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

WAI-ARIA Accessible Rich Internet Applications

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. 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 (Kelsey-Adkins & Thompson, 2022; 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 accessibility checkers are intended to be used for the whole page or website. In the case study conducted during this research automatic accessibility tests will be run 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. Parallels can be drawn 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.

This thesis will explore the potential pros and cons of accessibility testing in the CI of a component library by trying out an automated accessibility evaluation tool in Pipedrive's 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 WCAG?

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

2 Theoretical Background

This chapter aims to provide a comprehensive summary of the concepts and principles that are relevant to this research. This includes an giving an overview of what is accessibility together with the various tools and methodologies used for evaluating it, the format used for defining accessibility standards and explaining the concept of component libraries their purpose and their importance in contemporary software engineering practices.

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. In addition, temporary disabilities should also be mentioned - 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 needs 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 most commonly used standard for considering people with different abilities and making web content accessible for different ways of consuming is 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. This means version 2.2 will not be entirely backward compatible with the previous versions.

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 the regulations there are most relevant to this 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). All evaluations in this thesis will be conducted against WCAG 2.1 because it is the latest published version of the most widely used standard.

WCAG is intended for anyone who is involved in developing 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.

Manual testing with experts or users is essential to making websites truly accessible to everyone. Currently there is no tool available that can replace the human evaluation of many WCAG success criteria, like how well the image's text alternative conveys the picture meaning or if an input field's error messages clearly suggest what needs to be corrected.

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. These tools 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. Some of these 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 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*
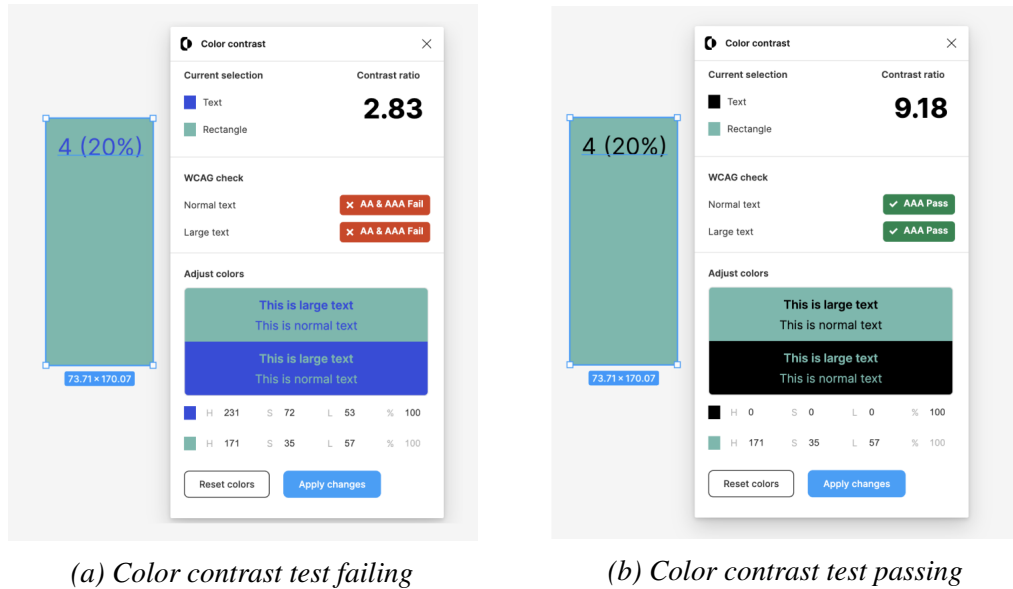
Figure 1: Figma plugin for testing color contrast compliance with WCAG

Authoring tool plugins can be for example plugins for Figma that check if the colors used in the design conform with WCAG requirements (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.

CLI tools 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 and in most tools a link to the relevant standard or rule it violated.

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% of organizations 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. Deques results show that 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 that the different 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.

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). To understand this we need to look at how they define and measure coverage. 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. Deque Systems suggests that the coverage should be the percentage of all the issues found on a site. A study where 2000 audit results from testing pages with axe-core and manual testing methodology were compared was conducted to prove the validity of this way of calculating the coverage of accessibility tests(Deque Systems, 2021). The average percentage of the number of issues detected by axe-core for each data set is 57.38%.

In the report, they also state 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). No studies were found about other tools that would have used the same method for calculating the coverage so in this paper the most widely used coverage calculation method will be used when making comparisons between different accessibility evaluation tools.

There is still a limit to what can be detected no matter how we define the coverage. There are aspects of accessibility that need human interpretation. The tools that are currently available can't interpret test to understand how well it conveys the meaning of images or even if the image on the webpage is purely decorative or a part of meaningful content on the page. The goal of using automated testing tools should be to help make the whole process easier, by catching automatically testable problems with tools. This gives the human evaluator more time to focus on the remaining issues.

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

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

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 the purpose of this research. 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 and announced in relevant channels to raise awareness about it.

In parallel to observing how the tool was being used a manual accessibility audit was conducted with a team of 3 designers and 1 developer on the same component library. These results were then compared with the automated testing results that were obtained from testing the same version of the library with an automated accessibility tool. As the last step, a questionnaire was sent out to gather information about how the tool had been adopted. To get more details some developers and designers that were known to have used the tool during that period were interviewed.

