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 WAS

Tallinn 2023

Abstract

Acknowledgements

Table of Contents

List of Figures

List of Tables

List of Abbreviation

ACT Accessibility Conformance Testing

AI artificial intelligence

API application programming interfaces

CDD Component Driven Development

CI Continuous Integration

CLI command-line interface

CRM Customer Relationship Management

DOM Document Object Model

HTML HyperText Markup Language

IAAP International Association of Accessibility Professionals

IAAP WAS International Association of Accessibility Professionals Web Accessibility Specialist

ML machine learning

SVG Scalable Vector Graphics

UI User Interface

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

According to Internet World Stats 67.9% of the global world population uses the internet (Miniwatts Marketing Group, 2023) and a big part of these people might face difficulties accessing the content that is provided on the web. Web accessibility focuses on making the web equally accessible to everyone.

The most commonly used standard for making webpages accessible is the Web Content Accessibility Guidelines Web Content Accessibility Guidelines (WCAG), and there are various methods and tools available for testing web content's compliance with it. One such method is automated testing. The aim of this research is to investigate the limitations of automated accessibility testing when evaluating a component library's compliance with common accessibility standards.

To investigate this, a case study and manual accessibility audit will be conducted in Pipedrive, a software company that develops a web-based sales Customer Relationship Management (CRM) and the workplace of the author of this thesis. The case study will involve integrating an automated accessibility testing tool into the company's component library and observing its usage. This will provide valuable insights into the practical implications of using automated accessibility testing tools in a real-world setting.

The "Introduction" chapter will outline the research problem and objectives, and conclude by stating the research questions that this thesis aims to answer. The subsequent chapter, "Theoretical Background", will provide context for the research by exploring the meaning of web accessibility, the common standards and regulations for achieving it, and the various methods and tools for evaluating it. Since this study focuses on accessibility evaluations in a component library, the final section of this chapter will explain the concept, usage, and importance of component libraries in web development.

The "Research" chapter will begin by describing the methodology used in this study, followed by an explanation of the three main components: a survey conducted at Pipedrive to gather insights into the current level of knowledge and awareness around accessibility, an exploratory case study on the usage of automated accessibility testing in Pipedrive's component library, and an overview of a manual accessibility audit on the same component library. The "Results" chapter will analyze the findings of this research to understand the capabilities of automated accessibility testing tools and to find answers to the research questions posed earlier.

The final chapter, "Discussion", will focus on exploring the limitations of this research and possibilities for future research on similar or related subjects. It will also explain the contributions of this thesis to the research of automated accessibility testing. The chapter will conclude with a summary of the key findings and a reevaluation of the statements proposed at the beginning of this paper.

1.1 Research Problem and Goal

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 also 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 designed to be used for the entire website or page. However, in the case study conducted during this research, automatic accessibility tests were run on the individual components of a component library, as this is a crucial aspect of most Design Systems (Yew et al., 2020). While certain aspects of accessibility, such as the proper usage of headings or the page language, can only be evaluated within the context of the entire page, there is still much to gain from this type of testing.

Parallels can be drawn between isolated component testing and unit tests in software development. The goal of unit tests is to verify that small units of code function properly (Humble & Farley, 2010, p.60). Similarly, accessibility testing of components 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

To address the research goals of this thesis, stated above, four research questions have been formulated.

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?

The first research question will be addressed by conducting a survey in Pipedrive. The remaining three research questions will be answered by analyzing the results of the case study and manual accessibility audit.

2 Theoretical Background

This chapter aims to provide a comprehensive summary of the concepts and principles that are relevant to this research. This includes giving an overview of accessibility, including the tools and methodologies used to evaluate it, the format for defining testable accessibility rules, and 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 approximately 7% of people aged 16 and over living in the EU have a severe disability, and about 17.5% have a moderate disability (Grammenos, 2020). This number gets significantly higher as people get older. Disability prevalence among people aged 65 and over is about 47.8%. Disability is a part of being human and people with disabilities have very different needs. In addition, temporary disabilities should also be considered - these might include 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 has 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 or devices (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.

Discrimination against disabled people is addressed in various laws and directives around the world. Many of them are based on or derived from some version of WCAG. Pipedrive operates in the United States and Europe, making these regulations most relevant to this case study.

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

European Union has two directives that outline what accessibility should look like and leave it to each member state to make national laws based on that. European Union Web Accessibility Directive states that the public sector should follow EN 301 549 which includes WCAG 2.1 Level AA and European Accessibility Act aims to ensure that essential products and services traded in and between European Union member states are equally accessible to everyone (Siteimprove, n.d.).

WAI gives an overview of the laws and policies around the world. 25 out of 40 laws and policies listed on their webpage are based on WCAG 2.0 or WCAG 2.0 derivative (Mueller et al., 2018). This thesis will use WCAG 2.1 as the basis for all evaluations, given that it is the latest published version of the most commonly used accessibility 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 should be able to 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 a 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. In the case of an atomic rule, this might be an HTML tag name, computed role, or distance between two elements. For composite rules, applicability is determined by a union of the atomic elements that it combines.

ACT Rules can be implemented in automated test tools and test methodologies. Tools and methodologies based on the same set of rules will produce (largely) the same results. Each rule includes a number of test cases that are used to check if an implementation is correct. The ACT Rules Community keeps a list of implementations that are actively implementing ACT Rules. By comparing how many implementations rules have, we work out which rules are widely agreed upon, and which ones need further discussion.

According to The ACT Rules Community website, 8 organizations have implemented these rules in their automated accessibility testing tools or test methodologies. Among these are developers of widely used accessibility testing tools - Deque Systems, IBM Accessibility, Level Access and Siteimprove.

2.2 Accessibility Evaluation

Different evaluation methods are used to determine if a website, digital document 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 the most widely used method (Brajnik, 2008). It involves analyzing web content to verify whether it meets a list of criteria. The results depend on the evaluator's opinions and 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 in achieving their goal on the website (Brajnik, 2008). An accessibility barrier can be any condition that makes it difficult for a person 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 by 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.

Choosing a suitable method 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 these 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, 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.

