Conducting accessibility testing on complex widgets

AccessU

Introduction

I'd like to acknowledge the **Traditional Owners of the lands** on which we meet today.

In my case, this is the **Cammeraygal people** of the Guringai tribe of the Eora nation.

I'd like to pay my respects to **Elders past and present**, and extend that respect to all First Nations people present today.

Don't be shy!

Feel free to **add comments or questions into chat** at any time!

Maybe start now by adding your **name and current role** in chat?

What is my role?

I am not an **accessibility auditing specialist**!

If I don't audit websites, what the hell **have I been doing** with my time?

Since 2003, I have focussed on **two areas of accessibility**.

1. Conducting user testing sessions and interviews

Over the years, a colleague and I conducted user testing sessions with a **wide range of people with disabilities**.

- Decide what we wanted to test
- Decide on the type and number of participants
- Write recruitment brief
- Find participants
- Create a script
- Arrange meeting times and locations
- Conduct user testing session and post-testing interview with each participant
- Compile results
- Publish results

We found that our testing was most effective when we went to people's homes or places of work and **watched them in action**.

This allowed us to observe people **in their natural environment**.

The sessions taught us a lot about **how people engaged with websites**, and their **barriers and frustations**.

2. Building and testing accessible UI components

Over the years, I've worked with many organisations to **help build and test a wide range of UI components**.

The term **"components"** means any self-contained UI concept.

I mainly focussed on components that involved **user interaction**.

This could mean simple components such as **form controls and their associated labels**.

Or complex components such as **sortable tables, dropdown-menus, date pickers or accordions**.

A testing framework

For this presentation, I'm going to focus on the **testing aspect** of my work with UI components.

We will specifically focus on **"accessibility unit testing"**.

I found that by setting up a rigorous testing process, I was able to **confidently define the accessibility of each component**.

As you will see, the testing process is **focussed around different types of users** and how they could interact with any component.

Over time, I began presenting this framework to teams **before new components were designed or developed**.

And this allowed teams to think about different types of users **during the design and development process**.

So, this testing framework can also be adapted to be used as a **design and development framework for UI components**.

What is accessibility unit testing?

Accessibility unit testing is a process where the tiniest parts of a component are **individually and independently evaluated**.

The aim is to determine if all aspects of the component **meet the needs of people with disabilities**.

Accessibility unit testing is typically conducted when **reviewing an existing component** within the Design System.

Or when **reviewing a new component** to determine if it can be added to the Design System.

So, how can you **systematically test components** to ensure they are accessible?

You can create **a series of tests** based on two concepts:

1. How the overall component or aspects are **presented to users** - either on screen or via assistive technologies.

2. How users should be able to **interact with a component** to complete a task.

These tests must include **all possible happy and unhappy paths** associated with the component.

A **happy path** is the error-free path users take to complete a task.

An **unhappy path** is where users may encounter errors or issues that stop them from completing the task.

Let's use an example of a **date picker**.

A **high-level task** might be something like:

The user can successfully add a date using the date picker.

But **numerous small interactions** associated with this overall task should be tested.

And there are **different methods that people use** to achieve these interactions - mouse, keyboard, assistive technologies etc.

Each of these needs to be **defined as a test**.

For example, what tests would be needed to determine if the date picker overlay **could be closed via keystrokes only**?

The `ESC` key can be used to close the date picker overlay: PASS/FAIL

When the `ESC` key is triggered, focus returns to the date field: PASS/FAIL

When the user has chosen a value before triggering the `ESC` key, and focus returns to the date field with the newly injected value, this value announced: PASS/FAIL

An unhappy path could be where the date-picker question is required, but the user **does not enter a date value** for the field.

What tests would be needed to **determine if users could identify the field as required**, before they decide whether to enter data or not?

A visual means of identifying that the field is required is present: PASS/FAIL

Screen reader users can programmatically determine that the field is required: PASS/FAIL

And what tests would be needed to **determine if the field is accessible** once it is in a state of error?

The form field uses methods other than colour alone to visually identify that it is in a state of error: PASS/FAIL

When the field is in a state of error, an error message presented on screen: PASS/FAIL

When the field is in a state of error, the error message provides a precise description of the exact problem: PASS/FAIL

When the field is in a state of error, the error message provides constructive advice on how to fix the problem: PASS/FAIL

When the field is in a state of error, the error message is in close visual proximity to the form field: PASS/FAIL

When the field is in a state of error, the error message is programmatically associated with the field: PASS/FAIL

As you can see, every test must be written as a statement where the only outcome is either a **pass or fail**.

