Tallinn University
MSc Human-Computer Interaction

Improving Web Accessibility Through Continuous Evaluation of Component Libraries

Master's Thesis

Author: Mariann Tapfer
Supervisors: Mustafa Can Özdemir, MSc
Mari-Ell Mets, IAAP WAP

Tallinn 2023

Notes

Abstract

Acknowledgements

Table of Contents

List of Figures

List of Tables

List of Abbreviation

ACT Accessibility Conformance Testing

API application programming interfaces

CDD Component Driven Development

CI Continuous Integration

CLI command-line interface

CRM Customer Relationship Management

DOM Document Object Model

EU European Union

HTML HyperText Markup Language

IAAP International Association of Accessibility Professionals

IAAP WAP International Association of Accessibility Professionals Web Accessibility Specialist

UI User Interface

USA United States of America

UX user experience

W3C World Wide Web Consortium

WAI Web Accessibility Initiative

WCAG Web Content Accessibility Guidelines

WHO World Health Organization

1 Introduction

Overview of the entire thesis. It ends by describing the organization of this document.

1.1 Research Problem and Goal

Research Problem and Significance: What is the problem you are addressing? Why is this a problem? Why is this important to HCI? Supported scientific references? ate 3 4 ½ page max
Research Goal and Motivation: Why are you doing this research? what will be your contributions? Supported scientific references? ate 3 4 ½ page max

According to World Health Organization (WHO) 1 billion people or about 16% of the world's population are estimated to have a significant disability (World Health Organization, 2022). They may not be able to use websites and mobile applications in a conventional way. To provide equal possibilities, digital environments need to be made accessible. There is proof that focusing on accessibility has a profound effect on the overall quality of products and services and therefore provides a big business value (Miesenberger et al., 2020).

Developers find it hard to maintain a high level of accessibility over time as it is considered more at the first release of the product (Paternò et al., 2020). Continuous Integration (CI) is used to speed up development and maintain general code quality (Zhao et al., 2017) and it could also help in maintaining a high level of accessibility by testing the added code against predefined accessibility requirements. There is some initial research into the current state of accessibility testing in CI, but it could be improved by doing a case study for an organization (Sane, 2021).

Design Systems are an increasingly common way to build websites because they help in producing a consistent user experience (Yew et al., 2020). Influencing how these are built is important in the field of HCI because it will allow us to play a role in shaping the future of User Interfaces.

Many companies have design systems to help maintain consistency both in what the end users experience and how the code is written. This is a good place to start with establishing a basic level of accessibility. Making changes there will have the widest impact and will build a solid foundation. Most automated accessibility testing tools are intended to be used for the whole page or website. I want to run accessibility tests on the individual components in a component library. However some aspects of accessibility, like correct usage of headers or page language, can only be evaluated on a full page. I draw parallels between isolated component testing and unit tests in software development.

Most accessibility checkers are intended to be used for the whole page or website. I want to

run these tests on the individual components in a component library because that's a key part of most Design Systems (Yew et al., 2020). Some aspects of accessibility, like correct usage of page headers or page language, can only be evaluated on the full page. I draw parallels between isolated component testing and unit tests in software development. The goal of unit tests is to verify that small pieces of code function properly (Humble & Farley, 2010, p.60). Similarly, component testing should ensure that this specific component does not have any issues related to accessibility.

The next step would be running end-to-end tests on the whole page or application that is made up of these small parts. End-to-end tests should imitate what would happen when end users are using the product. When individual component tests are working well then there should already be fewer issues when you test the whole page.

My research goal is to test an automated accessibility tool in Pipedrive's component library to explore the potential pros and cons of accessibility testing in the CI of a component library.

1.2 Research Questions

RQ1: How good is the knowledge about accessibility standards, tools and best practices in the company before integrating the accessibility testing tool?

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

RQ3: To what extent can integrating automated testing into a component library's development pipeline help improve its compliance with Web Content Accessibility Guidelines (WCAG)?

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

2 Theoretical Background

This chapter will give a short overview of web accessibility, accessibility evaluation and component libraries. The goal of this is to understand the context around accessibility, how it is regulated and defined and how it can be evaluated. The case study that has been done as a part of this thesis focuses on a component library's compliance with common accessibility standards. I will go over what a component library is and why are they used.

2.1 Web Accessibility

Accessibility is about providing equal access to goods and services to everyone regardless of age, gender, knowledge or disabilities. We all have different abilities, some run faster than others, some see more clearly and some might not be able to hear sounds. A report published by the European Commission in 2020 estimates that about 7% of people aged 16 and over who live in the European Union have a strong disability and about 17.5% a moderate disability (Grammenos, 2020). This number gets significantly higher when people get older. Disability prevalence among people aged 65 and over is about 47.8%. Disability is a part of being human and persons with disabilities have very different needs. We can also talk about temporary disability - this might be a mother who can only use one hand because she is holding a child in the other or someone who broke an arm and can't use it until it heals. The experiences during a temporary disability may be very similar to a permanent condition. Everyone who designs the physical and virtual environment around us should consider these different limitations.

Accessibility principles can be applied to various fields. For example, in architecture, it could be designing buildings in a way that can be accessed by people who can walk as well as the ones who need to use a wheelchair. In digital products, it is more often related to the senses we use to navigate and consume content. Everything should be equally accessible regardless of the sense someone wants or need to use for consuming the information.

The biggest strength of the web lies in its availability to everyone. Tim Berners-Lee, WCAG Director and inventor of the World Wide Web said: "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect" (World Wide Web Consortium, 1997). The virtual world has the potential to be much more inclusive than the physical world.

Web accessibility considers auditory, cognitive, neurological, physical, speech and visual disabilities - everything that might affect someone's ability to access the Web and people using different devices, who have reduced abilities because of aging, temporary disabilities, situational limitations or are using limited or slow internet (Henry, 2022).

2.1.1 Web Accessibility Standards

The standards for considering people with different abilities and making web content accessible for different ways of consuming are defined in WCAG (Kirkpatrick et al., 2018). The guidelines are developed by Web Accessibility Initiative (WAI) which is part of World Wide Web Consortium (W3C).

The first version of WCAG (1.0) was published in 1998 (Vanderheiden et al., 1998). The current version is WCAG 2.1 and the next version 2.2 is scheduled to be finalized in May 2023 (Henry, 2023). All WCAG versions are backward compatible, meaning that all requirements that were in 2.0 are included in 2.1 and some new requirements are added. Version 2.2 is currently in the draft stage, but the planned changes also include some minor changes to the current guidelines.

Legally discrimination against disabled people is covered by different laws and directives around the world. Many of them are based on or derived from some version of WCAG. Pipedrive operates in the US and European markets so regulations there are most relevant to my case study.

In the United States of America (USA) accessibility is addressed in two laws (Siteimprove, n.d.). Amendment of Section 508 regulates accessibility related to Federal agencies and organizations that receive federal funds or are employed under contract with a federal agency and includes WCAG. Title III of the Americans with Disabilities Act ensures that public goods and services are equally accessible to everyone.

European Union (EU) has two directives that outline what accessibility should look like and leave it to each member state to make national laws based on that. EU Web Accessibility Directive states that the public sector should follow WCAG 2.1 Level AA and European Accessibility Act aims to ensure that essential products and services traded between EU member states are equally accessible to everyone.

WAI gives an overview of the laws and policies around the world. 25 out of 40 listed on the page are based on WCAG 2.0 or WCAG 2.0 derivative (Mueller et al., 2018). In this thesis, I will test against WCAG 2.1, because it is the latest published version of the most widely used standard.

WCAG is intended for developers of web content, authoring tools or web accessibility evaluation tools and others who need a standard for accessibility (Henry, 2023). The 4 principles of accessibility addressed in the guidelines are:

1. Perceivable - information and user interface should be presented in a way that users don't have to rely on a single sense to perceive it.
2. Operable - user can interact with the interface,

3. Understandable - user should understand the information and how the user interface operates
4. Robust - content should be robust enough that it can be interpreted reliably by a wide variety of user agents and assistive technology.

