examples:

• Bad: Two small buttons placed immediately next to each other without padding.

• Good: A set of buttons spaced with 8–12px margins, visually grouped and easy to activate.

- Accessibility Considerations:

• Motor-impaired users rely on spacing to reduce unintended taps or clicks.

• Low-vision users benefit when targets are separated and consistently placed.

• All users gain efficiency from predictable, well-spaced layouts.

- Best Practices:

• Group related controls with spacing that reduces error risk but maintains usability.

• Ensure touch targets do not overlap or extend beyond visible boundaries.

• Test spacing with users of screen magnifiers, touch devices, and alternative inputs.

• Follow platform guidelines (iOS, Android, web) for spacing recommendations.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Target Size and Spacing → Summary

- Accessible target size and spacing ensure interactive elements are easy to activate, reducing errors and frustration.

- Key requirements:

• Targets MUST be at least 24x24 CSS pixels (WCAG 2.2 AA).

• Targets SHOULD be at least 44x44 CSS pixels for mobile/touch.

• Adequate spacing MUST be provided to prevent accidental activation.

• High-risk actions MUST include additional spacing or confirmation.

• Placement MUST be consistent and predictable.

- Accessibility Benefits:

• Motor-impaired users avoid accidental taps with larger, well-spaced controls.

• Low-vision users using magnification benefit from increased hit areas.

• Touchscreen users gain smoother, error-free interactions.

• All users enjoy more efficient, forgiving interfaces.

- Regulatory Alignment:

• WCAG 2.2 Level AA specifies minimum target size requirements.

• Section 508 and EN 301 549 mandate target accessibility in digital interfaces.

- Best Practices:

• Design mobile-first with larger default targets and spacing.

• Use padding to increase hit areas while maintaining clean visuals.

• Group controls logically and avoid placing critical actions too close together.

• Test on touchscreens, magnifiers, and with alternative input devices.

- Key Takeaway: Proper target sizing and spacing make interfaces more forgiving, inclusive, and usable for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Motion Actuation → Introduction

- Purpose:

• Motion actuation features (e.g., shaking, tilting, gesturing) must be accessible and have alternatives for users who cannot perform physical gestures.

• Without alternatives, users with motor impairments or limited mobility may be excluded.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate.

- Prerequisites: knowledge of motion sensors, input methods, and device APIs.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible alternatives to motion-based input.

• W3C Device and Sensors API guidance supports accessibility considerations.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Motion Actuation → Alternatives to Motion Input

- Purpose:

• Features triggered by motion must provide alternative input methods so all users can perform the same actions.

- Guidelines:

• Any motion-activated functionality MUST have an equivalent control available via standard input (e.g., button, link, menu option).

• Motion inputs MUST NOT be the only method to complete a task.

• Users MUST be able to disable motion actuation if it causes difficulty or unintended triggers.

• Alternatives SHOULD be as efficient as the motion input.

• Documentation MUST inform users about both motion and alternative input options.

- Examples:

• Bad: A mobile app that only refreshes content by shake gesture.

• Good: A mobile app that supports both shake-to-refresh and a “Refresh” button.

- Accessibility Considerations:

• Motor-impaired users may be unable to perform shaking, tilting, or gesturing.

• Cognitive users may forget or struggle to recall motion-only inputs.

• All users benefit from consistent, discoverable alternatives.

- Best Practices:

• Always pair motion controls with standard input alternatives.

• Ensure alternatives are visible, intuitive, and easy to access.

• Provide a settings option to disable motion-based triggers.

• Test with motion disabled to confirm full functionality remains accessible.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Motion Actuation → Preventing Accidental Activation

- Purpose:

• Motion-based interactions must be designed to prevent accidental or unintended activation, especially for users with motor impairments.

- Guidelines:

• Motion inputs MUST include thresholds (e.g., intensity, duration) to avoid unintentional triggers.

• Users MUST be able to confirm or cancel motion-based actions.

• Critical actions triggered by motion MUST provide confirmation dialogs.

• Sensitivity settings SHOULD be configurable to meet user needs.

• Motion actuation SHOULD NOT override or conflict with assistive technologies.

- Examples:

• Bad: A phone call canceled by an accidental device shake.

• Good: Shake-to-undo feature that requires confirmation before applying changes.

- Accessibility Considerations:

• Motor-impaired users may trigger motion inputs unintentionally.

• Users with tremors or limited control need safeguards to avoid errors.

• All users benefit from confirmable and forgiving interactions.

- Best Practices:

• Implement thresholds to differentiate intentional vs. accidental motions.

• Provide clear undo or cancel options after motion triggers.

• Allow users to adjust sensitivity or disable motion actuation entirely.

• Test with diverse user groups, including those with motor impairments.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Motion Actuation → Summary

- Accessible motion actuation ensures users can interact with motion features without exclusion or unintended triggers.

- Key requirements:

• Motion inputs MUST have equivalent standard input alternatives.

• Users MUST be able to disable motion actuation if needed.

• Accidental activation MUST be prevented through thresholds or confirmations.

• Critical actions triggered by motion MUST require confirmation.

- Accessibility Benefits:

• Motor-impaired users gain alternatives to shaking, tilting, or gesturing.

• Users with tremors or limited control avoid unintended triggers.

• Cognitive users benefit from discoverable, consistent alternatives.

• All users gain from safer, more forgiving interactions.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA require accessible alternatives to motion inputs.

• Section 508 and EN 301 549 mandate inclusive design for sensor-based interactions.

- Best Practices:

• Always pair motion controls with visible, intuitive alternatives.

• Provide settings to disable or adjust motion sensitivity.

• Use thresholds and confirmations to reduce accidental triggers.

• Test with assistive technologies and with users who have motor impairments.

- Key Takeaway: Motion actuation should enhance—not limit—accessibility by ensuring alternatives, preventing errors, and supporting user control.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Orientation and Reflow → Introduction

- Purpose:

• Orientation and reflow ensure that content remains usable in both portrait and landscape modes, and that it adapts without loss of information or functionality.

• Restricting orientation or breaking layouts during reflow creates barriers for users with mobility, vision, or cognitive impairments.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate.

- Prerequisites: knowledge of responsive design, CSS media queries, and adaptive layouts.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require orientation flexibility and reflow support.

• W3C responsive design principles reinforce adaptability for accessibility.

- Conformance definitions:

• MUST = Required

• SHOULD = Strongly recommended

• MAY = Optional/conditional

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Orientation and Reflow → Orientation Flexibility

- Purpose:

• Content must be usable in both portrait and landscape orientations, supporting users who rely on fixed device positions or assistive technology mounts.

- Guidelines:

• Applications MUST NOT restrict content to a single orientation unless essential (e.g., banking check deposit, games).

• Layouts MUST adapt seamlessly when switching between portrait and landscape.

• Orientation changes MUST preserve functionality and data entry.

• Users MUST be able to complete tasks regardless of device orientation.

• Orientation SHOULD be tested on multiple devices and screen sizes.

- Examples:

• Bad: An app that forces portrait orientation and breaks when rotated.

• Good: A responsive website usable in both portrait and landscape without loss of features.

- Accessibility Considerations:

• Wheelchair or mobility device users may lock devices in one orientation.

• Low-vision users using screen magnification may prefer specific orientations.

• All users benefit from flexibility when devices are rotated.

- Best Practices:

• Use responsive CSS media queries to handle orientation changes.

• Test with device rotation on phones, tablets, and desktops.

• Preserve form input and state when switching orientation.

• Provide clear documentation if orientation locking is functionally required.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Orientation and Reflow

Introduction

- Purpose:

• Orientation and reflow ensure content remains usable in both portrait and landscape modes, adapting without loss of information or functionality.

• Restricting orientation or breaking layouts during reflow creates barriers for users with mobility, vision, or cognitive impairments.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate.

- Prerequisites: responsive design, CSS media queries, adaptive layouts.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require orientation flexibility and reflow.

• W3C responsive design principles reinforce adaptability.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Orientation Flexibility

- Guidelines:

• Applications MUST NOT restrict content to a single orientation unless essential.

• Layouts MUST adapt seamlessly between portrait and landscape.

• Orientation changes MUST preserve data entry and functionality.

• Users MUST be able to complete tasks regardless of device orientation.

• Orientation SHOULD be tested on multiple devices.