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. Pipedive has 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.

This chapter gives an overview of the research methodologies that were used to conduct this research and explains the reasoning behind choosing them and how they were employed.

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).

A literature review was conducted as a part of this research to understand the current landscape of automated accessibility testing. These kinds of tools have been available for a long time and the literature review part of this thesis intended to understand how much is known 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?

The 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. A similar search was conducted on Google to find any nonscientific articles about the subject. The references in all of these articles were explored to identify any relevant research that should also be included.

This produced sufficiently extensive results and very soon the conclusions and references in the articles started to 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, articles that were published in the last 10 years were focused on and older ones were only looked at when they gave more high-level overviews or methods and did not focus on comparing specific tools.

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

Many studies (Alsaeedi, 2020; Duran, 2017; Ismailova & Inal, 2022; Rybin Koob et al., 2022; Sane, 2021; Vigo et al., 2013) have been done to compare different accessibility testing tools using various methods. The results of these studies were looked at and enough tools were assessed as a part of the exploratory phase to find one that would be usable for this case study, but the intention was not to compare them methodically as a part of this work. The comparisons that have been made by others in the past will be used as a foundation to decide what tools to use.

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, but the richness of detail they provide makes them fascinating and often there is a lot to learn from them (Range, 2023). Sometimes these insights can be applied to other similar cases. It is a good method for exploratory or critical and unusual cases.

This method was chosen to explore the possibilities of automated accessibility evaluation. An organizational case study was conducted that focused on the process of evaluating and improving the accessibility of CRM tool called Pipedrive. In the scope of this study, an automated accessibility evaluation tool was added to the component library of Pipedrive and a manual accessibility evaluation of the same library was conducted with a team of 3 designers and 1 developer working in the company.

Before implementing the new tool a questionnaire was sent out 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 the case study another questionnaire was sent out with a focus on gaining information from people in the organization who use the tool that was added during that period. The aim of this was to understand if the tool 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.

The data collected from automated and manual testing was organized to make it comparable. Statistical analysis was used 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

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

As the first step of the research, a survey was sent out in Pipedive's Slack channels (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, the component library and who are interested in accessibility (Table 1).
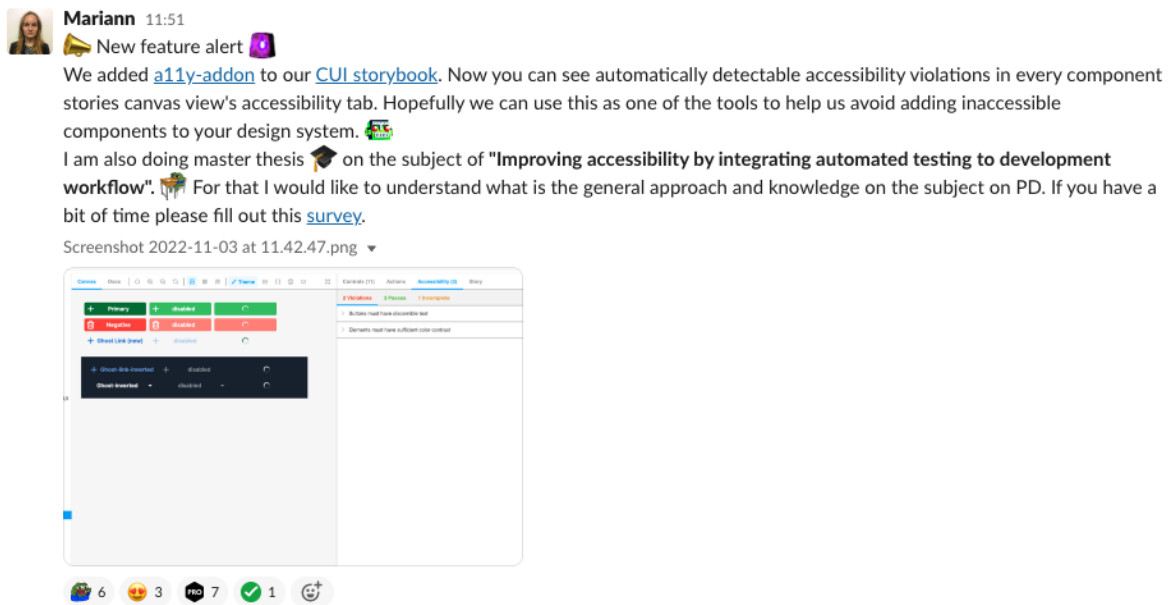
Figure 2: Message in company Slack announcing adding accessibility tool and inviting people to reply to the survey.

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

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

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 (Appendix A.1).

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 are interested in accessibility were more likely to respond.

