Design and Evaluation of a Semi-autonomous Accessilility Tool for Web Development

Master's Thesis in Medieninformatik
at Institute for Information and Media, Language and Culture (I:IMSK)

Handed in by:	Mathias Götz
Address:	Halfingerstr. 24, 81825 München
E-Mail (University):	mathias-christian.goetz@stud.uni-regensburg.de
E-Mail (private):	mathias.goetz@outlook.com
Student Number:	2244750
Primary Corrector:	Prof. Dr. Christian Wolff
Secondary Corrector:	Prof. Dr. Niels Henze
Supervisor:	Prof. Dr. Christian Wolff, Markus Müller (TÜV SÜD Product Service GmbH)
Current Semester:	6. Semester Medieninformatik M.Sc.
Date handed in:	23.06.2023

Contents

Task 8

1. Introduction 9
1.1. Problem statement 9
1.2. Approach outline 11
1.3. Problems and results 11
1.4. Structure of this work 11

2. Background 13
2.1. Digital Accessibility 13
2.2. W3C Web Content Accessibility Guidelines (WCAG) 2.1 14

3. Related Work 16
3.1. Automation of Accessibility 16
3.1.1. Testing 16
3.1.2. Implementation 17
3.2. Automated accessibility tools 19
3.2.1. Axe 19
3.2.2. Wave 19

4. System and design 20
4.1. Basic Algorithm Details 20
4.2. Selected Accessibility Problems 21
4.2.1. Language missing 23
4.2.2. Region missing 23
4.2.3. H1 Missing 24
4.2.4. Heading skipped 24
4.2.5. Heading Empty 25
4.2.6. Heading Possible 25
4.2.7. Text Small 26
4.2.8. Contrast 26
4.2.9. Link Empty 27
4.2.10. Link Suspicious 27
4.2.11. Alt Link Missing 28
4.2.12. Link redundant 28
4.2.13. Link internal broken 29
4.2.14. Alt missing 29
4.2.15. Alt suspicious 30
4.2.16. Alt duplicate 30
4.2.17. Alt redundant 31
4.2.18. Table layout 31
4.2.19. Text justified 31
4.2.20. Button empty 32
4.2.21. Label missing 32

4.2.22. Title redundant 33
4.2.23. Event handler 33
4.2.24. Noscript 34

5. Implementation 35
5.1. Algorithm Details 35
5.2. Algorithm parts 36
5.2.1. Basic Page Information 36
5.2.2. Headings 36
5.2.3. Text Content 37
5.2.4. Links 38
5.2.5. Alt Text (Images) 39
5.2.6. Layout 39
5.2.7. Buttons 40
5.2.8. Assistive Tags 40
5.2.9. Scripting 41

6. System Validation 42
6.1. Evaluation 42
6.1.1. Data set 42
6.1.2. Webcrawler 43
6.1.3. Technical Implementation Details 47
6.1.4. Checking Algorithms 47
6.2. Results 47

7. Discussion 51
7.1. Results 51
7.2. Limitations and Further Work 53

8. Conclusion 55

Bibliography 57

A. Appendix 60
A.1. Webaim Million Data 61

Erklärung zur Urheberschaft 62

Erklärung zur Lizenzierung und Publikation dieser Arbeit 63

List of Figures

1. Two screenshots of the same webpage. The screenshot on the left shows inadequate contrast, the right picture shows sufficient contrast between the background and text. Website: https://www.tuvsud.com 9
2. WCAG guideline 1.4.3 Contrast (Minimum) (World Wide Web Consortium, 2018) 15
3. Most frequently used tools for automatic accessibility evaluation (M. et al., 2019) 17
4. Table showing the difference between parameters needed by automatica11y and manual code refactoring on 5 web pages (Ikhsan & Catur Candra, 2018) 18

List of Tables

1. All selected accessibility problems categorized. Percentage data according to (WebAim, 2023) 22

2. Top 100 websites in Germany (Sereda, 2022) resulting in usable html files 46
3. Data on errors-types compared between the WebAim million study and the checker created during this study 48
4. Accessibility Issues before & after 49
5. Data on frequency of errors-types compared between the websites before and after running the checker 50

6. All selected accessibility problems. Data based on WebAim million (WebAim, 2023) 61

Zusammenfassung

Digitale Barrierefreiheit bedeutet, Barrieren bei der Nutzung von digitalen Diensten und Webseiten zu beseitigen. Barrierefreiheit fokussiert sich auf die Inklusion von Menschen mit Behinderungen, aber auch normale Benutzer können von sinnvoll umgesetzter Barrierefreiheit profitieren. Die meisten aktuell bestehenden Webseiten weisen Barrierefreiheitsprobleme auf, innerhalb der Top Million Webseiten tritt auf 96,8% mindestens ein Fehler auf (WebAim, 2023). Der in dieser Arbeit beleuchtete Ansatz um diese Problem zu lösen, ist Automatisierung. Dazu werden 24 unterschiedlich Barrierefreiheitsprobleme ausgewählt, und ihre Relevanz beschrieben. Für diese Probleme wird ein theoretisches Konzept zur automatischen Behebung im HTML Quellcode aufgestellt. Alle diese Teilalgorithmen werden in Python zu einem Tool zusammengefasst und implementiert. Anhand der Top 100 Webseiten in Deutschland wird das Tool evaluiert. Dabei werden die Webseiten automatisch auf Barrierefreiheitsprobleme vor- und nach Anwendung des Tools überprüft Die Ergebnisse zeigen, dass die Verwendung des Tools die Barrierefreiheit dieser Websites deutlich verbessert. Daraus lässt sich folgern, dass die Automatisierung der Barrierefreiheit eine valide Option für Webseiten ist.

Abstract

Digital Accessibility is the principle of removing barriers in the usage of digital services and websites. Accessibility has a focus on the inclusion of disabled users, but able-bodied people can also profit from the implementation of accessibility.

Currently existing and evaluated websites show accessibility problems, with 96.8% of the top million websites showing some kind of accessibility failure (WebAim, 2023). This thesis focuses on solving this problem by automating the process of implementing accessibility. As not all guidelines can be implemented, the focus is on the most impactful. Therefore 24 accessibility problems are selected, evaluated for relevance and a theoretical algorithm concept created to fix these problems automatically. This concept is implemented in python to create a working tool for automated accessibility implementation. Using the top 100 web sites in Germany the tool is evaluated, with the websites checked automatically for accessibility problems before and after. Results show that using the tool improves accessibility on these websites significantly. This leads to the conclusion of automating accessibility being a valid option for making websites more accessible.

Task

Hintergrund

Der Kerngedanke (Digitaler) Accessibility ist es, allen Menschen gleichermaßen ohne fremde Hilfe Zugang zu digitalen Systemen zu ermöglichen. Daher gibt es bereits bestehende Guidelines, die bspw. für Webseiten umgesetzt werden können, um Menschen mit Einschränkungen den Zugang zu ermöglichen oder zu erleichtern. Diese Methoden werden allerdings in einem Großteil der bereits bestehenden Systeme nicht umgesetzt. Um Entwicklern bei der Umsetzung zu helfen, existieren Tools, um die Accessibility (teil-)automatisiert zu testen. Diese können allerdings nur die Korrektheit einer zuvor manuell durchgeführten Implementierung überprüfen.

Zielsetzung

Um das Problem der fehlenden Accessibility in einem Großteil der bereits bestehenden Webseiten zu lösen, soll ein Algorithmus entwickelt werden, damit bereits bestehende Accessibility Guidelines (teil-)automatisiert umgesetzt werden können. Dieser Algorithmus soll dann mit einem Datensatz aus bestehenden Webseiten validiert werden, indem die Accessibility der Webseiten vor und nach dem Einsatz des Algorithmus bewertet wird.

Konkrete Aufgaben:

- Aufbereiten relevanter Literatur
- Erarbeiten eines Algorithmus für (teil-)automatisierte Umsetzung von Accessibility für Webseiten anhand bestehender Guidelines
- Prototypisches Umsetzen dieses Algorithmus
- Validierung des Algorithmus an bestehenden Webseiten

1. Introduction

In Germany alone, 7.8 million people suffer from disabilities. This equals to 9.4 % of the population (Destatis, 2022). With the digital world playing an increasingly big role in everyday life, those people need to have equal access to digital technologies as everyone else. To make the internet usable, and have those people be able to fully take part in digital life, a certain level of digital accessibility is needed. This means for example taking account of the needs of visually impaired people, and having an adequate level of contrast, so the information can be perceived by everybody. Additionally, accessible websites do not only benefit people with disabilities, well designed websites with a focus on accessibility can also improve usability for regular users.

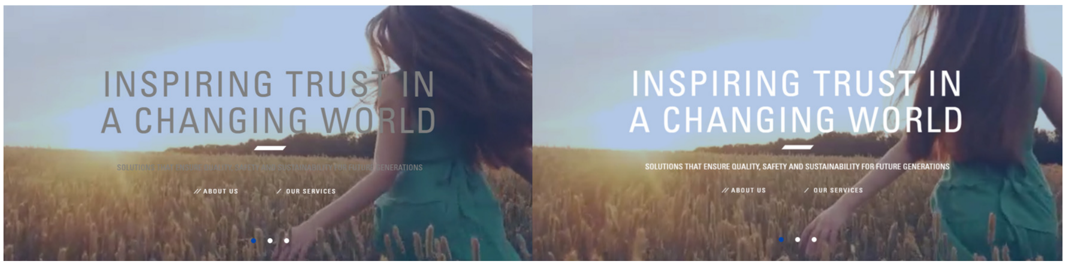


Figure 1.: Two screenshots of the same webpage. The screenshot on the left shows inadequate contrast, the right picture shows sufficient contrast between the background and text. Website: https://www.tuvsud.com

1.1. Problem statement

Accessibility is needed to include everybody in the everyday digital life. To ensure a base level of accessibility, the european commission published a regulation, the 'european accessibility act' (EAA). This regulation requires consumer facing companies to implement digital accessibility for their products or services starting June 2025 (European Parliament, 2019). This means these digital products or services like computers, E-Commerce or media websites need to be accessible by this date, to avoid fines. These regulatory requirements as well as the social responsibility of inclusion showcase the importance of digital accessibility. This statement of importance stands in stark contrast with the situation in reality. A study on the top million websites worldwide shows, that 98.6% of these websites show some kind of accessibility failure (WebAim, 2023). This raises the question why accessibility is not implemented at all or not implemented correctly in these web pages. Research shows the main problems to be time and budgetary constraints (Ikhsan & Catur Candra, 2018), lack of knowledge (Ikhsan & Catur Candra, 2018), lack of awareness (Antonelli et al., 2018) and websites no longer being in active development (Centeno et al., 2005). All these problems except awareness can be solved by adding automated implementation of digital accessibility.