Under each principle is a list of guidelines that address that principle (Accessibility Guidelines Working Group (AG WG) Participants, 2022). Under each guideline, there are Success Criteria that describe specifically what conformance to it means. Each Success Criterion is a statement that will be either true or false for specific Web content.

2.1.2 Web Accessibility Rules Format

W3C Accessibility Guidelines Working Group has developed Accessibility Conformance Testing (ACT) Rules Format to provide developers of evaluation methodologies and testing tools a consistent interpretation of how to test for conformance with accessibility requirements like WCAG (Fiers et al., 2019). The format describes both manual and automated tests. The aim of this is to make accessibility tests transparent and results reproducible.

For example, accessibility testing tools check if the provided HyperText Markup Language (HTML) meets the requirements defined in WCAG. These requirements are different for each element, but they might be also combined and include more criteria. Each element needs a specific set of requirements to be checked.

ACT Rules include atomic rules that define an element to be tested for a single condition and composite rules that can combine multiple atomic rules to determine if a single test subject satisfies an accessibility requirement (Fiers et al., 2019). Each rule defines when it should be applied. For atomic rules, this could be an HTML tag name, computed role or distance between two elements for example and composite rules it is a union of the applicability of the atomic elements it combines.

2.2 Accessibility Evaluation

Different evaluation methods are used to determine if a website or mobile application is accessible. These include expert review, user testing, subjective assessment, screening techniques or barrier walkthrough. Each method has its pros and cons depending on the subject of the evaluation, available experts and other factors.

Expert review, also called, conformance, standards or guidelines review or manual inspection is most widely used (Brajnik, 2008). It means checking if a page satisfies a checklist of criteria. The results depend on the evaluator's opinions and depend on the chosen guidelines. Skillful evaluators are needed for this method to be effective.

Barrier walkthrough is a technique where the evaluator has to assess how seriously some predefined barriers impact the user to achieve their goal on the website (Brajnik, 2008). An Accessibility barrier can be any condition that makes it difficult for people to achieve a goal.

Screening techniques consist of evaluating a website by using the interface while some sensory, motor or cognitive capabilities of the user are artificially reduced (Brajnik, 2008). Subjective assessment is based on a panel of users exploring a website themselves and giving feedback on what worked and what did not. User testing means conducting usability tests with disabled people while adopting a think-aloud method to get feedback on their experience.

Which method is suitable depends on several factors including the availability of skilled auditors and other resources. Evaluations with experts are very dependent on the knowledge and experience of the evaluators. Research into the effect of expertise on web accessibility evaluation conducted by Brajnik et al. shows that when web pages are evaluated with non-experts we see a drop in reliability and validity (Brajnik et al., 2011). Even with experts, the result will vary a bit, but the results should even out when at least 3 experts are used. For the same level of reliability, at least 14 evaluators that are not considered experts in the field of web accessibility are needed.

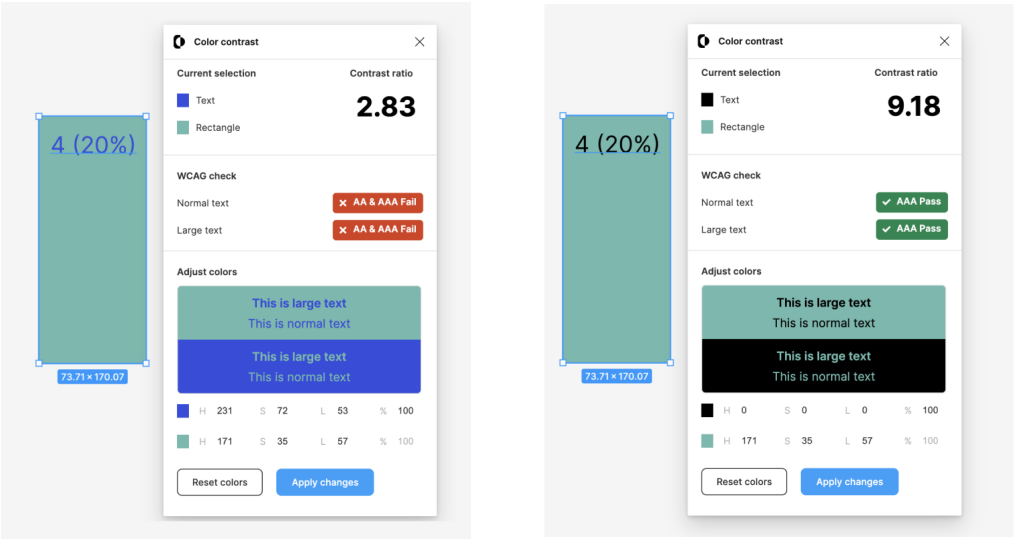
Mari-Ell: Here I would add the passage about how manual testing is essential as not all the WCAG criteria is machine-testable - some of the criteria is subjective and needs human opinion. Today there are no tools that can interpret if the image's text alternative really conveys its meaning or if the error message of the input field is clear enough.

2.2.1 Tools for Accessibility Evaluation

There are software programs and online services that help determine if web content meets accessibility guidelines. These tools may support various standards. They might be browser or authoring tool plugins, command line tools, code linters, open source application programming interfaces (API), desktop or mobile applications or online services (Abou-Zahra et al., 2017). Some tools are aimed at non-technical content creators and these are often built into the tools they use daily. Others are online services, where you can enter the URL of your website to be evaluated or services that regularly check your website and that are hosted by the provider or in a company's internal network.

The results can be presented as a report in HTML, CVS, PDF, XML or other formats, as in-page feedback with temporary icons and markup or as a step-by-step evaluation where the user is prompted to assess the parts that can't be automated (Abou-Zahra et al., 2017). The tool might transform the page, show only text or take away all the colors, to help identify issues. The tools can evaluate either a single page or a group or related pages at the same time. Some tools are even capable of accessing password-protected content.

Browser extensions and online tools usually evaluate one page at a time. This might work well for small websites and one-time audits but can be quite ineffective to use as a continuous long-term solution. Command-line interface (CLI) tools need more setup, but can potentially be seamlessly integrated into the development workflow.



(a) Color contrast test failing *(b) Color contrast test passing*

Figure 1: Figma plugin for testing color contrast compliance with WCAG

Authoring tool plugins for Figma that check if the colors used in the design conform with WCAG requirements (see Figure 1) or plugins for integrated development environments that provide immediate feedback to a developer in their working environment when they miss something related to accessibility in their code.

Some tools do a very specific task or work inside a very certain tool and others try to tackle accessibility as thoroughly as possible. In most cases, using many different tools in combination will give the best result.

2.2.2 Continuous Accessibility Evaluation

CI is a software development practice where members of the team integrate their work frequently and each integration is verified with an automated build that includes tests to detect errors as quickly as possible (Fowler, 2006). 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 codebase for bugs. These tests need to be easy to trigger and indicate any failures.

This last principle could be applied well to accessibility evaluation. This practice would follow modern software development principles. The automated tests 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.

Tools that can be set up to perform checks every time changes have been made automatically with minimal manual effort can be used for continuous accessibility testing. The earlier in the development process these can be triggered, the better. Not all of the tools described in the previous chapter can be used for that. Browser extensions and plugins in authoring tools need to be manually triggered to perform checks.

CLI based tools are most suitable for this purpose. They can be customized to specific needs and integrated to run together with other tests. The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web (MDN contributors, 2023). These programs scan the rendered DOM of a website against accessibility standards like WCAG to find violations. These errors are reported back with a reference to the code that produced the error.

According to the survey conducted by Level Access about the state of digital accessibility 67% of organizations that practice CI also include accessibility tests (Level Access et al., 2023). This has gone up from 56% reported in 2021. 91% of all organizations that participated in the survey say they use accessibility testing tools and most of them prefer the free options available. Browser extensions are by far the most popular tool with 82.4% of organizations using them. 23.4% has reported that they use testing technology that integrates with their CI tool and 22.7% uses testing technology that is compatible with their test framework. Organizations that have an accessibility program in its early stages (2-6 years) or have International Association of Accessibility Professionals (IAAP)-certified personnel are more likely than the average to have implemented a CI testing process.