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 and none of the responders judge their knowledge about accessibility to be very good (Figure 3a). 35% of people who participated in the survey know where to find resources about accessibility standards, 15% don't know and 50% know, but think they need more (Figure 3b).
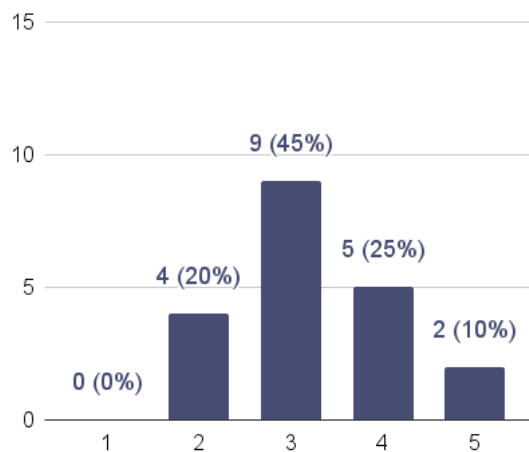
(a) Pre-case study survey: How much do you know about accessibility (standards, best practices, importance, testing)? 1- "Almost no knowledge" to 5 - "I have very good knowledge"
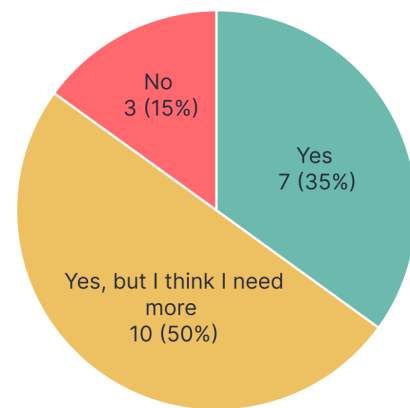
(b) Do you know where to find resources about accessibility standards? (no one chose the 4th option "No I don't think I need them")

Figure 3: Accessibility knowledge in Pipedrive

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 that Pipedrive should be putting more focus on following common accessibility standards (Figure 4). They elaborate a bit more about what they think are the reasons behind this in the free text answers to questions 4 and 6.

According to these replies, accessibility has always been at the end of Pipedrive's 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 this subject 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. All replies can be seen in Appendix B.2.
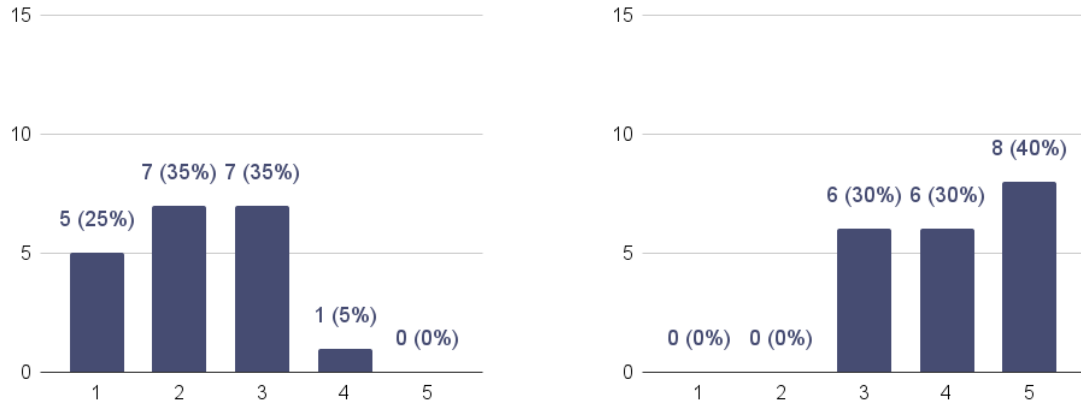
(a) Do you think accessibility is a priority in our company? 1-No not at all to 5-Yes very much

(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

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 the evidence would suggest that 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 Testing to Pipedrive's Component Library

The next step was adding some automated tests to Pipedrive's component library development workflow and observing their usefulness. The aim was to find something that can be integrated easily and that does not have a steep learning curve or add unnecessary complexity to the development workflow. Two open-source CLI tools that were mentioned in the previous research were tested to see if they would be suitable.

Pa11y CI is an accessibility test runner that is focused on CI environments (Team Pa11y, 2022). It is said to work very well in CI. The tool supports WCAG 2.0 A, AA and AAA standards and the tests can be run on multiple URLs. Axe-core is an accessibility testing engine for websites and other HTML-based user interfaces (Deque Systems, 2023). It supports WCAG 2.0 and 2.1 on levels A, AA, AAA, but needs a bit more initial setup to start testing webpages.

Storybook is used as the component explorer for our reusable component library's 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. The developers in Pipedrive use it to preview the components they are developing locally. A version of Storybook is also published internally with all the components available for anyone in the company to see.

To run these tests with either tool the components first need to be rendered in a browser. This can be done by using Storybook. 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 that to test the isolated components with pa11y CI or axe-core Storybook's UI that gets rendered around the component needs to be excluded so we can run the tests on only the actual relevant example.

Both of these tools are more suitable for testing whole web pages. Testing isolated components with the current setup would have required some extra steps. First, a list of all the examples for each component together with their URLs would need to be obtained in a way that it can be kept up to date every time changes are made. The next challenge would have been to isolate the components in Storybook UI to test only them and not the UI around them. This all seemed like an extensive thing to build and there probably were better more suitable solutions available already, because component libraries and using Storybook for developing them is quite common.

Storybook has a wide ecosystem of add-ons and one of them is addon-a11y. It uses the same accessibility testing engine, axe-core to audit the HTML rendered for an isolated component (Chromatic, n.d.-a). The issues that were mentioned before have already been solved here. The add-on tests only the relevant part of the HTML and also provides a nice UI inside Storybook (Figure 5).