A lot of previous research shows automated testing of accessibility for websites, and helping the developers getting their websites to an accessible state in that way (M. et al., 2019). This still means, that they need to implement everything themselves. Research projects on automating the implementation of accessibility are scarce, and no software for the automatic implementation exists, in comparison to a lot of automated testing tools. Accessibility can not be automated for everything, as some parts need human judgement or input. For these aspects, creating a semi autonomous approach where the developer needs to give information and the changes are performed automatically can still decrease the workload, when compared to fully implementing everything without assistance.

This leads to the question: To what degree can a semi-autonomous accessibility tool help improve the website landscape?

This thesis sets out to show, if automating accessibility for web pages can be a viable option to increase website accessibility, while not adding a lot of work for the developers, and thus lowering barriers for a more accessible web.

1.2. Approach outline

The thesis starts by defining a subset of accessibility problems to be addressed, with the intent to use the most frequent problems to make a noticeable improvement on different website types. For those problems, a theoretical algorithm concept is established. This theoretical algorithm is then implemented in python for testing. To evaluate if this proof of concept offers the expected benefits and can be used in the real world, a study with the top 100 web pages in Germany is conducted, and the results are discussed. With these results, the functionality and meaningfulness of the algorithm can be confirmed.

1.3. Problems and results

As there are 78 different accessibility guidelines as defined in WCAG 2.1 (World Wide Web Consortium, 2018), the most impactful accessibility problems have been selected using the WebAim million study. Using these guidelines, the issues occurring in 10% of websites or more are selected. This leads to 24 accessibility problems focused in this thesis. For these problems, the theoretical relevance and a concept for (semi-)automatic code changes to fix the problem is discussed. For 23 of these problems, this theoretical concept could be implemented in python. To validate the implementation in the real world, the algorithms are tested on the top 100 websites in Germany by visits. To see if the code changes result in a real improvement, all websites are checked for accessibility problems before and after the appropriate changes are performed. For this we use an algorithm developed during this thesis as well as off-the-shelf software. This leads to an improvement of 74.4 % when using the specific checker developed during this thesis, and an improvement of 20.7% using off the shelf testing software.

1.4. Structure of this work

Background on digital accessibility and the impact of digital accessibility as well as current digital accessibility guidelines is given in chapter 2. Chapter 3 describes the relevant related work, like implementation standards for digital accessibility, or other scientific articles on automated testing or implementation of digital accessibility. In chapter 4 the basic algorithm design as well as the relevance and theoretical concept to fix the defined accessibility problems is described. The implementation and step from theoretical algorithms to a real implementation is described in chapter 5. This design is then used to validate the concept using a study of 75 websites described in chapter 6. Chapter 7 then discusses the results and limitations of the algorithms as well as the real world impact as seen in the study. Chapter 8 concludes this work by giving a summary and an outlook on future work.

2. Background

This chapter gives an overview on the relevant background topics needed to understand the further work in this thesis. It defines the basic concept of digital accessibility in chapter 2.1 and the W3C Guidelines published by the world wide web consortium and used to achieve an accepted level of digital accessibility in Chapter 2.2.

2.1. Digital Accessibility

Digital Accessibility is the principle of removing barriers in the usage of digital services and websites, to make them usable for everyone, regardless of their abilities. Accessibility has a focus on the inclusion of disabled users, but able-bodied people can also profit from the implementation of accessibility, as a clear and easy to use website offers benefits to everybody. None of these guidelines compromise the usability for regular users, but can offer benefits even for users that do not suffer from disabilities. So it does not only help the 7.8 million users affected by disabilities in Germany, it also has the potential to help every user. With the percentage of disabled people increasing with age (Destatis, 2022), the number of people benefiting from accessible websites is expected to rise as time goes on. Assistive technologies are used by over a quarter of site visitors, when looking at the statistics for the UK government page, with 30% of users reporting to use a screen magnifier and 29% of users reporting the use of a screenreader (UK Government Propriety and Civil Service, 2016).

The current standard for accessibility testing WCAG 2.1 includes 4 principles with 50 success criteria (A&AA) (World Wide Web Consortium, 2018). A part of these success criteria need to be tested by hand, as human judgement is needed for criteria that require context knowledge.

In the best case, implementing digital accessibility has been considered by developers early on in the development process, as adding these features afterwards may require more work and can lead to a worse implementation (Centeno et al., 2005). Websites currently implemented and developed without a focus on accessibility have a lot of criteria to implement into already existing websites to offer accessibility. This amount of work, especially for a website that already exists and works for regular users puts off most companies and developers. This can be a reason why currently 96.8% of existing websites in the WebAim million study show at least one accessibility failure (WebAim, 2023).

2.2. W3C Web Content Accessibility Guidelines (WCAG) 2.1

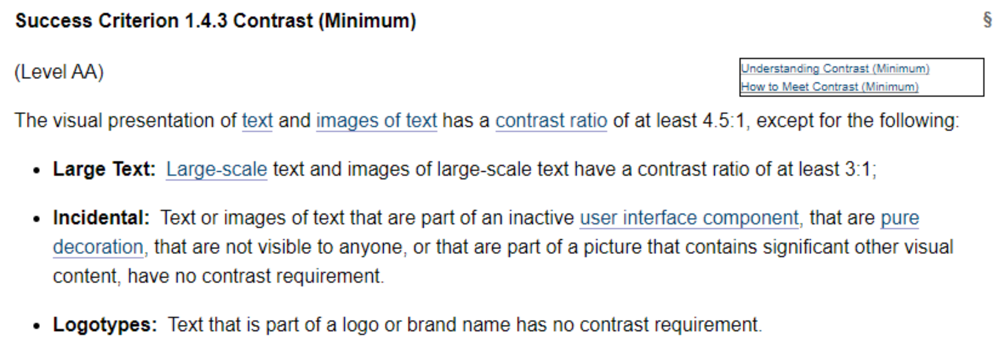
WCAG is a set of success criteria created by the World wide web consortium (W3C).

Criteria can be class A, AA, or AAA with decreasing importance. A and AA levels are required for compliance, with AAA considered as ”going the extra step”. The current version is WCAG 2.1, released in 2018 with WCAG 2.2 in a draft status as of march 2023 (World Wide Web Consortium, 2023). WCAG consists of 4 main classes of principles:

- Perceivable:
  Information and user interface components must be presentable to users in ways they can perceive.
- Operable:
  User interface components and navigation must be operable.
- Understandable:
  Information and the operation of user interface must be understandable.
- Robust:
  Content must be robust enough that it can be interpreted by by a wide variety of user agents, including assistive technologies.

The current version 2.1 contains 50 required success criteria (A&AA) and 28 optional criteria (AAA) to be implemented and tested (World Wide Web Consortium, 2018).

These guidelines give information on the requirements that need to be fulfilled when designing a website, together with exceptions and definition on when to apply these rules. The guideline to fix the contrast problem shown in the introduction (Fig. 1) is defined in WCAG 2.1 Success Criterion 1.4.3 can be seen below (Fig. 2).



Success Criterion 1.4.3 Contrast (Minimum) §

(Level AA)

Understanding Contrast (Minimum)
How to Meet Contrast (Minimum)

The visual presentation of text and images of text has a contrast ratio of at least 4.5:1, except for the following:

- **Large Text:** Large-scale text and images of large-scale text have a contrast ratio of at least 3:1;
- **Incidental:** Text or images of text that are part of an inactive user interface component, that are pure decoration, that are not visible to anyone, or that are part of a picture that contains significant other visual content, have no contrast requirement.
- **Logotypes:** Text that is part of a logo or brand name has no contrast requirement.

Figure 2.: WCAG guideline 1.4.3 Contrast (Minimum) (World Wide Web Consortium, 2018)

As these guidelines have a main focus on Web, the mobile sector (Apps) is mainly left out, with WCAG stating "Mobile accessibility is covered in existing W3C WAI accessibility standards/guidelines. There are not separate guidelines for mobile accessibility" (Initiative (WAI) W3C Web Accessibility, 2021). This means, the guidelines are very general, to cover everything.

As these guidelines are the baseline for accessible web development, they are also referenced in the web-part of the harmonized standard EN 301 549 'Accessibility requirements for ICT products and services' (chapter 9).

Conformity with all criteria of WCAG 2.1 automatically proves compliance to the EAA, therefore these guidelines need to be fulfilled to comply with the regulation starting 2025.

In the past, WCAG has been criticized to formalize too much, with no real impact when working with affected people (Rømen & Svanæs, 2012). Another problem of WCAG is, with the guidelines requiring human judgement for some of the guidelines, one tester may deem a point as successfully implemented, while other testers may not accept the implementation (A. S. Al-Khalifa & Al-Khalifa, 2011), (Alonso et al., 2010), (Brajnik et al., 2010).