It is important to mention that automated tests can detect only a part of all the possible violations. The UK Government Digital Service Accessibility team compared 13 automated tools' performance in 2018 on a page that had 142 deliberately introduced accessibility issues and found the automated tools were able to detect 13-40% of issues. Abbott compared the two most popular accessibility tools that can be used in CI using the same deliberately inaccessible site in 2021 and reported that axe-core caught 27% and pa11y 20% (Abbott, 2021). Vigo et al. compared 6 different tools in 2013 and found that they were able to detect 23-50% of WCAG 2.0 Success Criteria (Abbott, 2021; GDS Accessibility Team, 2018; Vigo et al., 2013).

The tools and WCAG standards have evolved during this time, but the common conclusion to most of the similar comparisons is still that most tools will be able to detect errors on 20-30% or WCAG Success Criteria.

Find citation

Axe-core accessibility testing engine promises to find on average 57% issues, which is

significantly higher than other tools on the market (Deque Systems, 2023). They claim that the current statistics are founded on an inaccurate definition of accessibility coverage - the percentage of individual WCAG Success Criteria. In reality, some types of issues are found much more frequently and the issues found by automated tests form a higher percentage of all issues compared to those discovered by manual detection. They suggest that the coverage should be the percentage of all the issues found on a site. They conducted a study where they compared 2000 audit results from testing pages with axe-core and manual testing methodology. The average percentage of the number of issues detected by axe-core for each data set is 57.38%.

They also say that looking at the results with current, in their opinion, inaccurate, definition of coverage would be in the range of 20-30% (Deque Systems, 2021). I did not find any studies about other tools that would use the same method for calculating the coverage so for this paper we will still compare the tools according to the most widely used coverage calculation method.

There is still a limit to what can be detected no matter how we define the coverage.

Mari-Ell: Add something here about the fact that it is not that the tools are BAD, they just cannot interpret things that need human opinion. However, many things that in my opinion could be tested with tools, for some reason, cannot - image alt texts that are just "object" or "image" are usually nor marked as error, and icons and input fields that are almost invisible, are also not marked as errors.

Manual testing will always be required to ensure that the content is fully accessible. The main strength of automated tests is that they can be set up to run automatically, and they provide measurable results. Therefore, it is a good way to monitor compliance with WCAG rules consistently without much extra effort. This can help avoid unwanted changes and highlight issues in code that should be straightforward to fix.

2.3 Component Libraries

start from design systems (Churchill, 2019)

Component Driven Development (CDD) is a development methodology that anchors the build process around components (Coleman, 2017). In 2017 Coleman called CDD the biggest trend in User Interface (UI) development.

A component is a well-defined and independent piece of UI, like a button, checkbox or card (Ella, 2019). This approach correlates well with other similar widespread principles that promote modularity in software development like atomic design and micro-front-ends. Designing each component as a standalone unit improves maintenance, reusability, and testing,

shortens the learning curve and makes development faster.

The key to success here is the separation of concerns and isolating a logical piece so that it can be worked without distractions. It promotes concentrating on details and refining the element as much as possible. These pieces can be combined into more complex components if needed and when combined they can make up views that become the whole site or web page.

It is not very straightforward to preview one single separated component and if you need to set up a test or live environment for the whole webpage it would defeat the purpose of dividing the big problem into manageable chunks. In development tools called component explorers are often used to mitigate that.

Component Explorer is a separate application that showcases the components in various states (Coleman, 2017). Working on a single component by manipulating the entire web page or app to a certain state can be difficult without a tool like that. This allows developers to test a given component in all the states that have been defined in isolation and make it easy to build one component at a time. It also makes it easier to go through all the possible states in one component and promotes the reusability of these elements.

Bit, Storybook and Styleguidist are popular tools used in component library development (Ella, 2019). Bit lets you pack the bundled and encapsulated components and share them on the cloud where your team can visually explore them. Storybook supports multiple frameworks and provides a rapid component development and test environment. The environment also allows you to present and document your library for better reusability. Styleguidist is useful for documentation and demoing different components.

Design Systems are a popular way to help companies scale design work and build products with consistent user experience (Yew et al., 2020). Yew et al. defines it as: "...a repository of reusable components that follow a set of shared design principles". Many companies build their own Design System. It should include a component library, style guide, design and content guidelines, design token library, interactive prototyping tools and accessibility guidelines.

Many big companies have a design system with an open source code that includes a component library, making it possible to reuse and extend them. Some of the more known ones include Material-UI (Material UI SAS, n.d.), Adobe Spectrum (Adobe, n.d.) and Bootstrap (Collings et al., n.d.).

Using a component library provides consistency and better quality. These components can be polished over time to provide the best user experience to end users when they are integrated into the final product. The effort that can be put into developing a component that will be used in multiple projects is bigger than what would be reasonable for a single use case.

3 Research

Come up with a list of things that should be tested manually, based on my case study

The research conducted for this thesis can be divided into 6 steps:

1. Researching automated testing tools
2. Pre-case study questionnaire
3. Setting up automated testing tools in Pipedrive's component library
4. Manual accessibility audit
5. Comparing manual and automated testing results
6. Post-case study questionnaire

The first thing was to look at different automated accessibility tools that are available, test them out and see what would best fit our purpose. The tool needed to be easy to use for everyone and not block development. The next step was to send out a questionnaire to better understand the current approaches toward accessibility among the people who work on it the most. After that, an automated accessibility tool was added to the component library repository and announced in relevant channels to raise awareness about it.

In parallel to observing how the tool was being used I conducted a manual accessibility audit together with 3 designers on the same component library. These results were then compared with the automated testing results that we got from the same version of the library. As the last step, I sent out a questionnaire to gather information about how the tool had been adopted. To get more details I also interviewed some developers and designers I knew had used the tool during that period.

Pipedrive has been developing a sales Customer Relationship Management (CRM) using mostly typescript and React. Accessibility has never been a high priority and at this point, it would not be very easy to get started. We have a design system and a React-based component library to keep the look and user experience (UX) consistent. This seemed like a good place to start solving accessibility issues. The library is used widely in the company and developers from different teams contribute to it. If a button in the reusable library gets fixed, most 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 as a good first step to get started. Making changes in a reusable UI library should have a wide impact on the product's overall accessibility and without establishing a basic level of accessibility there, it would be difficult to start testing individual pages of the final product.

3.1 Research Methodology

Methodology – The overall strategy to conduct the research, e.g. experimental, quasi-experimental, correlational, descriptive, action research, design research, ethnography – Determined by the research question - summarise the research strategy and methodology for conducting the research.

3.1.1 Literature review

A literature review is a critical analysis of existing research and scholarly literature on a particular topic. It involves systematically reviewing, evaluating, and synthesizing existing knowledge and research findings in a specific field to identify gaps in the current knowledge and identify topics for further research (Luft et al., 2022).

In my work literature review was needed to understand the current landscape of automated accessibility testing. These kinds of tools have been available for a long time and I wanted to understand how much we already know about their strengths and weaknesses. Have they been tested in real-world situations and what are the results? What is the difference between these tools? Have some been proven to be better than others?

My method for finding relevant scientific articles was to use EBESCO discovery service for Academic Library of Tallinn University to find any articles with keywords accessibility, automated evaluation and continuous integration. I also conducted a similar search on Google to find any nonscientific articles about the subject. I went through the references in all of these articles to identify any relevant research that I should also include.

This gave me pretty solid results and very soon I started to see the conclusion references in the articles I read repeat themselves. The tools that are used evolve fast and this would make older articles irrelevant to the current state of continuous accessibility testing. As a general rule, I focused on articles published in the last 10 years and only looked at old ones when they gave more high-level overviews or methods and did not focus on comparing specific tools.