As mentioned before axe-core is an accessibility testing engine for websites and other HTML-based user interfaces that supports WCAG 2.0 and 2.1 on levels A and AA (Deque Systems, 2023). 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 Continuous Accessibility Evaluation chapter.

This seemed like the best solution in our situation so addon-a11y was added to our component library's Storybook. The accessibility tab is visible in 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.
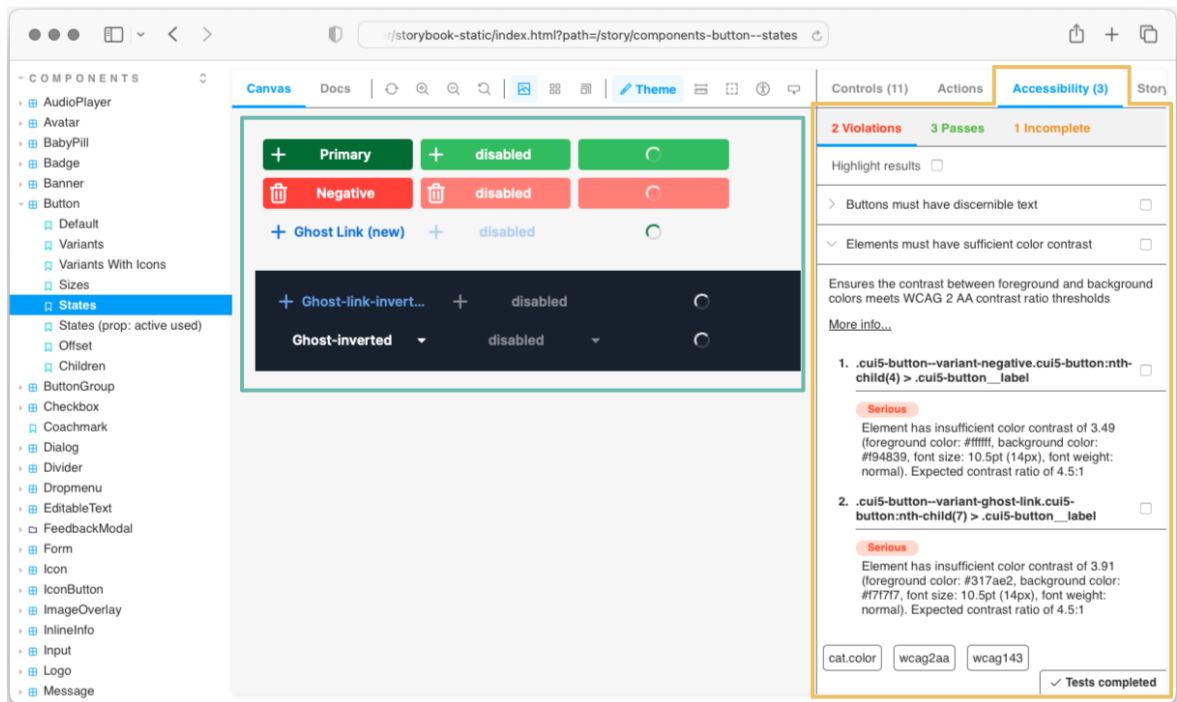
Figure 5: Accessibility add-on (addon-a11y) in Storybook. The accessibility panel is highlighted in yellow and the component example (story) is highlighted in green.

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 Success Criteria 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).

The addon in Storybook provided a good way to test all the examples, but no way to get an overview of how many issues we have in the whole library. A summary of all the potential accessibility problems was needed to evaluate the effort that would be needed to make the components provided by this library accessible and to measure the impact of making improvements there.

To get an overview of all the issues Storybook-a11y-report (Karube, 2020) was used to generate a report of all the violations in the whole component library. This CLI tool is meant to be used together with addon-a11y and it goes through all the examples and runs 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 (Appendix C).

To get some data that could be compared with the results from manual testing, all the examples were examined in detail and the following data were extracted for each component (Appendix E):

- **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. All the violations in each component example were looked at to decide if they were relevant and valid. All violations that come for other components or were related to elements added just to illustrate the usage of the component were excluded.

- **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. All the passes in each component example were looked at to decide if they were relevant and valid. All passes that come for other components or were related to elements added just to illustrate the usage of the component were excluded.

- **explanation of false violations 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

At the same time, a very simple component library, brick-ui was set up to test anything that might not be very easy to try out in a real component library that is widely used and includes more than 50 components. This library was used as a quick playground to test out ideas and it provided an easy way to see if the latest version of Storybook which was still in beta at the beginning of the case study would have solved some of the issues that were found. Continuous improvements are being made in both Storybook and axe-core through the maintainers and active communities around them. Brick-ui provided an opportunity to set up simple examples to illustrate some questions that came up while setting up the tooling. Examples from Pipedrie's component library could not be shared, because the code is not

publicly available. As a result, Storybook's maintainers advised to use pseudo states addon for testing different states of components and a bug that was found during testing could be reported to axe-core maintainers.

While this research was going on another new component library development started in Pipedrive and because the latest versions of Storybook had been explored in the simple component library the team had the confidence to use this prerelease version of the library as the component explorer of this new library. This provided some invaluable insight into an improved automated accessibility testing workflow inside Storybook and also an opportunity to observe what benefits could setting up automated accessibility tests from the start of development have on the final result. In this scenario, there would be no need to fix already existing issues, but rather the developers would be able to avoid them from the start.