3. Related Work

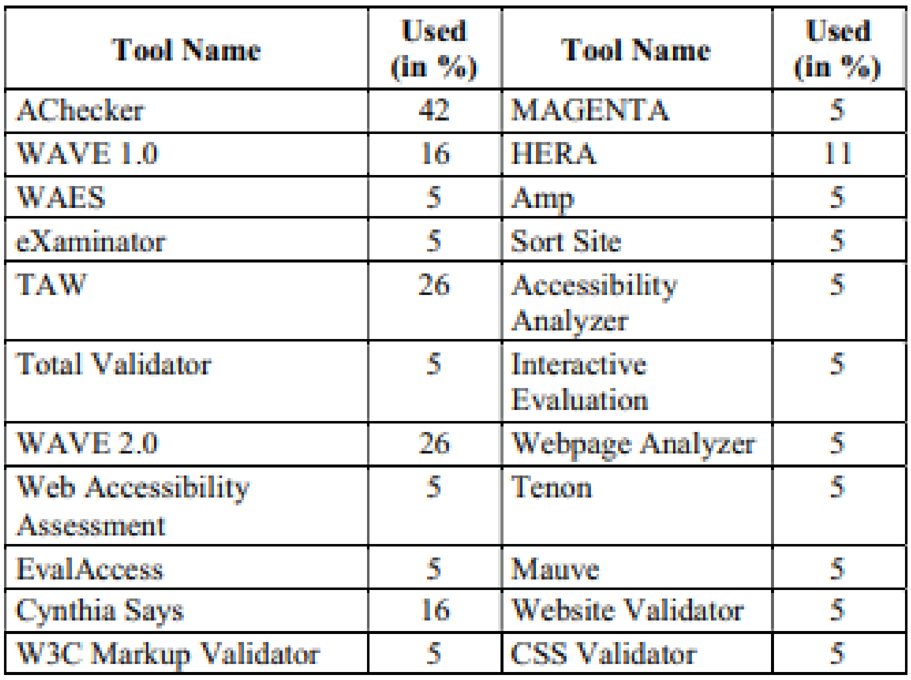
This chapter describes related work in the field of this thesis. This includes related research projects and articles, as well as off the shelf software in the field of accessibility automation. All these related projects serve as inspiration for the thesis and add to the knowledge needed to create this concept.

3.1. Automation of Accessibility

This chapter gives an overview on the scientific research in the field of accessibility automation for websites, focusing on the field of automated testing and the smaller field of automated code refactoring to handle and automatically fix accessibility issues.

3.1.1. Testing

Automated testing for website accessibility is widely available, and mostly based on the WCAG guidelines. For testing, several automated tools and research papers exist. Nagaraju et al. conducted a literature review on the most frequently used automated testing tools used in research work. The authors look at 43 different research projects evaluating different data sets of websites for evaluating their accessibility levels. They give an overview over the 22 automated checking tools, with usage frequency within all considered research projects (Fig. 3).



Tool Name	Used (in %)	Tool Name	Used (in %)
AChecker	42	MAGENTA	5
WAVE 1.0	16	HERA	11
WAES	5	Amp	5
eXaminator	5	Sort Site	5
TAW	26	Accessibility Analyzer	5
Total Validator	5	Interactive Evaluation	5
WAVE 2.0	26	Webpage Analyzer	5
Web Accessibility Assessment	5	Tenon	5
EvalAccess	5	Mauve	5
Cynthia Says	16	Website Validator	5
W3C Markup Validator	5	CSS Validator	5

Figure 3.: Most frequently used tools for automatic accessibility evaluation (M. et al., 2019)

From the 43 research projects, 19 have been selected for more thorough review. This resulted in information on the sample, levels of evaluation and findings. The most frequent guidelines used for evaluation is the WCAG Standard in different levels of coverage, with 16 out of 19 research papers using some kind of WCAG based testing. (M. et al., 2019). These automatic testing tools are limited in use cases and results. When creating a new tool for arabic websites Al-Khalifa tested the created checking algorithm against other checking algorithms. This lead to finding significant differences between the algorithms (H. Al-Khalifa, 2012).

3.1.2. Implementation

The most similar research project on automate code refactoring is done by Ikhsan and Catur Candra with the title "Automatica11y: An Automated Refactoring Method and Tool for Improving Web Accessibility". The authors identify the research gap in the field of automated accessibility, with most previous research only focusing on automated testing, instead of automated refactoring of websites to implement common accessibility criteria automatically. To prototype the system, the authors modify the existing tool 'HTML Code Sniffer' to find accessibility issues when looking at html source code. All WCAG techniques that can be found by HTML Code Sniffer are evaluated for automation potential, leading to a total of 40 techniques implemented. This tool is evaluated on a set of 5 predefined websites, with 4 already existing websites and a website specifically created for evaluating the tool. It is run multiple times, until there are no changes performed anymore, as a website may need more passes of the algorithm to be fixed completely. The tool shows significant improvement when changing the websites using the tool (Ikhsan & Catur Candra, 2018). Figure 4 shows the number of parameters needed to enter into the tool to change the code, in comparison to the manual code changes needed to fix the accessibility issues.

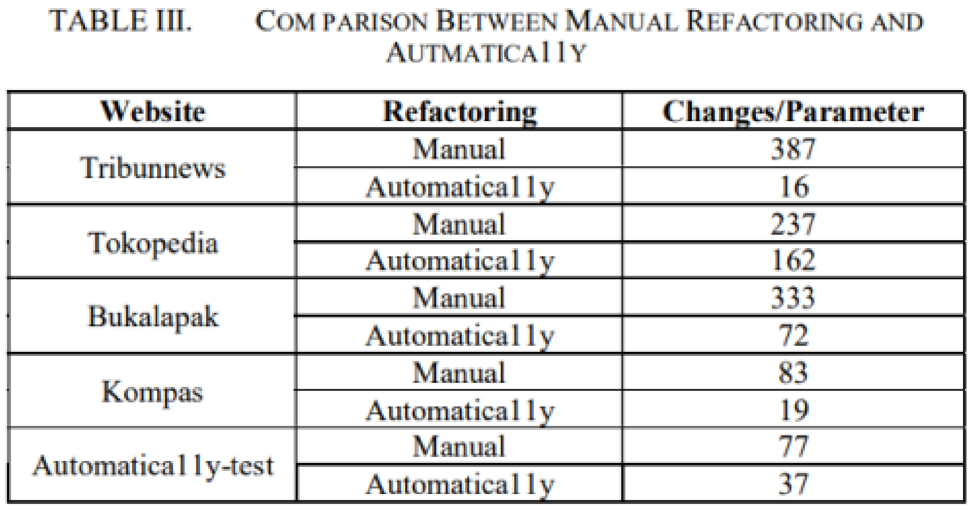


TABLE III. COM PARISON BETWEEN MANUAL REFACTORING AND AUTMATICAl lY

Website	Refactoring	Changes/Parameter
Tribunnews	Manual	387
	Automatical ly	16
Tokopedia	Manual	237
	Automatical ly	162
Bukalapak	Manual	333
	Automatical ly	72
Kompas	Manual	83
	Automatical ly	19
Automatical ly-test	Manual	77
	Automatical ly	37

Figure 4.: Table showing the difference between parameters needed by automatica11y and manual code refactoring on 5 web pages (Ikhsan & Catur Candra, 2018)

There is a significant difference between automatic and manual code changes, showing that the system decreases the workload. This means using an automated tool makes changing websites to conform to digital accessibility easier and more efficient. This research shows that automating accessibility implementation can lead to a fast way to improve web accessibility in a defined use case.

3.2. Automated accessibility tools

This chapter describes two different ready to use software packages, that offer automated or semi-automated accessibility testing solutions. The selection of these two tools has been made as they have an impact on the development of this thesis.

3.2.1. Axe

The Axe software suite is an automated tool for accessibility testing developed by Deque Systems [1]. The software offers different services like a browser extension as paid products. It developed axe-core, an open source accessibility testing API. This API receives active support and development from Deque systems. Axe-core covers 57% of WCAG 2.1 issues automatically, with assistance for manual testing for the remaining guidelines (dequelabs, 2023). On this foundation, python libraries like axe-selenium-python are developed, to have a powerful tool for different use cases, like automated checking of websites using a python script.

3.2.2. Wave

The Wave evaluation tool[2] is an automated accessibility testing toolkit, developed by WebAim, a part of the Institute for Disability Research at Utah state university. This University is the creator of the WebAim million study used later in this thesis. The tool is offered as an api-service as well as a browser extension. The Wave error categories are different than the WCAG 2.1 success criteria, but aim to cover the full spectrum. Every category can be differentiated between errors, warnings or just informational aspects. All categories are documented, with a description of the issue, help for fixing the issue and the relevant WCAG 2.1 criteria (WebAim, 2023).

[1]https://www.deque.com/axe/
[2]https://wave.webaim.org/

4. System and design

This chapter covers the theoretical concept for the algorithms created in this thesis. It gives an insight on basic algorithm considerations, as well as the selection of the most influential accessibility problems. The relevance of the selected problems is discussed and the concept for a theoretical implementation algorithm evaluated.

4.1. Basic Algorithm Details

The algorithm finds accessibility flaws within a specified website. For the predefined accessibility problems, smaller sub-algorithms are defined to change the html source code and fix the problem automatically without breaking functionality. The algorithm is designed to be used by developers, so every change made to the website that changes the appearance or functionality in any way can be checked by the developer. As the tool needs supervision from the author of the website, usage for individuals to change any website every time it is opened is not in scope of this thesis.

As the accessibility problems need to be identified first, this leads to two algorithms created during this thesis. The first algorithm checks for accessibility problems, finds the cause of the problem in the source code and logs the number of problems into a CSV file. The second algorithm uses the part of the first one to find the problems and their location. It contains the algorithms for fixing these problems. All actions are logged, and the algorithm returns the changed html file containing the full website with the identified issues fixed.