I looked through any research papers on the subject of automated accessibility testing and also any relevant articles or blog posts. Things change fast in software development and I was not expecting to find the most up-to-date information from scientific publications. That's why I thought that it was necessary to also include resources that have been published on reputable sites.

Many studies have been done to compare different accessibility testing tools (Alsaeedi, 2020; Duran, 2017; Ismailova & Inal, 2022; Rybin Koob et al., 2022; Sane, 2021; Vigo et al., 2013). I took a look at the results of these studies and tried out enough tools on my own to find one that would be usable for my purposes, but my intention was not to compare them methodically as a part of this work. I will rely on the comparisons that have been made by others in the

past.

3.1.2 Case study

A case study is an in-depth analysis of a bounded system (Range, 2023). It involves multiple forms of data collection like observations, interviews, documents, reports and analysis. The case can be a specific individual, group, community, business, organization, event or phenomenon. It can be chosen because of its uniqueness or typicality. The goal of using case study methodology is to investigate something contemporary in its real-life context.

The results of a single case study might be very subjective to that particular case and to the biased opinions of the researcher and can't always be generalized and applied to other similar situations (Range, 2023). The richness of detail they provide makes them fascinating and often there is a lot to learn from them. Sometimes these insights can be applied to other similar cases. It is a good method for exploratory or critical and unusual cases.

I chose this method to explore the possibilities of automated testing. I conducted an organizational case study that focused on the process of evaluating and improving the accessibility of CRM tool called Pipedrive. In the scope of this study, I added an automated accessibility evaluation tool in our component library and conducted a manual accessibility evaluation of the same library together with 3 designers working in the company.

Before implementing the new tool I sent out a questionnaire to understand the knowledge about and approaches towards the subject of web accessibility in the company among people who are most likely to be doing work related to it. At the end of my research, I sent out another questionnaire with a focus on gaining information from people in the organization who use the tool that I added during that period. I wanted to know if it was helpful and if they had any issues using it and what other opinions they might have about it. The goal of both of these was to get more detailed information about the experience of the real users of the tool.

I organized the data collected from automated and manual testing to make it comparable. I used statistical analysis to gain valuable insight into how well these two methods work and how they might differ from one another.

3.2 Current state of awareness about accessibility in Pipedrive

First, I sent out a survey in our company's Slack channels (see figure 2) to understand what is the knowledge and general approach to web accessibility in the company. It was shared in 4 channels to reach people who are most likely to be dealing with accessibility, our component library and who are invested in accessibility (see more details in table 1).

Channel members or theme	number of members
Front end developers	212
Dedicated to our component library	124
Designers	73
Accessibility channel	31

Table 1: List of all the channels the survey was shared in.

Some people might also be on 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 contribute to the library. There were 7 questions and some of them also included a field for free text to give more details on the subject if they wanted.

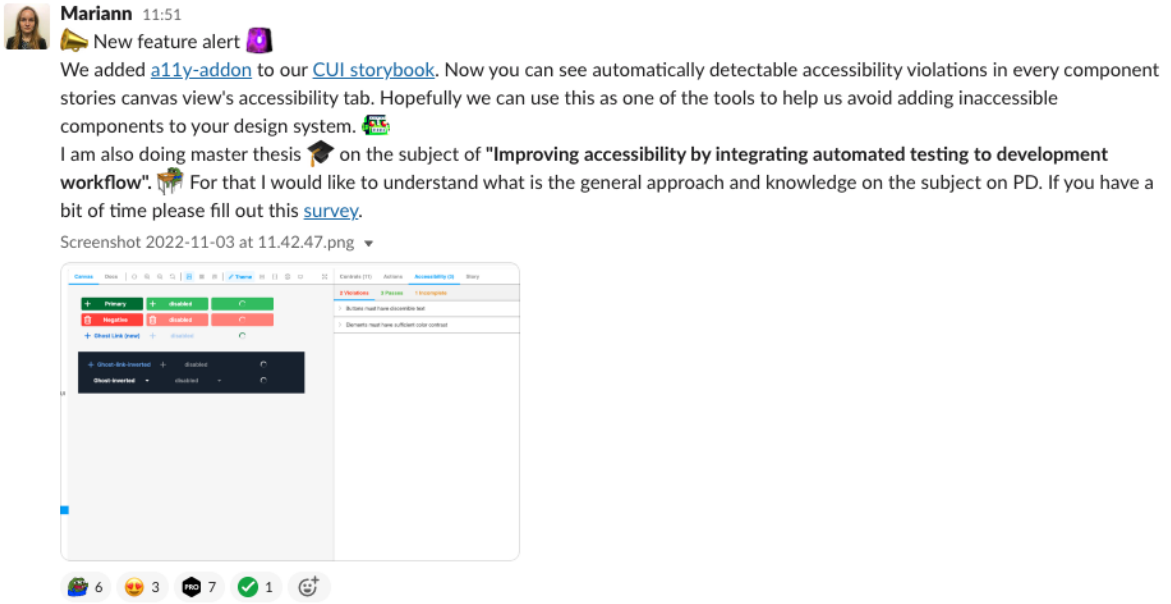
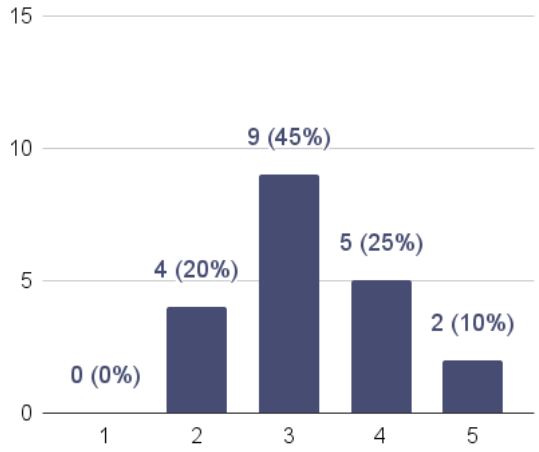
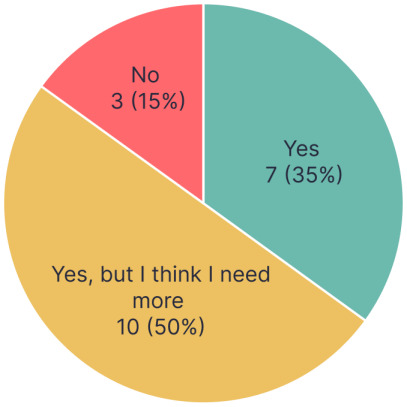


Figure 2: Message in company Slack announcing adding accessibility tool and inviting people to reply to the survey.

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 the general opinions regarding accessibility. Likely, developers and designers that are more involved with our component library and/or interested in accessibility were more likely to respond.



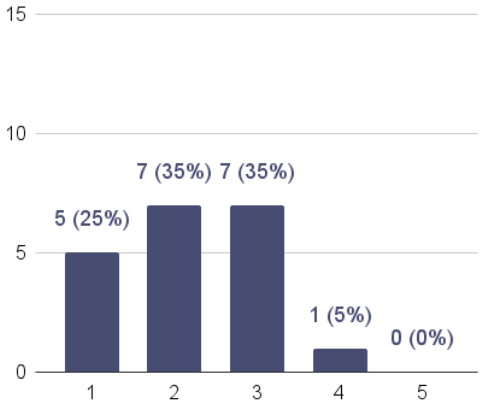
(a) Pre-case study survey: How much do you know about accessibility (standards, best practices, importance, testing)? 1- "Almost no knowledge" to 5 - "I have very good knowledge"



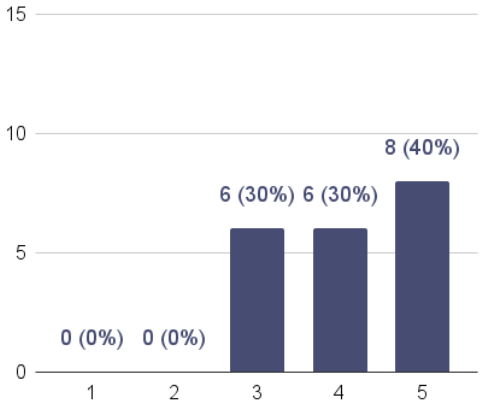
(b) Do you know where to find resources about accessibility standards? (no one chose the 4th option "No I don't think I need them")

Figure 3: Accessibility knowledge in Pipedrive

The results show that 10% of people who responded think their knowledge of accessibility is very good, while most think that their knowledge level is average (see Figure 3a). 35% of responders know where to find resources about accessibility standards, 15% don't know and 50% know, but think they need more (see Figure 3b).



