From Remediation to Proliferation: Mainstreaming the Accessibility of Web

University Library Innovation Fund Report

August 21, 2017

University of Illinois Library Innovation Fund
A11yFirst
<for CKEditor>
http://a11yfirst.library.illinois.edu/

Submitted by:

- JaEun Jemma Ku, Ph.D., Principal Investigator, A*11yFirst Project*
- Jon Gunderson, Ph.D., Principal Investigator, *A11yFirst Project*
- Nicholas Hoyt, Development and Design Lead, *A11yFirst Project*

In partnership with:

- Dena Strong, Technology Services, University of Illinois

Table of Contents

Abstract

The goal of this project is innovating the web content authoring model through changes in the Content Management System's (ie. WordPress) WYSIWYG (What You See Is What You Get) Editor user interface and through just-in-time feedback to authors during the content authoring process.

To accomplish these goals, the project team started with a thorough conceptual modeling process while benchmarking the ONENET text editor developed by the State of Illinois Department of Human Services. After requirement gathering, the project team documented text editor features and chose the code base, CKSource, to work with.

Project deliverables included content authoring interface modifications including: text editor toolbar design, heading plugin with accessibility help information, link plugin, block format plugin, inline style plugin, element level style features, and inserted image plugin (in progress). In addition, helpful usability testing data, which validates and improves on the A11yFirst Editor design, was delivered. It is expected that A11yFirst editor will be deployed after fall semester 2017 in the University of Illinois Library's WordPress instance, rather than immediately, to minimize current workflow disruption.

Going forward, Disability Resources and Educational Services (DRES) at the University of Illinois will provide financial resources for a student programmer's work throughout the year. After that, developers on campus will be invited to contribute. Eventually, the A11yFirst project will became an open source project in a GitHub repository so any developer can contribute.

Ongoing evolution for the project includes multi faceted collaboration: (1) user research for the conceptual model of the A11yFirst editor with School of Information Science, (2) A11yFirst Editor deployment to campus WordPress services (such as publish.illinois.edu) and Drupal CMS services and (3) sharing the project knowledge with CKsource, the creator of CKEditor, as well as sending feedback on UI design improvement.

The project team appreciates the University Library's support very much — it made this innovative project possible.

Introduction

The goal of A11yFirst editor project is changing the content authoring model through changes in Content Management System (ie. WordPress) WYSIWYG (What You See, What You Get) Editor user interface and through just in time feedback to authors during the authoring process.

Project Goals

1. Decreasing the likelihood that inaccessible content pages are created.

The learning curve to understand accessibility requirements and recommendations (i.e. WCAG rules and ARIA roles) is steep. Once the principles are understood, they must be implemented in efficient working code, a task that is beyond the capability of most end users and requiring time to develop expertise that most developers do not have.

2. Forestalling expensive and ineffective retrofitting of accessibility.

Past practice has been to hold a small number of Library IT staff solely responsible for ensuring the accessibility of numerous web pages created by library subject experts and staff. This requires that inefficient and largely ineffective remediation work be completed after services have been developed, typically via the application of one-off solutions.

3. Creating an inclusive online environment for students, faculty and staff with disabilities.

Inclusive Illinois 2015 impact report addresses well how “people from diverse backgrounds working together identify more creative solutions to problems.” These efforts will also contribute to enhancing web content quality in teaching and research.

4. Prevention of Office for Civil Right (OCR) complaints.

Higher educational institutions face liability for inaccessible web content and technologies. Several universities have already gone through OCR complaints for not ensuring equal access to its website for individuals with disabilities.

5. Growth of In-house web accessibility expertise in the University Library.

The University Library has been getting immense assistance from the Disability and Resources and Educational Services (DRES) to release IT projects such as the Library Gateway Website project, Easy Search and the Interlibrary Loan site. DRES helps with IT projects across all campus units, not only library web projects. As a result of being a campus-wide service, DRES is in high demand and it usually takes a few weeks to receive final accessibility reports. If the University Library can grow in-house experts through this innovation project, Library web projects will be more efficient while ensuring IT accessibility with the close partnership with DRES.

Project Process

Conceptual Modeling

As the first step of designing the A11yFirst editor, the project group started with the "Conceptual modeling" process. *Conceptual modeling* represents a group of vocabularies that relate to a user's task domain when using a text-editor. It attempts to provide terminology for the objects, attributes and available actions that map a user's tasks when creating or editing a document. Conceptual modeling of the A11yFirst editor is documented in A11y First Project: CKEditor Modifications — Conceptual Model v4.1

The core idea of the conceptual model defines the functionality of A11yFirst editor.

Functionality

- Proactively promote thinking in terms of structural blocks as the main components of a document
- For specific block types, prompt the user for semantic clarification (e.g., whether list is bulleted or numbered, image text alternative), or when appropriate, simply limit the available choices (e.g., heading levels)
- Make it possible to style inline content, but discourage users from reaching for this type of action on first impulse (i.e., first, think semantically)

Here is the example of how conceptual model distinguishes various types of actions for A11yFirst editor.

Actions

1) Block actions vs. text/inline actions

- Need to make a clear distinction between text/inline actions and block actions.
- For example, for all of the block objects that contain text/inline, all of the text/inline actions are available, and thus can be factored out when considering the available actions on a particular block.
- Need to encourage/prioritize block actions (insert block, convert to another block) over text/inline styling actions.

2) Block insertions and splits

- A block can only be inserted at the top level of the parent context, which may be (1) the document, (2) a table data cell or (3) a block quote.
- A paragraph can be split into two paragraphs by using the enter key.
- A list can be split into two lists using the enter key twice in succession.

- These actions are necessary in order to insert a block, i.e. the user is not permitted to insert a block within a paragraph or list.
- Likewise, tables need a mechanism for splitting into two tables.
- In general, blocks need to maintain their integrity (you can't easily demolish them by inserting random blocks within them) but must also be easily divided.

3) Block selection, conversion and deletion

- Need to make it clear how blocks are selected, to allow modification of their properties, and what needs to happen in order to delete them.
- For example, a paragraph can be converted to a heading or list item by placing the cursor at the beginning, end or anywhere in between and choosing an action.
- On the other hand, a paragraph can be deleted only by deleting all of its text/inline content. A list can be deleted only by deleting all of its list items, and list items are deleted by deleting their text/inline content.

4) Relationships

- A simple block (heading, paragraph, list item) contains text/inline only
- A compound block (list, table, table column, table row) contains sub-blocks (list items, data cells)
- A table data cell is a hybrid -- it may contain text/inline or blocks (e.g. paragraphs, list/list items)
- An image is unique -- no visible text is required (optional caption?) but it must have alt text (a short description) and may have a long description

Requirement Gathering

The project team worked on high level project goals as summarized by three aspects.

1. Technical restrictions (not allowing users to skip heading levels)
2. Guides for users in creating accessible content via Q & A wizard style interface or dialog UI
3. Accessibility evaluations such as just-in-time checks for accessibility.

Feature Documentation

The team also spent time on analyzing the OneNet Text editor user interface and implementation by the State of Illinois by referring to the OneNet Functional Requirement page. As a result, we documented each of the A11yFirst plugin features and created guidelines pertain to project priority, type of checker, code requirements, and user interface/user's work flow.[1] Later this documentation became design specific documentation in the A11yFirst project GitHub repository. Here are some guidelines we used for feature documentation.