4.2. Selected Accessibility Problems

To find a selection of accessibility problems, the WebAim million study has been taken into account. This study looks at the top million websites worldwide, and lists the most influential and frequent issues (WebAim, 2023). These issues are categorized in errors and warnings. For the purpose of this thesis, the team behind the study has been contacted, to get the list of problems with a overall percentage of which problems are the most frequent, as these numbers are not published within the study [3]. The problems are categorized using WebAims own categories. These categories can be identified using the WAVE analytics tool. All categories can be mapped to W3C WCAG 2.1 Guidelines (WebAim, 2023). The following table shows the errors and warnings from the WebAim 2023 study, together with the total percentage of the million websites showing these problems. For the sake of this study, problems occurring on more than 10% of total web pages are considered to have the most significant impact, as they are the most frequent and occur on all types of websites. With this selection, 24 different accessibility problems, composed of 8 errors and 16 warnings remain. Some of those errors and warnings need more information or approval from the developer, because they either need a decision or alter the websites layout. These problems are categorized by similarity and order for the full algorithm in table 1. The full list sorted by error frequency can be found in table 6. For every selected issue, the relevance and the concept for a theoretical implementation algorithm is discussed. For this, the WebAim documentation is used as starting point (WebAim, 2023).

[3]All data can be found in the contents of the attached flash disk

Guideline Name	Type	Error Type	Percentage
language_missing	**Basic Page information**	Error	22.3%
region_missing		Warning	26.0%
h1_missing	**Headings**	Warning	23.8%
heading_skipped		Warning	40.5%
heading_empty		Error	13.0%
heading_possible		Warning	19.7%
text_small	**Text Content**	Warning	29.2%
contrast		Error	83.9%
link_empty	**Links**	Error	50.1%
link_suspicious		Warning	18.0%
alt_link_missing		Error	38.7%
link_redundant		Warning	71.7%
link_internal_broken		Warning	12.0%
alt_missing	**ALT Text (Images)**	Error	33.8%
alt_suspicious		Warning	13.5%
alt_duplicate		Warning	14.5%
alt_redundant		Warning	18.1%
table_layout	**Layout**	Warning	13.9%
text_justified		Warning	10.8%
button_empty	**Buttons**	Error	27.2%
label_missing	**Assistive Tags**	Error	46.1%
title_redundant		Warning	38.4%
event_handler	**Scripting**	Warning	11.4%
noscript		Warning	48.5%

Table 1.: All selected accessibility problems categorized. Percentage data according to (WebAim, 2023)

4.2.1. Language missing

Relevance

Language tags are a kind of meta-tag, informing the user in which language the text of a webpage is presented. The usage of language tags is mainly important for users of screen readers, with the screen readers using different voices and pronunciations for different languages. If a language tag is set incorrectly or does not exist at all, the pronunciation of text may not make sense, and therefore be difficult to understand for users.

Approach

This issue can be fixed fully automatically, by looking at a text paragraph of the website and determining the language using language-recognition libraries. As this only adds a tag to the html-header, no layout change or other side-effects occur.

4.2.2. Region missing

Relevance

Website regions, for example 'header','footer', 'navigation' or 'content' are parts of a webpage, that exist on every sub page of the website. Because of that, it makes no sense for screenreaders to read the contents of recurring parts like a footer every time, when the user knows the structure from the last page. Screen readers can only read out the 'content' that is different on every sub page in this case. This clear structure is also needed for navigating the webpage.

Approach

To automatically determine page regions, it is possible to look at different subpages of the same website, and try to determine which elements are on every sub page. With this information, it is possible to determine which parts are content and which are parts of the layout.

4.2.3. H1 Missing

Relevance

The heading structure is an important part of conveying the website structure for screen reader users and users relying on other forms of assistive tools. Screen readers or other assistive technologies may group the content part in chapters using these headings. If the heading is missing, it can be hard for some users to understand the website structure.

Approach

Different headings can be determined inside the html-structure automatically. This way, it can be checked if a H1 element exists. If this is not the case, and H2-elements are present, the first H2 element can be converted to a H1 element, while keeping the appearance (size, text style) of a H2 element. This does not change the layout or appearance, while providing more structure.

4.2.4. Heading skipped

Relevance

Like with H1 missing, when a heading level between H1 and H6 is skipped, for example there is no H3 element, it just continues from H2 to H4 and makes it difficult to understand the website structure for some users. That can be a conscious layout decision, but makes the structure harder to understand for screen reader users. If the different heading labels are just used for styling purposes, simple html or CSS styling can be used instead of different heading tags.

Approach

Like with H1 missing, it is possible to determine all headings and their respective levels automatically. If a level is skipped, it is possible to change the level, so there are no gaps, without changing the layout. This makes the structure easier to understand for users of assistive technologies like screen readers, while keeping the same appearance.

4.2.5. Heading Empty

Relevance

As described for previous categories, headings add structure to websites. When a heading element is present that has no content, it may still be read to screenreader users when navigating the heading structure. That may cause confusion on the website structure.

Approach

As an empty heading will not be shown to users that view the website regularly (without the use of assistive technologies), there is no information lost when the element is deleted. These empty headings contain no extra information and can be deleted, so every user has the same experience.

4.2.6. Heading Possible

Relevance

A part of text styled like a headline makes no difference to a seeing user, as there is no visual difference. On screenreaders, information like text size or text-styling like 'bold' or 'italic' are mostly lost. Users of screenreaders will not see this information, and it will be treated like regular text. If it is intended to be interpreted like a heading and is supposed to add structure or context, this information will be lost for these users.

Approach

Only parts like text size and style can be determined automatically, not the intention behind these style choices. If a full paragraph is styled like a heading, and contains only one sentence or less, it can be suspected to be a heading. If such occurrences are found, the assumption of it being a heading can be made. This still needs input on the intention, before it can be automatically changed to a heading with the appropriate level. This level can be determined by looking at the text size when compared to the styles of different heading levels.

4.2.7. Text Small

Relevance

Small text might not be seen by users with a vision impairment and, depending on the text-size, can even be problematic for regular users. On the other hand, text with a size of 0 will not be seen by regular website users, but might be read out to screenreader users. In this case information may be lost for different user groups or information may be read out that is not supposed to be seen. This leads to a different usage experience for these groups.

Approach

Text size inside a html document can be programmatically determined. If the size is 0, it can just be deleted, without losing any information, as it would not be seen anyway. If it is too small to be seen for users with lower vision, the text-size can be increased automatically. This can change or break the layout, so this change is not without side-effects and needs careful consideration by the developer overseeing the process.

4.2.8. Contrast

Relevance

Good contrast is needed for visually impaired and even regular users. If the contrast is too low, it makes the content harder to perceive and one may not be able to read the text presented on a webpage.

Approach

The contrast can be calculated automatically, when looking at the text and background colors. If the contrast is not sufficient, the colors can be changed automatically. As the background is more likely to be an intentional layout decision, the text color can be changed. This still carries the risk to destroy intentional layout decisions, and needs oversight from the developer.

4.2.9. Link Empty

Relevance

An empty link has no descriptive link-text, and will therefore not be shown correctly. This problem exists for regular as well as screen reader users, with regular users not having access to the link or users of screenreaders getting read a link that has no text.

Approach

As the purpose of the link is not known, it can either be deleted or the link destination programmatic determined. In this case, the links destination will be determined, with the goal to add a meaningful description. When opening the link, the algorithm can either take the site title or description to get information on the contents of the link. This information can then be used to create a link description to use as new link text. This may change the layout, so the developers approval is needed.

4.2.10. Link Suspicious

Relevance

Link suspicious can be when the link text consists of 'Click', 'Here' or only the URL. That means the link purpose can not be discerned only from the link text. This can be needed by users of assistive technologies, that may read out the links in a different way or order than the content is presented to seeing users. Normal sighted users can have a hard time understanding the purpose of the link, when there is no context provided, and may only understand the links purpose after clicking on it.

Approach

If the link text can not be used to convey the correct link information, the text can be programmatically changed, as described in chapter 4.9 .

4.2.11. Alt Link Missing

Relevance

ALT-Link missing describes the problem arising when the only content of a link is an image, without alternative text. That means, the link will be presented to screen-reader users like it has no content, when there is no text and no alternative text for the image to read out.

Approach

This problem can either be fixed by adding alternative text to the image (see 4.2.14) or by determining the links destination (see 4.2.9). As the intended use of the link is for an image, adding alternative text is the preferred option in this case.

4.2.12. Link redundant

Relevance

A redundant link may not disturb regular users, but to screen reader users the link may be read twice. This is especially problematic when the links are next to each other, because one link will be read out right after the other. This adds time for screenreader users, while offering no extra information.

Approach

As links can be identified, link elements can be checked for directly adjacent links. If these links go to the same destination, one can be deleted without losing information. For the selection of which link is more useful and should be kept, developer input may be needed.

4.2.13. Link internal broken

Relevance

Internal links provide a way to jump to a predefined point inside of the web page. If these links are broken, they only add confusion and do not offer any value to the user, as the intended function does not work. This can happen if the link or corresponding id is broken or forgotten.

Approach

As it is not possible to get the original intention of the link, when the corresponding target does not exist on the webpage, no functionality is lost when the broken link is removed. These links start with href="#", and if the corresponding ID is not present on the webpage, the link can simply be removed.

4.2.14. Alt missing

Relevance

ALT-Tags are important for blind users or users with a sighting impairment. This user group can only perceive parts of images, or not percieve them at all. For this, every image in a website has to have an alt tag, containing a description of the image to give another method to get the information otherwise only conveyed by the picture.

Approach

Inside HTML code, it is easy to find <img>-tags, that do not contain an alt-element. An AI description service can then be used to transcribe the contents of these pictures. Any context between pictures and text is lost, because these libraries only transcribe what they can see in the image, without connection to the context of the website.

4.2.15. Alt suspicious

Relevance

Like ALT-Text missing, a not suspicious ALT-Text is needed to convey the same information the image shows. Suspicious ALT-Text might be alt text that is very short, only states 'Image' or just the Filename.

Approach

The ALT text can be checked for length, keywords like 'Image' or if contains a filename by checking if it ends in a file extension like '.JPG' or '.PNG'. This means the information in the ALT-text does not provide the information it is supposed to. If this is the case, the Algorithm from 4.2.14 can be used, to generate a new ALT-Text.