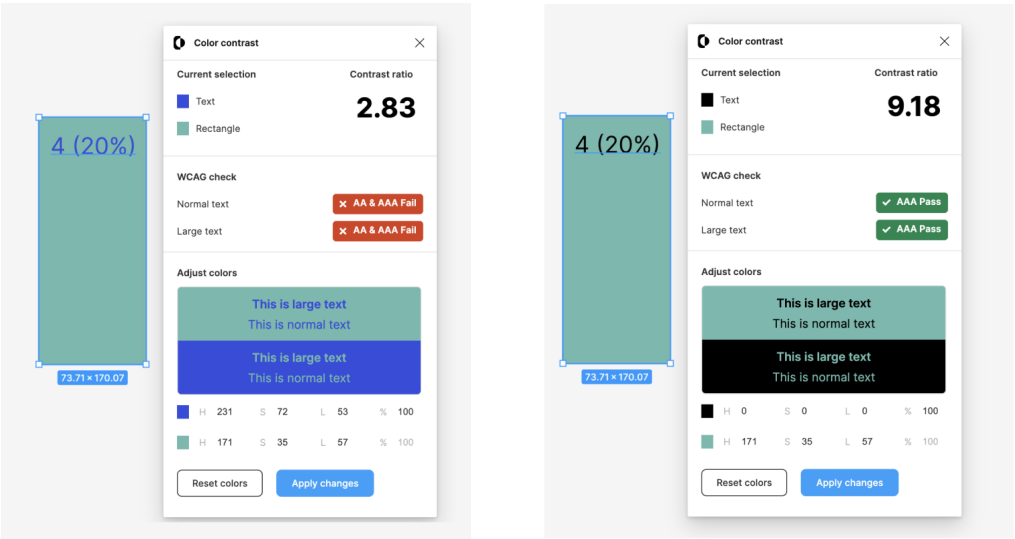
Manual testing with experts or users is essential to making websites truly accessible to everyone. Currently, there is no tool available that can replace human evaluation of many WCAG success criteria. For example, determining if the link text is informative enough or if an input field's error message suggests what needs to be corrected clearly 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 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 are often built into the tools they use daily. Others are online services, where the user can enter the URL of the website to be evaluated or services that regularly check the website and are hosted by the provider or in a company's internal network. These tools may support various accessibility standards.

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, for example, to help identify issues. These tools can usually evaluate either a single page or a group of 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 developers 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. This practice is known to improve the quality of software development and accelerate the development process (Fowler, 2006). As Fowler points out in his article, one of the practices in CI is making your code self-testing by 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 report any failures.

This last principle could be applied well to accessibility evaluation while following modern software development principles. The automated tests will not catch all accessibility issues, but they will catch enough to make it worthwhile. After the initial setup, they should not need a lot of maintenance and would 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 automatically with minimal manual effort every time changes are made to the code, can be used for continuous accessibility testing. The earlier these can be triggered in the development process, the better. Not all of the tools described in the previous chapter can be used for that, for example, browser extensions and plugins in authoring tools need to be manually triggered to perform checks. CLI based tools are the most suitable for continuous accessibility testing.

CLI tools can be customized to meet 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). CLI tools scan the rendered DOM of a website against accessibility standards like WCAG to find violations. The errors are reported back with a reference to the code that produced the error and, in most tools, a link to the accessibility rule it violated.

According to the survey conducted by one of the leading digital accessibility solution providers - 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. The survey states that 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. Level Access's

survey 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 accessibility 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. 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 about 20-30% of 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. To understand the significance of this number, the logic behind how Deque Systems defines and measures coverage needs to be understood. They claim that the current statistics are founded on an inaccurate definition of accessibility coverage - the percentage of individual WCAG Success Criteria (Deque Systems, 2023). According to Deque Systems, 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. To prove the validity of this way of calculating the coverage of accessibility tests, a study was conducted where 2000 axe-core and manual testing audit results were compared (Deque Systems, 2021). The results show that the average percentage of issues detected by axe-core is 57.38%.

In the report, they also state that with the current, in their opinion, inaccurate definition of the coverage, the results would be in the range of 20-30% (Deque Systems, 2021). No studies about other tools that would have used the same method for calculating the coverage were found so in this paper percentage of tested WCAG Success Criteria 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 how well image alternative conveys the meaning of the image or even if the image on the webpage is purely decorative or a part of meaningful content. The goal of using automated testing tools should be to help make the testing process easier by catching problems that can be detected automatically. 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 strengths of automated tests are that they can be set up to run automatically and provide quantitative 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 is consistent with other common principles that promote modularity in software development, such as atomic design and micro-frontends. Designing each component as a standalone unit improves maintenance, reusability, and testing, reduces the learning curve, and speeds up development. The key to success is separating concerns and isolating each logical piece to allow for focused work and refinement. These pieces can be combined into more complex components, which can then be used to create entire web pages or sites.

It is not very straightforward to preview one single component on its own and if the developer needed 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). Without a tool like that developers would often need to manipulate the entire webpage or app to a certain state just to work on a single component. A Component Explorer allows developers to test a given component 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 allows the user to pack the bundled and encapsulated components and share them on the cloud where the 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 their design work and build

products with consistent user experience (Yew et al., 2020). Yew et al. defines a design system as '...a repository of reusable components that follow a set of shared design principles.' Many companies build their own design systems, which could include a component library, style guide, design and content guidelines, design token library, interactive prototyping tools, and accessibility guidelines. While the first three items on this list are generally considered the most essential components of a design system, other components may not always be included, according to a survey conducted by Yew et al. at the leading conference for design systems.

Many big companies have a design system with an open source code that includes a component library, making it possible to reuse and extend the components. 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 design and code quality. These components can be polished over time to provide the best user experience 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 which of them 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 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.

Pipedrive has been developing a sales CRM using mostly typescript and React. Accessibility had never been a high priority and at this point, it would have not been easy to get started. Pipedrive 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 the accessibility issues. The library is used widely in the company and developers from different teams contribute to it. For example, 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. 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

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. It also explains the methods used for collecting and analyzing data.

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).

As part of this research, a literature review was conducted to understand the current landscape of automated accessibility testing tools. While such tools have been available for a long time, the review aimed to determine what is already known about their strengths and weaknesses. Specifically, it sought to answer questions such as whether these tools have been tested in real-world situations and what the results were, as well as what are the main differences between them and whether some have been proven to be better than others.

EBESCO discovery service for the Academic Library of Tallinn University was used to find relevant scientific articles with the 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. Accessibility testing tools evolve fast and this would make older articles irrelevant to the current state of continuous accessibility testing. As a general rule, the focus was on articles that were published in the last 10 years. 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 scientific publications cannot be expected to contain the most up-to-date information. That's why it was thought to be necessary to also include non-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 reviewed and enough tools were assessed during the exploratory phase to find one that would be suitable for this case study.