3.4   Manual Accessibility Audit

The next part was conducting a manual accessibility audit of the same library to get a better overview of all the issues. This approach will enable a comparative analysis of the results of the manual testing and the automatically generated test report to determine what are the tool's strengths and weaknesses and if testing isolated components poses any limitations.

Storybook was used to render an isolated preview of each component. The accessibility add-on had already been installed, and all the violations reported there were also explored. The team that conducted the audit consisted of 4 people - 3 designers and 1 developer. There was one designer in the team who considered himself to be an accessibility expert and the rest had a lower level of knowledge. Brajnik et al. warned about a significant drop in validity when testing with non-experts compared to experts (Brajnik et al., 2011). According to him, at least 3 expert evaluators should be used to get reliable results and when the evaluators are not experts then there should be at least 14 to reach the same quality in results. The audit conducted as a part of this research had 4 evaluators and not all of them can be considered experts, but they all had a basic level of knowledge in the field of accessibility. Daily discussions and 2 longer review sessions were carried out to mitigate the lack of experience and ensure that the results from the audit are as good as possible.

Tasks were created for each component - 53 tasks in total. Everyone tested the components individually and wrote down any issues they found in a shared table. Each morning the team had a meeting to discuss any questionable issues and to share interesting insights. 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 the team members looked over as many of them as they thought was necessary to get an extensive overview of that specific component.

At the beginning of the audit, an add-on in Storybook for mocking a screen reader (Lara,

n.d.) was used. It showed the output that a screen reader would play as audio, in text form in the add-ons panel in Storybook. 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.

During the rest of the audit, VoiceOver - the built-in screen reader for Apple products was used 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. The barrier walkthrough method where the barriers that were focused on were defined by how the user interacts with the webpage was used. Evaluators looked at what issues different types of users might encounter with each component and separated the results 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

This separation was seen as a good way to prioritize fixing the issues in the future. The mouse user is the user who is currently considered in all of our development. 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. The functionalities that the user should be able to use with a mouse were explored by using only the 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 it might not be usable by a screen reader. Some components actually worked better with a screen reader. Screen readers have more possibilities for navigating through elements and nested elements and this helps access come nested focusable elements that were unreachable by using regular keyboard navigation.

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, the issues needed to be prioritized 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

4.1 Results from Manual Audit and Automated Tests

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

A summarized table from all violations and passes in automated tests and issues found in the manual audit was created. In this table, the results have been reduced to counts that can be analyzed using statistical methods (Appendix E).

The results show 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%) (Figure 7a). On average the automated testing tool found 1 violation per component. The maximum number of violations reported 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 7b). 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* components (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.
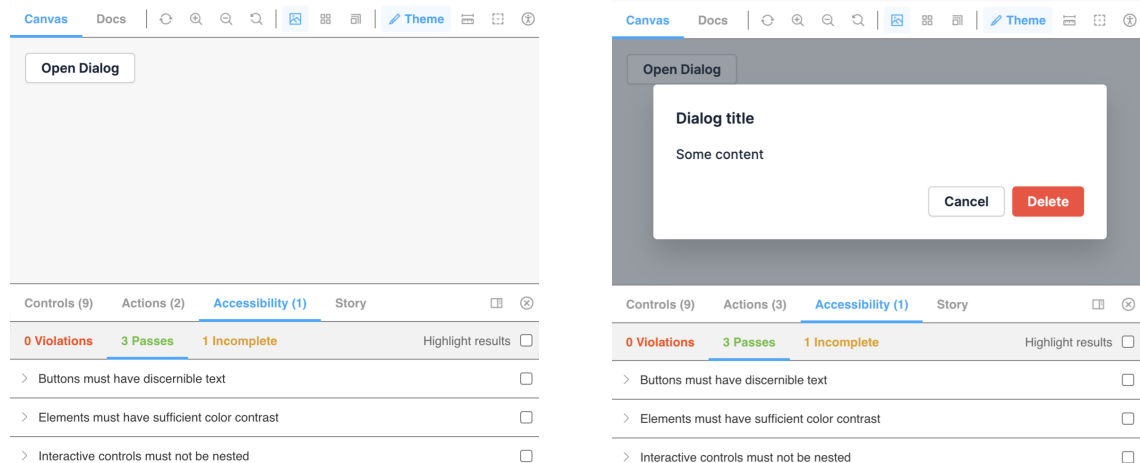
(a) Inital view of example. This is tested by addon-a11y.