- Examples:

• Bad: An app forces portrait orientation and breaks when rotated.

• Good: A responsive site usable in both orientations with no loss of function.

- Accessibility Considerations:

• Wheelchair/mobility device users may lock devices in one orientation.

• Low-vision users with magnification may prefer specific orientations.

• All users benefit from flexibility when rotating devices.

- Best Practices:

• Use CSS media queries to handle orientation changes.

• Test rotation across devices and OS platforms.

• Preserve user input/state when orientation changes.

• Document orientation locking only if functionally required.

Reflow and Responsive Layouts

- Guidelines:

• Content MUST reflow without loss of information up to 400% zoom (WCAG requirement).

• Horizontal scrolling MUST NOT be required for blocks of text.

• Layouts MUST adapt across breakpoints without clipping or overlap.

• Interactive elements MUST remain usable after reflow.

• Content SHOULD adapt gracefully to narrow viewports (mobile-first design).

- Examples:

• Bad: A two-column layout forcing horizontal scroll at 200% zoom.

• Good: Layout reflows to a single column at higher zoom levels.

- Accessibility Considerations:

• Low-vision users rely on zoom without forced horizontal scrolling.

• Mobile users require responsive layouts for small viewports.

• Cognitive users benefit from uncluttered, simplified reflow views.

- Best Practices:

• Use relative units (em, rem, %) for fluid scaling.

• Design mobile-first with progressive enhancement for larger screens.

• Test zoom at 200%–400% to ensure content reflows correctly.

• Validate with screen magnifiers and responsive design tools.

Summary

- Key requirements:

• Orientation MUST be flexible unless essential.

• Content MUST reflow without loss up to 400% zoom.

• Horizontal scrolling MUST NOT be required for reading text blocks.

• Layouts MUST remain functional and accessible across orientations and sizes.

- Accessibility Benefits:

• Low-vision and magnifier users gain zoom flexibility.

• Motor-impaired users can keep devices fixed in one orientation.

• Cognitive users benefit from simplified reflow and uncluttered layouts.

• All users enjoy better responsive design and device adaptability.

- Regulatory Alignment:

• WCAG 2.2 AA: Orientation (SC 1.3.4), Reflow (SC 1.4.10).

• Section 508 and EN 301 549 mandate adaptability.

- Best Practices:

• Apply responsive design principles across all breakpoints.

• Use mobile-first design with progressive enhancement.

• Preserve content and state during orientation and reflow.

• Test with zoom, screen magnifiers, and real devices.

- Key Takeaway: Orientation and reflow must not restrict users — accessible design adapts to user needs, device positions, and zoom levels without breaking functionality.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Visual Presentation

Introduction

- Purpose:

• Visual presentation must be flexible, clear, and perceivable for all users.

• Inflexible or cluttered presentation creates barriers for low-vision, dyslexic, or cognitive users.

- Intended audience: developers, designers, content authors.

- Technical level: easy to moderate.

- Prerequisites: CSS styling, typography, responsive design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible visual presentation.

• W3C guidance emphasizes readability, contrast, and user control.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Text Presentation and Contrast

- Guidelines:

• Text MUST maintain a contrast ratio of at least 4.5:1 (AA) or 3:1 for large text.

• Users MUST be able to resize text up to 200% without loss of content or function.

• Backgrounds and patterns MUST NOT obscure text legibility.

• Justified text SHOULD be avoided; left alignment supports readability.

• Line height MUST be at least 1.5x font size; spacing must remain adjustable.

- Examples:

• Bad: Gray text on a light gray background.

• Good: Dark text with 4.5:1 contrast on a white or solid background.

- Accessibility Considerations:

• Low-vision users require high contrast and resizable text.

• Dyslexic users benefit from increased line spacing and left alignment.

• Cognitive users gain readability from simple, uncluttered text presentation.

- Best Practices:

• Use relative font sizes (em, rem).

• Validate contrast using automated tools.

• Provide user-controlled style overrides.

• Test with zoom and high-contrast settings enabled.

Layout and Clarity

- Guidelines:

• Layouts MUST avoid horizontal scrolling for text blocks.

• White space SHOULD be used to separate sections and reduce clutter.

• Decorative content MUST NOT interfere with readability.

• Content MUST adapt to different screen sizes and orientations.

• Information SHOULD be grouped logically for cognitive clarity.

- Examples:

• Bad: Overlapping text and images in a fixed layout.

• Good: Clean, responsive layout with white space and logical grouping.

- Accessibility Considerations:

• Low-vision and screen magnifier users require uncluttered layouts.

• Cognitive users benefit from simplified, consistent presentation.

• All users find clearer layouts easier to read and navigate.

- Best Practices:

• Use flexible grids and spacing.

• Avoid placing text over moving backgrounds.

• Validate layouts with screen magnifiers and responsive tools.

• Provide options to toggle simplified views.

Summary

- Key requirements:

• Text MUST meet WCAG contrast and resizing standards.

• Layouts MUST remain functional without horizontal scrolling.

• Line spacing and clarity MUST support readability.

• Visual clutter MUST be minimized with logical grouping and whitespace.

- Accessibility Benefits:

• Low-vision users access high-contrast, resizable text.

• Dyslexic and cognitive users benefit from simplified, uncluttered presentation.

• All users enjoy clearer, more usable visual design.

- Regulatory Alignment:

• WCAG 2.2 (SC 1.4.3, 1.4.4, 1.4.10, 1.4.12).

• Section 508 and EN 301 549 require readable, adaptable presentation.

- Best Practices:

• Test color contrast and resizing in real scenarios.

• Apply mobile-first, responsive design.

• Provide style override support for personalization.

• Test layouts with assistive technology and user groups.

- Key Takeaway: Visual presentation must prioritize clarity and adaptability — ensuring text remains readable, layouts uncluttered, and contrast sufficient for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Reading and Focus Order

Introduction

- Purpose:

• Reading and focus order must follow a logical, meaningful sequence so users can perceive and interact with content correctly.

• Disordered focus or reading paths confuse users, disrupt comprehension, and block task completion.

- Intended audience: developers, designers, QA testers.

- Technical level: moderate.

- Prerequisites: knowledge of DOM order, tab index, ARIA landmarks, and screen reader navigation.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require logical reading and focus order.

• W3C accessibility guidance emphasizes predictable navigation flow.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Reading Order

- Guidelines:

• DOM order MUST match the visual reading order.

• Content read by screen readers MUST convey meaning equivalent to the visual layout.

• CSS positioning MUST NOT disrupt logical reading flow.

• Headings, lists, and tables MUST be structured to preserve reading order.

• Dynamic content MUST be inserted logically into the DOM.

- Examples:

• Bad: Visually reordering content with CSS flex but leaving DOM order inconsistent.

• Good: Structuring DOM to match intended visual flow, ensuring screen reader consistency.

- Accessibility Considerations:

• Screen reader users depend on DOM sequence to follow content meaningfully.

• Cognitive users require logical reading flow to maintain comprehension.

• All users benefit from predictable content ordering.

- Best Practices:

• Validate DOM vs. visual reading order regularly.

• Use semantic HTML structure to reinforce natural reading order.

• Avoid CSS tricks that disrupt reading flow (absolute positioning, flex reordering).

• Test with screen readers and keyboard-only navigation.

Focus Order

- Guidelines:

• Focus MUST follow a logical, predictable sequence matching visual flow.

• Tab order MUST NOT skip or loop unintentionally.

• Interactive elements MUST be reachable in a consistent order.

• Modal dialogs MUST trap focus until closed.

• Dynamic content MUST receive focus appropriately when triggered.

- Examples:

• Bad: Focus jumps unpredictably between unrelated controls.

• Good: Tabbing progresses logically through form fields, buttons, and links.

- Accessibility Considerations:

• Keyboard-only users depend on logical tabbing order.

• Screen reader users rely on focus sequence to interact effectively.

• Cognitive users require consistent, predictable focus flow.

- Best Practices:

• Use tabindex sparingly; rely on DOM order for natural flow.

• Ensure modals and overlays trap and restore focus correctly.

• Validate focus order with keyboard-only testing.

• Test across browsers and assistive technologies.

Summary

- Key requirements:

• DOM reading order MUST match visual order.

• Focus order MUST follow logical, predictable sequences.

• Modals and dynamic content MUST manage focus responsibly.