4.2.16. Alt duplicate

Relevance

Duplicate alt texts can be a problem, if different images have the same alt-text. That means, for users with impaired vision, two different pictures could have the same descriptions. In this case, they miss out on information seeing users can perceive.

Approach

All images in the webpage can be checked if adjacent images contain the same alt-text. If this is the case, the alt-text for one or both images can be changed automatically using Computer Vision, or the developers input.

4.2.17. Alt redundant

Relevance

Redundant ALT-Text means, the contents of the alt text are also seen in nearby text. This means, the same information is seen more than once. This can add confusion to users trying to take advantage of the alt-text.

Approach

The images alt-text can be checked. If it appears in nearby text, it can be removed without losing any information. It can also be changed to 'Image providing context to XYZ' or completely regenerated using a ComputerVision library as seen in 4.2.14.

4.2.18. Table layout

Relevance

Layout tables are created for layout purposes, for example to get two text-areas or images next to each other. When compared to regular tables, these tables do not come with a header. These tables are read out to screenreader users like a real table. This adds confusion, as the tables are not intended to be presented as such, and should only change the layout.

Approach

Layout tables can be found, as they contain no <th> elements. This content can then be changed into regular html elements using CSS to preserve the styling choices, without altering the layout.

4.2.19. Text justified

Relevance

Text justification adds white spaces, so that every line of text has the same length. This can offer a cleaner look, but different spacing can make it more difficult to read longer texts.

Approach

Fully justified paragraphs can be automatically found, checked for length and the justification can be switched to regular text layout. This makes the text easier to read, but does potentially change an intentional styling decision and the layout.

4.2.20. Button empty

Relevance

Empty buttons have no descriptive Text. So the user does not know what the purpose of the button is, if the context does not provide additional information. This can be confusing to every user type, if the purpose can only be understood when looking at the context or the button is pressed.

Approach

Empty buttons can be found, but the purpose of the buttons can not be automatically determined. In this case, developer input on the purpose of the button may be needed. With this information, the Button-Text can be changed automatically.

4.2.21. Label missing

Relevance

When a Website has forms with input-fields for user input, they need a label attached to it. This label provides information on the purpose of the corresponding input field, and is needed for every user to understand the forms purpose.

Approach

The intention behind the label can not be found automatically. It is possible to add the input type, for example 'Text' or 'Password' as a label, to add a little bit of extra information, to make it easier for the user to understand the purpose of that input field. Another option is to make the developer describe the input field, and input this information into the source code automatically.

4.2.22. Title redundant

Relevance

The title tag for html elements adds extra information to html elements. If it is the same as image alt-texts or the text in general, it adds no extra information. It is then just another element that may be shown to the user, whilst not providing any benefits or new information.

Approach

All html elements can be searched for the title tag. If it contains the same text as image alt text or the adjacent text, it can just be deleted without losing information or changing the layout in any way.

4.2.23. Event handler

Relevance

Event handlers, like 'onclick' of 'onmouseover' start an action, when the mouse is clicked or hovers over the content. This can lead to content not being accessible for users that access the website only using a keyboard or in other ways that do not include the mouse-cursor like touchscreen operation.

Approach

To make this content accessible, elements like 'onfocus', that trigger when the element recieves focus using keyboard controls, can be added. For this, every element with only mouse accessible event handlers can be found and more accessible options like 'onfocus' can be added that serve the same function. An 'onfocus' element triggers when the element gets focus, regardless of the user interaction activating said focus.

4.2.24. Noscript

Relevance

Noscript elements contain information, that will only be presented to users with JavaScript disabled. This tries to convey the same information as the JavaScript parts. As only 1.3% of users have JavaScript disabled (Hein, 2010), this may not be needed. Most assistive technologies can handle JavaScript when implemented correctly and accessible (de Oliveira, 2018). This means, a noscript element does not necessarily add accessibility. This just adds extra bulk to the website.

Approach

Noscript elements can be found and deleted without changing the layout. No information will be lost, as these elements are only shown if JavaScript is disabled, and try to provide the same information as the JavaScript parts.

5. Implementation

This section describes the implementation of the theoretical algorithm developed in section 4. The result of this chapter is a checking algorithm, that checks for accessibility issues in the given html source code, as well as a fixing algorithm that fixes all defined issues.

5.1. Algorithm Details

The algorithm is implemented in python. As input, the website needs to be available as html-file. The html file is parsed using BeautifulSoup (bs4)[4], as it offers the possibility to address the different html elements by type as well as search for different parts of the html tag, like meta information and contents. The order of the different part algorithm is designed in a way that the part algorithms work together, instead of influencing each other in a negative way. This is especially important for issues like 'alt-link-missing', where the alt-tag is fixed before the part is reached and tries to fix it in the link part. The output of the tool is another html-file, with the accessibility issues fixed in the source code. As it is needed to find the accessibility issues before fixing, a bi-product of this algorithm is a separate algorithm used only for finding and logging these accessibility issues. This tool is referred to as 'checker'. It only takes the website as input, and prints the issues by category into a csv-file as well as to the console.

[4]https://www.crummy.com/software/BeautifulSoup/

5.2. Algorithm parts

All previously described theoretical implementations (see chapter 4) for fixing the accessibility problems are implemented in python. The implementation for every accessibility problem is described in the following chapter. All algorithm implementations are described in detail, using the categorization and order described in chapter 4.2. The developers input for the semi-autonomous part is not implemented, instead assumptions are made for the input to have the algorithm usable for the study described in chapter 6.

5.2.1. Basic Page Information

Using bs4, the html-header can be found. This tag can then be checked for the 'lang' attribute. If the attribute is not present (*language-missing*), a paragraph with text can then be loaded into the python library langdetect[5] by requesting the text of the first paragraph using bs4. This library returns the appropriate ISO language-code from the provided text. This code can then be added as lang-attribute to the html tag.

For the issue of a missing page region (*region-missing*), no autonomous or semi autonomous algorithm could be created. It was not possible to discern the page structure when comparing the given source code with other subpages of the same website. This issue can be a point of future research.

5.2.2. Headings

All headings can be found using bs4. This leads to an array of all different headings. If no heading of type 'H1' can be found (*h1-missing*), the first heading in the list can be converted to a 'H1' element by changing the name of the html element to 'H1'.

For *heading-possible*, the text size of text paragraphs is evaluated using the algorithm defined in 5.2.3. If the text size is higher than 19, or higher than 15 and the elements text contains '<b>' for bold text or '<it>' for italicised text, the length of the text is checked. If the length is below 50 characters, the element is changed to

[5]https://github.com/Mimino666/langdetect

a heading of class 'H6' or another heading type that is not present on the webpage, to fill in the gap.

All different heading levels can be checked as well. If a level returns an empty array, with the subsequent heading level arrays containing headings (*heading-skipped*), the empty array can be filled by changing all headings in the subsequent level array to the empty level. All headings get the CSS attribute 'class' with the original heading level, so they keep their appearance. This algorithm is iterated over every heading level. This leads to the only empty heading levels being of the lower heading types, with gaps between different heading levels.

With the array of all headings, the string can be checked for contents. If the heading is empty (*heading-empty*) the element can be extracted using bs4.

5.2.3. Text Content

Using bs4, every html element with a style tag can be found. This style tag can then be used like a string, and checked for a font-size declaration. If this is present, the text in between 'font-size:' and 'px' can be extracted, and the font size saved. If the font size is lower than 11 (*text-small*), there are two ways to handle the situation. If the text size is 0, the element is extracted from the file using bs4. If the size is higher than 0, the value is replaced with a text size of 12 in the style tag. This changes the text-size to 12, and makes the content readable.

To check the text contrast (*contrast*), the sites background color needs to be evaluated first. If the background color is not defined in the header, it can be saved as white, otherwise the appropriate value is stored. All html elements without their own background color are stored, and the content and children's text-color is stored. If a html-element has no specified text-color, it is saved as black, which is the standard value for html. The same steps are performed for elements with a specified background color. This leads to a list of background and foreground colors, with values for every html element. These values are then evaluated, with the background and text-color used as input values for the library wcag-contrast-ratio[6].

[6]https://github.com/gsnedders/wcag-contrast-ratio

It returns the contrast value for the specified element. If this value is below 4.5, the text color needs to be changed with WCAG requiring a contrast level higher than 4.5. The background color is then checked for contrast with white and black, and the better color is used if the contrast value exceeds 4.5. In this case, the text-color can be changed by replacing or adding the better text-color attribute to the corresponding html element.

5.2.4. Links

All links in the html code can be found using bs4. Iterating over all links, the link-text can be checked if it is present. If this is not the case (*link-empty*) and a 'href' element is present, meaning the link has a valid destination, the link can be opened using python requests. This content is then parsed into bs4, and the website title and description extracted. If the description does not exceed 150 characters, it is used as new link text. Otherwise the website title is used. This information can then be set as text for the link. When looking at the links, the link text can be evaluated whether it contains words like 'click', 'more' or 'www.' to check if the link is suspicious (*link-suspicious*). For suspicious link text, the text is replaced using the method discussed above.

The issue of *alt-link-missing* is fixed in Chapter 5.2.5, with alt-texts getting included automatically.

When looking at the array of all links, it can be checked if the succeeding link has the same destination by looking at the 'href' attribute for both html elements (*link-redundant*). If this is the case, the link text length is compared, and the shorter link is extracted using bs4.

When checking the links, if the destination starts with a '#' it is an internal link. For internal links the corresponding id needs to exist (*link-internal-broken*) in another html element. A list of every html-element with an id can be created using bs4, and the ids checked against the link using python string operations. If the corresponding id to the link is not found, the internal link is extracted using bs4.