However, our intention was not to systematically compare them as part of this work. The comparisons that have been made by others in the past will be used as a foundation to decide which tools to use.

3.1.2 Case study

A case study is an in-depth analysis of a bounded system (Range, 2023). According to Range, 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 a CRM tool called Pipedrive. In the scope of this study, an automated accessibility evaluation tool was added to Pipedrive's component library 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 among the company's designers and developers. At the end of the case study, another questionnaire was sent out with a focus on gaining information from people in the organization who used the tool that was added during that period. The aim was to understand if the tool was helpful, whether they had any issues using it and what was their opinion on it. The goal of both of these questionnaires was to get more detailed information about the experience of the real users of the tool.

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

3.2 Adding Automated Testing to Pipedrive's Component Library

The next step was adding an automated accessibility testing tool to Pipedrive's component library development workflow and observing their usefulness. The aim was to find a tool 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 and Axe-core.

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 levels and the tests can be run on multiple URLs simultaneously. 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 Pipedrive's reusable component library's development. It is an open-source software for UI development that allows teams to work on one component at a time (Chromatic, n.d.-b). It allows rendering 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 on an internally available website, with all the components available for anyone in the company to see.

To run accessibility tests with either tool, the components first need to be rendered in a browser. This can be done using Storybook. Each component has stories – examples of how the component would be used in real life. The 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 own 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 accessibility testing 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 could 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 would mean building a custom solution that could be hard to maintain over time and there would have probably been more suitable solutions available already because using component libraries and Storybook 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 as axe-core to audit the HTML of an isolated component

(Chromatic, n.d.-a). The above-mentioned issues have already been solved here. The add-on tests only the relevant part of the HTML and also provides a nice UI inside Storybook.

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 the Continuous Accessibility Evaluation chapter.

Addon-a11y seemed like the best solution in our situation, so it was added to our component library's Storybook. The accessibility tab is visible in every component example (Figure 2). 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.

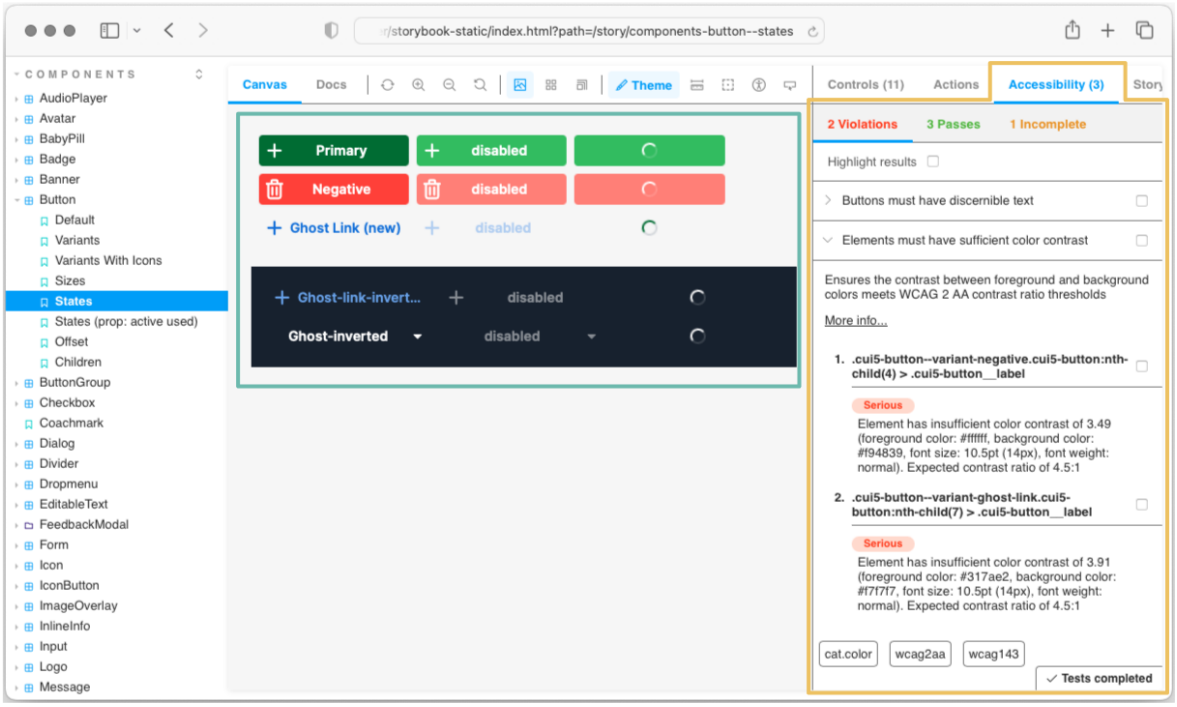


Figure 2: Accessibility add-on (addon-a11y) in Storybook. The accessibility panel is highlighted in yellow and the component example (story) is highlighted in green.

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 guidelines 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 be included in WCAG guidelines (Fiers & Lambert, 2023).

maybe one sentence about what the defaults are

The addon in Storybook provided a good way to test all the examples, but no way to get an overview of how many issues were found in the whole library. A summary of all the potential accessibility problems was needed to evaluate the effort and impact of making the components in this library accessible.

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 the component's 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. Duplicate issues can occur when the component is present in more than one example.
- **violations** - Count of all unique violations detected for the component. Violations that were reported in more than one example of the same component were only counted once.
- **true_violations** - Count of how many of these violations are valid and relevant to this component. Violations that came from other components or elements that were included in the example just to illustrate the usage of the component were excluded.
- **passes_detected_for_component** - A list of unique passes visible in the component's Accessibility tab
- **passes** - Count of all unique passes detected for the component. Passes reported in many examples were 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 reviewed to decide whether they were relevant and valid. Passes that came from other components or elements that were included in the example 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. Issues that were detected using addon-a11y, were excluded because these are listed in the automated testing report.
- **manual_count** - Count of violations found in manual testing.