• Interactive elements MUST be consistently reachable.

- Accessibility Benefits:

• Screen reader users perceive content meaningfully.

• Keyboard-only users navigate predictably.

• Cognitive users follow clear, sequential flows.

• All users gain consistency across tasks and devices.

- Regulatory Alignment:

• WCAG 2.2 (SC 1.3.2, 2.4.3).

• Section 508 and EN 301 549 mandate logical reading and focus order.

- Best Practices:

• Align DOM and visual order.

• Validate tabbing sequences with QA and assistive tech.

• Document focus patterns in design systems.

• Test modals and dynamic updates rigorously.

- Key Takeaway: Reading and focus order must remain logical and predictable, aligning DOM and visual flow to support comprehension and interaction for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Pointer Gestures and Cancellation

Introduction

- Purpose:

• Pointer gestures (swipe, pinch, drag) and input cancellation must be accessible to ensure users can operate controls without exclusion or errors.

• Complex gestures without alternatives exclude users with motor or cognitive impairments.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate.

- Prerequisites: pointer events, ARIA, alternative input design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require gesture accessibility and cancellation.

• W3C pointer event guidance emphasizes alternatives and error prevention.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Pointer Gestures

- Guidelines:

• Complex gestures (pinch, multi-finger swipe) MUST NOT be the only method of input.

• Single-point alternatives (tap, click, button) MUST be provided.

• Gesture-based inputs MUST remain discoverable and consistent across contexts.

• Instructions MUST describe gestures clearly if required.

- Examples:

• Bad: Photo viewer requires pinch to zoom with no button alternative.

• Good: Photo viewer supports both pinch and plus/minus zoom buttons.

- Accessibility Considerations:

• Motor-impaired users may not manage multi-finger gestures.

• Cognitive users may forget or misinterpret gestures.

• Voice input and switch device users require simple alternatives.

- Best Practices:

• Pair complex gestures with standard controls.

• Use consistent, documented gestures.

• Test gesture accessibility with multiple input methods.

Pointer Cancellation

- Guidelines:

• Users MUST be able to cancel or reverse pointer actions before finalization.

• Critical actions (delete, purchase) MUST require confirmation before execution.

• Down-events (touchstart, mousedown) MUST NOT trigger irreversible actions.

• Interfaces MUST provide undo or cancel options where feasible.

- Examples:

• Bad: Purchase completes immediately on tap down, no cancel.

• Good: Tap selects, release confirms, with Cancel option provided.

- Accessibility Considerations:

• Motor-impaired users may trigger inputs unintentionally.

• Cognitive users benefit from safeguards against accidental actions.

• All users gain confidence with reversible, forgiving interactions.

- Best Practices:

• Trigger actions on up-event (touchend, mouseup) rather than down-event.

• Provide undo/redo or confirmation dialogs for critical actions.

• Test pointer cancellation across touch, mouse, and pen inputs.

Summary

- Key requirements:

• Complex gestures MUST have simple, single-point alternatives.

• Actions MUST NOT trigger irreversibly on down-event.

• Cancel/undo MUST be provided for critical interactions.

• Gestures MUST remain consistent and documented.

- Accessibility Benefits:

• Motor-impaired users avoid exclusion from multi-finger gestures.

• Cognitive users gain clarity with simple, discoverable inputs.

• All users avoid accidental, irreversible actions.

- Regulatory Alignment:

• WCAG 2.2 (SC 2.5.1, 2.5.2).

• Section 508 and EN 301 549 require gesture accessibility and error prevention.

- Best Practices:

• Pair gestures with standard controls.

• Trigger actions on up-events and provide undo.

• Test gestures and cancellation with multiple AT and inputs.

• Document expected gestures in the design system.

- Key Takeaway: Gestures and pointer inputs must include simple alternatives and safeguards — ensuring accessibility, preventing errors, and supporting user confidence.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Error Prevention, Help, and Support

Introduction

- Purpose:

• Error prevention, help, and support ensure users can avoid mistakes, receive guidance, and complete tasks successfully.

• Lack of safeguards and help documentation creates barriers, especially for cognitive, motor, or low-vision users.

- Intended audience: developers, designers, content authors, QA testers.

- Technical level: moderate.

- Prerequisites: knowledge of form design, ARIA, and usability principles.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require error prevention and accessible help.

• W3C usability principles emphasize support and forgiveness.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Error Prevention

- Guidelines:

• Forms collecting critical data (legal, financial, submissions) MUST allow review, confirmation, and correction before final submission.

• Required fields MUST be clearly indicated and validated in real time.

• Critical actions (delete, purchase) MUST provide confirmation dialogs or undo.

• Input formats MUST be enforced consistently (e.g., phone number, date).

• Defaults SHOULD reduce errors (e.g., pre-filled known information).

- Examples:

• Bad: Submitting a payment form immediately with no review or undo.

• Good: Payment form shows confirmation screen with editable details.

- Accessibility Considerations:

• Cognitive users benefit from reduced complexity and undo options.

• Motor-impaired users avoid repeated effort when defaults reduce typing.

• All users gain confidence in completing critical tasks.

- Best Practices:

• Provide confirmation before irreversible actions.

• Use inline validation and clear required field indicators.

• Preserve user input on error or reload.

Help and Support

- Guidelines:

• Context-sensitive help MUST be available at point of need.

• Instructions MUST be provided in plain, concise language.

• Tooltips, labels, and descriptions MUST be accessible to screen readers.

• Contact and support options MUST be accessible via multiple channels (chat, phone, email).

• FAQs and help documentation SHOULD be structured with headings and search.

- Examples:

• Bad: Help text only shown as an inaccessible hover tooltip.

• Good: Help icon linked to accessible, searchable documentation.

- Accessibility Considerations:

• Screen reader users require programmatically associated help text.

• Cognitive users benefit from simplified, clear instructions.

• All users gain efficiency from context-sensitive help.

- Best Practices:

• Provide persistent help icons or links near complex fields.

• Ensure tooltips are accessible and keyboard operable.

• Use progressive disclosure for help to reduce clutter.

• Offer multiple contact methods for support.

Summary

- Key requirements:

• Critical actions MUST include confirmation or undo.

• Required fields MUST be clear and validated accessibly.

• Context-sensitive help MUST be provided in plain, accessible formats.

• Support MUST be available through accessible, multimodal channels.

- Accessibility Benefits:

• Cognitive users gain support and error recovery guidance.

• Motor-impaired users reduce input effort with defaults and error prevention.

• All users benefit from confidence, forgiveness, and accessible help.

- Regulatory Alignment:

• WCAG 2.2 (SC 3.3.1–3.3.6).

• Section 508 and EN 301 549 require error prevention and help access.

- Best Practices:

• Build confirmation, undo, and inline validation into workflows.

• Provide help and support consistently across platforms.

• Test with assistive technologies and real user scenarios.

• Document help and support patterns in the design system.

- Key Takeaway: Accessible design prevents errors, preserves input, and provides clear help — ensuring all users can complete tasks with confidence and support.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Authentication and Cognitive Support

Introduction

- Purpose:

• Authentication must be accessible, avoiding reliance on memory, complex tasks, or inaccessible methods.

• Inaccessible logins or verifications can completely block access for users with disabilities.

- Intended audience: developers, security engineers, UX designers.

- Technical level: moderate to advanced.

- Prerequisites: knowledge of authentication patterns, security requirements, and ARIA.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessible authentication.

• W3C guidance emphasizes cognitive accessibility in authentication.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Accessible Authentication Methods

- Guidelines:

• Authentication MUST NOT rely on cognitive tests (puzzles, CAPTCHAs without alternatives).

• Password entry MUST allow paste from password managers and assistive technologies.

• Multi-factor authentication MUST provide accessible methods (e.g., SMS, app, hardware key).

• Biometric authentication SHOULD include alternatives for users unable to use fingerprints or face recognition.

• Login flows MUST remain consistent and avoid unnecessary complexity.

- Examples:

• Bad: CAPTCHA requiring users to identify distorted text with no audio alternative.

• Good: CAPTCHA providing accessible audio or logic puzzle alternative.

- Accessibility Considerations:

• Cognitive users struggle with memory-based tests or puzzles.

• Screen reader users need accessible multi-factor authentication flows.