[1] These information can be be found at feature documentation page

Development Priority[2]

- High: Must have feature for launch of version 1.0
- Medium: Nice to have, but not required for version 1.0
- Low: Feature we'd like to have eventually, but is likely high development cost to lower benefit

Type of Accessibility Check

- Real Time: These checks are done without any user initiation of the check. It would be tied to event listeners (in the CKEditor API) for element insert/modification, and would either correct problems automatically or give the user an immediate warning/encouragement to modify.
- Manual Check: The user must initiate this check either from the checks menu or a "check everything" option, or it will be triggered on save/update.

Code Requirements

- Automatically produce valid, semantically-correct HTML 5 code.
- Allow authorized users to edit HTML directly.
- Use HTML Tidy (or a similar code validator) to check/correct HTML code edited by users.

User Workflow

- Leverage vertical menu: Where it makes sense, present drop down (like) menus.
- Where it is more complex, open a modal window on selecting that feature/group (and in some cases a hybrid, where links in the menu can themselves open a modal).
- When prompting users about errors, visually display the elements to make it easier to see how they are are related (see OneNet)–most likely as separate visualizations (one for headings, one for lists, etc.)
- Run some checks on requesting publish (not on save draft/autosave) and prompt user to verify/remediate (possible) errors.

Choice of Codebase

Two Editors: TinyMCE and CKEditor

To code the A11yFirst WYSIWYG editor, our team researched options for a code base. We could not use OneNet Text source code since it was developed in-house for specific document needs and the code was developed more than 10 years ago. Therefore, we decided to code from existing WYSIWYG code bases not to reinvent the wheel. The competing two candidates were TinyMCE by Ephox and CKeditor by CKsource. In spite of TinyMCE as default editor for WordPress, the team decided to go with CKeditor source code base due to well documented API, more options for plugins (ie: 464 plugin for Ckeditor vs 48 plugins for TinyMCE), active developers community in CKSource(116,648 registered developers in

[2] This uses a prioritization technique, MoSCoW method

CKsource vs 23,140 in TinyMCE), and free plugin for core functionality such as “Paste from Word” in CKsource. (This functionality comes with paid enterprise version in TinyMCE, but is free in CKEditor).

Comparison

This comparison is based on internet articles[3] and feature and API documentation from TinyMCE and CKEditor sites.

Pre-packaged/Pre-existing Features

Criteria	**CKEditor**	**TinyMCE**
Most Recent Version	4.4.1	4.2.7
Documentation	http://docs.ckeditor.com/	https://www.tinymce.com/docs/
License	GPL, LGPL, MPL / Commercial	LGPL, Commercial
Developer Guide	CKEditor Guides	TinyMCE Developer Guides
API Documentation	CKEditor API documentation	TinyMCE API Reference
Other Available SDKs (Software Development Kit)	Plugin, Widget, Skin	Plugin, Themes and Skins

[3] *http://www.krizalys.com/article/ckeditor-vs-tinymce
**http://socialcompare.com/en/comparison/javascript-online-rich-text-editors
(It is also confirmed that all the bugs are closed for CKEditor)

Accessibility Checker	a11y checker (Available as Add-on and with payment) Documentation/Demo	a11y checker (Available as plugin and with Enterprise subscription) Demo (check the bottom of the page)
Installing Add-ons/Plugins Options	Add-ons repository (464 plugins)	Plugin list (48)
Expert Opinion/Recommendation	Matt King, APG member (Accessibility expert is working on CKEditor project)	Default text editor for WordPress. WordPress is the choice of University of Illinois Library Content Management System (CMS).
Keyboard Accessibility	Keyboard shortcuts	Keyboard shortcuts
Browser Compatibility	Desktop environments: -Internet Explorer (8.0 and 9.0 – close to full support, 10 and 11 – full support, 9.0 Quirks Mode and 9.0 Compatibility Mode – limited support.) -Firefox, Chrome, Safari, Microsoft Edge, Opera: Latest stable release – full support. Reference	Firefox (3.0+), IE (8+), Chrome (1.0+), Opera (11.0+), Safari (5+) 3.x version reference

Copy-Paste (from Word)	Available as *Free* add-on function	Copy-paste from Word plugin is available in *Premium(payment required)* features
Image Upload and Handling	Feature rich*	Limited options* MoxieManager: Image/File manager server side component. (Paid component)
Developer Community	Forum ● Registered developers: 116,648 ● Number of downloads: 16,419,009 ● StackOverflow CKEditor forum	TinyMCE forum ● Registered users: 23,140 ● Number of downloads: unknown ● WordPress default editor

Functionality/Methods

Criteria	**CKEditor**	**TinyMCE**
Functionality Overview	End-user features/Integration features	N/A

Saving Content	Getting and Saving Data in CKEditor (Use CKEditor Javascript API, Ajax)	When the <form> is submitted the TinyMCE editor mimics the behavior of a normal HTML <textarea>during the post.
Filtering Content	Advanced Content Filter (Filters incoming HTML content by transforming and deleting disallowed elements, attributes, classes, and styles.)	Content filtering (The way the editor handles the input and output of content.)
How Plugin Works	CKEditor Plugins	Internal Plugin Code Example (hint: use plugins string) External Code Example (hint: use external_pluginsobject)
Copy and Paste from Word	Paste from Word (Free plugin for the editor)	Paste From Word (included in the TinyMCE Enterprise download)
Consistency (Content Editable, Editable Content)	● CKEDITOR.dom.range ● CKEDITOR.plugins.context ● Menu CKEDITOR.htmlParser	tinymce.dom.Selection.getRng()

Test Case

Document Object Model (DOM) selection/manipulation and Copy and Paste functionality were tested for two editors. In particular, DOM selection and manipulation was easier in CKEditor.

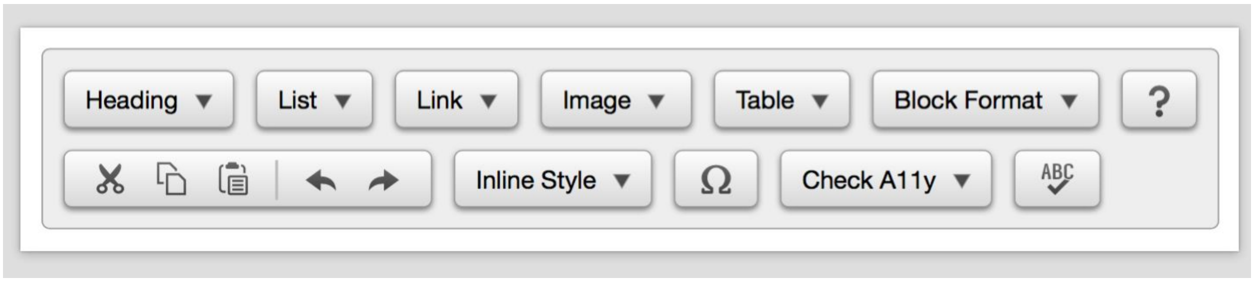
Project Deliverables

Editor Toolbar Design/Configuration

Based on the conceptual model, the team configured the CKeditor toolbar differently from the default CKeditor toolbar to integrate accessible content authoring concepts into practice.