(a) Do you think accessibility is a priority in our company? 1-No not at all to 5-Yes very much



(b) Do you think we should prioritize following accessibility standards (like WCAG, EN 301 549) in our product? 1- "No I think they are irrelevant for Pipedrive customers" to 5 - "Yes I think following accessibility standards should be a high priority"

Figure 4: Priority of accessibility in Pipedrive

Most people who replied to the survey don't think accessibility is being prioritized in Pipedrive

and at the same time, most of them think we should be putting more focus on following common accessibility standards (see Figure 4). They elaborate a bit more about what they think the reasons behind this are. Accessibility has always been at the end of our priorities and there is no clear company-wide strategy on how to manage and improve it. There has never been a dedicated project manager to bring focus to it and right now developers and designers are the ones that seem to be responsible for it. Two people also mentioned that the priority might be low because as a private service, Pipedrive does not have a legal obligation to comply with any specific accessibility standard. Two people think that it would not affect Pipedrive's customers much.

It seems like the biggest obstacle to improving the accessibility of Pipedrive is the lack of knowledge about the business impact. More knowledge would help prioritize solving these problems. There is a lot currently that could be improved, but in my opinion implementing a way of continuously testing for at least some basic issues from the beginning of all new developments would help to ensure that the technical debt related to accessibility would not grow to be even bigger. There were several comments in the free text sections expressing saying they appreciate this subject being brought to focus and even some offering to help.

3.3 Adding automated accessibility tests to component library

The next step was adding some automated tests to our component library's development workflow and observing their usefulness. The aim was to find something that we can integrate easily and that does not have a steep learning curve or add unnecessary complexity.

We use Storybook as the component explorer for our reusable component libraries development. It is open-source software for UI development that allows teams to work on one component at a time (Chromatic, n.d.-b). It allows us to render isolated UI components without integrating them into 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 with all the components available for anyone in the company to see. As this is the first place where elements are rendered it seemed logical to try to find a way to start testing them there.

Each component has stories – examples of how the component would be used in real life. These stories will be rendered in a browser inside Storybook UI. It also has a sidebar for navigating between different examples, and controls for additional tools and documentation. This means it would not be very easy to use a browser extension for accessibility testing because it would also test the Storybook UI around the actual relevant example.

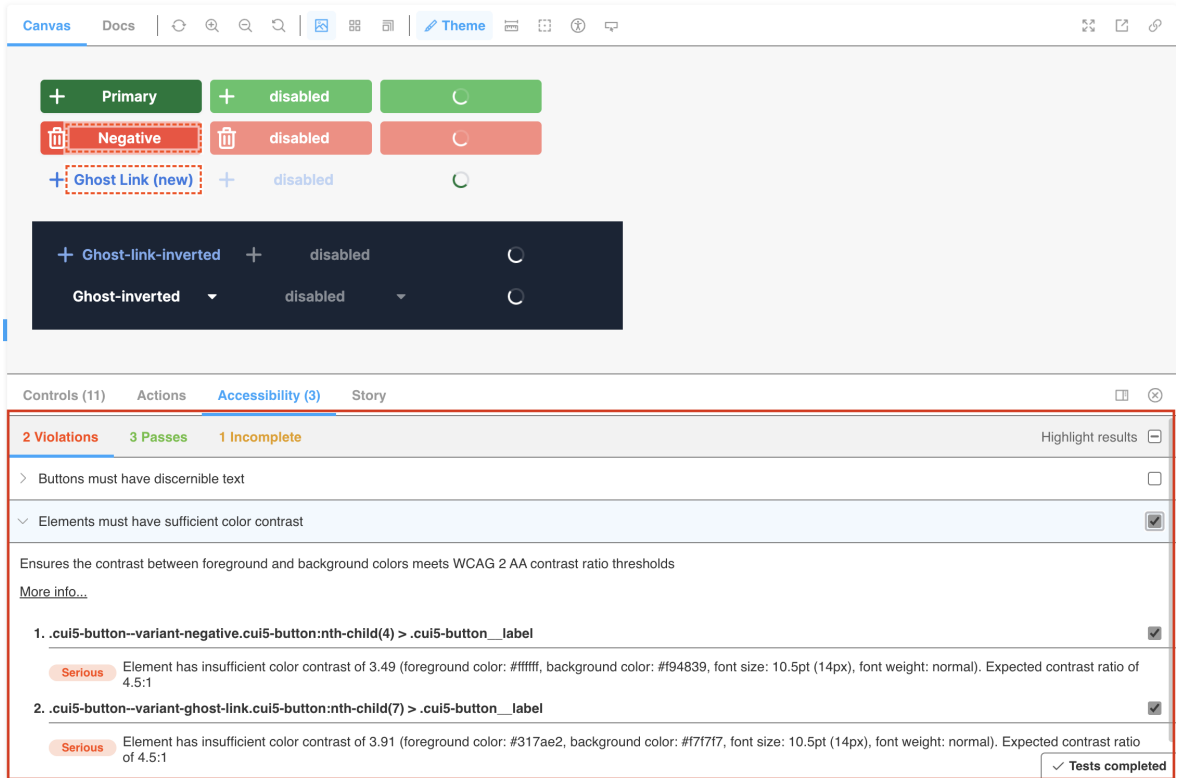


Figure 5: Accessibility add-on for Storybook (addon-a11y)

Storybook has a wide ecosystem of add-ons and one of them is addon-a11y (see figure 5). It uses Deque's axe-core to audit the HTML rendered for an isolated component (Chromatic, n.d.-a).

Axe-core is an accessibility testing engine for websites and other HTML-based user interfaces (Deque Systems, 2023). It includes WCAG 2.0 and 2.1 on levels A and AA and it has a coverage of about 20-30% if you calculate it in a commonly used way, but they claim it to be 57% according to their new and improved calculation method. This has been explained in more detail in the Automated accessibility testing chapter.

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 add-ons sidebar of every component example. It shows all the checks that the component has passed, all the violations that were 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 the Deque webpage with examples. This should be very useful in understanding and fixing these issues.

The rules format that axe-core uses is developed by Deque Systems and is an adoption of the ACT rules format developed by WAI (Fiers, 2017). They have a set WCAG that can be evaluated in a fully automated way. They are divided by WCAG standards version (2.0, 2.1, 2.2) and Level (A & AA, AAA) and they also have some rules for best practices in the industry that improve the user experience but might not conform to WCAG success criterion

(Fiers & Lambert, 2023).

I used Storybook-a11y-report (Karube, 2020) to generate a report of all the violations in the whole component library. It went through all the examples and ran the same tests as addon-a11y to generate a summarized report. The final report links back to each component's story where you can see the example and addon panel with all the information mentioned before.

add the report to Appendices and reference it here.