• Motor-impaired users require alternatives to biometrics or touch-only verification.

- Best Practices:

• Support password managers and paste functionality.

• Offer multiple, accessible MFA options.

• Avoid reliance on inaccessible visual CAPTCHAs.

• Document accessible authentication methods in design standards.

Cognitive Support During Authentication

- Guidelines:

• Authentication tasks MUST be simple, clear, and free of unnecessary steps.

• Instructions MUST be provided in plain language and programmatically associated with fields.

• Error recovery MUST preserve entered data.

• Timeout policies MUST allow extensions or easy re-authentication.

• Help and support MUST be accessible during authentication.

- Examples:

• Bad: Session timeout forces re-entry of all login data without warning.

• Good: Timeout warning with option to extend session or save progress.

- Accessibility Considerations:

• Cognitive users benefit from reduced steps and clear instructions.

• Screen reader users require form labels and error associations.

• All users need flexible, forgiving authentication flows.

- Best Practices:

• Provide progress indicators for multi-step logins.

• Allow session extension before timeout.

• Keep error feedback concise, specific, and actionable.

• Test authentication with multiple assistive technologies.

Summary

- Key requirements:

• Authentication MUST avoid memory or puzzle-based barriers.

• MFA MUST include accessible alternatives.

• Password entry MUST support password managers and paste.

• Timeout and error handling MUST preserve progress and support recovery.

- Accessibility Benefits:

• Cognitive users avoid exclusion from memory-intensive logins.

• Screen reader and motor-impaired users gain full access to authentication tasks.

• All users enjoy smoother, more forgiving login experiences.

- Regulatory Alignment:

• WCAG 2.2 (SC 3.3.7, 3.3.8).

• Section 508 and EN 301 549 mandate accessible authentication globally.

- Best Practices:

• Provide multiple authentication pathways.

• Ensure CAPTCHAs have accessible alternatives.

• Support assistive technologies in all login and MFA flows.

• Document and test authentication patterns for accessibility.

- Key Takeaway: Authentication must remain accessible, supporting all users with inclusive methods, clear instructions, and flexible recovery options.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Timeouts and Interruptions

Introduction

- Purpose:

• Timeouts and interruptions must be accessible to ensure users do not lose progress or become disoriented during tasks.

• Rigid or unannounced time limits exclude users with cognitive, motor, or assistive technology needs.

- Intended audience: developers, designers, product managers.

- Technical level: moderate.

- Prerequisites: knowledge of session management, form handling, and alerts.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible timeout management.

• W3C usability principles emphasize flexibility and predictability.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Timeouts

- Guidelines:

• Users MUST receive warnings before timeouts occur.

• Users MUST have the option to extend sessions or save work.

• Timeout warnings MUST be accessible to screen readers and assistive tech.

• Sessions with critical data MUST allow re-authentication without data loss.

• Timeout durations SHOULD be generous to support cognitive and motor needs.

- Examples:

• Bad: Session ends abruptly after 5 minutes with no warning.

• Good: Warning provided 2 minutes before timeout with “Extend Session” option.

- Accessibility Considerations:

• Cognitive users require clear warnings and recovery paths.

• Motor-impaired users may need more time to complete tasks.

• Screen reader users must receive timely, programmatic alerts.

- Best Practices:

• Provide customizable timeout settings when possible.

• Use ARIA live regions for accessible timeout announcements.

• Preserve input and progress across re-authentication.

Interruptions

- Guidelines:

• Interruptions MUST NOT automatically take focus or disrupt user tasks without warning.

• Users MUST be able to dismiss or postpone non-critical notifications.

• Critical alerts MUST be presented accessibly and without blocking controls.

• Background auto-refresh MUST NOT reset focus or input state.

- Examples:

• Bad: A modal ad takes focus mid-form entry without dismissal option.

• Good: Notification appears non-intrusively with “Remind me later” option.

- Accessibility Considerations:

• Cognitive users need control over timing of interruptions.

• Screen reader users require accessible alerts that don’t hijack focus.

• All users benefit from reduced disruption during tasks.

- Best Practices:

• Use ARIA role="alert" or live regions for notifications.

• Allow users to control notification timing.

• Avoid resetting focus or input on refresh.

• Test with screen readers and keyboard-only navigation.

Summary

- Key requirements:

• Timeouts MUST provide warnings and extension options.

• Critical data MUST be preserved during session recovery.

• Interruptions MUST be dismissible, non-intrusive, and accessible.

• Auto-refresh MUST NOT disrupt input or focus.

- Accessibility Benefits:

• Cognitive and motor-impaired users gain flexibility to complete tasks.

• Screen reader users receive accessible, predictable alerts.

• All users avoid lost progress and disruptive interruptions.

- Regulatory Alignment:

• WCAG 2.2 (SC 2.2.1, 2.2.6).

• Section 508 and EN 301 549 mandate accessible session management.

- Best Practices:

• Warn and extend sessions accessibly.

• Provide user control over interruptions.

• Test timeout and notification flows with assistive tech.

• Document timeout policies in accessibility guidance.

- Key Takeaway: Accessible timeouts and interruptions preserve progress, reduce disruption, and keep users in control of their tasks.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Animations and Motion

Introduction

- Purpose:

• Animations and motion effects must be accessible and not cause harm or disorientation.

• Overuse of motion or lack of user control can trigger vestibular disorders, distract cognitive users, or block task completion.

- Intended audience: developers, designers, UX testers.

- Technical level: moderate.

- Prerequisites: CSS animations, JavaScript interactions, user preference APIs.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible animation handling.

• W3C guidance emphasizes reduced motion and user control.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Reduced Motion Support

- Guidelines:

• Interfaces MUST respect prefers-reduced-motion settings at the OS/browser level.

• Animations MUST be minimized, simplified, or disabled when reduced motion is requested.

• Essential animations (e.g., progress indicators) MUST remain perceivable in accessible alternatives.

• Motion-triggered parallax or background effects MUST be disabled with reduced motion.

- Examples:

• Bad: Parallax scrolling persists even with reduced motion enabled.

• Good: Motion disabled; static alternative visuals presented.

- Accessibility Considerations:

• Vestibular disorder users may experience dizziness from motion.

• Cognitive users may be distracted by excessive animations.

• All users gain control and comfort when motion is adjustable.

- Best Practices:

• Use CSS prefers-reduced-motion media query.

• Provide manual settings to disable animations.

• Test reduced motion with OS/browser settings enabled.

Animation Control

- Guidelines:

• Users MUST be able to pause, stop, or hide non-essential animations lasting more than 5 seconds.

• Animations critical to understanding content MUST have accessible alternatives.

• Motion effects MUST NOT trigger without user initiation unless essential.

• Repetitive or flashing animations MUST avoid seizure risks (flash threshold compliance).

- Examples:

• Bad: Auto-playing banner animation with no pause control.

• Good: Banner includes Pause/Stop buttons accessible via keyboard and screen reader.

- Accessibility Considerations:

• Epilepsy and seizure-prone users must be protected from flashing content.

• Low-vision users may miss content if overwhelmed by continuous motion.

• All users gain focus and control with adjustable animation settings.

- Best Practices:

• Provide visible controls to pause/stop/hide animations.

• Test animations against flash thresholds (no >3 flashes per second).

• Validate with keyboard, screen readers, and reduced motion settings.

Summary

- Key requirements:

• Interfaces MUST respect reduced motion settings.

• Users MUST be able to pause/stop animations >5s.

• Essential animations MUST have accessible alternatives.

• Motion effects MUST NOT trigger harm or distraction.

- Accessibility Benefits:

• Vestibular disorder users avoid dizziness and discomfort.

• Cognitive users focus without distraction.

• All users gain choice and control over motion.

- Regulatory Alignment:

• WCAG 2.2 (SC 2.3.1, 2.2.2, 2.3.3).

• Section 508 and EN 301 549 mandate safe, controllable animations.

- Best Practices:

• Implement prefers-reduced-motion support.

• Provide manual controls for motion.

• Test with accessibility settings and flashing guidelines.

• Document motion policies in design standards.

- Key Takeaway: Animations must support user control and reduced motion preferences, ensuring inclusivity, safety, and focus for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Status Messages and Alerts

Introduction

- Purpose:

• Status messages and alerts must be accessible so users are aware of important updates without losing context.