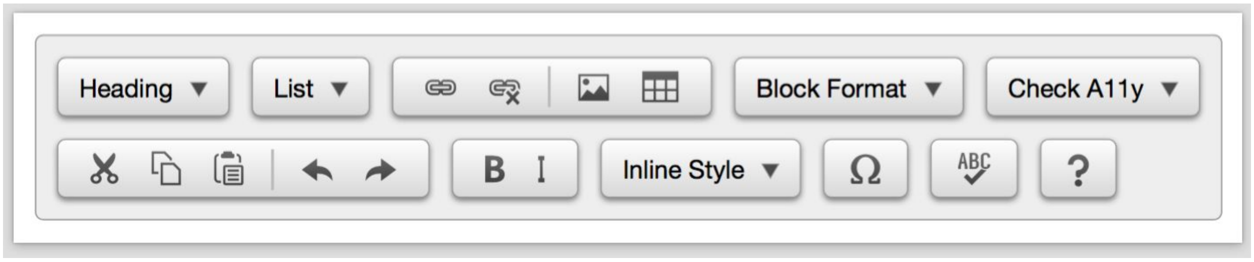
Editor Design

[Figure] Initial Toolbar configuration on November 2016



[caption: Started with menu style action items based on objects.]

[Figure] Toolbar configuration on January 2017



[caption: Started to deploy button-type menus, which is the default UI for CKEditor. Heading, List, Block format, inline style object menus were retained.]

Current editor design is reflected in the A11yFirst editor demonstration site and it will continue to be modified as we add more A11yFirst plugins.

Editor Configuration

The process for configuring the toolbar using CKbuilder is below.

1. Use the standard configuration as the starting point
2. Select the following from the available plugins
    - Code Snippet
    - Content Templates
    - Find / Replace
    - Language
    - Show Blocks
3. These plugins may be needed later
    - Accessibility Checker
    - Dropdown menu manager
    - List Style
    - Panel Button
4. A11yFirst plugins and their dependencies

Plugins

- A11yFirst editor configuration
- Blockformat
- Heading
- Inline style
- Element level styling features
- Inserted image plugin

API Dependencies

- A11yFirst editor configuration
- Blockquote
- Code snippet
- Menu button
- Richcombo
- Remove format

Design Specification

Some of the most intense work by the project team focused on the design specification for the A11yFirst Editor.
For example, heading plugin design specification states accessibility requirements, user interface components, task descriptions such as insert heading or convert an existing text block to a heading.

Example of Heading Plugin

Overview The Heading plugin is rendered as a menu button with a menu that provides the following items:

- Level n: allows the user to insert a new heading, convert an existing text block to a heading element, modify the level of an existing heading element or remove the heading format of an existing heading element.
    - When the cursor is positioned on an empty line, inserts a new heading element of the chosen level.
    - When the cursor is positioned on an existing text block that is not already a heading, converts it to a heading element of the chosen level.
    - When the cursor is positioned on an existing heading element:
        - If the menu item that corresponds to the current level of the heading (denoted by a check mark) is selected, removes the heading format;
        - Otherwise, changes the level of the heading element to the chosen level.
- Remove format: enabled only when the cursor is positioned on an existing heading element. Removes the heading format from the text block that corresponds to the cursor position.
- Help: provides help documentation on working with headings.

Accessibility Requirements

Heading levels should convey consistent structure in the document according to the following rules:
1. The level of the first heading should be the highest level permitted in the document.
2. If the highest level permitted is Level 1, there should typically be only one heading of that level.
3. Any subsequent heading should have either the same level (unless that would violate the multiple use restriction for Level 1), a higher level (again, avoiding the multiple use restriction for Level 1), or the next-lower level (without skipping any levels) in relation to the heading that precedes it.

User Interface Components

1. Menu Button
- Appearance
    - Label: Heading
    - To the right of the label: a down arrow to indicate that the button displays a menu when activated
- Behavior

When the menu button is activated it displays a menu

2. Menu

Menu items

- Level 1 or Level 1 (Reserved for Document Title) based on configuration
- Level 2
- Level 3
- Level 4
- Level 5
- Level 6
- Normal text

- -------
- Help (calls A11yFirst Help with param for headings)

Behavior

- When the menu button is activated, the menu it displays contains all possible heading levels (2 through 6) but only the available heading levels are enabled.
- Also, if the current context (based on cursor position or unambiguous selection) is an existing heading, the menu item corresponding to its level should be checked.
    - Example 1
    - There are currently no headings in the document. Menu displays only Level 2 enabled, i.e. it's the only available choice.
    - Example 2
    - The closest previous heading has level 2. Menu displays Level 2 and Level 3 enabled.
    - Example 3
    - The closest previous heading has level 3. Menu displays Level 2, Level 3 and Level 4 enabled.
    - Task Descriptions

Task 1 — Insert heading

User Actions
- Position cursor on a blank line within the document.
- Select Heading menu button: menu is displayed (see Menu — Behavior)
- Select desired heading level from menu
- Type heading text and optionally press return to end heading block and start a new paragraph.

Result : A new block of text formatted as a heading with the select level is created.

Task 2 — Convert an existing text block to a heading

User Actions
- Position cursor at the beginning or end of, or within, a block of text.
- Select Heading menu button: menu is displayed (see Menu — Behavior)
- Select desired heading level from menu

Result : The block of text defined by the cursor location is converted to a heading element of the selected level.

Task 3 — Remove heading format from existing heading

User Actions
- Position cursor at the beginning or end of, or within, a heading block.
- Select Heading menu button: menu is displayed (see Menu — Behavior)
- Select one of the following in the menu:
    - The checked Level n item, i.e., the item that corresponds to the level of the heading defined by the cursor position

- Remove format

Result : The block of text defined by the cursor location is converted from a heading element to a plain paragraph.
Task 4 — Get help on adding headings to a document
User Actions

- Select Heading menu button: menu is displayed
- Select Help from menu

Result : A dialog box is displayed that contains help documentation for the Heading feature.

To Do

- Generalize the algorithm for getAvailableHeadings to account for various configuration options.

