Table of Contents

Introduction

1 Research goal and methodology

1.1 Research goal

Accessibility is important, but not easy to maintain. Implementing truly accessible solutions demands good cooperation between designers and developers. In software development code quality and best practices are checked with linters and tests. This helps maintain a high quality and when set up right demands a low effort on a daily basis. Accessibility consists of different parts and a part of it is the html that the end user interacts with. This is quite testable.

I would like to see if automated accessibility test can potentially help improve the overall accessibility in out component library. Most testing tools also provide links and instructions about the issue that was detected. Can this help improve the developer knowledge about accessibility?

Many companies have design systems to help maintain consistency both in what the end users experience and in the code. This seems like a good place to start with establishing a basic accessibility standard. Most accessibility checkers are intended to be used for a website. How effectively can components be tested? What limitations might there be?

There have been various studies that have compared different accessibility testing tools (**TODO: Find refs) and studies that have tried to find reliable ways to measure websites conformance with accessibility standards. I will take a look at enough tools to find one that would be usable for my purposes, but I am not intending to compare them as a part of this work.

I intend to test out and automated accessibility tool in my workplace on our React component library - Convention UI React (CUI for short) **TODO: should I use this?. This will probably pose some limitations on the tool I can use.

1.2 Research questions

RQ1: How good is the knowledge about accessibility standards, tools and best practices in the company currently?

RQ2: What kind of errors can be caught by running automated accessibility tests on a component library?

RQ3: To what extent can integrate automated testing to a component library's development pipeline help improve its compliance with WCAG standards?

RQ4: What are the biggest problems of integrating automated accessibility testing to a component library development workflow?

1.3 Methodology

2 Background

2.1 Web accessibility

What is it, and why should we care about it?

2.1.1 What standards should be followed?

2.2 Automated testing

Continuous integration or CI for short is a software development practice where members of the team integrate that work frequently and each integration is verified with and automated build that includes test to detect and errors as quickly as possible. This is believed to help develop high quality software more rapidly. One of the practices in CI is making your code Self-Testing - adding a suite of automated tests that can check a large part of the code base for bugs. These tests need to be easy to trigger and indicate of any failures. (Fowler, 2006)

This last principle could be applied to accessibility. It goes well modern software development principles. Test will not catch everything, but they will catch enough to make it worthwhile. After the initial setup they should not need a lot of maintenance and will be run every time someone contributes to the codebase. This can act as a very effective gatekeeper for any potential accessibility issues.

2.2.1 How does it work?

Web content is essentially code and any code can be analyzed and compared against rule sets. Accessibility testing can be done by experts, real users or by code. The last one is what we call automated testing.

It can be set up in different way - as a browser plugin that check the site you are currently on against a rule set or a test written to ensure that a piece of code does not violate a certain rule or unit test that will ensure that the sites' compliance with a set of rules is consistent.

These kinds of test can detect up to 60% of violations. **TODO: Find citation There is a limit to what can be detected. More testing will always be required to ensure that the content is fully accessible. These kinds of tests can be set up to run automatically, and they provide measurable results. This means it a good way to monitor compliance with WCAG rules consistently without any extra effort. This can help avoid unwanted changes and show easy fixes in your code.

WCAG 2.0 was made in a way that works better for automated testing. **TODO: Find citation

2.2.2 Types of tools

(Sane, 2021) (Abou-Zahra et al., 2017) (Abou-Zahra & Henry, 2020) (Alsaeedi, 2020) (WebAIM, 2019) What are the most common methods for testing?

- Testing with users
- Testing using experts
- Testing by using checkers that run in a browser

2.2.3 Automated accessibility testing.

- How can accessibility testing be automated?
- In software development CI/CD (Continuous integration / Continuous delivery) is commonly used. How can we use the same principle for automated testing. Look et this ()
- What are common strategies used by other companies?

2.2.4 Limitations of automated tests

The automated accessibility checkers that are available now will not catch all the errors. They work on certain types of issues very well and not so well on others. The use of AI in could make these tools even more effective, **TODO: Citation but until then they should always be combined with testing with users or experts to ensure that the products that you are developing is truly accessible.