Each test must be **repeatable** so that it can be re-tested at a later date.

These tests may need to be performed in **complex combinations** of different assistive technologies, browsers and operating systems.

Each of these different test combinations must be **documented separately**.

The end result should be a document that **defines the following**:

- The **test**.
- The **environment** in which the task was performed.
- The **result of the test** within the environment.

This approach means that **anyone in the team** should be able to:

- **Review** the results.

- Determine which aspects of the component are **accessible and inaccessible**.
- **Repeat** the task in the future.

Let's look at an example of an **autocomplete component** from the last design system I worked on.

Starting with people and their needs

How could you create **a complex series of tests** for any component?

You can start by looking at **different groups of people and their needs**.

This means you can focus on one user type and explore **how they may need to interact with the component**.

These **groups of people** could include:

1. People with some sort of **cognitive impairment** or **neurodivergence**.
2. People with some sort of **colour vision deficiency**.
3. People with **low vision**.
4. People with **limited mobility**.
5. People with **limited or no sight**.

For each group, there are some **high levels questions** that can be asked for any component to help determine possible tests.

1. People with some sort of cognitive impairment or neurodivergence

1. Is the purpose of the component clear?
2. Is the component easy to use?
3. If additional instructions are required, are they clear?
4. If users make a mistake, can they quickly and easily recover?
5. Is there any complex language used associated with the component?
6. Are there any animations or movements that could distract the user?

2. People with some sort of colour vision deficiency

Does the component:

1. Have sufficient colour contrast?
2. Use alternative methods to display colour information?

3. People with low vision

In some cases, these people may need to **customise the user interface** or **magnify the display** in some way.

Is the component operable/understandable:

1. When the text is scaled to 400%?
2. When the overall layout is scaled to 400%?
3. When aspects of the component are magnified?
4. When displayed in low contrast?

5. When displayed in different viewport sizes?

4. People with limited mobility

In some cases, these people may rely on **keyboard interactions** or **voice recognition software**.

1. Does the component need to receive focus?
2. Is focus managed into, within and out of the component?
3. Does focus order follow a meaningful sequence?
4. Can all actions be executed using keystrokes only?
5. Can all actions be executed using voice controls?
6. Are keystrokes intuitive for keyboard-only users?
7. If focus is relevant, are all visible focus states clearly defined?

5. People with limited or no sight

These people may rely on **screen reader software**.

1. Does the component have an accessible name?
2. Does the accessible name clearly define the component's purpose?
3. Does the component have a role?
4. Are all of the components possible states defined?
5. If a value can be added within the component, is it exposed?
6. If the component has any dynamically added content, is this announced at the appropriate time?

Creating tests from each question

The next step is to break down each question into **a series of individual tests**.

In some cases, the question **can become the test**. For example:

Question:

Does the component have an accessible name?

Test:

The component provides an accessible name: PASS/FAIL

In other cases, you may need **multiple tests associated with a question**. For example:

Question:

Can all actions be executed using keystrokes only?

This question may need a **wide range of individual tests**, depending on the complexity of the component.

Let's explore a **modal component** as an example.

A modal component

Let's just focus on people with **limited mobility** and how they could interact with a modal that has been triggered.

Add your answers in chat

What tests would you include when the modal has been opened?

When the modal window is triggered, the modal window receives focus: PASS/FAIL

When the modal window is triggered, the first focusable element inside the modal receives focus: PASS/FAIL

When the modal window is open, `TAB` and `SHIFT` + `TAB` keystrokes are contained within the modal: PASS/FAIL

When the modal window is open, all actions within the modal be executed using keystrokes only: PASS/FAIL

When the modal window is open, focus order follows a meaningful sequence: PASS/FAIL

All visible focus states within the modal are clearly defined: PASS/FAIL

When the modal window is open, the `Esc` keystroke can be used to close the modal: PASS/FAIL

What tests are needed **when the modal has been closed**?

When the modal is closed, focus returns to a meaningful and intuitive location within the page below: PASS/FAIL

After the tasks are finished?

We need to determine if each test can be **conducted using a single browser**, or if multiple browsers are required.

We also need to determine if any **assistive technologies required** for any tests, and if these need to be conducted across multiple browser options.

We can then **create a testing spreadsheet** and begin testing!

Circling back to design and development

As mentioned earlier, two things can help with the **design and development process for new components**:

1. Considering how **different types of people** may use the component.

2. Applying the **same testing rigour** during the design and development phases.

Where possible, **try to include real users** as part of the design and development process.

Questions/discussion?