At the same time, a simple component library, brick-ui was set up to test anything that might not be very easy to try out in a real, actively used component library. Pipedrive's component library had more than 50 components and 414 component examples at the time of the case study. This means that any changes that affected the examples needed to be compatible with all the examples without breaking anything and any changes made to the components or extra steps added to the development workflow needed to be carefully reviewed to make sure that future deploys don't get blocked. For example, updating the component library to use the latest version of Storybook was not possible without making updates to a lot of component examples. These changes are out of the scope of this case study. Most od the concepts can be tried out with the combination of the current component library that is using Storybook 6 and brick-ui using Storybook 7.

Brick-ui 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 tools. Examples from Pipedrive's component library could not be shared, because the code is not publicly available. This made it possible to ask for help from the community and maintainers of axe-core and Storybook.

While this research was going on, development started on a new component library 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 pre-release version of the library as the component explorer for this new library. This provided invaluable insights into an improved automated accessibility testing workflow within Storybook and an opportunity to observe the benefits of setting up automated accessibility tests from the start of the development. In this scenario, there would be no need to fix existing issues, and developers would be able to avoid them from the beginning. While this provided interesting insights the main focus of this thesis is Pipedrive's main component library because the development of the new library is still ongoing and that makes it hard to draw any definitive conclusions.

3.3 Manual Accessibility Audit

The next task was conducting a manual accessibility audit of Pipedrive's main component library to get a better overview of all the issues. This approach will enable a comparative analysis of the results from manual testing and the automatically generated test report to determine the tool's strengths and weaknesses, as well as to identify any limitations when testing isolated components.

Storybook was used to render an isolated preview of each component. The accessibility add-on had already been installed, and all violations reported there were also explored. The audit team consisted of one developer and three designers. One designer considered themselves an accessibility expert. Brajnik et al. cautioned that non-expert evaluators may lead to a significant drop in validity compared to expert evaluators. They suggested using at least three expert evaluators to obtain reliable results, while at least 14 evaluators may be required when the evaluators lack expertise. The audit conducted as part of this research had four evaluators, and although not all of them were experts, they all had a basic level of knowledge in the field of accessibility. Daily discussions and two longer review sessions were conducted to mitigate the lack of experience and ensure the audit's results were as reliable 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. Every morning, the team had a meeting to discuss any questionable issues and to share interesting insights. 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. The evaluators checked violations in the accessibility add-on panel, using a keyboard, and also a screen reader to navigate.

A screen reader is a software application that converts text and other on-screen information, like the layout and structure of a page, into synthesized speech, which is then delivered to the user through a speech synthesizer or refreshable braille display (Watson, n.d.). It allows users to navigate and interact with web content in a non-visual way.

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, which focused on barriers related to user interaction with the website, was used. Evaluators considered various aspects of user interactions, such as navigation, forms, and multimedia content, and looked at what issues different types of users might encounter with each component. The results were separated 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 type of user was simulated by 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 helped access some nested focusable elements that were unreachable using a keyboard.

In real life, these 3 types of users might not be so clearly separated and many issues would affect all of them, 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 Current state of awareness about accessibility in Pipedrive

As the first step of the research, a survey was shared in Pipedive's Slack channels (Figure 3) 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).

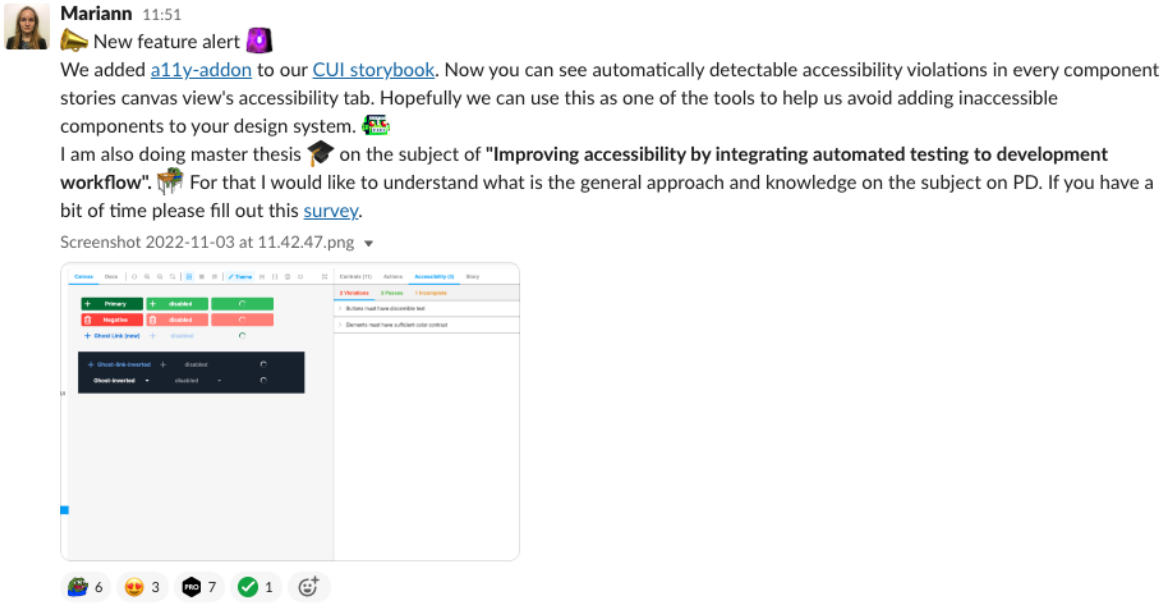


Figure 3: Message in company Slack announcing adding accessibility tool and inviting people to reply to the survey.

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

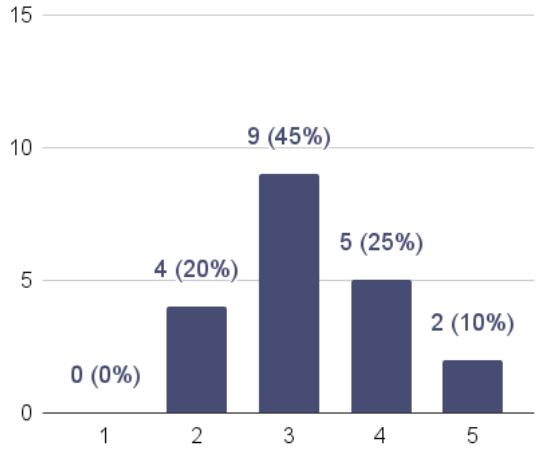
Table 1: List of all the channels the survey was shared in.

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 (Appendix A.1).