3 Case study

Steps

1. Set up automates accessibility issue detection in storybook adding a11y-addon + find a way to generate report of all issues.
   Data gathered from automated testing report:
   - How many occurrences in the accessibility violations report. This will show how many violations where detected from all the examples. Might contain the same issue multiple times.
   - How many unique issues will only count different violations for each component.
   - How many passed checks – this together with violations will show how many things where tested for each component – Most component have more than one story – the list will contain all different passed checks listed
   - How many valid checks – are the passed checks relevant to the component – only count the ones that are related to the component that the example is about.
2. Manual accessibility audit with other team members. We already had the add-on set up, and we used storybook preview of components for testing, so we looked at the violations reported by the addon-a11y also.
3. Comparison between manual and automated report
4. Research other possibilities. Testing new storybook test-runner to automatically run all a11y tests. How can we ignore some issues without losing the info.
5. Set up a new solution in a new component library. Will it help to ensure better accessibility from the beginning?

Pipedrive has been developing a sales CRM using mostly typescript and React. Accessibility has never been high priority and at this point it is not very easy to get started. We have a design system and a React based component library to keep the look and UX consistent. This seems like a good place to start with solving accessibility issues. The library is used widely in the company and developer from different teams contribute to it. If a button in the reusable library gets fixed 95% of the buttons in the web app that our customers use should be improved.

This should not be taken as a way to solve all accessibility problems, but a good first step to get started. Making changes in a reusable UI library should have a wide impact on the products overall accessibility and without reaching some basic level there first it could be hard to start testing views in the final product.

3.1 Current state of awareness about accessibility in the company

First I sent out a survey in our company Slack channels (see figure 1) to understand what is the knowledge and general approach on web accessibility in the company. It was shared in one front end developers channel with 212 members, designers channel with 73 members, accessibility channel with 31 members and in out component library channel with 124 members. Some people might also be in more than one of these channels. The aim was to reach people in the company who would be most likely to be using these tools and who would be likely contributors to the library. There were 7 questions and some of they also included a field for free text to give more details on the subject if they wanted.

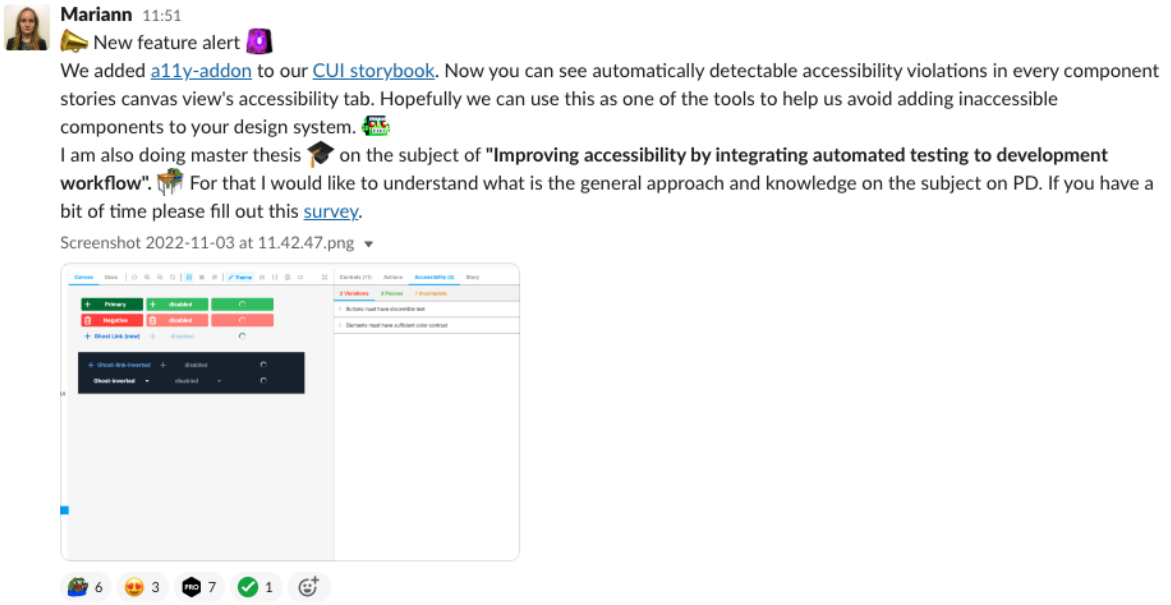

Figure 1: Message in company Slack announcing adding accessibility tool and inviting people to reply to survey.

In total 20 people replied to the survey - 6 designers and 14 developers, including one engineering manager. This does not give a full overview of the company, but it should give a good insight into what general opinions regarding accessibility might be. It is likely that developers and designers that are more involved with our component library and/or interested in accessibility where more likely to respond.

The results show that 10% of people who responded think their knowledge on accessibility is very good, while most think that knowledge level is average. 35% of responders know where do find resources about accessibility standards, 15% don't know and 50% know, but think they need more. They don't see that accessibility has high priority in the company currently, but at the same time 40% of responders think that following accessibility standards should be prioritized. There were several comments in the free text sections expressing saying they appreciate this subject being opened.