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

Figure 6: Storybook example for Dialog component using button trigger
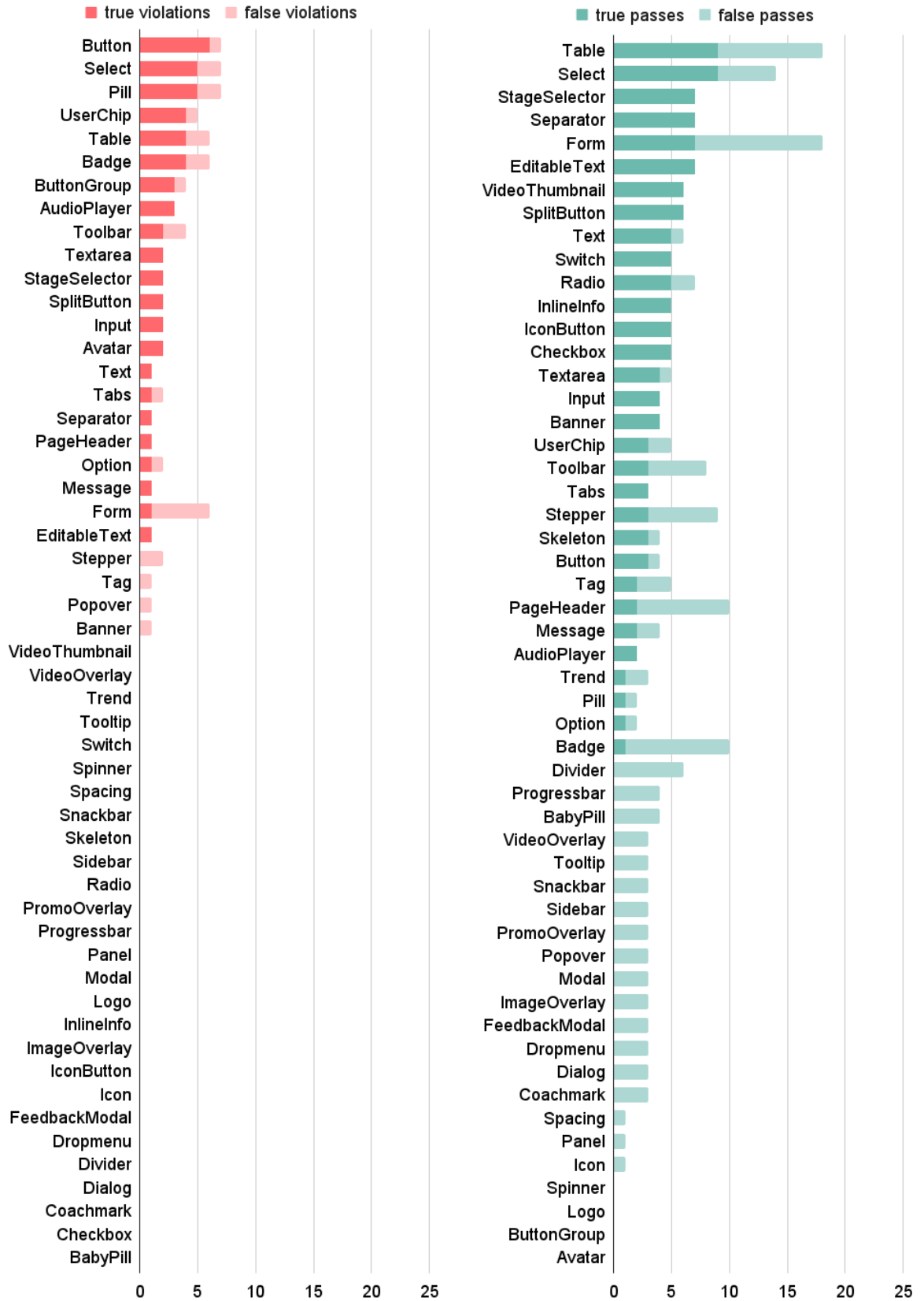
(a) All violations reported by addon-a11y and how many of them are valid.

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

Figure 7: All issues checked by addon-a11y

Looking at all the fails and passes gave a good overview of how many tests were run for each component. Out of 53 components that were tested, only 2 (4%) did not get checked by addon-a11y at all, but some of these included false violations or passes. After validating the results the number of components not tested is 20 (38%). This included components that become visible only when triggered by another element, like modals and popups that were displayed with a button as a trigger in the Storybook examples (Figure 6). In total, there were 11 components with examples like that and they can be seen in Figure 7b starting from *VideoOverlay* and ending with *Coachmark*.

The remainder 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.

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 (Figure 8).

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

(b) How many of the passes are valid?

(c) How many of all violations and tests combined are valid?
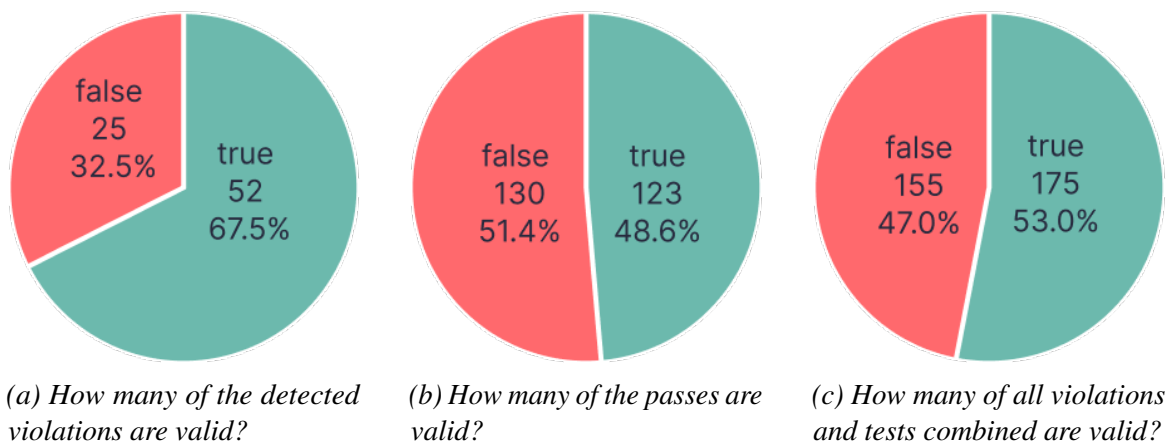
Figure 8: Valididty of tests performed by addon-a11y