• Inaccessible alerts prevent screen reader and cognitive users from receiving critical feedback.

- Intended audience: developers, designers, QA testers.

- Technical level: easy to moderate.

- Prerequisites: knowledge of ARIA roles, live regions, and dynamic content.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible status communication.

• W3C ARIA guidance defines best practices for alerts and live regions.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Status Messages

- Guidelines:

• Status messages (confirmation, success, error) MUST be programmatically conveyed to assistive technologies.

• Messages MUST NOT steal focus unless necessary.

• Live regions (aria-live) MUST be used for updates without requiring focus changes.

• Status categories MUST align with ARIA roles (alert, status, log, progressbar).

• Messages SHOULD be concise and in plain language.

- Examples:

• Bad: “Form submitted successfully” only shown visually with no ARIA.

• Good: Success message in aria-live region, read by screen readers.

- Accessibility Considerations:

• Screen reader users depend on programmatic announcements of status.

• Cognitive users benefit from short, plain-language feedback.

• All users need awareness of system state without losing context.

- Best Practices:

• Use role="status" for neutral messages, role="alert" for critical.

• Avoid focus changes unless required for safety or recovery.

• Keep status messages persistent until user acknowledges.

Alerts and Notifications

- Guidelines:

• Alerts MUST be delivered accessibly without blocking current tasks.

• Critical alerts MUST use role="alert" or aria-live="assertive".

• Non-critical notifications SHOULD use aria-live="polite".

• Users MUST be able to dismiss or control recurring alerts.

• Visual and auditory cues MUST be paired with programmatic announcements.

- Examples:

• Bad: Pop-up alert steals focus during typing.

• Good: Non-intrusive alert announced with aria-live="polite".

- Accessibility Considerations:

• Screen reader users require alerts without disruptive focus shifts.

• Cognitive users need control over timing and clarity of alerts.

• All users benefit from dismissible, predictable notifications.

- Best Practices:

• Provide “close” or “dismiss” options for alerts.

• Use progressive levels of urgency (polite → assertive).

• Test alerts with multiple screen readers and assistive technologies.

• Document alert patterns in the design system.

Summary

- Key requirements:

• Status messages MUST be programmatically conveyed via ARIA live regions.

• Alerts MUST align with urgency (status, polite, assertive).

• Focus MUST NOT be stolen except when essential.

• Alerts MUST be dismissible and predictable.

- Accessibility Benefits:

• Screen reader users gain awareness of updates in real time.

• Cognitive users benefit from concise, plain-language feedback.

• All users experience smoother workflows with non-intrusive alerts.

- Regulatory Alignment:

• WCAG 2.2 (SC 4.1.3).

• Section 508 and EN 301 549 mandate accessible status communication.

- Best Practices:

• Match ARIA roles with urgency of the alert.

• Avoid disruptive focus changes.

• Provide dismiss controls and persistence until read.

• Test across devices and assistive tech.

- Key Takeaway: Status messages and alerts must be accessible, timely, and non-intrusive — ensuring all users stay informed without disruption.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Drag-and-Drop

Introduction

- Purpose:

• Drag-and-drop functionality must be accessible so users can manipulate items regardless of input method.

• Exclusive reliance on pointer dragging excludes keyboard, switch, and screen reader users.

- Intended audience: developers, designers, QA testers.

- Technical level: moderate to advanced.

- Prerequisites: DOM events, ARIA drag-and-drop patterns, keyboard interaction design.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 mandate accessible drag-and-drop alternatives.

• W3C ARIA Authoring Practices provide guidance on accessible drag-and-drop.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Keyboard and Alternative Input Support

- Guidelines:

• Drag-and-drop MUST be operable via keyboard alone.

• Users MUST be able to select, move, and drop items without pointer input.

• Controls MUST provide clear focus and state feedback (e.g., "Item selected for dragging").

• Drop targets MUST be reachable via keyboard navigation.

• Alternatives (buttons, menus) SHOULD be provided for non-drag workflows.

- Examples:

• Bad: File upload only allows dragging files into a box.

• Good: File upload allows drag-and-drop and a “Browse Files” button.

- Accessibility Considerations:

• Motor-impaired users may be unable to drag with precision.

• Screen reader users require programmatic announcements of drag/drop states.

• All users benefit from consistent, predictable alternatives.

- Best Practices:

• Use ARIA drag-and-drop roles and states (aria-grabbed, aria-dropeffect).

• Provide redundant controls like Add/Remove or Move Up/Down buttons.

• Test with keyboard-only and assistive technologies.

Feedback and State Announcements

- Guidelines:

• Users MUST receive programmatic feedback during drag-and-drop (e.g., "Item grabbed").

• Drop targets MUST announce availability and acceptance.

• Errors (invalid drop) MUST be communicated clearly.

• Visual feedback (highlighting) MUST be paired with programmatic cues.

- Examples:

• Bad: Visual highlight only, no screen reader feedback.

• Good: “Move Item A before Item B” announced when hovering drop target.

- Accessibility Considerations:

• Screen reader users rely on ARIA feedback for orientation.

• Cognitive users benefit from explicit instructions and feedback.

• All users gain clarity when actions and states are announced.

- Best Practices:

• Pair visual cues with ARIA live regions or role announcements.

• Provide undo functionality for drag-and-drop actions.

• Keep instructions concise, persistent, and accessible.

Summary

- Key requirements:

• Drag-and-drop MUST be operable without a pointer.

• Programmatic feedback MUST announce drag states and drop availability.

• Alternatives (buttons, menus) MUST be provided for all tasks.

• Errors and successes MUST be announced accessibly.

- Accessibility Benefits:

• Motor-impaired users gain non-drag methods for task completion.

• Screen reader users receive feedback on drag/drop state.

• Cognitive users benefit from clear instructions and undo options.

• All users enjoy predictable, forgiving drag-and-drop interactions.

- Regulatory Alignment:

• WCAG 2.2 (SC 2.1.1, 4.1.2).

• Section 508 and EN 301 549 mandate accessible alternatives to drag-and-drop.

- Best Practices:

• Implement ARIA drag-and-drop roles/states with redundant controls.

• Provide undo for drag/drop errors.

• Test with keyboard-only and multiple assistive technologies.

• Document drag-and-drop accessibility patterns in design systems.

- Key Takeaway: Drag-and-drop must include accessible alternatives, programmatic feedback, and forgiving recovery — ensuring usability for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Custom Widgets and Components

Introduction

- Purpose:

• Custom widgets and components (sliders, tabs, accordions) must be accessible to all users.

• Inaccessible custom controls block interaction for screen reader, keyboard, and switch users.

- Intended audience: developers, component library maintainers, QA testers.

- Technical level: advanced.

- Prerequisites: ARIA authoring practices, JavaScript, semantic markup.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require accessible custom widgets.

• W3C ARIA Authoring Practices define design patterns for custom components.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

Keyboard Accessibility

- Guidelines:

• All widgets MUST be fully operable via keyboard.

• Focus management MUST follow ARIA practices (e.g., roving tabindex for menus).

• Keyboard shortcuts MUST be documented and intuitive.

• Tab, arrow, and Enter/Space behaviors MUST align with platform conventions.

• Escape MUST close modals, popups, or menus consistently.

- Examples:

• Bad: Custom slider works only with mouse drag.

• Good: Slider operable with Left/Right arrow keys and programmatically labeled.

- Accessibility Considerations:

• Keyboard-only and switch users depend on correct tab/focus flow.

• Cognitive users benefit from consistent, predictable key behavior.

• All users expect platform-standard keyboard conventions.

- Best Practices:

• Follow WAI-ARIA keyboard interaction design patterns.

• Provide visible focus indicators for active states.

• Test with keyboard-only navigation.

Screen Reader and ARIA Support

- Guidelines:

• Widgets MUST use appropriate ARIA roles, properties, and states.

• Labels and instructions MUST be programmatically associated.

• State changes (expanded, selected) MUST be announced via ARIA.

• ARIA MUST NOT be misused (avoid duplicating semantics with roles).

• Complex widgets MUST follow ARIA design patterns (e.g., tabs, accordions).

- Examples:

• Bad: Accordion with no ARIA-expanded state.

• Good: Accordion with role="button" and aria-expanded announced correctly.

- Accessibility Considerations:

• Screen reader users rely on accurate ARIA semantics.