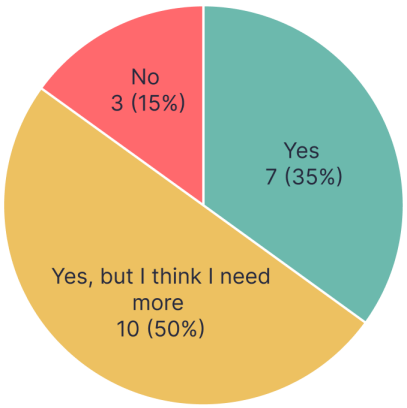
In total, 20 people replied to the survey - 6 designers and 14 developers, including one engineering manager. This does not give a full overview of the company, but it should give a good insight into the general opinions regarding accessibility. Likely, developers and designers

that are more involved with our component library and/or are interested in accessibility were more likely to respond.

RQ1: "How good is the knowledge about accessibility standards, tools and best practices in the company before integrating the accessibility testing tool?" is answered by the results of this survey. They 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 4a). 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 4b).



(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"



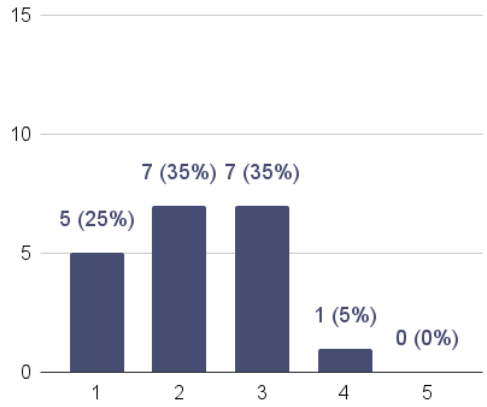
(b) Do you know where to find resources about accessibility standards? (no one chose the 4th option "No I don't think I need them")

Figure 4: Accessibility knowledge in Pipedrive

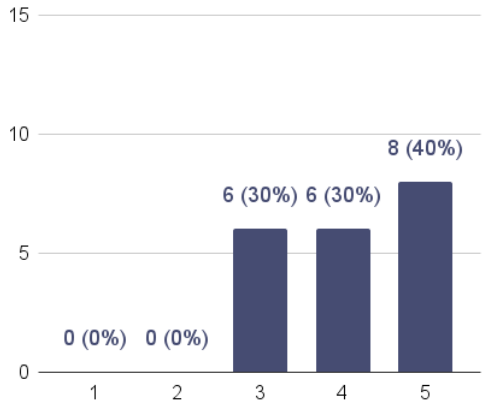
Most people who replied to the survey don't think accessibility is being prioritized in Pipedrive and at the same time, most of them think that Pipedrive should be putting more focus on following common accessibility standards (Figure 5). They elaborated on what they think are the reasons behind this in the free text answers to questions 4 and 6.

Based on the replies received, accessibility has always been seen as something that is low on Pipedrive's list of priorities and there has not been any 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 think that it would not affect Pipedrive's customers much. All replies can be seen in Appendix B.2.

It seems like with most things more knowledge about the business impact of accessibility would help prioritize solving these problems. There is a lot currently that could be improved,



(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 5: Priority of accessibility in Pipedrive

but it could be argued that implementing a way of continuously testing for at least some accessibility issues from the beginning of all new developments could 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 that they appreciate this subject being brought to focus and even some offering to help.

4.2 Results from Manual Audit and Automated Tests

A summary table was created from all violations and passes found while running the automated accessibility tests and all the violations found in the manual audit. The listed violations and passes were counted to obtain numerical data 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%). 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 (Figure 6).

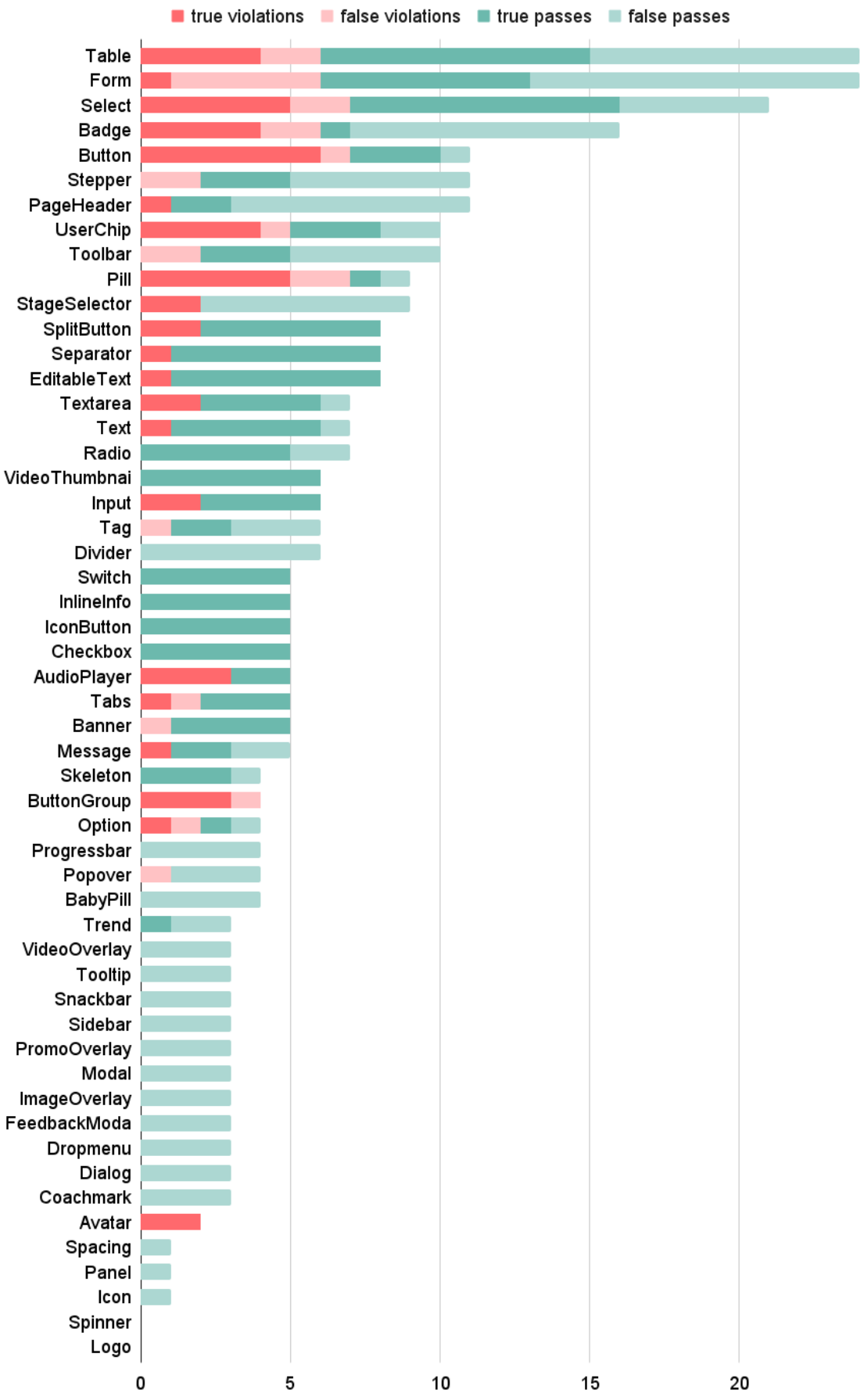
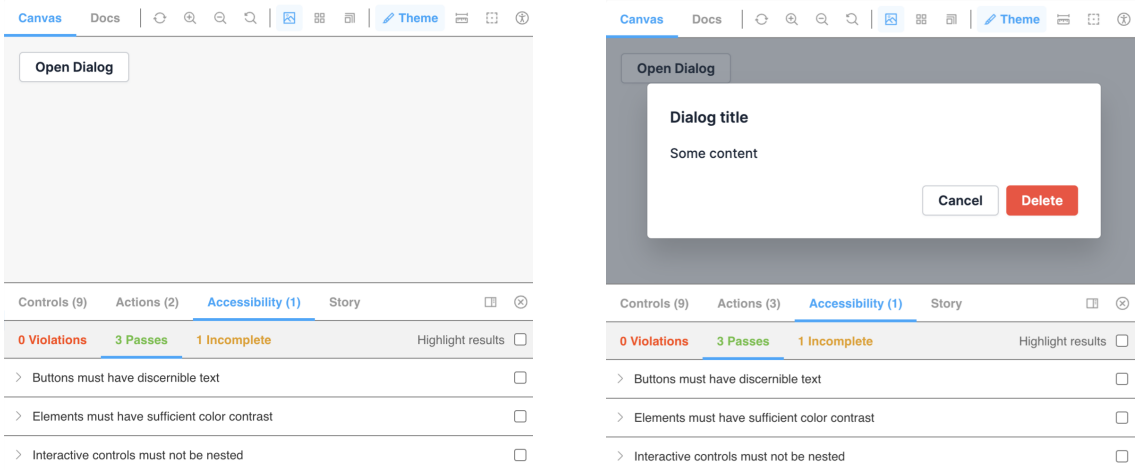