RQ2: "What kind of errors can be caught by running automated accessibility tests on a component library?" can be answered based on these results and the results from manually evaluating the same components. The WCAG Success Criterion, correct usage of Accessible Rich Internet Applications (WAI-ARIA) and other best practices that proved to be reliably testable in Pipedrive's component library are:

- Success Criterion 1.1.1 Non-text Content
    - Images must have alternate text

- Success Criterion 1.3.1 Info and Relationships

  - Certain ARIA roles must contain particular children

- Success Criterion 1.4.3 Contrast (Minimum)

- Success Criterion 1.4.6 Contrast (Enhanced)

  - Color contrast

- Success Criterion 1.4.12 Text Spacing

  - Inline text spacing must be adjustable with custom stylesheets

- Success Criterion 2.1.1 Keyboard

- Success Criterion 2.1.3 Keyboard (No Exception)

  - Scrollable region must have keyboard access

- Success Criterion 2.4.4 Link Purpose (In Context)

- Success Criterion 2.4.9 Link Purpose (Link Only)

  - Links with the same name must have a similar purpose

  - Links must have discernible text

- Success Criterion 4.1.1 Parsing

  - IDs used in ARIA and labels must be unique

- Success Criterion 4.1.2 Name, Role, Value

  - Buttons must have discernible text

  - Form elements must have labels

  - Form field must not have multiple label elements

  - Form elements should have a visible label

  - ARIA input fields must have an accessible name

  - ARIA roles used must conform to valid values

  - ARIA commands must have an accessible name

  - Interactive controls must not be nested

- Correct usage of WAI-ARIA

  - ARIA attributes must conform to valid values

  - ARIA attributes must conform to valid names

- – Elements must only use allowed ARIA attributes

- – Required ARIA attributes must be provided

- Best Practices Rules

  - – ARIA role should be appropriate for the element

  - – Elements should not have tab index greater than zero

  - – Alternative text of images should not be repeated as text

  - – Headings should not be empty

  - – Heading levels should only increase by one

There might be other WCAG Success Criteria that the tool is also capable of testing, but that just was not represented by this certain set of components and examples. The most common valid test result was color contrast with 45,1% of all violations and passes being color contrast issues. It is also important to note that this tool can reliably test for the existence of the alt attribute in an image, but it is not capable of judging how accurate the meaning is or even if the image is purely decorative and should have an empty alt value.

From the additional issues found in the manual audit, it could be concluded that the following issues should be manually tested:

- Focus states

- Navigating by using a keyboard

- Navigating and interacting by using a screen reader

- Recognizable color differences between semantically different variants of one component

- Images and what should be the value of their alt text

- Svgs and if they should have an aria-label

Evaluating the accessibility of focus states with automated testing could potentially be improved by using pseudo states add-on and writing better examples for components that would also display it in all relevant focus states.

Inaccurate results and missed issues indicated problems with how the component examples have been written. The following things could be improved to get better results from automated testing:

- variants of components that are meant to be used on dark backgrounds should always be displayed on a dark background

- Examples where the component gets triggered by a button should be avoided when possible or made visible to the testing tool by writing interactions

- pseudo-classes (like hover, focus etc) of interactive elements should be displayed in examples

- Examples should be simple and display one component at a time to reduce the number of irrelevant tests being run

4.2 How Much Was Compliance with WCAG 2.1 Improved?

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

One of the benefits of using automated testing tools is improved observability of the state of accessibility. Multiple reports were generated in Pipedrive's component library during the time that the usage of the automated testing tool was observed. The report counts different issues and the for the purpose of this study and for observing the overall process that was made over time the whole number of detected violations were compared. (Figure 9).
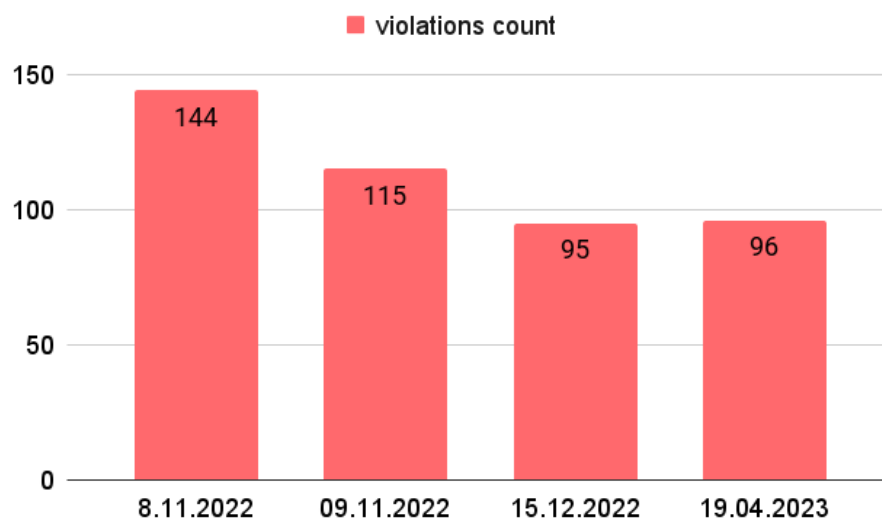
Figure 9: How much have automated audit report results improved since adding the tool?