To get some data that could be compared with the results from manual testing, I went through all the examples and extracted the following info for each component:

add the table to Appendices and reference it here

- **unique violations detected for component** - A list of unique violations visible in components Accessibility tab
- **report_violations** - Count of how many times the component was mentioned in the report that included all the components with all their examples. There might have been duplicates here when the same issue is visible in more than one example for the same component.
- **violations** - Count of all unique violations detected for the component. One violation reported in many examples was only counted once.
- **true_violations** - Count of how many of these violations are valid and relevant to this component. I went through all the examples with violations and validated if they were relevant. I excluded all violations that come for other components or are related to additional elements added just to illustrate the usage of the component better.
- **passes detected for component** - A list of unique passes visible in components Accessibility tab
- **passes** - Count of all unique passes detected for the component. One pass reported in many examples was only counted once.
- **true_passes** - Count of how many of these passes are valid and relevant to this component. I went through all the examples with violations and validated if they were relevant. I excluded all violations that come for other components or are related to additional elements added just to illustrate the usage of the component better.
- **explanation of false violation and passes** - An explanation of why each false violation and pass was not considered relevant
- **violations from manual testing for component** - A list of violations listed for each

component in the manual audit report. Left out issues that were detected using addon-a11y, because these are listed in automated testing.

- **manual_count** - Count of these manually found violations

3.4 Manual accessibility audit

The second part was conducting a manual accessibility audit of the same library to get a better overview of all the issues. This would also allow me to compare the results of the manual testing with the automatically generated test report to determine what are the tool's strengths and weaknesses and if testing isolated components poses any limitations.

3.4.1 Methodology for manual audit

We used Storybook to render an isolated preview of each component. The accessibility add-on had already been installed, and we looked at the violations reported there also. We worked in a team of 4 people - 3 designers and 1 developer. We created a task for each component - 53 tasks in total.

Everyone worked individually and each morning we had a quick meeting to discuss any questionable issues. Each evaluator checked violations in the accessibility add-on panel, using a keyboard, and also a screen reader to navigate. Each component had 1 or more examples and we looked at as many as we thought relevant to get a good overview.

At the beginning of the audit, we tested an add-on in Storybook for mocking a screen reader (Lara, n.d.). It had the option to show the output that the screen reader would play as audio, as text. Initially, it seemed like a convenient solution with rather reliable results, but further investigation revealed that the output was very different from what an actual screen reader would say.

For the rest of the audit, we used VoiceOver - the built-in screen reader for Apple products, because our work computers are MacBooks and VoiceOver was conveniently available for us to use. There was an initial learning curve, but after that, it went quite smoothly. All the components that had already been tested with the faulty add-on were reviewed again using VoiceOver.

The audit results were documented in a table. We used the barrier walkthrough method where the barriers we focused on were defined by how the user interacts with the webpage and looked at what issues different types of users might encounter. We separated them into 3 sections:

1. Mouse user issues - using a mouse to interact with the page
2. Keyboard user issues - using only a keyboard to do the same things

3. Screen reader user issues - trying to do the same things and getting the same information while only using a screen reader

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

The second type of user would encounter all the issues from the first category plus everything that is unusable for them by using a keyboard. We took the functionalities that the user should be able to use with a mouse and tried to use them with a keyboard only.

Mari-Ell: But it is OFTEN the case that screen reader works better than keyboard

The third user was imitated using a screen reader. The prerequisite for this was keyboard accessibility - if it was not usable with a keyboard then most likely it would not be usable by a screen reader.

In real life, these 3 types of users might not be so clearly separated and many issues 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 by solving mouse user issues, then keyboard user issues and then screen reader user issues.

4 Results

Follow up survey + reach out to devs I know have used addon-a11y

This section should have answers to all my research questions

Limitations of manual testing - resources, we only had this many people to perform the review

4.1 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.

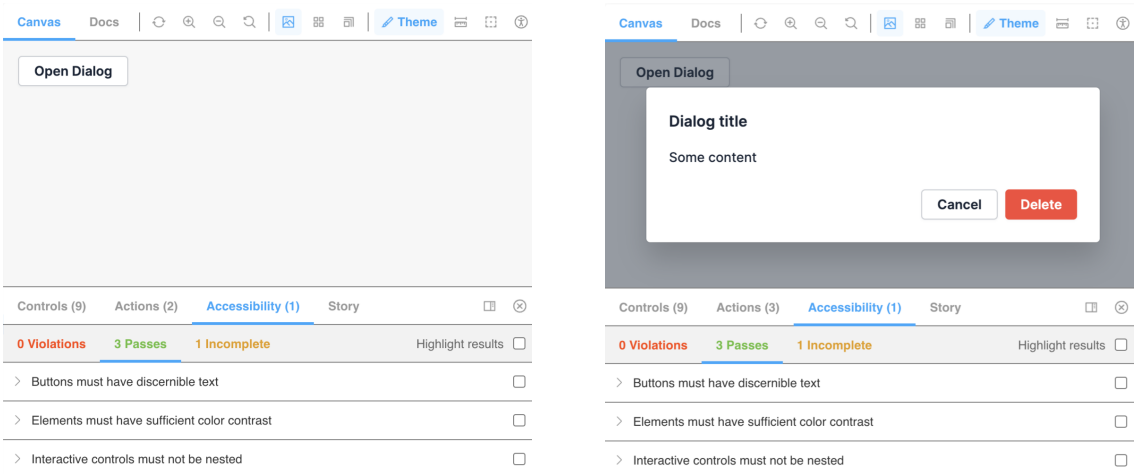
I prepared a comparison table from both results. For automated accessibility tests, I recorded the number of times it was mentioned in the report, the number of different violations and the number of passed checks for each component. In most cases, there were more examples with violations because the same thing was reported in more than one example (see figure 8a).

The next step was to go over all the violations as passes to verify if they are correct and relevant to the specific component. If a component uses another component inside it and that component has a violation or a pass reported then I did not count that.

Addon-a11y did not report any violations for 27 (51%) components out of 53 components after verifying the validity of these issues the number goes up to 31 (58%) (see Figure 8a). On average the automated testing tool found 1 violation per component. The maximum number of violations for a single component was 7 and the maximum number of valid violations was 6.

4 components out of 53 did not have any passed checks and 22 did not have any valid passed checks (see figure 8b). This means that the number of components with no valid passed checks changed from 4% to 42% after manually validating the results. The highest number of passes were detected for Form and Table component (18). After validating the number of passes changed to 9 for Table and 7 for Form. The average number of passes for each component was 5 before and 2 after validating the results.

Looking at all the fails and passes gives an overview of how many tests were run for each component. Out of 53 components, only 2 (4%) did not get checked by addon-a11y at all, but after validating the results this number changes to 20 (38%). This includes components that become visible only when triggered by another element, like modals and popups that are currently displayed with a button as a trigger in the Storybook examples (see Figure 6). These components are seen in figure 8b starting from *VideoOverlay* and ending with *Coachmark* - 11



(a) Inital view of example. This is tested by addon-a11y.

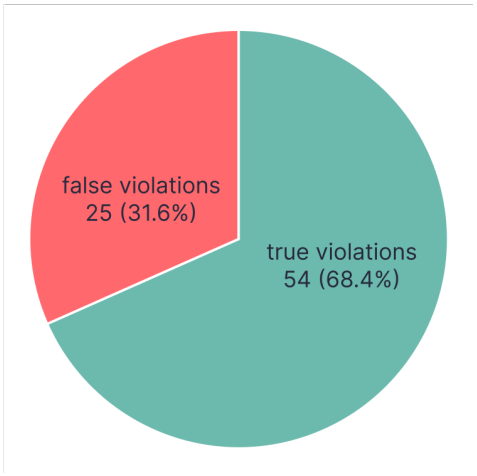
(b) Example after clicking the button and revealing the actual component.

Figure 6: Storybook example for Dialog component using button trigger

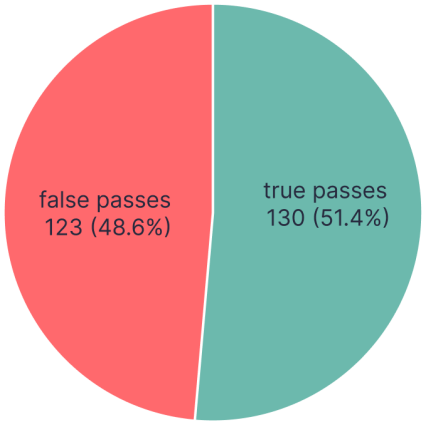
overall. The rest of the components that did not get tested by the addon included very simple components for spacing and visuals. These need further testing when they are being used in context to make sure that the visual info they are conveying is also included in text form. The results from manual testing did not reveal any further issues for 6 of these components. The components that have a trigger button in the example had the most additional issues. This is expected as due to the limitation of the examples the correct component was not evaluated by the tool. The remainder of the components had passed or failed issues, but often not both.