• Cognitive users gain clarity from clear, consistent labeling.

• All users benefit from standardized widget behavior.

- Best Practices:

• Use semantic HTML where possible; ARIA only when needed.

• Validate ARIA with automated and manual testing.

• Provide instructions for custom interaction patterns.

Summary

- Key requirements:

• Widgets MUST be keyboard operable.

• ARIA roles and states MUST be applied correctly.

• Labels and state changes MUST be programmatically conveyed.

• Widgets MUST follow WAI-ARIA Authoring Practices.

- Accessibility Benefits:

• Keyboard and screen reader users gain equal access to custom components.

• Cognitive users benefit from predictable, documented behaviors.

• All users enjoy reliable, consistent interaction with widgets.

- Regulatory Alignment:

• WCAG 2.2 (SC 2.1.1, 4.1.2).

• Section 508 and EN 301 549 mandate accessible custom widgets.

- Best Practices:

• Prioritize semantic HTML over ARIA when possible.

• Align keyboard/focus behavior with platform conventions.

• Document widget accessibility patterns in design systems.

• Test with real assistive technologies and users.

- Key Takeaway: Custom widgets must be accessible through semantic markup, ARIA, and consistent keyboard support — ensuring full usability for all.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Personalization and Adaptability

Introduction

- Purpose:

• Personalization and adaptability allow users to adjust interfaces to meet their accessibility needs without breaking functionality.

• Inflexible designs that lock layouts or styles prevent users from creating usable experiences.

- Intended audience: developers, designers, content authors.

- Technical level: moderate.

- Prerequisites: CSS customization, responsive design, AT settings.

- Standards alignment:

• WCAG 2.2 success criteria (Levels A and AA).

• Section 508 and EN 301 549 require personalization support.

• W3C guidance highlights adaptability as a core accessibility principle.

- Conformance definitions:

• MUST = Required | SHOULD = Strongly recommended | MAY = Optional/conditional

User-Controlled Presentation

- Guidelines:

• Users MUST be able to override styles (fonts, colors, spacing).

• Text resizing, color inversion, and custom stylesheets MUST NOT break content.

• Interfaces MUST NOT disable browser zoom or OS personalization features.

• Users SHOULD be able to adjust line height, font, and spacing for readability.

• Documentation SHOULD explain personalization options.

- Examples:

• Bad: Page disables pinch-to-zoom on mobile.

• Good: Page respects OS-level large text settings without clipping.

- Accessibility Considerations:

• Low-vision users depend on zoom and color contrast personalization.

• Dyslexic users benefit from custom fonts and spacing adjustments.

• All users gain comfort from flexible presentation.

- Best Practices:

• Respect browser/OS accessibility settings.

• Avoid fixed layouts that break with personalization.

• Test with custom CSS, zoom, and OS text scaling.

Summary

- Key requirements:

• Interfaces MUST support user overrides for styles and scaling.

• Personalization MUST NOT break layouts or functionality.

• Users SHOULD be able to adapt spacing, fonts, and colors.

- Accessibility Benefits:

• Low-vision and dyslexic users access interfaces tailored to their needs.

• Cognitive users benefit from simplified, personalized layouts.

• All users gain comfort and control over their experience.

- Regulatory Alignment:

• WCAG 2.2 (SC 1.4.4, 1.4.12).

• Section 508 and EN 301 549 mandate personalization features.

- Best Practices:

• Allow zoom, scaling, and custom styles.

• Test personalization features across platforms.

• Document supported personalization patterns.

- Key Takeaway: Personalization and adaptability ensure accessibility by letting users control how content is presented without breaking usability.

Domain I: Creating Accessible Web Solutions

Subsection: Accessible Consistency, Error Handling, and Wrap-Up

Consistency

- Purpose:

• Consistency ensures repeated elements (labels, icons, navigation) behave predictably across a product.

• Inconsistent interfaces create confusion and cognitive load.

- Guidelines:

• Identical components MUST have consistent roles, labels, and behavior.

• Navigation MUST remain consistent across all pages.

• Dynamic elements MUST retain predictable structure.

• Changes in terminology or order SHOULD be avoided unless functionally required.

- Examples:

• Bad: Search icon labeled differently (“Find” vs. “Search”) across pages.

• Good: Same icon, same label, same role across the interface.

- Accessibility Considerations:

• Screen reader users rely on consistent labels and landmarks.

• Cognitive users benefit from predictable naming and structure.

• All users save time and reduce errors through consistency.

Error Handling (Recap)

- Guidelines:

• Errors MUST be programmatically linked to their input fields.

• Users MUST receive clear, plain-language instructions for recovery.

• Data MUST be preserved during errors to avoid re-entry.

• Critical actions MUST allow confirmation or undo.

- Examples:

• Bad: Error message “Invalid” shown with no context.

• Good: “Error: Email address is required” linked to the input field.

- Accessibility Considerations:

• Screen reader users require programmatic error associations.

• Cognitive users rely on actionable recovery steps.

• Motor-impaired users benefit when input is preserved.

Wrap-Up

- Key requirements:

• Consistency in labels, navigation, and components MUST be enforced.

• Errors MUST be clearly identified, linked, and recoverable.

• Interfaces MUST provide forgiving, predictable user experiences.

• Accessibility MUST be embedded across design, development, and QA.

- Accessibility Benefits:

• Screen reader users navigate predictably with consistent patterns.

• Cognitive and motor-impaired users benefit from forgiving error recovery.

• All users gain from clarity, consistency, and predictability.

- Regulatory Alignment:

• WCAG 2.2 Levels A and AA cover consistency, errors, and recovery.

• Section 508 and EN 301 549 mandate consistent, accessible design globally.

- Best Practices:

• Document consistency in the design system.

• Regularly audit interfaces for consistent labeling/navigation.

• Test error handling flows with assistive technologies.

• Validate predictable behavior across pages and platforms.

- Key Takeaway: Accessible web solutions demand consistency and error forgiveness — ensuring users can navigate predictably, recover from mistakes, and complete tasks with confidence.

Domain II: Identifying Accessibility Issues in Web Solutions

Subtopic: Web Accessibility Testing Methods and Tools (Deque Course Content)

- Accessibility testing should be integrated throughout the development lifecycle. Early testing during planning/design prevents costly late-stage fixes.

- Planning phase: define accessibility requirements and testing scope; create tests from design annotations (color contrast, headings, labels, keyboard interaction).

- Build phase: developers should run automated tools (e.g., axe DevTools), unit tests, and keyboard/screen reader checks. Catching issues here reduces cost.

- Testing phase: validate against WCAG 2.2 AA using both automated and manual methods. Include QA staff, developers, accessibility experts, and end users with disabilities.

- Reports must be detailed, actionable, and understandable to non-specialists.

Role-Based Checklists

- Designers: ensure descriptive page titles, semantic headings, skip links/landmarks, logical reading/focus order, meaningful link text, sufficient color contrast, visible focus indicators, alt text, accessible forms and dynamic content, standard widgets preferred over custom.

- Developers: use semantic HTML, ensure logical reading order, provide alt text (including background images), maintain visible keyboard focus, logical tab order, avoid traps, mark up tables correctly, associate labels with controls, provide accessible error handling, avoid unexpected context changes, support timing controls.

- QA Testers: confirm automated checks (axe), semantic structure, logical reading order, correct alt text, visible focus, keyboard-only operation, table markup, programmatic labels, error handling, accessible custom controls, no unexpected context changes, timing accommodations.

Content-Based Checklists

- Semantic Structure & Navigation: meaningful titles, lang attribute, landmarks, hierarchical headings, semantic links, unique navigation, logical reading/tab order, proper list and table markup, unique IDs, valid HTML.

- Non-Text Content: all non-text content must have text alternatives. Informative/actionable images need concise meaningful alt text; decorative images must be null alt or role="presentation"; complex images need long descriptions. SVGs, canvas, icon fonts, multimedia, plug-ins, and documents must have appropriate alt text, roles, and accessibility API support.

- Visual Design & Colors: information not by color alone, maintain 4.5:1 contrast for text (3:1 large text), controls and focus indicators ≥3:1 contrast, high contrast mode compatible, logical layout, adequate spacing, readable typography, avoid full justification, user-customizable fonts.