Accessibility Feature Examples

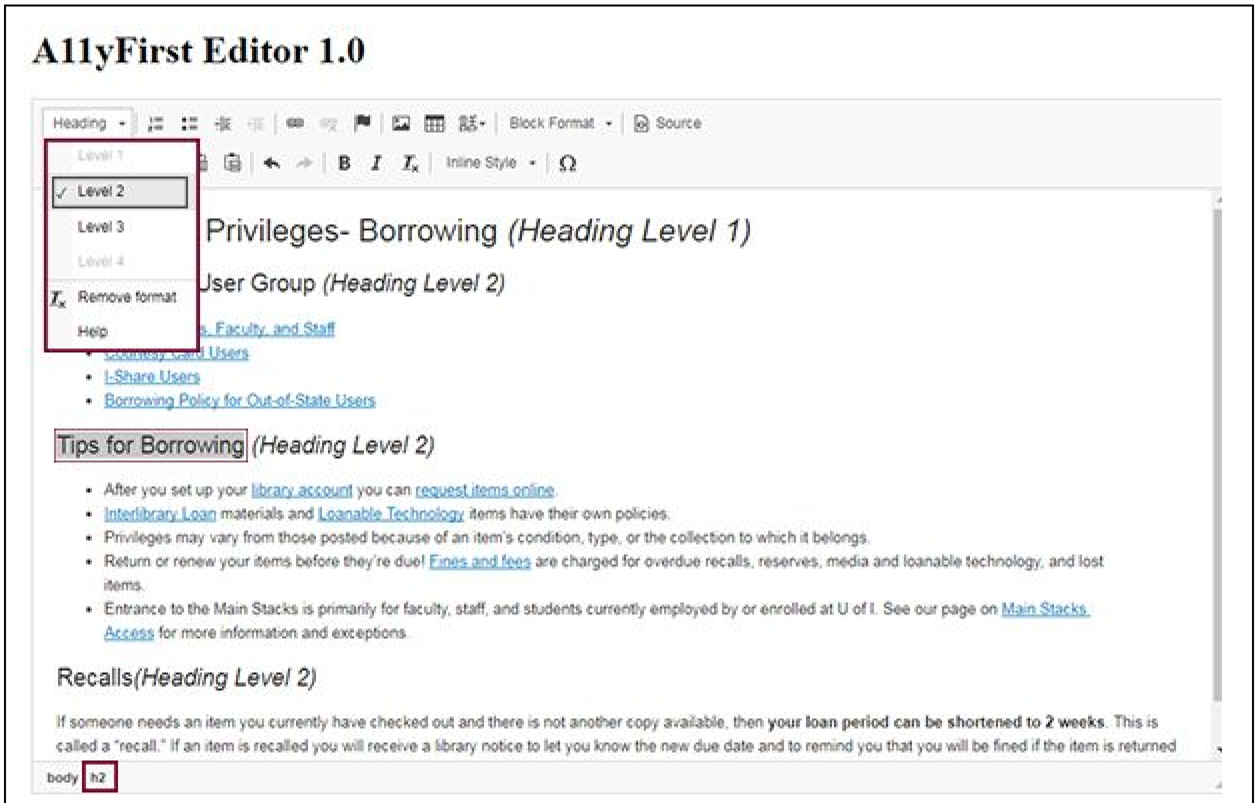
A11yFirst Heading plugin

1. Restricting Author's Choices for Heading Level

The first principle of Web accessibility is based on semantic structure of the documents. Well structured documents with appropriate heading levels help users with disabilities understand the content in an easy manner. To enforce good heading levels for a document, the A11yFirst editor programmatically restricts heading level selection. Accessibility requirements for heading levels are that Heading levels should convey consistent structure in the document according to the following rules:

1) The level of the first heading should be the highest level permitted in the document.
2) If the highest level permitted is Level 1, there should typically be only one heading of that level.
3) Any subsequent heading should have either the same level (unless that would violate the multiple use restriction for Level 1), a higher level (again, avoiding the multiple use restriction for Level 1), or the next-lower level (without skipping any levels) in relation to the heading that precedes it.

[Figure] Heading Restriction for Structured Document



A11yFirst Editor 1.0

Privileges- Borrowing *(Heading Level 1)*

User Group *(Heading Level 2)*

- s, Faculty, and Staff
- Courtesy Card Users
- I-Share Users
- Borrowing Policy for Out-of-State Users

Tips for Borrowing *(Heading Level 2)*

- After you set up your library account you can request items online.
- Interlibrary Loan materials and Loanable Technology items have their own policies.
- Privileges may vary from those posted because of an item's condition, type, or the collection to which it belongs.
- Return or renew your items before they're due! Fines and fees are charged for overdue recalls, reserves, media and loanable technology, and lost items.
- Entrance to the Main Stacks is primarily for faculty, staff, and students currently employed by or enrolled at U of I. See our page on Main Stacks Access for more information and exceptions.

Recalls*(Heading Level 2)*

If someone needs an item you currently have checked out and there is not another copy available, then **your loan period can be shortened to 2 weeks**. This is called a "recall." If an item is recalled you will receive a library notice to let you know the new due date and to remind you that you will be fined if the item is returned

[Caption: This image demonstrates that author has choice of only heading level two or three because selected heading level is level 2. Heading level 2 with checkmark indicates the current heading level.See also disabled(grayed out) heading level 1and level 4 to prevent skipping the heading level.]

2. Heading Menu Features

- The Heading menu only enables the **allowed** heading levels, based on the position of the cursor in the document, to support proper nesting of headings.
- If the cursor is in a block of text that is not already a heading, selecting one of the enabled heading levels in the menu will convert the block to a heading of that level.
- If the cursor is on a heading, the menu item with a checkmark indicates its **current level**. Additional menu items that correspond to **any allowed changes** to the current level are enabled.
- When the cursor is on a heading, selecting the **Remove format** menu item, or selecting the **current level** menu item, converts the heading block to a plain text paragraph.

3. Using Heading Levels

1) Heading levels identify the structural relationships between sections of content in a document.
2) Higher-level headings (Levels 1 and 2) identify the main topics of a document and lower-level headings (Levels 3, 4, 5 and 6) identify subsections of the document.
3) A subsection is identified by using the next lower-level heading. For example, subsections of Level 2 headings use Level 3 headings, subsections of Level 3 headings use Level 4 headings, and so on to Level 6 headings.

4) Break content into subsections when there are two or more ideas or concepts that correspond to the topics covered in the section. Use headings of the same level to label each subsection.
5) Heading levels should **never** be used for inline visual styling of content (e.g. larger or smaller font size, bold or italic). Instead, use the "Inline Style" options.