5.2.5. Alt Text (Images)

All images in the html code can be searched using bs4. It is then possible to iterate over every picture, and check the alt text. These alt-attributes can then be checked if they exist at all (*alt-missing*). If an alt-attribute exists, it can be checked if it is suspicious by checking the string for content like '.jpg', '.png' or 'http' using string operations to see if the alternative text is just an url or a filename (*alt-suspicious*).

If these problems occur, an alternative text can be created using the image link. This link is appended to the current URL, if the image is at the same web-location as the webpage. Using this link, the Microsoft Azure Cloud Vision API[7] can be used, to create an image caption. This caption needs to be translated from English to German using deep-translator[8] when the source website is german, because the Microsoft API only offers the captioning service in English.

If the alt-text is sufficient, it still needs to be checked if it is the same as the alt-text of adjacent images. This is done using the bs4 array of images created earlier by looking at the alt-text of the previous or succeeding image (*alt-duplicate*) and comparing it to the current alt text. If this is the case, the alt-text for the current image is changed using the method above.

The last check on the alt-text is to see if the alternative text is contained verbatim in the previous or next html element (*alt-redundant*). This is done by checking if the string can be found in the text-contents of these sibling, using the bs4-method returning these siblings and comparing the strings using regular python operations. If this is the case, the alt-text is changed automatically using the computer vision method explained above.

5.2.6. Layout

All table elements can be found using bs4. Every table can be checked if a table header element ('<th>') exists by looking at the element as text. If this is not the case, it is a layout-table and needs to be changed (*table-layout*). This is done by

[7]https://learn.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-describe-images-40?tabs=image
[8]https://github.com/nidhaloff/deep-translator

removing the rows from the table and changing every column into a 'span' element with ';display:inline-block;' added to the style to make the columns displayed next to each other. Every row is followed by a break ('<br>') to end the line. This mimics the layout-tables appearance by placing the cells next to each other, and the rows below each other. With all contents changed, the empty table can be extracted.

Using bs4, every element can be checked if the element has a style-attribute containing 'text-align: justify' and text is longer than 500 characters (*text-justified*). If this is the case, the justification can be removed by replacing the justification in the style-attribute with an empty string.

5.2.7. Buttons

Using bs4, all button elements can be searched. These are checked for the text. If a button without text is found (*button-empty*), it can be changed to the developers desired text. In this case, it is replaced with 'button', to make the algorithm usable for the study.

5.2.8. Assistive Tags

All forms on the website can be found using bs4. When checking these forms, an input field can be checked for the 'id' attribute. If this attribute is present, a list of labels can be checked to see if a label exists having the 'for' attribute with the same id. If this is not the case, or the input does not have an 'id' atribute (*label-missing*), a new label with the id can be created using bs4. If needed, this id can be created for the input field. This label can either be set using developer input or by using the function of the input-field as new label-text. These functions can be values like 'text' or 'password'.

All html elements with a title attribute can be listed. The text of the attribute can be compared with the alt-text of the element. If this title attribute is the same as the alt-tag (*alt-redundant*), the title attribute is deleted using the extract function of bs4.

5.2.9. Scripting

Every html element can be checked, if an attribute like 'onmouseover' or 'onclick' exists. If these attributes exist, it needs to be checked if an accessible version like 'onfocus' is implemented. This can be done using basic bs4 functions. If no accessible version exists (*event-handler*), it is possible to add an 'onfocus' attribute to the code with the same content as the non-accessible event-handler.

If noscript elements are present (*noscript*) they can be found using bs4. These elements can then be extracted.

6. System Validation

To validate the algorithm, the top 100 websites in Germany are automatically evaluated. To have a level of repeatability, all Websites are downloaded and saved into a data set. All websites are first evaluated for accessibility problems, and these scores are saved into a CSV file. After that, the fixing algorithm defined in section 5 is used to fix all implemented accessibility problems automatically. For this, the number of changes is also recorded. After the Algorithm, all changed websites are once again checked for accessibility problems, and the score is recorded. Using this method, a before and after comparison for the algorithm can be performed. This check is done using the checker created in this thesis as well as the axe testing software, to get independent results.

6.1. Evaluation

This section describes the data set and the general implementation of the study used to evaluate the fixing tool created in this thesis.

6.1.1. Data set

The data set is created using the top 100 websites in Germany by visitor numbers (Sereda, 2022). This is done, so a diverse data set of different websites and website purposes can be evaluated, while still testing with relevant websites. This has the goal to see the accessibility issues on major German websites, as well as get a diverse set of different errors. The algorithm has not been trained or tested on these websites. This data set can also be compared against the WebAim million, to find differences in a German data set in comparison to international values.

6.1.2. Webcrawler

To have comparable results, the top 100 Websites in Germany have been crawled using python requests and bs4 on 28.05.2023. This leads to only the main page (index.html or similar) saved as html. These html files can then be opened like a regular webpage in a browser. This is done so there are no changes, in comparison to working with the real webpage and people continuously adding new content or changing the website. This leads to comparable results if the web pages need to be reevaluated later. This working with files instead of the URL simulates the process the tool is intended for, with developers putting in their html source code themselves. All websites which could not be crawled or resulted in an 'error' or 'browser not supported' page have been removed from the data set. Websites occurring twice in the data set, but with different top-level-domains (for example ebay.com and ebay.de) have only been considered once, with the lower ranking html file filtered out. This leads to a full data set of 75 web pages out of a total 100 as seen in table 2.

Ranking	Website	Usable
1	google.com	X
2	youtube.com	X
3	facebook.com	X
4	amazon.de	X
5	wikipedia.org	X
6	bild.de	X
7	google.de	X
8	ebay.de	X
9	t-online.de	X
10	instagram.com	X
11	ebay-kleinanzeigen.de	
12	web.de	X
13	spiegel.de	X
14	tagesschau.de	X

Ranking	Website	Usable
15	focus.de	X
16	dhl.de	X
17	xhamster.com	X
18	samsung.com	X
19	twitter.com	X
20	twitch.tv	X
21	gmx.net	X
22	paypal.com	X
23	wetter.com	
24	otto.de	X
25	welt.de	
26	n-tv.de	X
27	pornhub.com	X
28	derwesten.de	X
29	merkur.de	X
30	wetteronline.de	X
31	chefkoch.de	X
32	sport1.de	X
33	idealo.de	X
34	live.com	X
35	accuweather.com	
36	fandom.com	X
37	teads.tv	X
38	chip.de	X
39	zdf.de	X
40	vodafone.de	X
41	immobilienscout24.de	X
42	ndr.de	X
43	netflix.com	X

Ranking	Website	Usable
44	mobile.de	
45	tz.de	X
46	booking.com	X
47	yahoo.com	X
48	kaufland.de	X
49	whatsapp.com	X
50	pressekompass.net	
51	ebay.com	X
52	stern.de	X
53	pinterest.de	
54	zeit.de	X
55	xvideos.com	X
56	faz.net	X
57	bahn.de	X
58	rtl.de	X
59	giga.de	X
60	bing.com	
61	mediamarkt.de	
62	lidl.de	X
63	check24.de	X
64	ikea.com	X
65	linkedin.com	X
66	tvspielfilm.de	X
67	finanzen.net	X
68	tiktok.com	X
69	zalando.de	
70	telekom.de	X
71	transfermarkt.de	
72	sueddeutsche.de	X

Ranking	Website	Usable
73	telekom.com	X
74	stackoverflow.com	X
75	amazon.com	X
76	gutefrage.net	
77	wetter.de	X
78	xhamsterlive.com	
79	demdex.net	
80	joyclub.de	X
81	kicker.de	X
82	chaturbate.com	X
83	wunderweib.de	X
85	netdoktor.de	X
86	bunte.de	X
87	youporn.com	X
89	br.de	X
90	mydealz.de	
91	markt.de	X
92	rnd.de	X
93	fr.de	X
94	weather.com	X
95	github.com	X
96	gala.de	X
99	livejasmin.com	
100	tripadvisor.de	

Table 2.: Top 100 websites in Germany (Sereda, 2022) resulting in usable html files

6.1.3. Technical Implementation Details

To evaluate all web pages, a full algorithm with logging is implemented. It starts by evaluating a webpage with two different checking algorithms. Then the fixing tool created in chapter 5 is used on the webpage. For every case where the algorithm would need developer input, assumptions are made to get the algorithm running fully autonomous. The resulting HTML file with fixed accessibility issues is then checked again, with the scores logged. This process is carried out for every webpage on the list.

6.1.4. Checking Algorithms

To have comparable results, the checking algorithm created during this thesis is used as well as the independent automatic test solution axe-selenium-python 2.1.6[9] with data reprocessing to have comparable results. This checking algorithm uses the axe-core-api by Deque Systems[10] and can be called inside python. As this algorithm can detect all accessibility problems, and not just the problems fixed by the tool created in this thesis, the total improvement can be seen. These categories are different to the WCAG categories as well as the WebAim categories. This is beneficial to avoid bias in this study. Additionally, the categories used by axe are filtered manually, to show only the problems covered in this thesis. This leads to three different scores for before and after. These different results are then compared to see the scale of improvement. The data on the frequency of the different accessibility problems can be compared to the data of the WebAim million study, to see if these findings line up to the findings in the top 100 websites in Germany.

6.2. Results

When looking at the different error types, *link-empty* is the most frequent issue, being found on 93.3% of web pages. The least frequent issue is *text-justified*, with no website showing this error.

[9]https://pypi.org/project/axe-selenium-python/
[10]https://www.deque.com/axe/

When looking at the different error types and frequencies, the values from the German subset do not line up with the values of the WebAim million counterparts, with only *alt-missing* and *link-internal-broken* being inside a 5% difference margin between each other. In total, 9 issues are more frequent in the WebAim million study and 14 issues are more frequent for the checking tool created in this thesis. In general, there is an average deviation of 27.44% between the respective higher and lower value per category. The values are a lot higher in the German data, with the values of 11 categories over 50%, when compared to the WebAim million with only 3 categories higher than 50%.