- Responsive Design & Zoom: support text reflow without horizontal scrolling at 320 CSS px, no zoom restrictions, orientation not locked, scalable icons/graphics, functional at 200% zoom, touch targets ≥44x44 px.

- Multimedia & Motion: captions, transcripts, audio descriptions as required (see WCAG levels); avoid auto-play >3s; prevent flashing >3Hz; allow pause/stop/hide; minimize parallax/motion; accessible players with keyboard and screen reader support.

- Input Methods: support mouse, keyboard, touch, and assistive tech. Must ensure focus visibility, logical tab order, no traps, focus management in dynamic updates, non-conflicting custom shortcuts, sufficient target sizes, operable gestures, motion actuation alternatives.

- Forms: labels programmatically associated and visible; group labels for related inputs; meaningful instructions; required fields indicated; input purposes defined; error and success messages visible, meaningful, and programmatically associated; custom widgets follow ARIA practices.

- Dynamic Content/AJAX/SPA: inform users of changes via page title, moved focus, or ARIA live; prevent silent updates; session timeouts must warn and allow extension; no auto-refresh; lazy load placeholders must inform; infinite scroll must allow keyboard navigation; SPA history must update for back/forward.

Testing Routine

- Automated scans (axe) catch ~1/3 of issues.

- Manual tests cover: keyboard input (tab order, focus, traps), touch input, visual focus, color use, alt text, media accessibility, landmarks, headings, links, form labels, form validation, custom widgets.

- Screen reader testing is essential; provide shortcuts and resources for JAWS, NVDA, Narrator, VoiceOver, TalkBack.

- Reports: clear, detailed, reproducible bug tickets with descriptive title, issue, expected result, platform/AT used, impact/severity, steps, visuals, remediation advice.

- Prioritize bugs by user impact, business priority, location/volume, remediation cost. Categories: Critical, Serious, Moderate, Minor.

Automated Tools

- axe DevTools and related extensions. Automated tests find missing attributes, contrast failures, etc., but must be validated manually.

- Types: unit testing, integration testing, regression testing, monitoring. Automation is a supplement, not a substitute.

Summary

- Accessibility testing requires integration in planning, design, build, and testing phases.

- Use both automated and manual methods, guided by WCAG 2.2 AA.

- Apply role-based and content-based checklists.

- Ensure coverage of key input methods, media, and dynamic content.

- Provide clear bug reporting and prioritization.

- Automated tools increase efficiency, but manual validation ensures user impact is addressed.

Domain II: Identifying Accessibility Issues in Web Solutions

Subtopic: Semantic Structure and Navigation (Deque Course Content)

- Semantic markup is essential for assistive technologies to convey page structure and content meaningfully.

- Page Titles: every page needs a <title>; must update on context change, be concise, unique, and match main heading.

- Language: primary page language must be identified in <html lang>; inline changes marked with lang attribute; language codes valid.

- Landmarks: use semantic elements (<header>, <nav>, <main>, <footer>) or ARIA roles; minimize duplication; multiple landmarks must be labeled distinctly.

- Headings: text that looks like headings must use heading tags in logical hierarchy; <h1> for main content start; headings concise, informative, not skipped in order.

- Links: semantic link markup required; link text must describe purpose, be programmatically determinable, not repeat “link”; indicate new windows, downloads, or external sites where relevant; visually distinguishable and focusable.

- Navigation: provide skip links; ensure logical reading and tab order; maintain consistent navigation order across pages; identify current page in navigation.

- Tables: data tables must have captions/names, headers marked with <th>, proper associations for simple and complex structures; layout tables must not use data semantics.

- Lists: proper <ul>, <ol>, <li> usage.

- Iframes: require non-empty, descriptive, unique titles; hidden if not meaningful.

- Other semantic elements: use <strong>, <em>, <blockquote>, <q>, <code>, <pre>, <del>, <ins>, <mark> correctly.

- Parsing/Validity: valid, complete tags; unique IDs; follow parent-child rules; avoid deprecated markup.

Bad Example: Heading structure created only with bold text, not <h> elements.

Good Example: <h1>Main topic</h1> <h2>Subtopic</h2>, providing logical outline.

---

Subtopic: Non-Text Content (Images, SVG, Canvas, Multimedia)

- All non-text content requires text alternatives.

- Informative images: must have meaningful alt (≤150 characters); concise, purpose-driven.

- Decorative/redundant images: alt="" or role="presentation" or CSS background; avoid duplicate adjacent text.

- Actionable images: alt must describe action/destination (e.g., “Search” not “Magnifying glass”).

- Form inputs type="image": require meaningful alt.

- Animated images: must allow pause/stop/hide if >5s; no flashing >3Hz.

- Complex images: brief alt + long description (on-page, link, expandable, or aria-describedby).

- Images of text: avoid; only allowed if essential (e.g., logos).

- CSS background images: informative/actionable require alt in HTML content; decorative should be ignored.

- Image maps: client-side maps need alt on <img> and <area> elements; server-side maps discouraged.

- SVG: <img role="img"> or inline with <title>/<desc> and aria-labelledby; concise alt ≤150 chars; long description for complex SVGs; decorative SVGs hidden with aria-hidden.

- Icon fonts: informative/actionable need role="img" + aria-label; decorative use aria-hidden.

- Canvas: assign role="img"; provide aria-label, aria-labelledby, or fallback text; long description with aria-describedby; background fill required; ensure keyboard accessibility; use ARIA roles for custom controls.

- Multimedia: prerecorded video requires captions; prerecorded audio requires transcript; live multimedia requires captions; videos must include audio descriptions for important visuals.

- Plug-ins: <object> requires alt; HTML alternatives preferred.

- Documents: Word, PDF, EPUB must follow accessibility principles; PDFs tagged; EPUB in EPUB 3 format.

Bad Example: <img src="final\_notice.png" alt=""> (missing meaningful alt).

Good Example: <img src="final\_notice.png" alt="Final Notice stamp">.

Domain II: Identifying Accessibility Issues in Web Solutions

Subtopic: Visual Design and Colors (Deque Course Content)

- Visual information must remain perceivable and understandable across abilities, devices, and zoom levels.

- Color: never rely on color alone to convey meaning (e.g., error states must include text/icons).

- Contrast:

- Normal text requires ≥4.5:1 contrast.

- Large text (≥18pt regular or ≥14pt bold) requires ≥3:1.

- Active control boundaries and visual focus indicators require ≥3:1.

- Graphics conveying information require ≥3:1 contrast.

- Links: must be visually distinguishable beyond color unless 3:1 contrast plus hover/focus styling provided.

- High Contrast Mode: essential content must remain visible; designs must not override OS contrast settings.

- Layout: content blocks should be distinct with margins/padding; one clear visual focus; labels adjacent to controls.

- Typography: fonts must be readable; line spacing ≥1.5; paragraph spacing ≥1.5x line spacing; max 80 characters per line; avoid full justification; fonts should be user-customizable.

- Target size: touch targets ≥44x44 px with ≥6 px spacing.

- Hidden/CSS-generated content: informative CSS text must have alt and aria-hidden; decorative must be aria-hidden.

- Hover/focus-triggered content must be dismissible, hoverable, and persistent.

Bad Example: pale gray 12pt text on white background (contrast too low).

Good Example: 16pt dark text on white with 4.8:1 contrast ratio.

---

Subtopic: Responsive Design and Zoom

- Responsive design benefits mobile users and those zooming in.

- Text, images, objects, tables, UI, and video must reflow without horizontal scrolling at 320 CSS px width.

- Simplify/reduce features for small screens or zoomed viewports.

- Zoom: page must remain functional and readable at 200% text zoom.

- Spacing: letter/word/line/paragraph spacing must be adjustable without content loss.

- Magnification: text, icons, and graphics should scale clearly without distortion.

- Orientation: content must not be locked to portrait or landscape unless essential.

- Touch target sizes must remain large enough under zoom.

Bad Example: Fixed-width table requiring horizontal scroll at 200% zoom.

Good Example: Flexible grid layout reflows to single column on narrow viewports.

---

Subtopic: Multimedia, Animations, and Motion (Summary & Checklist)

- Four main accessibility methods: captions, transcripts, audio descriptions, sign language interpretation.

- Captions: required for prerecorded multimedia and live video with narration; recommended for live audio. Must be synchronized, accurate, and identify speakers/background sounds.