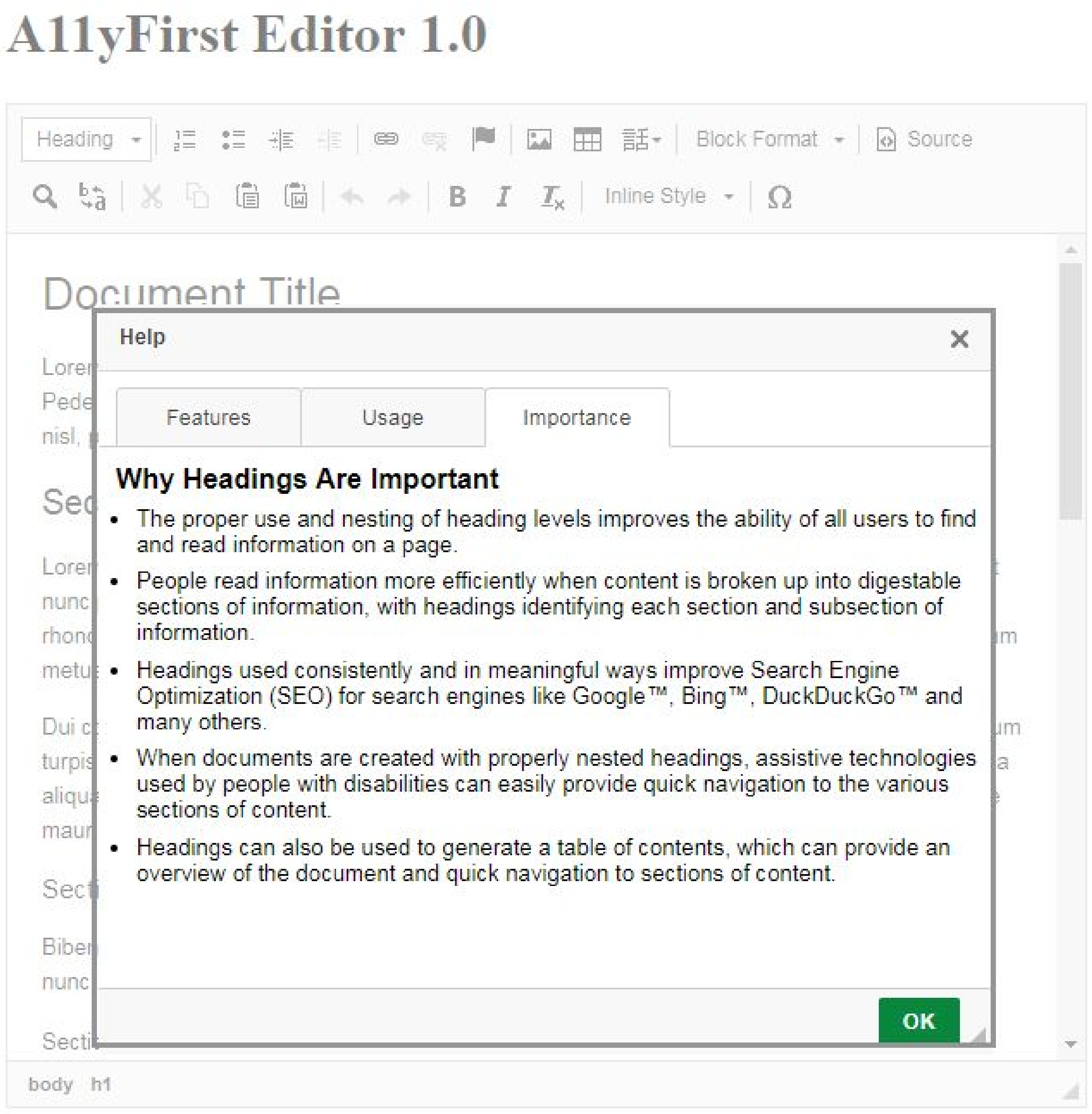
3. Focus on Accessibility Education/Guidance

One of the goals of A11yFirst Editor is educating authors and users about the text editor so that they can create accessible content both for visual readers and non visual readers. Therefore, the team added accessibility information such as the importance of good heading structure and its usage as two of the menu items. For example, the team tried to convey “Why Headings Are Important” as one option. Here is heading plugin “help” information.

Why Headings Are Important

- The proper use and nesting of heading levels improves the ability of all users to find and read information on a page.
- People read information more efficiently when content is broken up into digestible sections of information, with headings identifying each section and subsection of information.
- Headings used consistently and in meaningful ways improve Search Engine Optimization (SEO) for search engines like Google™, Bing™, DuckDuckGo™ and many others.
- When documents are created with properly nested headings, assistive technologies used by people with disabilities can easily provide quick navigation to the various sections of content.
- Headings can also be used to generate a table of contents, which can provide an overview of the document and quick navigation to sections of content.

One of the problems we found from usability testing is that authors did not notice this “help” menu under “heading” menu. The team needs to enhance the user interface to make this section more discoverable.



Why Headings Are Important

- The proper use and nesting of heading levels improves the ability of all users to find and read information on a page.
- People read information more efficiently when content is broken up into digestable sections of information, with headings identifying each section and subsection of information.
- Headings used consistently and in meaningful ways improve Search Engine Optimization (SEO) for search engines like Google™, Bing™, DuckDuckGo™ and many others.
- When documents are created with properly nested headings, assistive technologies used by people with disabilities can easily provide quick navigation to the various sections of content.
- Headings can also be used to generate a table of contents, which can provide an overview of the document and quick navigation to sections of content.

A11yFirst Link Plugin

1. Core Idea

Context based link text to website URLs or to document objects are more useful. Authors should avoid non informative link phrases such as: Click Here, Here, More, Read more, Link to [some link destination], Info. Also using URLs as links can present two types of challenges, readability and length.[4]

By default, CKEditor allows the user to create a link with an *empty Display Text field*. When this occurs, CKEditor uses the *URL specified for the link as the Display Text*.

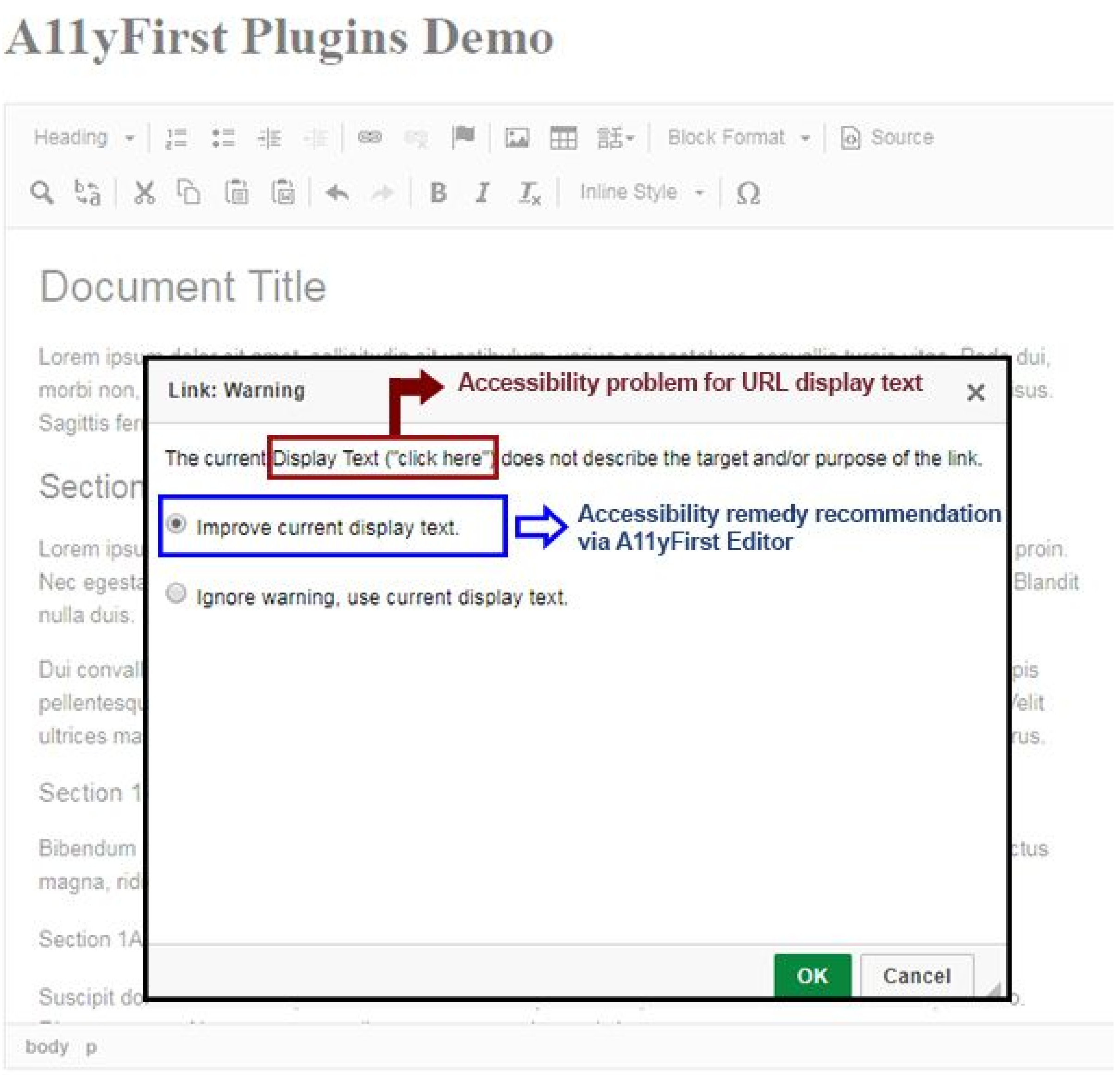
The modification we are proposing in A11yFirst link plugin is to check for an empty Display Text field in the Link dialog, and when detected, warn the user with a Warning dialog that contains the following:

- Title: Link: Warning
- Message: The Display Text input field for the link is empty.
- Radio button (1 of 2): Add text to input field
- Radio button (2 of 2): Use URL as display text
- OK button

2. User Flow and Warning Dialog Functionality

[Step 1] An author put a url with the display text, “click here.” The dialog box is brought up to inform the author that he/she needs to improve URL display text for web accessibility.

[4]Links and Hypertext Appearance, http://webaim.org/techniques/hypertext/link_text



The current Display Text ("click here") does not describe the target and/or purpose of the link.

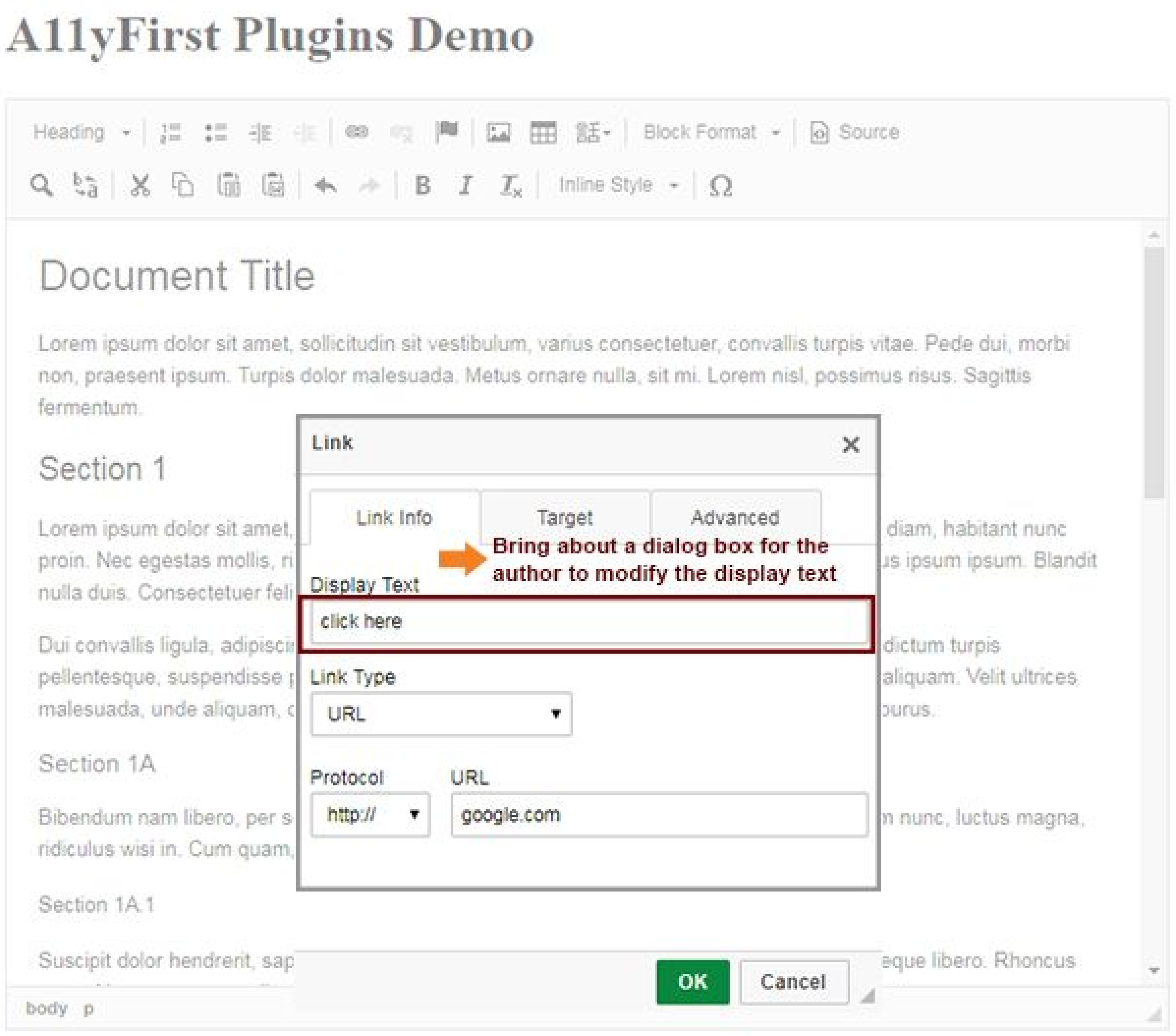
(●) Improve current display text.

( ) Ignore warning, use current display text.

Accessibility remedy recommendation via A11yFirst Editor

[Source] https://cdn.rawgit.com/a11yfirst/plugins-dev/feature/link/custom/index.html

[Step 2] An author chose to improve the display text, “click here.” Another dialog box is brought up to change existing value of display text, “click here” and author can change the display text to improve accessibility for who using the screen reader.



Display Text: click here

Link Type: URL

Protocol: http://

URL: google.com

[Source] https://cdn.rawgit.com/a11yfirst/plugins-dev/feature/link/custom/index.html

Milestones

Strategic direction

- Using CKeditor checker functionality from CKsource
- Submit heading plugin as add-on to CKeditor – minimum viable product
- Links – you have headings as anchor link target

- Use existing CKeditor checker for now to ensure accessibility (categories of checking [filter by object such as heading or images] vs CKEditor checker is like a laundry list)
- Enhance existing CKeditor UI

A11yFirst editor 1.0 - Completed

- Heading + code management
- Inline style
- Block format (including block quotation)
- Toolbar configuration (documentation re a11y)
- Project Milestones 1.0

A11yFirst editor 1.1 - Completed

- Dialog: empty link text with summary/detail help
- Dialog: non-descriptive link text with summary/detail help
- Help tab in link dialog
- A11yfirst editor links include warning

A11yFirst editor 2.0 - In process

- A11yfirst image plugin specification:
    - Revision of CKeditor image plugin to create a11yfirst image plugin
    - Integration of a11yfirst image plugin to WordPress
    - Project milestones 1.1 and 1.2
- A11yfirst tables plugin specification
- Enhance heading table of content, list of headings, revisiting the link dialog process, adding context for links (ie. adding title)

A11yFirst editor 2.X - Upcoming

- “Paste from Word”: how this works with accessible authoring (paste from pdf, google doc)

Library WordPress CMS Deployment

Schedule for Staged Deployment

A Library WordPress test serveron which A11yFirst editor will be deployed was set up. The schedule for deployment was discussed. Currently, production deployment was scheduled on after the fall semester 2017 and during winter break to minimize disruption and to start as a clean slate. The team will take the approach of staged deployment, starting the deployment from the A11yFirst WordPress server, then to Library WordPress CMS development server, then to WordPress stage server and finally the WordPress production server.