Analyzing the validity of checks performed by addon-a11y further shows that from all passes and fails together a bit more than half are valid while 68% of detected violations are correct and 51% of passed checks were confirmed to be relevant (see Figure 7).

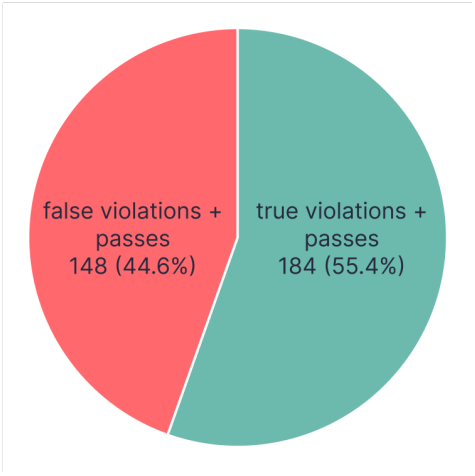
Mari-Ell: Do you have a solution on how to get them tested as well? Somehow display them without the need of the trigger? - Test out play functions with Storybook 7 and the built-in test runner.



(a) How many of the detected violations are valid?

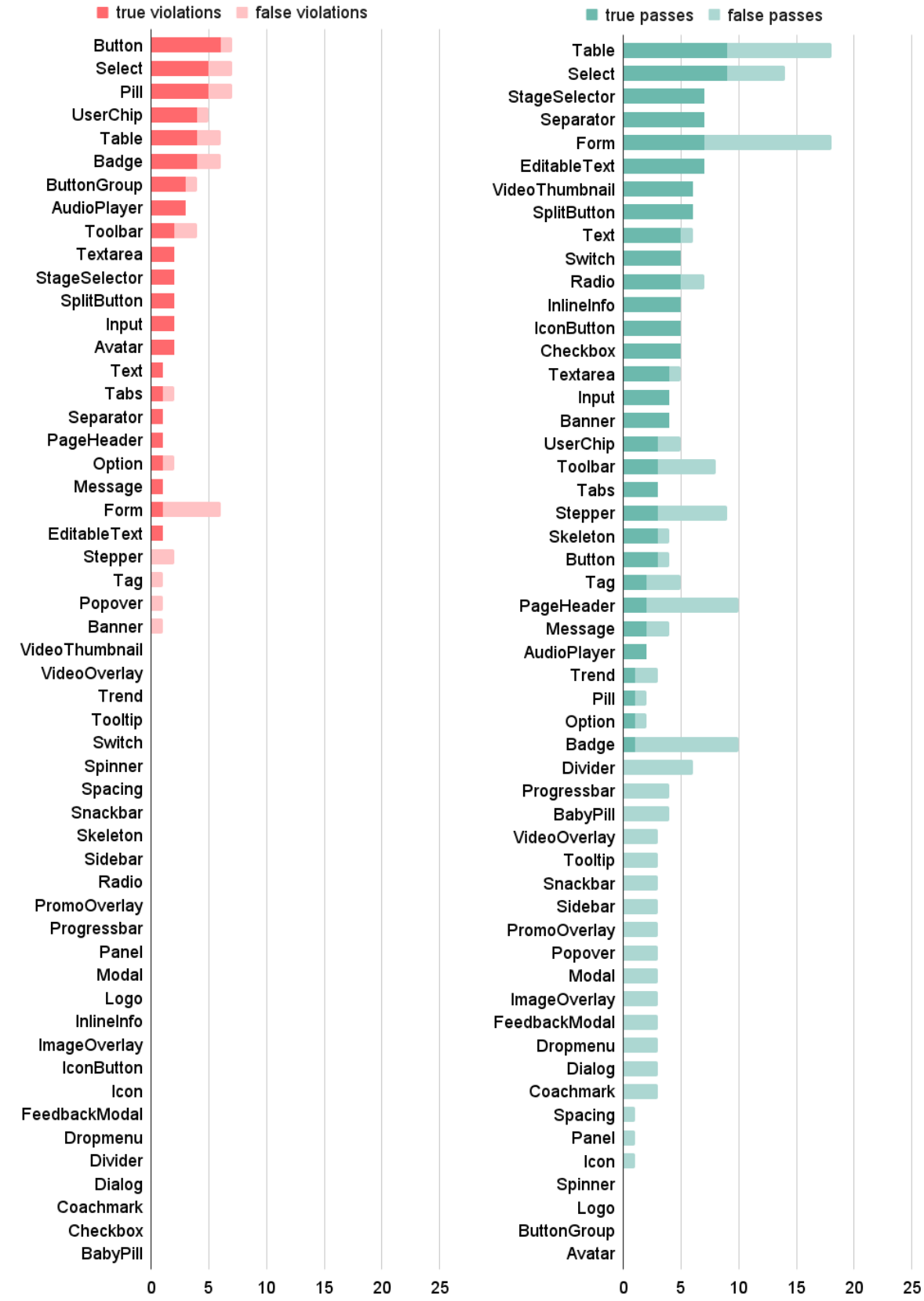


(b) How many of the checks passed are valid?



(c) How many of all tests are valid?

Figure 7: Valididty of tests performed by addon-a11y



(a) All violations reported by addon-a11y and how many of them are valid.

(b) All passed checks reported by addon-a11y and how many of them are valid.

Figure 8: All issues checked by addon-a11y

5 Discussion

5.1 Limitations of the study

5.1.1 Limitations Storybooks addon-a11y

The accessibility add-on in Storybook analyzes the examples that have been made for the component and unsuitable examples 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 viewed 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 the 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 our component library to that version would 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 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.

Reasons for automated check not being effective

5.2 Future research

1. Testing storybook 7 and its test runner.
2. Components that only show a button-play function + test-runner? Would it work?
3. Adding accessibility tests as a part of library setup - would that have a bigger impact

AI for accessibility evaluation

Conclusion

References

Abbott, C. (2021). Axe-core vs PA11Y. Retrieved 03/12/2023, from https://craigabbott.co.uk/blog/axe-core-vs-pa11y/

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

Accessibility Guidelines Working Group (AG WG) Participants (Ed.). (2022). *Wcag 2.1 understanding docs: Introduction to understanding wcag*. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/WCAG21/Understanding/intro#understanding-the-four-principles-of-accessibility

Adobe. (N.d.). *React Spectrum: A react implementation of spectrum, adobe's design system.* Retrieved 03/19/2023, from https://react-spectrum.adobe.com/react-spectrum/index.html

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

Brajnik, G. (2008). A comparative test of web accessibility evaluation methods. *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*, 113–120. https://doi.org/10.1145/1414471.1414494

Brajnik, G., Yesilada, Y., & Harper, S. (2011). The expertise effect on web accessibility evaluation methods. *Human-Computer Interaction*, *26*(3), 246–283. http://ezproxy.tlu.ee/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=65184804&site=eds-live

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

Churchill, E. F. (2019). Scaling ux with design systems. *Interactions*, *26*(5), 22–23. https://doi.org/10.1145/3352681

Coleman, T. (2017). *Component-Driven Development*. Chromatic. Retrieved 03/20/2023, from https://www.chromatic.com/blog/component-driven-development/

Collings, S. J., Honnibal, M., & Vanderwerff, P. (N.d.). *React-Bootstrap*. Retrieved 03/19/2023, from https://react-bootstrap.github.io/

Deque Systems. (2021). *The automated accessibility coverage report: Why we need to change how we viewaccessibility testing coverage.* (research rep.). Deque Systems. https://accessibility.deque.com/hubfs/Accessibility-Coverage-Report.pdf