- Transcripts: required for prerecorded audio; recommended for video; must include essential visual events.

- Audio descriptions: required where visual info is not in audio track.

- Sign language: optional, but useful for Deaf users.

- Clear audio: avoid or reduce background noise/music, especially during narration.

- Seizures: avoid flashing >3Hz; keep flashing areas <21,824 px²; use tools like PEAT.

- Motion: avoid excessive parallax or auto-motion; allow pause/stop/hide; ensure moving backgrounds have sufficient text contrast.

- Media players: must be fully keyboard-accessible, expose names/roles/values, support accessibility features, allow caption customization, remember preferences, and provide full screen mode.

- Auto-play: discouraged; if audio auto-plays >3s, must provide stop/pause/mute control at top of page.

Bad Example: Video autoplays with loud music, no captions, no stop control.

Good Example: HTML5 player with keyboard focus order, captions toggle, transcript link, audio description track, and customizable captions.

Domain II: Identifying Accessibility Issues in Web Solutions

Subtopic: Device-Independent Input Methods (Deque Course Content)

- All functionality must be operable through keyboard, mouse, touch, and assistive technologies.

- Keyboard:

- All controls must be focusable and operable without mouse.

- Logical tab order matching visual order.

- Visible focus indicator required.

- Focus management must be logical (e.g., dialog focus and return).

- Avoid keyboard traps; users must always move focus away.

- Provide instructions for unusual custom controls.

- Touch:

- Touch targets large (≥44x44 px).

- Targets must not be too close together.

- Touch functionality must have keyboard/touchscreen equivalents.

- Custom gestures must have alternatives since screen readers override gestures.

- Motion actuation (shake, tilt) must have alternate input methods.

- Input purpose:

- Identify input purpose programmatically where possible (e.g., autocomplete attributes).

- Provide consistent labeling and instructions.

- Single-key shortcuts: must be remappable, off-switchable, or active only when relevant UI element is focused.

- Users must be able to turn off, pause, or adjust motion-triggered input features.

Bad Example: Custom swipe gesture to delete email with no keyboard alternative.

Good Example: Swipe gesture plus visible “Delete” button focusable by keyboard.

---

Subtopic: Forms, Labels, and Validation

- Labels:

- Every form control must have a programmatically associated visible label.

- Related fields grouped with fieldset/legend.

- Labels must be adjacent to controls.

- Instructions:

- Must be visible and clear; avoid relying only on placeholder text.

- Instructions must be programmatically associated with inputs.

- Input purposes must be communicated (e.g., email, phone).

- Errors:

- Descriptions must be programmatically associated with inputs.

- On submission, errors must be announced to screen readers.

- Real-time validation must be announced immediately.

- Error messages must be descriptive and actionable.

- Success feedback must be perceivable to all users.

- Autofill should be supported.

- Custom widgets must follow ARIA authoring practices.

- Controls must remain near content they modify.

Bad Example: Required input marked only with red border, no text label.

Good Example: “Email (required)” label with aria-required, plus error message linked with aria-describedby.

---

Subtopic: Dynamic Content, AJAX, and Single-Page Applications (SPAs)

- Users must be informed of updates caused by dynamic content.

- Acceptable methods:

- Updating page <title>.

- Moving focus to new content.

- ARIA live regions.

- Dynamic updates must not be silent or invisible to AT users.

- Session timeouts: must warn users and allow extension.

- Auto-refreshing or redirecting pages without warning must be avoided.

- Lazy loading/infinite scroll: provide clear placeholder messaging and keyboard-accessible loading controls.

- SPAs must update browser history and state to allow back/forward navigation.

- Modal dialogs: must trap focus inside until closed, then return focus logically.

- Forms: AJAX error messages must be announced.

- Prevent unexpected context changes on focus or input without user initiation.

Bad Example: SPA loads new page content without updating history or informing screen readers.

Good Example: SPA announces “New content loaded” in live region and updates document title/history.

---

Subtopic: Accessibility Bug/Issue Management

- Bug reports must be clear, specific, and reproducible.

- Essential details:

- Descriptive title.

- Description of issue.

- Expected behavior.

- Platform, browser, AT used.

- Steps to reproduce.

- Screenshots or recordings.

- Severity/impact classification.

- Suggested remediation guidance.

- Prioritization factors:

- User impact (critical for blocking tasks vs. cosmetic).

- Business risk or legal exposure.

- Frequency of occurrence.

- Cost/feasibility of fix.

- Categories:

- Critical: blocks primary functionality (e.g., cannot log in with keyboard).

- Serious: major barrier but with workarounds.

- Moderate: causes confusion but not a complete block.

- Minor: low impact, cosmetic, or alternative path available.

Bad Example: Bug filed as “Keyboard issue” with no reproduction steps.

Good Example: Bug filed as “Tab key skips Submit button in Chrome + NVDA; expected sequential tab order. Impact: critical for form completion.”

Domain II: Identifying Accessibility Issues in Web Solutions

Subtopic: Automated and Manual Testing Methods

- Automated testing:

- Tools like axe DevTools can quickly flag missing alt text, low contrast, missing labels, and ARIA misuse.

- Automated scans typically catch ~30% of WCAG failures; manual testing required for full coverage.

- Automated tests should be integrated into development (unit tests, integration, regression checks, monitoring).

- Automated monitoring can detect regressions but does not replace expert review.

- Manual testing:

- Essential for confirming usability and end-user impact.

- Includes keyboard-only navigation checks, screen reader testing, magnification, high-contrast settings, and testing across browsers/devices.

- Requires understanding of how people with disabilities actually use AT and coping strategies.

- QA should validate that automated tool findings are real and impactful.

- Assistive technology testing:

- Screen readers (JAWS, NVDA, VoiceOver, TalkBack) must be tested with browsers where support is strongest (e.g., NVDA with Firefox).

- Screen magnifiers: check zoom, panning, and content reflow.

- High contrast modes: confirm content remains visible and meaningful.

- Speech input: confirm all functionality can be triggered without mouse or keyboard.

- End-user impact:

- Test with personas (low vision, cognitive, mobility, Deaf/HoH, blind).

- Verify readability, navigation ease, clear instructions, and absence of distractions.

- Documentation of testing:

- Provide detailed accessibility evaluation reports with clear pass/fail, supporting screenshots, code samples, and remediation guidance.

- Reports should map findings to WCAG 2.2 success criteria for traceability.

Bad Example: Only running an automated scan and declaring site accessible.

Good Example: Running automated scan, validating manually with keyboard and screen reader, confirming user impact, documenting with WCAG references.

---

Subtopic: Testing Routine (Step-by-Step)

1. Run an automated check (axe or similar).

2. Conduct screen reader tests for headings, landmarks, forms, tables, and dynamic content.

3. Test keyboard accessibility:

- Tab focus order, visibility, and logical navigation.

- Activation of all controls by Enter/Space.

- No keyboard traps.

- Focus management on dialogs, validation errors, AJAX updates.

4. Test touch device accessibility:

- Ensure controls work with gestures and alternatives.

- Check target size and spacing.

- Validate with screen reader on (TalkBack, VoiceOver).

5. Test for color reliance:

- Verify info is conveyed by text or symbols, not color alone.

6. Test alternative text quality for images and non-text content.

7. Test video/audio accessibility: captions, transcripts, audio descriptions.

8. Test landmarks, headings, and link text for clarity and programmatic determinability.

9. Test forms: labels, instructions, errors announced to AT, logical grouping.

10. Test custom widgets: ARIA roles, keyboard operability, proper states/values.

Bad Example: Tab order cycles illogically through page sections.

Good Example: Tab order follows visual reading order, focus indicator always visible.

---

Subtopic: Accessibility QA and Integration

- Accessibility QA must be continuous across SDLC.

- Designers provide accessibility annotations (color contrast, alt text, keyboard behavior).

- Developers run automated/unit tests and basic keyboard/AT checks.

- QA validates accessibility in staging with real AT.

- Accessibility experts or trained testers review results and guide remediation.

- Organizations should adopt accessibility testing methodologies such as WCAG-EM for enterprise-scale audits.

- Procurement must integrate accessibility conformance reviews (e.g., VPATs based on WCAG/Section 508).

Bad Example: Testing only after site launch.

Good Example: Accessibility considered in design specs, code reviews, automated pipelines, QA testing, and procurement.