Requirement by Library Wordpress CMS

Must Have Features

- Reduce list of special characters
  - Remove letters and numbers when English is the default language
- Add paragraph justify toolbar buttons (add after block format button)
  - Left
  - Right
  - Center
- Block Format Button
  - Button Labeling "Paragraph Format"
  - Options
    - Blockquote
    - Preformatted (<pre> tag)
    - Add "Normal text"
- Heading Menu
  - Issues: People don't understand why H1 is disabled
  - Changes: "Remove Format" -> "Normal Text"
- Image ALT text checking
  - Check document on insert for ALT text
  - This is a plugin that only allows editing of ALT text
- Accessibility Checker

Nice To Have Features

- Language change button
- Link Checker
  - Use of a URL as display test
  - Use of poor display test (e.g. click here)
- List style options
  - No-bullet
  - Extra line spacing
  - Highlight (colors and border)
- Paragraph
  - Highlight (colors border options)

Usability Testing

Introduction

Traditional usability testing focuses on end users and a waterfall-compatible method which offers the same interface to at least five users per audience at the same time. However, this project had five additional usability-related challenges to face:

Creator / author experience, not “user” experience: The A11yFirst project focus was on contributing-level content creator experience rather than “traditional” viewer-level user experience. Creator-specific experience is often given short shrift in the spectrum of usability, accessibility, and quality assessment tools and methodology, because there will always be more system viewer-type users than creator-type users. There is comparatively less guidance in this field, so we’re assessing a less-studied area.

The Innovation Fund’s support for A11yFirst enabled attention to be paid to this critical and often underserved group.

Struggling creators don’t create (or maintain) content well: In his book *Author Experience*, Rick Yagodich observes, “A difficult-to-use environment encourages [end users] to go elsewhere to fulfill their needs. If they have no choice but to use our poorly implemented system, they will become frustrated, leading to a combination of sloppy use and negative reputation…. Authors are users too; the same psychological consequences apply to them. ...We cannot afford to use a platform that frustrates authors and keeps them from developing high-quality content.” (p. 19)

This is sometimes referred to as “mental friction.” The harder it is to use a system and the more expectation-breaking hiccups a user experiences, the less frequently the system content will be updated.

It’s hard enough to have creators keep content up to date and consistent in a friction-free system that complies seamlessly with creator’s expectations. However, part of the A11yFirst mandate is to change both expectations and behavior in a way that produces an improved accessibility outcome. Changing long-established interaction behavior without creating struggle-and-resistance-inducing levels of mental friction is a fine needle to thread.

Everyone comes in to an editor interface with their own (differing) habits: That hypothetical “friction-free but behaviorally-changed and more-accessible system” is even more challenging to find in a world where Google and Microsoft have set user expectations for 20 years.

If A11yFirst had been inventing a brand new style of interface that no one had seen before, people would have learned it independently of their background experience and would approach it with fewer ingrained behaviors. However, nearly every content creation system offers a WYSIWYG editor interface -- and most of them have evolved to have a great deal of similarity in behavior, shortcuts, and organization over the years.

WYSIWYG editors also offer several ways to do any one thing. In our first six usability assessments, every single user displayed a new interaction method that none of the others had shown. So there are a lot of interaction methods to (carefully) adjust.

Not all the interface elements could be adapted: Because the A11yFirst project is built upon the existing CKEditor interface, the team was working within constraints imposed by the existing system in the interest of improving an editor with a substantial number of installations. Some adaptations could be made locally; others would need to be transferred to the CKEditor project itself for consideration.

Agile development means the interface changes quickly and responsively: When performing observational usability assessments, the traditional gold standard promoted by Nielsen et al is to watch five users interacting with the same interface. However, agile development cycles modify the interfaces so quickly that it’s not possible to follow this guidance. We’ve modified the procedure to refine our tests in sync with agile interface development and added a time dimension to our observation tracking so that we can discover whether newer developments have improved user experience with trouble spots.

Design

Dena Strong, a senior information design specialist with Technology Services, works with the Research Data Service four hours a week to foster cross-campus collaboration and communication and to help Technology Services support the RDS mission. With the approval of Dr. Heidi Imker of the Research Data Service and Janet Jones of Technology Services, some of her RDS time was offered to the A11yFirst project for usability assessment design, execution, and analysis.

During the development of the Illinois Data Bank, she created an agile-adapted version of usability assessments for the dataset contribution and API interfaces, and this agile model of usability assessment was used for the A11yFirst project as well.

Several methods of user experience testing were employed during the evaluations:

- **Heuristic analysis:** The interface was reviewed against behavioral standards for internal consistency, and issues identified were divided by the team into A11yFirst-specific or CKEditor-specific issues.

  (For example: Many embedded elements like images and links can be double-clicked to launch a more detailed properties-editing view. With that behavioral standard established by the interface, the lack of a double-click editing option for tables was noted at heuristic analysis, and reinforced by testers who encountered the same issue.)

- **Comparative analysis**: WYSIWYG editors are common enough to have acquired their own acronym. In order to reduce a user’s experience of mental friction, designers need to identify not only formally-created design standards but informally-established design “patterns” of the way people will expect an interface to behave -- things like iconography, menu language and contents, and click patterns.

  (For example: Faced with design decisions to make about interface labels or format removal behavior, a side by side comparison of the way Microsoft, Google, Box, and several other major providers handle the same issue can identify where people will arrive with strong expectations of standard behavior that will be hard to modify, or where different providers have made different decisions and users will have fewer pattern expectations.)

- **Video-mediated usability assessments**: Watching how users interact with the system, noting points of confusion or hesitation, and asking for their thoughts on how the system behaves is a classic usability study method. We designed three phases to the test, with different specific tasks in each phase: Exploration of the interface; content creation involving a simplified page structure with headlines, bullets and links; and recreation of an existing real page in the Library’s site.

  The simple test page’s structure helped to compare different user’s behavior with the same content and to focus user’s attention on the interface elements that had received accessibility-specific modification. The real page recreation test helped us to identify an assortment of real-world scenarios that the simplified test page wouldn't cover, such as complex layouts and content types.

(For example: Some of our test users had picked up keyboard shortcuts offered by the Library WordPress editor that weren't available in the A11yFirst demonstration system, and some of those shortcuts were used at the level of reflex. When asked to recreate what he'd just done, one tester couldn't consciously identify what he was trying to do in the test editor until he launched a Library WordPress window and established that the keyboard interaction he'd attempted did work there. An interview method without observing his hands on the keyboard wouldn't have captured that pattern.)

Process

Before the video-mediated assessments began, the team discussed which elements were critical to assess, which hadn't yet been modified for accessibility improvements, and what the team hoped to accomplish with both obvious and subtle accessibility improvements in the interface.

Six Library staff members with different levels of familiarity with the current WordPress editor interface agreed to perform usability assessments of the A11yFirst editor interface at its current state of development. We performed one assessment per week between July 5 and August 9, occasionally modifying the "sample text" phase of the assessment to adjust user's attention to the area of the interface that was of particular interest to the developers.