Figure 6: Violations and passes detected by addon-a11y

Four components out of 53 did not have any passed checks and 22 did not have any valid passed checks (see figure 6). 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 - 18 - was detected in *Form* and *Table* components. 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, which decreased to 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 7: Storybook example for Dialog component using a button trigger

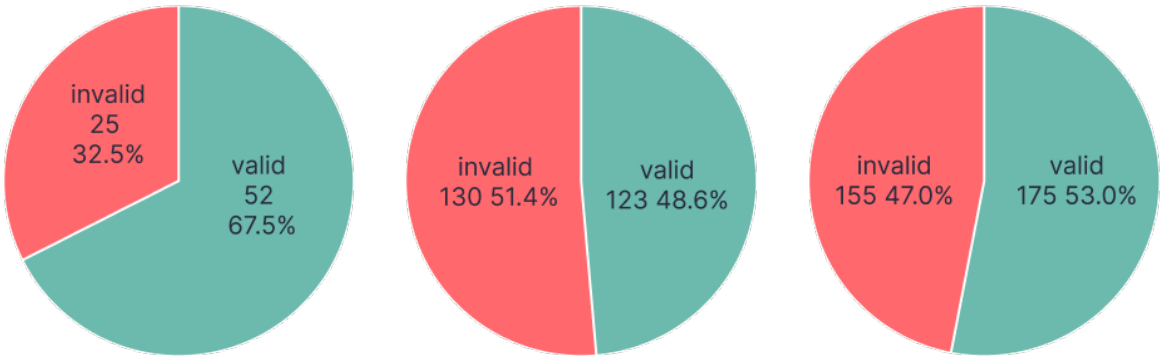
Looking at all the fails and passes gave a good overview of how many tests were run for each component. Out of 53 components that were tested, only 2 (4%) did not get checked by addon-a11y at all, but some of the ones that did get tested included false violations or passes. After validating the results,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

The components that are displayed with a trigger button would normally open when triggered by something, and the examples are intended to resemble how they would be used in real life. Brick-ui was used to explore different ways of mitigating this. Testing revealed that these kinds of examples could be improved in Storybook 7 by adding a user interaction that opens the component. Accessibility tests are run after this mocked user interaction has been executed. In Storybook 6 the same thing could be achieved by opening the example manually and triggering 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). All 11 components were retested in Storybook with addon-a11y with the changes described above made. Accessibility checks could only be triggered in one of the

examples of *Tooltip* and on of the examples of *Modal*. The rest of the components need more improvements to their examples. It is possible to render all the components except *VideoOverlay* and *FeedbackModal* inside the root element. This means that with improved examples, tests could be run for these components. At the time of the case study, there was a bug in axe-core that caused most of the components to be ignored even if all the requirements described above were fulfilled.

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 additional issues for 6 of the components that did not get tested. Manual testing revealed the most additional issues for the components that have a trigger button in the examples. 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 the passes and violations together, a bit more than half were valid while 68% of the detected violations were 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: Validity 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 that proved to be reliably testable in Pipedrive's component library are:

- Success Criterion 1.1.1 Non-text Content
- Success Criterion 1.3.1 Info and Relationships
- Success Criterion 1.4.3 Contrast (Minimum)
- Success Criterion 1.4.6 Contrast (Enhanced)

- Success Criterion 1.4.12 Text Spacing
- Success Criterion 2.1.1 Keyboard
- Success Criterion 2.1.3 Keyboard (No Exception)
- Success Criterion 2.4.4 Link Purpose (In Context)
- Success Criterion 2.4.9 Link Purpose (Link Only)
- Success Criterion 4.1.1 Parsing
- Success Criterion 4.1.2 Name, Role, Value

In addition to these, axe-core automated accessibility testing engine tested for the correct usage of Accessible Rich Internet Applications (WAI-ARIA) and best practices that are defined by the developers of the testing tool based on the accepted practices in the industry. Deque's Github repository lists a total of 87 testable accessibility rules that are enabled by default in axe-core (Fiers & Lambert, 2023). Of those, 58 check for conformance with WCAG 2.0 and 2.1 level A and AA success criteria and 29 are best practices that can't necessarily be mapped to WCAG but should indicate industry-accepted practices for improvements that can be made to the overall user experience. This shows that axe-core testing engine is capable of testing more than what was listed above, but these issues might just not have been represented in Pipedrive's components and examples.