3.2 Adding tests to our component library

The next step was adding some automated test to our component library development workflow and observing their usefulness. The aim was to find something that we can integrate to our development workflow without a need to learn a new tool or add unnecessary complexity.

We use Storybook for our UI development. It is an open source software for UI development tool that allows you to work on one component at a time (Chromatic, n.d.-b). It allows us to render isolated UI components without integrating them to the final product right away. Our developers use it to test out the components they are developing locally. We also have a version of storybook available for anyone in the company to see with all the components. If this is the first place where elements will be rendered it seems logical to try to find a way to start testing them there.

To show a component you need to write a story - this means use it the way you would use it in the real world, and it will be rendered in a browser inside Storybook UI. It also has a sidebar for navigating between different examples, controls for additional tools and documentation. This means using a browser extension for accessibility tests would also test the UI around the actual example.

Storybook has a wide ecosystem of add-ons and one of them is addon-a11y. It uses Deque's axe-core to audit the HTML rendered for an isolated component (Chromatic, n.d.-a). Axe is an accessibility testing engine for websites and other HTML-based user interfaces. It includes WCAG 2.0 and 2.1 on level A and AA and promises to catch 57% of WCAG issues on average. (Deque Systems, 2023) This should be taken with a grain of salt, because we are intending to test isolated element and not the whole webpage and other studies comparing accessibility testing tools effectiveness have found the coverage to be lower **TODO: needs citation. This seemed like the best solution in our situation so addon-a11y was added to our component library's storybook. The tool is visible in the sidebar of every example. It shows all the checks that it has passed, all the violations that where found and any issues that could not be checked and might need manual testing. Every rule listed there also has reference to the HTML node that had the violation, explanation and links to Deque webpage with examples. This should be very useful in understanding and fixing these issues. It does not generate a report of all the issues found across the whole library for this another tool was used that will go through all the examples and generate a summary of all the violations found.

3.3 Manual accessibility audit

The second part is conducting a manual accessibility audit in the same library and comparing the results with the automatically generated test report results to determine what are its strengths and weaknesses and if testing isolated components poses any limitations.

3.3.1 Methodology for manual audit

In order to look at each component in isolation we used storybook here too. The accessibility add-on had already been installed, and we looked at the violations reported there too. We worked in a team of 4 people - 3 designers and 1 developer. We created a task for each component - 53 tasks in total.

We checked violations in accessibility add-on panel (see figure **TODO: add figure), tried using only a keyboard to navigate, tried using a screen reader to navigate. Each component had 1 or more examples. As many as was relevant to get the whole picture where looked at.

In the beginning of the audit we tested an add-on in storybook for mocking a screen reader (**TODO: Reference for this add-on and maybe and article about the difficulty of using a screen reader). It had the option to show the output a normal screen reader would play as audio as text. Initially it seemed like convenient solution with rather reliable results, but further investigation revealed that the output was very different from what an actual screen readers. For the rest on the audit we used Voice over - the MacBook built in screen reader, because it was available to us. There was and initial learning curve, but after that it went quite smoothly. All the component that had already been tested with the faulty add-on where looked over again using Voice over.

The audit results where documented in a table. We approached the problems for the users perspective and looked at what issues different types of users might encounter. We separated them to 3 sections:

1. Mouse user issues
2. Keyboard user issues
3. Screen reader user issues

We see this separation as a good way to prioritize fixing the issues in the future. Mouse user is the user we are considering in all of our development currently. The issues they would encounter should be most critical. This category includes a lot of visual, color contrast and image text issues.

The second type of user would encounter all the issues from the first category plus everything that is unreachable to them by using a keyboard. We looked at what functionality is or should be working when you use a mouse and tried to do the same things by only using a keyboard.

The third user was imitated by using a screen reader. The prerequisite for this was keyboard usability - if it was not possible to navigate using a keyboard then most likely it would not be very usable by a screen reader.

In real life these users might not be so clearly separated and there were many issues that

would affect all types of users, but as the intention was to come out of the audit with an actionable list we needed to prioritize the issues while we found them in order of severity and current customer impact. These categories also mostly depend on each other, so it would make sense in most cases to start from solving mouse user issues, then keyboard user issues and then screen reader user issues.

3.3.2 Comparing results from manual audit and automated report

To get the whole list including all components a report was created that included the violations caught in each example of each component. This report was generated at the beginning of the manual audit, so the results obtained from both methods are based on the same source code.