Wave Category	**Percentage WebAim Million**	**Percentage Top 100 (Germany)**
contrast	83,9%	17,3%
link_empty	50,1%	93,3%
label_missing	46,1%	37,3%
alt_link_missing	38,7%	X
alt_missing	33,8%	32%
button_empty	27,2%	66,7%
language_missing	22,3%	8%
heading_empty	13,0%	70,7%
link_redundant	71,1%	77,3%
noscript	48,5%	62,7%
heading_skipped	40,5%	32%
title_redundant	38,4%	68%
text_small	29,2%	8%
region_missing	26,0%	65,3%
h1_missing	23,8%	33,3%
heading_possible	19,7%	1,4%
alt_redundant	18,1%	56%
link_suspicious	18,0%	93,3%
alt_duplicate	14,5%	64%
table_layout	13,9%	6,7%
alt_suspicious	13,5%	86,7%
link_internal_broken	12,0%	16%
event_handler	11,4%	26,7%
text_justified	10,8%	0%

Table 3.: Data on errors-types compared between the WebAim million study and the checker created during this study

When checking the results for the checker implemented in the study, an improvement of 74.4% can be seen. When comparing the websites before and after, the accessibility problems drop from a total of 32374 individual errors on all web pages to only 8292. This is an improvement from 431.65 to 110.56 average errors per page.

For the axe data set, a smaller improvement of 18.6% for all accessibility problems and 20.7% for the filtered problems can be seen. This reduces the errors from 3411 or 45.48 per page to 2776 or 37.01 per page for the unfiltered data set, and 2888 or 38.5 per page to 2291 of 39.88 per page for the filtered data set.

It also shows the different error types per page decreasing from 7.49 different error types to 6.93 different error types in the unfiltered data set.

The filtered data set covers 84.67% of errors found in the unfiltered data set (before), showing that most of the relevant accessibility problems are covered by the selection made in this study.

	Errors before	**Errors after**	**Improvement**
AXE	3411	2776	18,6%
AXE - filtered	2888	2291	20,7%
Checking Tool	32374	8292	74,4%

Table 4.: Accessibility Issues before & after

When looking at the different error categories before and after in detail, the issue with the most total occurrences is *link-empty* with 10720 occurrences. For the error categories *heading-empty*, *link-internal-broken*, *table-layout*, *button-empty*, *title-redundant* and *noscript* every instance could be fixed. The only categories not being able to be fixed are *heading-possible* and *text-small* staying the same with a number of 2. For *link-suspicious* an increase in errors from 4648 to 6586 can be seen. All other categories show an improvement when comparing the website before and after. Another notable thing is the categories *link-empty* and *heading-empty* being found 10720 and 3383 times on the web pages, meaning empty html elements are very frequent in the data set.

Guideline Name	**Total (before)**	**Total (after)**
language_missing	6	2
region_missing	96	96
h1_missing	25	18
heading_skipped	30	1
heading_empty	3383	0
heading_possible	2	2
text_small	23	23
contrast	389	160
link_empty	10720	74
link_suspicious	4648	6586
alt_link_missing	X	X
link_redundant	1960	43
link_internal_broken	26	0
alt_missing	523	7
alt_suspicious	1423	184
alt_duplicate	1379	219
alt_redundant	844	175
table_layout	30	0
text_justified	0	0
button_empty	1256	0
label_missing	117	115
title_redundant	4801	0
event_handler	646	587
noscript	47	0

Table 5.: Data on frequency of errors-types compared between the websites before and after running the checker

7. Discussion

This section discusses the results of the study, as well as other findings during the creation of this thesis. Limitations of this thesis and resulting suggestions for future work are discussed at the end.

7.1. Results

These results show a real world benefit when using the accessibility tool, with the fixing algorithm improving the data set by 70.4% for the checker created in this thesis and 20.7% when checked with axe-selenium-python. This discrepancy between the values can be attributed to the quality of the checking algorithms. It is expected for an algorithm being marketed to achieve better performance as an algorithm created in a masters thesis. This also leads to the conclusion that with the base idea of this thesis proven to be working, developing the algorithm further to get better performance using the findings of this thesis can lead to a tool that is usable and beneficial for use in the real world. Even when looking at an improvement of 20.7%, evaluating the fixing tool further can have significant benefits for the website landscape, with the fixing tool not taking up a lot of time to run on a website. The input of a developer can be considered when issues are present that can not be fixed fully automatically, meaning this input or information needed will then be implemented into the source code automatically. This offers benefits to users with limited technical knowledge.

The coverage of 84.67% of total accessibility issues when looking at the defined subset of accessibility problems shows that the assumption of these issues having the biggest impact when compared to other categories is correct. This has only been evaluated on one data set, with the assumption coming from another dataset, further cementing this thesis. But the data set being relatively small means further research in this direction may be needed. In comparison to thew WebAim million data set, the issue frequency per websites is very different, with an average deviation of 27.44% between these data sets. This can be attributed to the performance of the checking algorithm, with a lot of the problems being more frequent. Another explanation can be the small data set, as a million of websites is compared to a data set of only 75. Even when taking these things into account, it can be said that the accessibility is worse for the top 100 German web pages, when comparing it to international values. This can lead to the assumption, that other countries have a larger focus on accessibility.

When looking at the specific problems in detail 7 of the accessibility problems could be fixed to 0 remaining problems of these categories. This means these algorithms are fully working with no further improvement needed. The problem of *link-suspicious* needs to be evaluated further, with the errors increasing significantly after the fixing algorithm has been performed. This can be attributed to the algorithm being faulty or errors in the implementation. The same can be said for the problems of *heading-possible* and *text-small*, where the problem can not be improved automatically. This indicates an error in the implementation of the algorithm, as the theoretical implementation is simple and without errors. All other algorithms work, but are not able to fix every problem that is found. This leads to the assumption of the concept being correct, and the implementation simply failing on some edge-cases. Sources of these issues can be for example links that are not working anymore or pictures directly embedded inside the html source code.

The fixing tool not being able to fix all defined issues, and other issues still remaining, means using the algorithm is no replacement for accessibility planning beforehand and manual implementation with the goal to develop fully accessible websites. Using the algorithm can be seen as a small step into the right direction while not taking up much development time, but will not replace a careful focus on accessibility.

7.2. Limitations and Further Work

As this is a toolkit created for the purpose of a study, some assumptions are made and it does not include the level of human judgement needed to be used in the real world. To use this toolkit productively, most of the algorithms can be used to create a toolkit with a level of human supervision and input added. This can be done by having a developer approve changes or giving information and then performing the changes automatically. This can be changed in future work, with AI and Machine Learning constantly gaining more capabilities. This can be especially relevant for categorizing elements that can not be classified using only the source code. Other research projects use Screen Recognition and Machine Learning models to categorize accessibility elements using the user interface instead of source code (Zhang et al., 2021). As this study is designed to work without user-interaction, not all issues can be detected and therefore fixed automatically.

With this thesis not focusing on every accessibility problem and accessibility in general, but only 24 specifically selected issues, these issues still cover over 80% of total issues. Therefore it is still worth while to look at more accessibility issues, build a full toolkit and evaluate its impact. This can then be compared to the tool only checking the selected issues, and the benefits compared when taking into account the development challenges.

When looking at the data set checked in this thesis, not every issue could be fixed automatically even if the sub-algorithm works in theory. This is because of some cases not considered in the implementation. This issue will continue to show and even increase when checking more websites, as not every website is built equal. The subset of pages only being 75 websites checked can be increased and diversified in further work, by increasing the number of pages or looking at pages from more countries than only Germany.

This algorithm is supposed to work on every html webpage, so some solutions may not be the best ones for a specific type of website, for example a news page. An algorithm for specific types of web pages could be examined in future research.

As the evaluation study was performed automatically, and only spot checks have been performed on the websites, unwanted layout changes may occur. This means if the tool is used by developers, every change that can alter the layout must be checked manually. Covering the websites before and after by checking the websites manually in addition to automatic checks, to spot problems the algorithm may have caused can be a point of future research. In further work, the algorithm can be evaluated together with developers, to get information on the real world usability.

Another limitation is only checking the front page of every website, so issues on other subpages of the same website are not covered by this study.

Using the extensive accessibility data created in this study, the top 100 websites in Germany can be evaluated at a later date. Further researchers can compare the accessibility problems at a later date to the data of May 2023 provided by this thesis, to evaluate a trend for accessibility issues in Germany.

8. Conclusion

Accessibility is an important part for websites, to make these websites usable for everyone. This stands in contrast to the low implementation rates of accessibility, with most websites failing when checked for accessibility.

This is a problem, as after 2025 conforming with these guidelines is required by law in the EU. On the other hand, a correct implementation is needed to fulfill the social responsibility to include and make usage possible for everyone regardless of their abilities.

Research shows that most of the problems holding back website accessibility can be fixed by using automation. Using this insight, this thesis implemented a semi-autonomous tool for website accessibility, as some information and interaction is needed to achieve the best results. This level of interaction and information needed from the developer can decrease in the future, with AI constantly gaining more capabilities.

The most frequent accessibility issues are evaluated for relevance and theoretical algorithm concept and an implementation in python is created. Checking this tool for improvement with the top 100 websites in Germany shows a significant improvement in website accessibility. Using this tool adds little work for developers when compared to manual code changes, but improves the websites for users in need of these features. The tool has shown to have several shortcomings that can be fixed by looking deeper into the algorithm and implementation. These problems need to be evaluated in future work and with further development.

As the base concept is proven to work, developing a tool using the concepts created and validated in this thesis can improve website accessibility dramatically, and should be considered in the future.

Concluding, the approach of automating accessibility can help make the needed step towards a more accessible internet. This can help with better including everyone, especially people affected by disabilities.