The most commonly 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 axe-core can reliably test for the existence of the alt attribute in an image, but it is not capable of judging how accurate or informative the content of the alt attribute is or even whether the image is purely decorative and should have an empty alt value.

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

- Focus states
- Navigating and interacting using a keyboard
- Navigating and interacting using a screen reader
- Images and what should be the values of their alt texts
- Scalable Vector Graphics (SVG)s and if they should have an aria-label (text alternative) and if so, what should it be

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 had 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 other means
- Examples should be simple and display one component at a time to reduce the number of tests being run that are irrelevant to the isolated component
- Pseudo-classes (like hover, focus etc) of interactive elements should be displayed in the examples

4.3 How Much Was Compliance with WCAG Improved?

One of the benefits of using automated testing tools is the improved observability of the state of accessibility. Multiple accessibility reports were generated in Pipedrive's component library during the time when the usage of the automated testing tool was observed. The reports counted different issues, but for the purpose of this study and for observing the overall accessibility improvement process, the total number of detected violations was compared. (Figure 9).

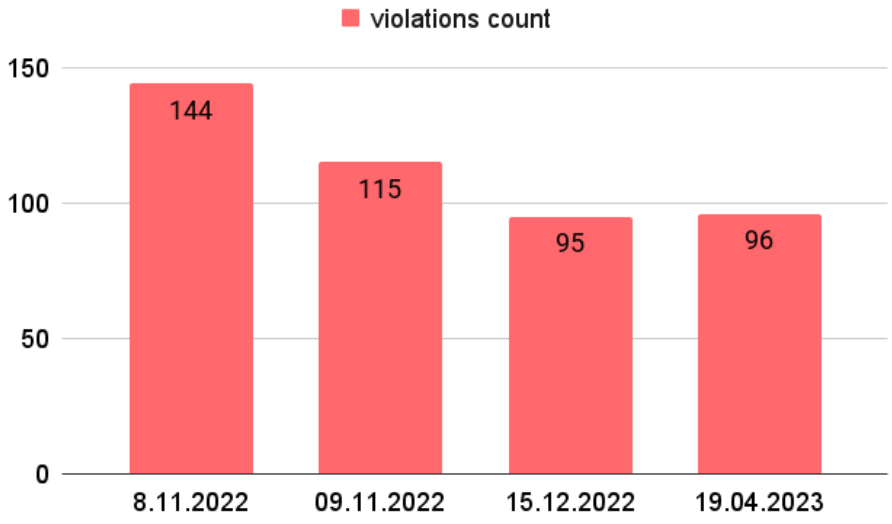


Figure 9: How much have automated audit report results improved since adding the tool?

Results show that over the course of 5 months, the number of accessibility issues was reduced while and immediately after conducting the manual accessibility audit. This is 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 were added to the library between the last two reports and the number of accessibility issues did not increase considerably.

As discussed in the previous chapters, automated accessibility testing tools cannot catch all issues. So the fact that almost no additional violations were introduced with the new components alone cannot provide certainty that the 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 automated testing tools in Pipedrive's component library, 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 only considered after the first release of the product. Having a continuous way to monitor and detect issues should help maintain the level of accessibility that was reached when there was more time to focus on it. 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 in a later phase is more difficult to justify to the business side and is usually more difficult than building an accessible component or webpage from the beginning.

Answering **RQ3**, "To what extent can integrating automated testing into a component library's development pipeline help improve its compliance with WCAG?" is not as simple as examining the reduced number of violations in the automated accessibility testing report. Based on the data gathered in this research, it is not possible to claim conclusively that implementing automated accessibility testing in a component library's development workflow will improve the accessibility of the final product. However, the data does suggest that such integration is highly likely to result in at least some improvement.

The experience of using automated accessibility testing tools and conversations with other developers in the company has suggested that a 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 and familiar way of testing for accessibility violations together with links to resources that help solve these issues makes it more likely for developers to take on the task of solving these issues.

4.4 Limitations of Testing in Component Libraries

For answering **RQ4:** "What are the biggest problems of integrating automated accessibility testing into a component library's development workflow?" we should look at the limitations that testing isolated components poses and also more specific issues that were faced during this case study and implementing automated accessibility testing in Storybook with addon-a11y.

Automated accessibility tests are not capable of evaluating all WCAG Success Criteria and there are also additional limitations that apply when these tests are being run on isolated components. The context around the elements on a webpage is important for understanding the purpose of each element. CDD development methodology promotes perfecting isolated elements before moving on to the next step where these elements are combined to build full views and logical interactions. This means that when testing a component library, it is not so important to focus on things that depend on the final context, but rather on what is contained in the component. The assumption is that there should be additional testing in the following stages of development.

In the case study that was carried out in Pipedrive's component library, Storybook was used as a way of rendering examples of the components and running automated tests on them. The accessibility add-on in Storybook analyzes the examples that have been made for the component which means unsuitable examples can cause false results, like components triggered by a button as described before.

Another limitation of addon-a11y is that currently the results can only be viewed inside Storybook. To see the number of passes and fails, the user needs to open the accessibility tab for each component. In the case study, an additional tool was used to generate a report that summarized all the violations, but generating the report needed some manual steps. It would be possible to automate this, but this would not be worth the effort, because the newest version of Storybook will provide better solutions. It will be possible to run all accessibility checks as a part of the component's unit tests making it possible to add them to the CI workflow.

The accessibility checks could not be run as a part of the CI workflow while using Storybook 6. This means that the usage of the tool was heavily dependent on the developer's knowledge about the existence of the tool. I could have been easily missed. This was one of the questions in the post-case study survey and the results show that at least one developer who made changes to the component library during that time didn't notice it. One designer also said that they did not have enough information about how the tool works and what they need to do if they see that it has detected potential issues.