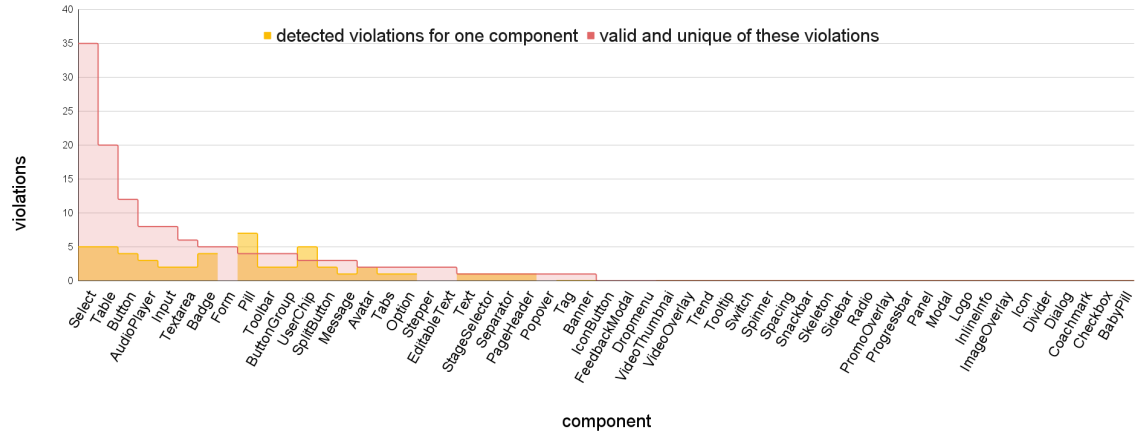

Figure 2: All violation found by addon-a11y and how many of them are valid.

I prepared a comparison table form both results. For automated accessibility tests I recorded the number of examples that included violations, the number of different violations and the number of passed checks for each component. In most cases there were more examples with violations because the same thing was reported in more than one example (see figure 2). Addon-a11y did not report any issues for 27 components out of 53 components.

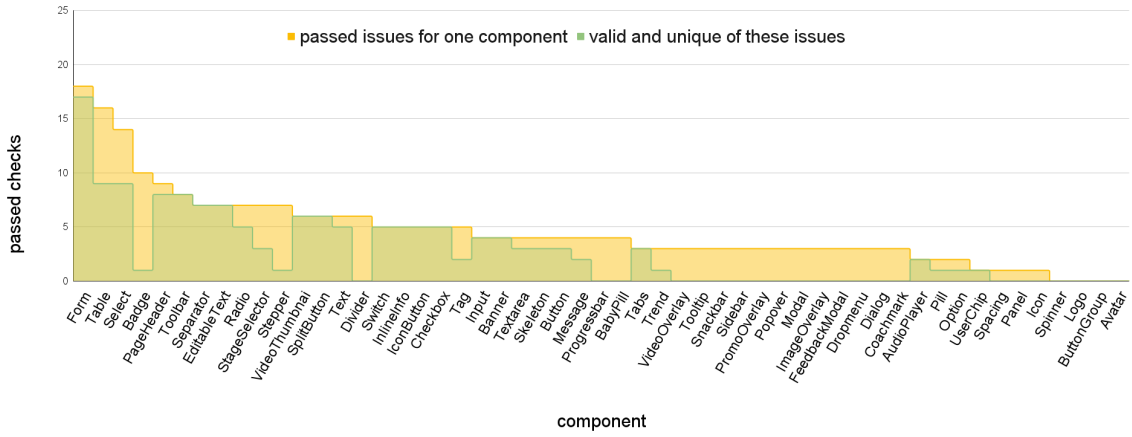

Figure 3: All passed checks reported by addon-a11y and how many of them are valid.

Passed issues where looked over to determine how many where valid (see figure 3). 4 components did not have any passed issues and 22 did not have any valid passed issues.

Looking at all the fails and passes gives an overview of what was checked for each component. Out of 53 components only 2 did not get checked by addon-a11y at all. In addition, components that become visible only when triggered by another element, like modals and popups that currently in our library displayed with a button as a trigger, don't get tested. These components are seen on figure 3 starting from *VideoOverlay* and ending with *Coachmark* - 27 over all. This means 29 components where not tested but this tool and rest of the components had passed or failed issues, but often not both.