Bibliography

Al-Khalifa, A. S., & Al-Khalifa, H. S. (2011, March). An educational tool for generating inaccessible page examples based on WCAG 2.0 failures. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility* (pp. 1–4). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-01-13, from `https://doi.org/10.1145/1969289.1969328` doi: 10.1145/1969289.1969328

Al-Khalifa, H. (2012, October). WCAG 2.0 Semi-automatic Accessibility Evaluation System: Design and Implementation. *Computer and Information Science*, *5*. doi: 10.5539/cis.v5n6p73

Alonso, F., Fuertes, J. L., González, □PBI L., & Martínez, L. (2010, April). On the testability of WCAG 2.0 for beginners. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)* (pp. 1–9). New York, NY, USA: Association for Computing Machinery. Retrieved 2023-01-13, from `https://doi.org/10.1145/1805986.1806000` doi: 10.1145/1805986.1806000

Antonelli, H. L., Rodrigues, S. S., Watanabe, W. M., & de Mattos Fortes, R. P. (2018, 6 20). A survey on accessibility awareness of brazilian web developers. In *Proceedings of the 8th international conference on software development and technologies for enhancing accessibility and fighting info-exclusion* (p. 71–79). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3218585.3218598` doi: 10.1145/3218585.3218598

Brajnik, G., Yesilada, Y., & Harper, S. (2010, October). Testability and validity of WCAG 2.0: the expertise effect. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility* (pp. 43–50). New York, NY, USA: Association for Computing Machinery. Retrieved 2022-10-23, from `https://doi.org/10.1145/1878803.1878813` doi: 10.1145/1878803.1878813

Centeno, V. L., Kloos, C. D., Gaedke, M., & Nussbaumer, M. (2005, May). Web composition with WCAG in mind. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)* (pp. 38–45). New York, NY, USA: Association for Computing Machinery. Retrieved 2022-10-23, from `https://doi.org/10.1145/1061811.1061819` doi: 10.1145/1061811.1061819

de Oliveira, D. (2018, March). *JavaScript, AJAX und Dynamik.* Retrieved 2023-06-22, from `https://www.netz-barrierefrei.de/wordpress/`

barrierefreies-internet/konzeption/javascript-ajax-und-dynamik/

dequelabs. (2023, June). *axe-core.* Retrieved 2023-06-22, from https://github.com/dequelabs/axe-core (original-date: 2015-06-10T15:26:45Z)

Destatis. (2022). *Behinderte menschen.* Retrieved 2023-06-20, from https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Behinderte-Menschen/_inhalt.html

European Parliament. (2019, April). *Directive (EU) 2019/882 of the European Parliament and of the Council of 17 April 2019 on the accessibility requirements for products and services (Text with EEA relevance).* Retrieved 2023-06-22, from http://data.europa.eu/eli/dir/2019/882/oj/eng

Hein, R. (2010, December). *How Many Users Have JavaScript Disabled.* Retrieved 2023-06-22, from https://www.searchenginepeople.com/blog/stats-no-javascript.html

Ikhsan, I. N., & Catur Candra, M. Z. (2018, November). Automatically: An Automated Refactoring Method and Tool for Improving Web Accessibility. In *2018 5th International Conference on Data and Software Engineering (ICoDSE)* (pp. 1–6). (ISSN: 2640-0227) doi: 10.1109/ICODSE.2018.8705894

Initiative (WAI) W3C Web Accessibility. (2021). *Mobile Accessibility at W3C.* Retrieved 2023-06-22, from https://www.w3.org/WAI/standards-guidelines/mobile/

M., N., Chawla, P., & Rana, A. (2019, February). A Practitioner's Approach to Assess the WCAG 2.0 Website Accessibility Challenges. In *2019 Amity International Conference on Artificial Intelligence (AICAI)* (pp. 958–966). doi: 10.1109/AICAI.2019.8701320

Rømen, D., & Svanæs, D. (2012, November). Validating WCAG versions 1.0 and 2.0 through usability testing with disabled users. *Universal Access in the Information Society*, *11*(4), 375–385. Retrieved 2022-10-23, from https://doi.org/10.1007/s10209-011-0259-3 doi: 10.1007/s10209-011-0259-3

Sereda, E. (2022). *Die meistbesuchten und meistaufgerufenen websites in deutschland — top 100 in 2021 und 2022 (website ranking).* Retrieved 2023-04-03, from https://de.semrush.com/blog/top-der-meistbesuchten-webseiten/

UK Government Propriety and Civil Service. (2016, November). *Results of the 2016 GOV.UK assistive technology survey - Accessibility in government.* Retrieved 2023-06-22, from https://accessibility.blog.gov.uk/2016/11/01/results-of-the-2016-gov-uk-assistive-technology-survey/

WebAim. (2023). *Wave documentation.* Retrieved 2023-05-06, from `https://wave.webaim.org/api/docs?format=html`

WebAim. (2023). *The webaim million - the 2023 report on the accessibility of the top 1,000,000 home pages.* Retrieved 2023-04-13, from `https://webaim.org/projects/million/`

World Wide Web Consortium. (2018). *Web content accessibility guidelines (wcag) 2.1.* Retrieved 2023-04-01, from `https://www.w3.org/TR/WCAG21/`

World Wide Web Consortium. (2023, June). *Web content accessibility guidelines (wcag) 2.2.* Retrieved 2023-06-01, from `https://www.w3.org/TR/WCAG22/`

Zhang, X., de Greef, L., Swearngin, A., White, S., Murray, K., Yu, L., ... Bigham, J. P. (2021, January). *Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels* (Tech. Rep.). Retrieved 2023-01-13, from `http://arxiv.org/abs/2101.04893` (arXiv:2101.04893 [cs] type: article) doi: 10.48550/arXiv.2101.04893

A. Appendix

A.1. Webaim Million Data

Wave Category	**Class**	**Percentage**
contrast	Error	83,9%
link_empty	Error	50,1%
label_missing	Error	46,1%
alt_link_missing	Error	38,7%
alt_missing	Error	33,8%
button_empty	Error	27,2%
language_missing	Error	22,3%
heading_empty	Error	13,0%
link_redundant	Warning	71,1%
noscript	Warning	48,5%
heading_skipped	Warning	40,5%
title_redundant	Warning	38,4%
text_small	Warning	29,2%
region_missing	Warning	26,0%
h1_missing	Warning	23,8%
heading_possible	Warning	19,7%
alt_redundant	Warning	18,1%
link_suspicious	Warning	18,0%
alt_duplicate	Warning	14,5%
table_layout	Warning	13,9%
alt_suspicious	Warning	13,5%
link_internal_broken	Warning	12,0%
event_handler	Warning	11,4%
text_justified	Warning	10,8%

Table 6.: All selected accessibility problems. Data based on WebAim million (WebAim, 2023)

Erklärung zur Urheberschaft

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie alle Zitate und Übernahmen von fremden Aussagen kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt.

Die vorgelegten Druckexemplare und die vorgelegte digitale Version sind identisch.

Von den zu § 27 Abs. 5 der Prüfungsordnung vorgesehenen Rechtsfolgen habe ich Kenntnis.

Regensburg, 23.06.2023

Signature

Erklärung zur Lizenzierung und Publikation dieser Arbeit

Name: Mathias Götz

Titel der Arbeit: *Design and Evaluation of a Semi-autonomous Accessilility Tool for Web Development*

Hiermit gestatte ich die Verwendung der schriftlichen Ausarbeitung zeitlich unbegrenzt und nicht-exklusiv unter folgenden Bedingungen:

- ☐ Nur zur Bewertung dieser Arbeit
- ☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre
- ☒ Unter einer Creative-Commons-Lizenz mit den folgenden Einschränkungen:
  - ☒ BY – Namensnennung des Autors
  - ☐ NC – Nichtkommerziell
  - ☐ SA – Share-Alike, d.h. alle Änderungen müssen unter die gleiche Lizenz gestellt werden.

(An Zitaten und Abbildungen aus fremden Quellen werden keine weiteren Rechte eingeräumt.)

Außerdem gestatte ich die Verwendung des im Rahmen dieser Arbeit erstellten Quellcodes unter folgender Lizenz:

- ☐ Nur zur Bewertung dieser Arbeit
- ☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre
- ☐ Unter der CC-0-Lizenz (= beliebige Nutzung)
- ☒ Unter der MIT-Lizenz (= Namensnennung)
- ☐ Unter der GPLv3-Lizenz (oder neuere Versionen)

(An explizit mit einer anderen Lizenz gekennzeichneten Bibliotheken und Daten werden keine weiteren Rechte eingeräumt.)

Ich willige ein, dass der Lehrstuhl für Medieninformatik diese Arbeit – falls sie besonders gut ausfällt - auf dem Publikationsserver der Universität Regensburg veröffentlichen lässt.

Ich übertrage deshalb der Universität Regensburg das Recht, die Arbeit elektronisch zu speichern und in Datennetzen öffentlich zugänglich zu machen. Ich übertrage der Universität Regensburg ferner das Recht zur Konvertierung zum Zwecke der Langzeitarchivierung unter Beachtung der Bewahrung des Inhalts (die Originalarchivierung bleibt erhalten).

Ich erkläre außerdem, dass von mir die urheber- und lizenzrechtliche Seite (Copyright) geklärt wurde und Rechte Dritter der Publikation nicht entgegenstehen.

- ☒ Ja, für die komplette Arbeit inklusive Anhang
- ☐ Ja, für eine um vertrauliche Informationen gekürzte Variante (auf dem Datenträger beigefügt)
- ☐ Nein
- ☐ Sperrvermerk bis (Datum):

Regensburg, 23.06.2023

Signature

Contents of added flash disk

`/1_Thesis`	Thesis as PDF and Latex File
`/2_Code`	Source Code for the Algorithm
`/3_Study/Design`	Source Code for the Study
`/3_Study/RAW`	Raw Data of the Study (Websites before & after etc.)
`/3_Study/CSV`	All evaluation CSV Files created for the study

The contents of the attached CD/flash disk can also be found in the Github repo MA-Semi-autonomous-Accessibility-Tool, excluding the downloaded and changed websites as well as WebAim data.