During each assessment, our volunteers provided background information about themselves and their familiarity with content editing systems, walked through the established tasks with Strong's guidance as a facilitator, and provided an exit interview to gather their final thoughts and suggestions for improvement. In order to allow the developers and other usability-interested people to observe the tests without crowding the room, a Skype for Business connection was established to allow for quiet and unobtrusive observation.

At the end of each test, the video recordings were processed and stored in Box for future reference and discussion, along with the observer's' notes. Specific observations were captured in an Excel spreadsheet to explore what percentage of users encountered particular confusion points, wished for certain behaviors, or expressed certain expectations.

Findings

As a result of the usability testing, we found places where icons didn't make sense, phrasing confused people, and interactions were inconsistent and confusing, giving us the ability to address them in sprints before initial release. We also found places where people sometimes chose to ignore the accessibility suggestions and guidelines provided by the editor. Usability testing gives us the ability to adjust our approach, our explanations, and our implementation in order to sometimes encourage and sometimes enforce greater compliance.

Future Usability Testing Plans

As the project's development work continues with other areas of interface design, we will refine the usability testing to focus on specific elements, allowing the usability testing to reveal unanticipated shortcomings of the A11yFirst editor.

Ongoing Evolution

Project Resources

The A11yFirst Project funding for a student employee will be provided by Disability Resources & Educational Services for the remainder of the year. Eventually, A11yFirst project will be transformed into an open source project in Github source repository.

Continuous Improvement

Continuous improvement is "the ongoing improvement of products, services or processes through incremental improvements". Among the most widely used tools for continuous improvement is a four-step quality model — the plan-do-check-act (PDCA) cycle, also known as the Deming Cycle or Shewhart Cycle. As the project team approaches next project milestone, more usability testing will be undertaken.

Collaboration with the School of Information Science

The project team will work with the School of Information Science to further user research on the A11yFirst Editor's conceptual model. The project team has already recruited graduate students to work on user research for the A11yFirst editor under the supervision of Dr. Twidale, a professor at the School of Information Science.

Collaboration with Other Campus Units

As the project team make progress on the A11yFirst editor, the project team are also planning to work with various campus units so that they can adopt the A11yFirst editor into their content management systems, including both WordPress and Drupal content management systems. The first target is the publish.illinois.edu services, which uses WordPress to provide a campus wide web service.

Collaboration with CKsource

The A11yFirst editor is based on API code from CKsource. The Project team has been gathering valuable research and usability testing data to submit as feedback to CKSource. We are hopeful CKsource may integrate this project deliverables into the CKSource core, ensuring accessible content creation via CKEditor. In this way, the A11yFirst Editor would be able to improve accessibility beyond the Library's WordPress CMS and achieve a global impact.

Project Websites

A11yFirst Editor Demo

http://a11yfirst.library.illinois.edu/ (Includes demo of text editor, code repo and brief project info)

A11yFirst Project Documentation

https://publish.illinois.edu/a11yfirst/ (includes project documentation, meeting minutes, design specification)

Original Proposal

From Remediation to Proliferation - Mainstreaming the accessibility of Web Resources

Project Expense

As of August 2017, total remaining project funds were $6092.69. $1881.68 was spent on student employees and academic hourly employees for the last payroll time period. $3832.17 was allotted to facilitate the project's presentation in Westminster, Colorado by three presenters. $365.36 was used to prepare project marketing materials distributed at the conference. The above figures are preliminary until all expenses are processed and posted to University Banner system, according to the University's Library Business office.

The majority of project expense was payroll for five student employees and three academic hourly employees, support for project outreach and marketing, conference attendance and presentation. Other miscellaneous expenses included gift cards for 36 usability testing sessions and a dinner meeting with Matt King of Facebook, the renowned accessibility guru, of keynote speaker at the University of Illinois Web Conference 2017.

Credits

First, I would like to thank the member of the University Library Executive Committee who made this project possible. This project established a strong foundation for innovating accessible content authoring practice on the web. The University Library Business and Human Resources Office also helped the project along the way, processing all the expenses and student hires.

I would like to give special thanks to Robert Slater(Technical Architect for Web Content, University Library IT), Mike Scott, (Director, Dept of Human Services, State of Illinois), Tim King, (Graphic Designer), Candice Woodrum and Susan Edwards (University Library Business office), Aneitre Johnson (University Library Human Resources office) and Kim Matherly(Administrative Assistant to the University Librarian and Dean of Libraries).

Appendix

Presentation

Jaeun Ku, *Why Conceptual Modeling? A Case Study of A11Y First Text Editor*
Annual Library Research Showcase, 2016
https://www.ideals.illinois.edu/handle/2142/95061

Workshop

Jon Gunderson, Open source Accessibility Tools, HighEdWeb 2017 Hartford, CT
*Dr. Gunderson will talk about A11yFirst Editor as one of open source accessibility tools at the conference

Accepted Conference Proposal

Conference : Accessing Higher Ground By Association on Higher Education and Disability(AHEAD), Westminster, Colorado 2017

Title: A11yFirst for CKEditor: Support Creation of Accessible Web Documents through User Interface Enhancements

Speakers: Jon Gunderson, JaEun Jemma Ku, and Dena Strong

Summary (350 characters max, including spaces)
CKeditor is a popular WYSIWYG editor using in web based Content Management Systems (CMS) like Drupal, Moodle, Blackboard and Desire2Learn. The A11yFirst project has redesigned and enhanced the user interface controls for use in CKEditor to support accessible authoring and document creation.

Abstract (1000 characters max, including spaces)
Most web content is made by people using web based WYSIWYG editors Embedded in content management systems like blogs, learning managements systems, and administrative websites. These users have little understanding of accessibility or the technical details of HTML accessibility. Embedded WYSIWYG editors like CKEditor often include accessibility checkers as an optional feature, but this requires an intentional action on the part of author to use them. The checker approach reinforces the remediation stereotype for accessibility, which is also extra work for the author. In contrast, the a11yFirst project changes the user interface features to guide authors in creating accessible content as they create documents. A11yFirst supports the creation of structured documents and providing just-in-time information on accessibility. The A11yFirst plug-ins are open source and can be used to upgrade current installations of CKEditor to improve accessible authoring.
Key Points

1. Accessible authoring versus accessibility remediation
2. User interface features that support and encourage accessible authoring

3. Support features to help authors understand accessibility

Submitted Conference Proposal

Conference : University of Illinois WebCon 2018
http://webcon.illinois.edu/

1. Presentation Title
Designing for Accessibility: Changing Users' Mental Models

2. Presentation Description (100-150 words)
This presentation is about design methodologies and how they interact with and impact development processes, and how design must sometimes compete with mental models that users may already be familiar with.

We will offer insights into how we dealt with the tension between usability testing focused on an agile development process, where user feedback may suggest specific design alterations, and a more goal-oriented design process that began with the creation of a conceptual model, and was based on interaction design principles and previous accessible text editor from State of Illinois.

As a case study, we will use the example of the A11yFirst project, which aims to support the creation of structured, accessible documents by web content authors who are using an embedded WYSIWYG editor within a content management system.

3. Presenters
JaEun Jemma Ku, Nicholas Hoyt, and Dena Strong