Results over 5 months show that we reduced the highest number of accessibility issues while and just after conducting the manual accessibility audit. This logical because the team involved in performing the audit also had dedicated time to focus on solving these issues. It is also worth noting that several new components have been added to the library during this time and the number of accessibility issues has not increased.

As discussed in previous chapters automated accessibility can't catch all issues. So the fact that almost no additional violations have been added alone can't provide certainty that this

component library is just as accessible as it was the last time the report was generated. It could be argued that the biggest value of using this in addition to finding certain kinds of accessibility violations has been bringing focus and attention to the subject of accessibility.

Paternò et al. said that developers find maintaining a high level of accessibility hard because it is then only considered after the first release of the product. Having a continuous way to monitor and detect issues should help keep the level that was reached in the phase where there was more time to focus on accessibility. The data collected in the short time that this automated accessibility tool has been in use in Pipedrive supports that theory.

A good level of accessibility is easier to achieve when developers consider it from the start of the process. Fixing things, later on, is much harder to justify on the business side and might be more difficult than it would have been to build a component or a webpage keeping accessibility in mind from the beginning.

With the data collected during this research, I can definitively say that implementing automated accessibility testing in a component library's development workflow will improve the accessibility of the end product where these elements are being used, but there is data to support that it is very likely that it could improve it to some extent.

The experience of using these tools and conversations with other developers in the company has suggested that one big benefit of using tools like this is that it makes the whole subject of web accessibility a bit more measurable and less daunting. Having an easy familiar way of testing for accessibility violations together with links to resources that help in solving these issues will make it more likely for developers to take on solving these issues.

4.3 Limitations of Automated Accessibility Testing

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

Follow up survey + reach out to devs I know have used addon-a11y, button issue, not possible to add to CI currently - easy to not notice

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 be solved with better examples. Brick-ui, the simple playgound for testing solutions in Storybook was used to explore different way of mitigating this. This revealed that these kinds of examples could be improved in Storybook 7 by adding a user interaction that opens the component, because accessibility tests are run after this interaction has been executed. In

Storybook 6 the same thing could be achived by opening the example manually and triggerig a rerun of the accessibility checks. It is important to make sure that the opened component is rendered inside the div element with id="root" (or "storybook-root" in version 7).

Another 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. Ways of automating this were explored 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 where accessibility test can be added to the built in test-runner.

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 Discussion

5.1 Limitations of the study

The tool that was used was chosen with the limitation of our current tooling. There are plenty of other options out there. Some free some paid and some of them might be more effective than the one we decided to use.

We had limited time. The tool was in use for a bit more than 5 months. It will continue to be used and it would be interesting to see how well it performs long term.

Manual accessibility audit might have not had enough highly qualified accessibility experts on the team.

5.2 Future research

Storybook 6 was used in Pipedrive's component library during the case study, but it seems that the next major version of this component exploer provides improved tools for testing. At the time of the study some of these possibilities where tested to get and overview of what might be possible. The new major verison has a feature to add play-functions or interactions to each story (example). These interactions are then run as unit tests. The asseccibility add-on works the same way and it it important to note that these accessibility tests will be run after the interaction finishes. This means that combining the add-on with interactions would allow to test some states that where hidden before. For example it would be possible to add an user interaction to press a button for the 11 examples that had a button trigger and where the real component did not get tested. This scenario was tried out in the example component library and it works.

Another added benefit that Storybook 7 provides is a test-runner is meant for running the initeractions that can be added to each story as unit tests. Accessibility test can be easily added to this globally without needing to do any extra steps in every component story. This will make it possible to run the same accessibility tests that can be seen in the add-ons panel in CLI and to CI testing phase.

Adding accessibility tests from the beginning - would that have a bigger impact?

Possibilities in SB 7 could be explored, test runner would even alllow to use a different accessibility testing engine

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 view accessibility 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

Kelsey-Adkins, E. R., & Thompson, R. H. (2022). Inter-rater reliability of command-line web accessibility
evaluation tools. *Proceedings of the 24th International ACM SIGACCESS Conference on Computers
and Accessibility*. https://doi.org/10.1145/3517428.3550395

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

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

Kelsey-Adkins, E. R., & Thompson, R. H. (2022). Inter-rater reliability of command-line web accessibility
evaluation tools. *Proceedings of the 24th International ACM SIGACCESS Conference on Computers
and Accessibility*. https://doi.org/10.1145/3517428.3550395

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/

Team Pa11y. (2022). *Pa11y CI* [ZSCC: NoCitationData[s0] original-date: 2016-06-02T21:07:27Z]. Retrieved 04/22/2023, from https://github.com/pa11y/pa11y-ci

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

A   Pre-case study questionnaire

A.1   Pre-case study questionnaire questions

A.2   Pre-case study questionnaire responses

B Post-case study questionnaire

B.1 Post-case study questionnaire questions

B.2 Post-case study questionnaire responses

C   Accessibility report

D   Manual Accessibility audit results

E   Table of summarized accessibility test results

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