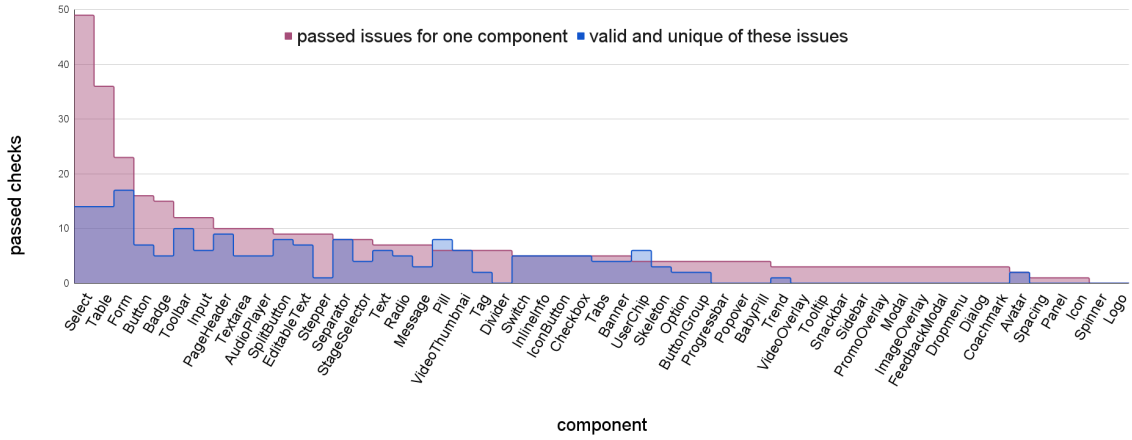

Figure 4: All issues that where tested by addon-a11y. This means both passed and failed issues combined.

In some cases there were 0 violations detected, and 0 valid checks passed – this means that the automated testing was not effective (**TODO: Make chart or table for this).

3.3.3 Limitations of using Storybook's addon-a11y

The accessibility add-on in Storybook analyzes the examples that have been made for the component and unsuitable example can cause false results. Like Components triggered by a button described before. This is because the initial HTML that the accessibility tests are being run on only has the button and the tests are not being run again after triggering the element. This could potentially be remedied with better examples.

The biggest limitation of this tool currently is that it can only be view-d in storybook. To see the number of passes and fails you need to open the accessibility tab for each component. I looked into ways of automating this so that the same checks could be run on every change to the library and added to continuous integration (CI) workflow. In the current version of Storybook there is no easy way to do this, but it will become much easier in the next major version.

Upgrading out component library to that version need some extra work to make it compatible, but I have tested out this solution on a test library, and it seems like it would definitely be an improvement. Running the tests in CI would ensure that they are run every time someone makes a change and not only when we choose to. We could also block changes that don't pass the required accessibility checks.

TODO: Reasons for automated check not being effective

4 Results

5 Discussion

5.1 Limitations of the study

5.2 Future research

Conclusion

(Alsaeedi, 2020)

References

Abou-Zahra, S., & Henry, S. L. (Eds.). (2020). *Accessibility Conformance Testing (ACT) Overview*. Retrieved 02/18/2023, from %5Curl%7Bhttps://www.w3.org/WAI/standards-guidelines/act/%7D

Abou-Zahra, S., Steenhout, N., & Keen, L. (Eds.). (2017). *Selecting Web Accessibility Evaluation Tools*. Web Accessibility Initiative (WAI). https://doi.org/10.1145/1061811.1061830

Alsaeedi, A. (2020). Comparing Web Accessibility Evaluation Tools and Evaluating the Accessibility of Webpages: Proposed Frameworks [ZSCC: 0000043]. *Information*, *11*(1), 40. https://doi.org/10.3390/info11010040

Chromatic. (N.d.-a). *Accessibility tests*. maintained by Chromatic. Retrieved 03/02/2023, from https://storybook.js.org/docs/react/writing-tests/accessibility-testing/

Chromatic. (N.d.-b). *Introduction to storybook for react*. maintained by Chromatic. Retrieved 03/01/2023, from https://storybook.js.org/docs/react/get-started/introduction

Deque Systems. (2023). *Axe-core*. Retrieved 03/02/2023, from https://github.com/dequelabs/axe-core

Fowler, M. (2006). Continuous Integration [ZSCC: NoCitationData[s0]]. *martinfowler.com*. Retrieved 03/05/2023, from https://martinfowler.com/articles/continuousIntegration.html

Sane, P. (2021). A brief survey of current software engineering practices in continuous integration and automated accessibility testing. *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. https://doi.org/10.1109/wispnet51692.2021.9419464

WebAIM. (2019). *Accessibility evaluation tools*. Institute for Disability Research, Policy, and Practice, Utah State University. Retrieved 02/27/2023, from https://webaim.org/articles/tools/

** All references are listed remove before submitting

Appendices

List of Figures