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

Duran, M. (2017). *What we found when we tested tools on the world's least-accessible webpage - Accessibility in government*. Retrieved 03/18/2023, from https://accessibility.blog.gov.uk/2017/02/24/what-we-found-when-we-tested-tools-on-the-worlds-least-accessible-webpage/

Ella, E. (2019). A Guide to Component Driven Development (CDD). Retrieved 03/20/2023, from https://dev.to/giteden/a-guide-to-component-driven-development-cdd-1fo1

Fiers, W. (2017). *W3c standardized rules*. Deque Systems. Retrieved 04/02/2023, from https://github.com/dequelabs/axe-core/blob/develop/doc/act-rules-format.md

Fiers, W., Kraft, M., Mueller, M. J., & Abou-Zahra, S. (Eds.). (2019). *Accessibility Conformance Testing (ACT) Rules Format 1.0*. World Wide Web Consortium (W3C). Retrieved 03/12/2023, from https://www.w3.org/TR/act-rules-format/

Fiers, W., & Lambert, S. (2023). *Rule descriptions*. Deque Systems. Retrieved 04/02/2023, from https://github.com/dequelabs/axe-core/blob/develop/doc/rule-descriptions.md

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

GDS Accessibility Team. (2018). *How do automated accessibility checkers compare?* Government Digital Service. Retrieved 03/19/2023, from https://alphagov.github.io/accessibility-tool-audit/

Grammenos, S. (2020). *European comparative data on europe 2020 and persons with disabilities: Labour market, education, poverty and health analysis and trends: Labour market, education, poverty and health analysis and trends*. European Commission. https://doi.org/10.2767/745317

Henry, S. L. (Ed.). (2022). *Introduction to Web Accessibility*. Web Accessibility Initiative (WAI). Retrieved 12/26/2022, from https://www.w3.org/WAI/fundamentals/accessibility-intro/

Henry, S. L. (Ed.). (2023). *WCAG 2 Overview*. Web Accessibility Initiative (WAI). Retrieved 03/18/2023, from https://www.w3.org/WAI/standards-guidelines/wcag/

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley Professional.

Ismailova, R., & Inal, Y. (2022). Comparison of Online Accessibility Evaluation Tools: An Analysis of Tool Effectiveness. *IEEE Access, Access, IEEE*, *10*, 58233–58239. https://doi.org/10.1109/ACCESS.2022.3179375

Karube, K. (2020). *Storybook-a11y-report*. Retrieved 04/11/2023, from https://github.com/kalbeekatz/storybook-a11y-report

Kirkpatrick, A., O'Connor, J., Campbell, A., & Cooper, M. (Eds.). (2018). *Web content accessibility guidelines (wcag) 2.1* (2.1). Web Accessibility Initiative (WAI). Retrieved 03/18/2023, from https://www.w3.org/TR/WCAG21/

Lara, V. (N.d.). *Storybook screen reader addon*. Retrieved 04/04/2023, from https://storybook.js.org/addons/addon-screen-reader

Level Access, eSSENTIAL Accessibility, The Global Initiative for Inclusive, & International Association of Accessibility Professionals. (2023). *2022 state of digital accessibility*. Retrieved 03/04/2023, from https://www.levelaccess.com/earesources/state-of-digital-accessibility-report-2022/

Luft, J. (1.), Jeong, S., Idsardi, R., & Gardner, G. (4.) (2022). Literature reviews, theoretical frameworks, and conceptual frameworks: An introduction for new biology education researchers. *CBE life sciences education*, *21*(3), rm33. http://ezproxy.tlu.ee/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-85132952396&site=eds-live

Material UI SAS. (N.d.). *MUI: The React component library you always wanted*. Retrieved 03/19/2023, from https://mui.com/

MDN contributors. (2023). *Introduction to the DOM - Web APIs | MDN*. MDN Web Docs. Retrieved 03/19/2023, from https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Miesenberger, K., Edler, C., Dirks, S., Bühler, C., & Heumader, P. (2020). User centered design and user participation in inclusive r&d. In K. Miesenberger, R. Manduchi, M. Covarrubias Rodriguez, & P. Peňáz (Eds.), *Computers helping people with special needs* (pp. 3–9). Springer International Publishing.

Mueller, M. J., Jolly, R., & Eggert, E. (Eds.). (2018). *Web Accessibility Laws & Policies*. Web Accessibility Initiative (WAI). Retrieved 04/02/2023, from https://www.w3.org/WAI/policies/

Paternò, F., Pulina, F., Santoro, C., Gappa, H., & Mohamad, Y. (2020). Requirements for Large Scale Web Accessibility Evaluation. In K. Miesenberger, R. Manduchi, M. Covarrubias Rodriguez, & P. Peňáz (Eds.), *Computers Helping People with Special Needs* (pp. 275–283). Springer International Publishing. https://doi.org/10.1007/978-3-030-58796-3_33

Range, L. M. (2023). Case study methodologies. *Salem Press Encyclopedia of Health*. http://ezproxy.tlu.ee/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=93871826&site=eds-live

Rybin Koob, A., Ibacache Oliva, K. S., Williamson, M., Lamont-Manfre, M., Hugen, A., & Dickerson, A. (2022). Tech Tools in Pandemic-Transformed Information Literacy Instruction: Pushing for Digital Accessibility. *Information Technology & Libraries*, *41*(4), 1–32. https://doi.org/10.6017/ital.v41i4.15383

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

Siteimprove. (N.d.). *Overview of website accessibility laws and regulations*. Siteimprove. Retrieved 04/17/2023, from http://www.siteimprove.com/glossary/accessibility-laws/

Vanderheiden, G., Chisholm, W., & Jacobs, I. (Eds.). (1998). *Wai accessibility guidelines:page authoring*. Web Accessibility Initiative (WAI). Retrieved 04/05/2023, from https://www.w3.org/TR/1998/WD-WAI-PAGEAUTH-0203

Vigo, M., Brown, J., & Conway, V. (2013). Benchmarking web accessibility evaluation tools. *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*. https://doi.org/10.1145/2461121.2461124

World Health Organization. (2022). *Disability*. World Health Organisation. Retrieved 12/26/2022, from https://www.who.int/news-room/fact-sheets/detail/disability-and-health

World Wide Web Consortium. (1997). *World Wide Web Consortium Launches International Program Office for Web Accessibility Initiative*. Retrieved 12/26/2022, from https://www.w3.org/Press/IPO-announce

Yew, J., Convertino, G., Hamilton, A., & Churchill, E. (2020). Design systems: A community case study. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–8. https://doi.org/10.1145/3334480.3375204

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., & Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study [Publisher: IEEE]. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on*, 60–71. https://doi.org/10.1109/ASE.2017.8115619

Appendices

add automated test report, manual testing report and the comparison between them here

Component	fail	fail(valid)	diff	pass	pass(valid)	pass+fail	diff	manual
AudioPlayer	8	3	5	2	2	10	0	3
Avatar	2	2	0	0	0	2	0	0
BabyPill	0	0	0	4	0	4	4	1
Badge	5	4	1	10	1	15	9	0
Banner	1	0	1	4	4	5	0	0
Button	12	4	8	4	3	16	1	2
ButtonGroup	4	2	2	0	0	4	0	2
Checkbox	0	0	0	5	5	5	0	2
Coachmark	0	0	0	3	0	3	3	3
Dialog	0	0	0	3	0	3	3	3
Divider	0	0	0	6	0	6	6	0
Dropmenu	0	0	0	3	0	3	3	0
EditableText	2	1	1	7	7	9	0	1
FeedbackModal	0	0	0	3	0	3	3	1
Form	5	3	2	18	17	23	1	1
Icon	0	0	0	1	0	1	1	0
IconButton	0	0	0	5	5	5	0	2
ImageOverlay	0	0	0	3	0	3	3	4
InlineInfo	0	0	0	5	5	5	0	1