5 Discussion

The experience from the case study carried out as a part of this thesis, indicates that the tests fit well into the development workflow and don't cause any unnecessary complications - moreover, they help notice issues that are not only related to accessibility but may also indicate problems in code logic that could have otherwise been missed. This suggests that using automated accessibility testing this way has the potential to improve overall code quality in addition to ensuring that the testable accessibility issues get caught before changes are added to the codebase.

The evidence from relevant literature and this case study suggest that if possible, automated accessibility tests should be included in the development setup as early as possible. In the case of testing component libraries and using Storybook, an early start would definitely help avoid a lot of issues later on. For example, developers could write component examples to be more suitable for accessibility tests from the start to ensure that the tool can test for the maximum amount of issues possible.

5.1 Contributions

Despite the limitations of this thesis, it has made a considerable contribution to the research of automated accessibility testing. Firstly, there had been no previous research done on using automated testing tools in component libraries. CDD is a very common way of building web content and catching any issues related to accessibility in component libraries would ensure they are found as early in the process as possible.

Secondly, by doing a case study on a real UI building system, it brought out problems that others might face when they try to integrate automated testing tools into an already existing system. Building things in the correct way from the start is always easier than fixing them later.

5.2 Limitations

The study was carried out on an existing and in-use component library so all the changes to the library needed to be nonintrusive and not add any unnecessary complexity to the development workflow.

Another limitation that became apparent towards the end of the study was using Storybook 6 in the component library. The latest version would have provided much better possibilities, but it could not be used, because at the beginning of this research, this version was still in the first phases of development and a version this unstable could not have been used in a component library that is being actively developed. This was mitigated first by setting up a mock component library to try out any new options in a safe way and later by testing this new version in another development inside the company. Both of these examples provided valuable insights but were very limited on their own: the mock library did not use real examples and the second development was only observed over a short period.

The limitations of the manual accessibility audit on the component library were the time we had to go through all the components, the number of evaluators in the team and their level of expertise on the subject.

5.3 Future Research

At the time that the case study was being run, another new component library was built in Pipedrive and with the initial insights from the case study a better setup for evaluating accessibility could be implemented. Storybook 7 and its test-runner with accessibility tests were used. It will be interesting to see what the long-term effect of setting up these kinds of checks from the start of development will be. This means that there should be no need to rewrite components so that they would comply with accessibility standards, rather they would be built to comply from the start.

In this case study Storybook and axe-core were used as the main tools for running accessibility tests. It would be interesting to see how other tools perform in comparison to them. The test-runner in Storybook's latest version makes it possible to integrate any CLI tool to run tests on each component example. It would be even possible to run tests with multiple tools to potentially get better coverage of all automatically testable accessibility issues.

Another very promising new path in automated accessibility testing is the use of artificial intelligence (AI) and machine learning (ML). This could potentially cover aspects that currently rely on human evaluation. Deque has mentioned the use of AI and ML in the axe suite, but nothing more concrete could be found about what they use it for. The use of AI can enhance the accessibility of web content by generating text alternatives, captions, and translations. It can also improve the testing of these features. At the time of writing this thesis,

it appears that most AI-based accessibility testing tools are paid. However, as the technology is evolving rapidly, there will likely be free options available on the market soon. There is not much research about the efficacy of using AI-based automated accessibility testing tools, and it would be fascinating to explore their capabilities through research.

5.4 Conclusions

The goal of this thesis was to explore the possibilities of automated accessibility testing in component libraries. A case study that added an automated accessibility testing tool to Pipedrive's component library was carried out over 5 months. Before the new tool was added, a survey was done to understand the context where this component library is being used. The results of the survey showed that accessibility is seen as something that is not currently the main focus of the company but should be prioritized more. The designers and developers in the company assess their knowledge on accessibility to be average or higher and many think that they need more resources on the subject.

To validate and compare the results from the automated testing, a manual audit was performed by a team of 3 designers and 1 developer. The results show that automated tests can be helpful, but the tools need to be carefully selected and set up to maximize the checks being run on each component. Not all accessibility issues can be caught by automated testing and testing a component library sets its own limitations by taking elements out of context. Automated accessibility tests in the component library were run in Storybook with its accessibility addon that uses axe-core accessibility testing engine. Almost half (45,1%) of all the checks performed by the tool were color contrast issues. A significant part of the 20 components that did not pass or fail any tests had examples with a trigger button that prevented the tool from accessing the HTML of the component. This limitation was explored and there are ways of mitigating this with improved examples.

To validate and compare the results from the automated testing, a team of three designers and one developer performed a manual accessibility audit. Comparing the results indicated that automated tests can be beneficial, but the tools need to be carefully chosen and configured to optimize the checks performed on each component. Not all accessibility issues can be identified by automated testing, and testing a component library has its limitations because it tests HTML elements out of real context. In this case, study Storybook and addon-a11y, that uses the axe-core accessibility testing engine, were used to run automated accessibility tests in the component library. Color contrast issues accounted for almost half (45.1%) of all the checks that were run by the tool. A significant portion of the 20 components that did not pass or fail any tests included examples with a trigger button that prevented the tool from accessing the component's HTML. This limitation was examined, and there are methods for addressing it, such as improved examples.

There is no conclusive evidence from this research that automated accessibility testing in component libraries can reduce the number of accessibility issues on the website of other product that uses this library, but the results suggest that it might help bring more focus to the subject of accessibility and definitely catch some issues.

Accessibility is a complex issue that cannot be resolved by a single tool or approach. However, automated accessibility testing can provide an easy way to initiate addressing these issues. Using CLI based automated accessibility testing tools that can be integrated into CI workflows can minimize the amount of manual effort required to run these tests. This approach aligns well with modern software development principles and ensures that every time changes are made to the codebase, basic checks for compliance with accessibility standards are conducted.

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

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

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.

Miniwatts Marketing Group. (2023). *Internet usage statistics: The internet big picture* [World internet users and 2023 population stats]. Miniwatts Marketing Group. Retrieved 04/26/2023, from https://www.internetworldstats.com/stats.htm

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

Watson, L. (N.d.). *What is a screen reader?* Nomensa Ltd. Retrieved 04/30/2023, from https://www.nomensa.com/blog/what-screen-reader

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

Appendices should only be included for the committee to see and not be public

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